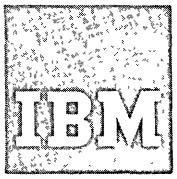
File No. S360-24
Form GC28-6398-1

DOS



Systems Reference Library

IBM System/360 Disk Operating System

American National Standard COBOL

Programmer's Guide

Program Number 360N-CB-482

This publication describes how to compile an American National Standard COBOL X3.23-1968 program using the IBM System/360 Disk Operating System American National Standard Full COBOL Compiler Version 2. It also describes how to linkage edit the resulting object module, and execute the program. Included is a description of the output from each of these three steps: compile, linkage edit, and execute. In addition, this publication explains features of the compiler and available options of the operating system. American National Standard COBOL was formerly known as USA Standard COBOL.

PREFACE

This publication is logically and functionally divided into two parts. Part I contains information useful to programmers who are running IBM American National Standard COBOL programs, i.e., programs compiled on the Version 2 Compiler, under the control of the IBM System/360 Disk Operating System. Part I covers such topics as job control language, library usage, interpreting output, and program debugging. Part I is intended solely as object-time reference material.

Part II contains supplemental information on the use of the language as specified in the publication IBM System/360 Disk Operating System: American National Standard COBOL, Form GC28-6394, and should be used in conjunction with this publication for coding IBM American National Standard COBOL programs. Part II covers in detail such topics as file organization, file label handling, and record formats. Part II is intended as source-time reference material for language features that are primarily system-dependent.

Wider and more detailed discussions of the Disk Operating System are given in the following publications:

IBM System/360 Disk Operating System: System Control and System Service Programs, Form GC24-5036

IBM System/360 Disk Operating System: Supervisor and Input/Output Macros, Form GC24-5037

IBM System/360 Disk Operating System: Data Management Concepts, Form GC24-3427

IBM System/360 Disk Operating System: System Generation and Maintenance, Form GC24-5033

IBM System/360 Principles of Operation, Form GA24-6821

The titles and abstracts of related publications are listed in the publication IBM System/360 Bibliography, Form GA22-6822.

Second Edition (February 1970)

This edition is a major revision of Form GC28-6398-0 and makes that edition and its associated Technical Newsletter, Form N28-0263, obsolete. The specifications in this publication correspond to Release 22/23 of the IBM System/360 Disk Operating System. This edition contains changes and additions that reflect the Version 2 Compiler's support of the following features: relative track addressing for direct files, spanned records on sequential tape files and on direct files, and forced end-of-volume for a sequentially organized file on a direct access device. In addition, changes have been made throughout the publication to correct and clarify specific items. All technical changes are indicated by a vertical line to the left of the change; revised or new illustrations are denoted by the symbol • to the left of the caption.

Changes are continually made to the specifications herein; any such changes will be reported in subsequent revisions or Technical Newsletters. Before using this publication in connection with the operation of IBM systems, refer to the latest SRL Newsletter, Form GN20-0360, for editions that are applicable and current.

Requests for copies of IBM publications should be made to your IBM representative or the IBM branch office serving your locality.

A form for readers' comments is provided at the back of this publication. If the form has been removed, comments may be addressed to IBM Corporation, Programming Publications, 1271 Avenue of the Americas, New York, New York 10020.

PART I

INTRODUCTION 11
Control Program 11
Supervisor 11
Job Control Processor 11
Initial Program Loader 11
Processing Programs 11
System Service Programs 11
Application Programs 12
IBM-Supplied Processing Programs . . . 12
Multiprogramming 12
Background vs. Foreground Programs . . 12

JOB DEFINITION 13
Job Steps 13
Compilation Job Steps 13
Multiphase Program Execution 13
Types of Jobs 14
Job Definition Statements 15
Other Job Control Statements 16

JOB PROCESSING 17
Compilation 17
Editing 17
Phase Execution 18
Multiphase Programs 18

PREPARING COBOL PROGRAMS FOR PROCESSING 19
Assignment of Input/Output Devices . . . 19
Job Control 21
Job Control Statements 21
Comments in Job Control Statements . 21
Statement Formats 21
Sequence of Job Control Statements . 22
Description and Formats of Job
Control Statements 22
ASSGN Statement 22
CLOSE Statement 24
DATE Statement 24
TLBL Statement 25
DLBL Statement 26
EXTENT Statement 26
VOL Statement 28
DLAB Statement 29
TPLAB Statement 30
XTENT Statement 30
JOB Statement 31
LBLTYP Statement 31
LISTIO Statement 32
MTC Statement 32
OPTION Statement 32
PAUSE Statement 34
RESET Statement 34
RSTRT Statement 35
UPSI Statement 35
CBL Statement -- COBOL Option
Control Card 35
Job Control Commands 37
Linkage Editor Control Statements . . . 37
Control Statement Placement 38
PHASE Statement 38
INCLUDE Statement 39
ENTRY Statement 39
ACTION Statement 39
Autolink Feature 40

LIBRARIAN FUNCTIONS 41
Librarian 41
Core Image Library 41
Cataloging and Retieving Program
Phases -- Core Image Library 41
Relocatable Library 42
Maintenance Functions 42
Cataloging a Module -- Relocatable
Library 42
Source Statement Library 43
Maintenance Functions 43
Cataloging a Book -- Source
Statement Library 43
Updating Books -- Source Statement
Library 45
Logical Unit Assignment and Control
Statement Placement: 47
UPDATE Function -- Invalid Operand
Defaults 47
Private Libraries 48
Source Language Considerations 48
Extended Source Program Library
Facility 48

PROGRAM CHECKOUT 51
Debug Language 51
Flow of Control 51
Displaying Data Values During
Execution 52
Testing a Program Selectively 54
Testing Changes and Additions to
Programs 54
Dumps 54
How to Use a Dump 55
Errors That Can Cause a Dump 55
Locating a DTF 56
Locating Data 57
Diagnostic Messages 64
Working with Diagnostic Messages . . . 64
Generation of Diagnostic Messages . . 64
Linkage Editor Output 64
Execution Time Messages 64
Recording Program Status 64
RERUN Clause 65
Taking a Checkpoint 65
Restarting a Program 66

INTERPRETING OUTPUT 67
Compiler Output 67
Object Module 75
Linkage Editor Output 75
Comments on the Phase Map 77
Linkage Editor Messages 77
COBOL Phase Execution Output 77
Operator Messages 78
STOP Statement 78
ACCEPT Statement 78

System Output 79

CALLING AND CALLED PROGRAMS 81
Linkage 81
Linkage In A Calling Program 81
Linkage In A Called Program 82
Entry Points 82
Correspondence of Arguments and Parameters 82
Linkage Editing Without Overlay 83
Assembler Language Subprograms 84
Register Use 84
Save Area 84
Argument List 84
In-Line Parameter List 85
Lowest Level Program 87
Overlays 87
Special Considerations When Using Overlay Structures 87
Assembler Language Subroutine for Accomplishing Overlay 88
Linkage Editing with Overlay 89
Job Control for Accomplishing Overlay 90

USING THE SORT FEATURE 95
Sort Job Control Requirements 95
Sort Input and Output Control Statements 95
Sort Work File Control Statements . . 96
Amount of Intermediate Storage Required 96
Improving Performance 96
Sort Diagnostic Messages 96
Linkage with the Sort Feature 96
Completion Codes 97
Checkpoint/Restart During a Sort 97

USING THE SEGMENTATION FEATURE 99
Operation 99
Output From a Segmented Program 100
Compiler Output 100
Linkage Editor Output 101
Cataloging a Segmented Program . . .101
Determining the Priority of the Last Segment Loaded into the Transient Area 101
Sort in a Segmented Program 101

PART II

PROCESSING COBOL FILES ON MASS STORAGE DEVICES 107
File Organization 107
Sequential Organization 107
Direct Organization 107
Indexed Organization 107
Data Management Concepts 108
Sequential Organization (DTFSD) 109
Processing a Sequentially Organized File 109
Direct Organization (DTFDA) 109
Accessing a Directly Organized File .110
ACTUAL KEY Clause 111
Randomizing Techniques 112
Actual Track Addressing Considerations for Specific Devices .125
Randomizing for the 2311 Disk Drive 125
Randomizing for the 2321 Data Cell .126

Indexed Organization (DTFIS) 127
Prime Area 127
Indexes 128
Track Index 128
Cylinder Index 128
Master Index 128
Overflow Area 128
Cylinder Overflow Area 128
Independent Overflow Area 128
Adding Records to an Indexed File .128
Accessing an Indexed File (DTFIS) . .130
Key Clauses 130
Improving Efficiency 131

ADVANCED PROCESSING CAPABILITIES 133
DTF Tables 133
Pre-DTF Switch 138
Error Recovery 138
Volume and File Label Handling 144
Tape Labels 144
Volume Labels 144
Standard File Labels 144
User Labels 144
Nonstandard Labels 144
Label Processing Considerations . . .148
Mass Storage File Labels 149
Volume Labels 149
Standard File Labels 149
User Labels 150
Label Processing Considerations . . .150
Files on Mass Storage Device Opened as Input 150
Files on Mass Storage Devices Opened as Output 150
Unlabeled Files 151

RECORD FORMATS 153
Fixed-length (Format F) Records 153
Undefined (Format U) Records 153
Variable-length (Format V) Records . . .154
APPLY WRITE-ONLY Clause 157
Spanned (Format S) Records 157
S-Mode Capabilities 158
Sequentially Organized S-Mode Files on Tape or Mass Storage Devices . . .159
Source Language Considerations . . .159
Processing Sequentially Organized S-Mode Files 159
Directly Organized S-Mode Files . . .161
Source Language Considerations . . .161
Processing Directly Organized S-Mode Files 162
OCCURS Clause with the DEPENDING ON Option 162

PROGRAMMING TECHNIQUES 165
General Considerations 165
Spacing the Source Program Listing .165
Environment Division 165
SELECT Sentence 165
APPLY WRITE-ONLY Clause 165
Data Division 165
Overall Considerations 165
Prefixes 165
Level Numbers 166
File Section 166
RECORD CONTAINS Clause 166
Working-Storage Section 166

Separate Modules 166
Locating the Working-Storage Section in Dumps 166
Data Description 167
REDEFINES Clause 167
PICTURE Clause 167
USAGE Clause 169
SYNCHRONIZED Clause 172
Special Considerations for DISPLAY and COMPUTATIONAL Fields 172
Data Formats in the Computer 172
Procedure Division 174
Modularizing the Procedure Division .174
Main-Line Routine 174
Processing Subroutines 174
Input/Output Subroutines 175
Intermediate Results 175
Intermediate Results and Binary Data Items 175
Intermediate Results and COBOL Library Subroutines 175
Intermediate Results Greater Than 30 Digits 175
Intermediate Results and Floating-point Data Items 175
Intermediate Results and the ON SIZE ERROR Option 176
Procedure Division Statements 176
COMPUTE Statement 176
IF Statement 176
MOVE Statement 176
NOTE Statement 176
PERFORM Statement 176
READ INTO and WRITE FROM Options . . .176
TRACE Statement 177
TRANSFORM Statement 177
Using the Report Writer Feature 177
REPORT Clause in a File Description (FD) Entry 177
Summing Techniques 177
Use of SUM 177
SUM Routines 178
Output Line Overlay 179
Page Breaks 179
WITH CODE Clause 179
Control Footings and Page Format . .180
NEXT GROUP Clause 181
Floating First Detail 181
Report Writer Routines 181
Table Handling Considerations 181
Subscripts 181
Index-names 182
Index Data Items 182
OCCURS Clause 182
DEPENDING ON Option 182
SEARCH ALL Statement 183
SET Statement 183
SEARCH Statement 185
Building Tables 186

APPENDIX A: SAMPLE PROGRAM OUTPUT . . .187

APPENDIX B: STANDARD TAPE FILE LABELS .201

APPENDIX C: STANDARD MASS STORAGE DEVICE LABELS 203

APPENDIX D: TRACK FORMATS FOR THE 2311, 2314, AND 2321 DIRECT-ACCESS STORAGE DEVICES 209

APPENDIX E: COBOL LIBRARY SUBROUTINES .211
Input/Output Subroutines 211
Printer Spacing 211
Tape and Sequential Disk Labels . .211
CLOSE WITH LOCK Subroutine 211
WRITE Statement Subroutines 211
READ Statement Subroutines 211
REWRITE Statement Subroutines . . .211
DISPLAY (EXHIBIT and TRACE) Subroutines 211
ACCEPT and STOP (literal) Statement Subroutines 212
CLOSE Subroutine 212
Multiple File Tape Subroutine . . .212
Input/Output Error Subroutines . . .212
Disk Extent Subroutines 212
Auxiliary Subroutines 212
Conversion Subroutines 212
Arithmetic Verb Subroutines 214
Sort Feature Interface Routine 214
Checkpoint (RERUN) Subroutine 214
Segmentation Feature Subroutine . . .214
Other Verb Routines 214
Compare Subroutines 214
MOVE Subroutines 215
TRANSFORM Subroutine 215
Class Test Subroutine 215
SEARCH Subroutine 215
Main Program or Subprogram Subroutine 215

APPENDIX F: DIAGNOSTIC MESSAGES 217
Compiler Diagnostic Messages 217
Object Time Messages 234
COBOL Object Program Unnumbered Messages 235

APPENDIX G: MACHINE CONSIDERATIONS . . .237
Minimum Machine Requirements for the Compiler 237
Execution Time Considerations 237
Sort Feature Considerations 237

APPENDIX H: COMMUNICATION REGION 239
Communication Region 239

APPENDIX I: SAMPLE JOB DECKS 241
Direct Files 242
Creating a Direct File 242
Retrieving and Updating a Direct File 242
Indexed Files 243
Creating an Indexed File 243
Retrieving and Updating an Indexed File 244
Files Used in a Sort Operation 244
Sorting an Unlabeled Tape File 244

INDEX 245

ILLUSTRATIONS

FIGURES

Figure 1. Sample Structure of Job Deck for Compiling, Linkage Editing, and Executing a Main Program and Two Subprograms 13
Figure 2. Sample Logical Unit Assignments 19
Figure 3. Possible Specifications for X'ss' in the ASSGN Control Statement . 24
Figure 4. Sample Label and File Extent Information for Mass Storage Files 28
Figure 5. Job Definition -- Use of the Librarian 38
Figure 6. Sample Coding to Calculate FICA 49
Figure 7. Altering a Program from the Source Statement Library Using INSERT and DELETE Cards 49
Figure 8. Effect of INSERT and DELETE Cards 50
Figure 9. Sample Output of EXHIBIT Statement with the CHANGED NAMED Option 53
Figure 10. Sample Dump Resulting from Abnormal Termination 58
Figure 11. Examples of Compiler Output . 68
Figure 12. Linkage Editor Output . . . 76
Figure 13. Output from Execution Job Steps 78
Figure 14. Calling and Called Programs 81
Figure 15. Example of Data Flow Logic in a Call Structure 83
Figure 16. Sample Linkage Routines Used with a Calling Subprogram 86
Figure 17. Sample In-line Parameter List 87
Figure 18. Sample Linkage Routines Used with a Lowest Level Subprogram . . 87
Figure 19. Example of an Assembler Language Subroutine for Accomplishing Overlay 88
Figure 20. Flow Diagram of Overlay Logic 89
Figure 21. Job Control for Accomplishing Overlay 90
Figure 22. Calling Sequence to Obtain Overlay Between Three COBOL Subprograms 91
Figure 23. Segmenting the Program SAVECORE 99
Figure 24. Storage Layout for SAVECORE 100
Figure 25. Compiler Output for SAVECORE 101
Figure 26. Linkage Editing a Segmented Program 102
Figure 27. Location of Sort Program in a Segmentation Structure 103
Figure 28. Structures of the Actual Key 111
Figure 29. Permissible Specifications for the First Eight Bytes of the Actual Key 112
Figure 30. Creating a Direct File Using Method B 116
Figure 31. Creating a Direct File with Relative Track Addressing Using Method B 121
Figure 32. Formats of Blocked and Unblocked Records 127
Figure 33. Adding a Record to a Prime Track 129
Figure 34. Standard Tape File Label and TPLAB Cards 145
Figure 35. Standard Tape File Label and TLBL Card (Showing Maximum Specifications) 146
Figure 36. Standard Tape File Label and TLBL Card (Showing Minimum Requirements) 147
Figure 37. Standard, User, and Volume Labels 148
Figure 38. Nonstandard Labels 148
Figure 39. Fixed-Length (Format F) Records 153
Figure 40. Undefined (Format U) Records 154
Figure 41. Unblocked V-Mode Records .154
Figure 42. Blocked V-Mode Records . .155
Figure 43. Fields in Unblocked V-Mode Records 156
Figure 44. Fields in Blocked V-Mode Records 156
Figure 45. First Two Blocks of VARIABLE-FILE-2 157
Figure 46. Control Fields of an S-Mode Record 158
Figure 47. One Logical Record Spanning Physical Blocks 159
Figure 48. First Four Blocks of SPAN-FILE 160
Figure 49. Advantage of S-Mode Records Over V-Mode Records 160
Figure 50. Direct and Sequential Spanned Files on a Mass Storage Device 161
Figure 51. Calculating Record Lengths When Using the OCCURS Clause with the DEPENDING ON Option 164
Figure 52. Treatment of Varying Values in a Data Item of PICTURE S9 . .174
Figure 53. Sample of GROUP INDICATE Clause and Resultant Execution Output .179
Figure 54. Format of a Report Record When the CODE Clause is Specified . . .180
Figure 55. Activating the NEXT GROUP Clause 181
Figure 56. Table Structure in Core Storage 184
Figure 57. Track Format 210
Figure 58. Communication Region in the Supervisor 239

Table 1. Job Control Statements . . . 16
Table 2. Symbolic Names, Functions, and Permissible Device Types 20
Table 3. Glossary Definition and Usage 73
Table 4. Symbols Used in the Listing and Glossary to Define Compiler-Generated Information 74
Table 5. System Message Identification Codes 79
Table 6. Conventional Use of Linkage Registers 84
Table 7. Save Area Layout and Word Contents 85
Table 8. Recording Capacities of Mass Storage Devices 107
Table 9. Partial List of Prime Numbers . . .114
Table 10. Fields Preceding DTFMT and DTFSD 134
Table 11. Fields Preceding DTFDA - ACCESS IS RANDOM - Actual Track Addressing 134
Table 12. Fields Preceding DTFDA - ACCESS IS RANDOM - Relative Track Addressing 135
Table 13. Fields Preceding DTFDA - ACCESS IS SEQUENTIAL - Actual Track Addressing 136
Table 14. Fields Preceding DTFDA - ACCESS IS SEQUENTIAL - Relative Track Addressing 137
Table 15. Fields Preceding DTFIS . . .137
Table 16. Meaning of Pre-DTF Switch . .138
Table 17. Errors Causing an Invalid Key Condition 139
Table 18. Meaning of Error Bytes for GIVING Option of Error Declarative . . .140
Table 19. Location and Meaning of Error Bits for DTFMT 142
Table 20. Location and Meaning of Error Bits for DTFSD 142
Table 21. Location and Meaning of Error Bits for DTFDA 142
Table 22. Location and Meaning of Error Bits for DTFIS 143
Table 23. Data Format Conversion . . .170
Table 24. Relationship of PICTURE to Storage Allocation 173
Table 25. Rules for the SET Statement .185
Table 26. Functions of COBOL Library Conversion Subroutines 213
Table 27. Functions of COBOL Library Arithmetic Subroutines 214

- INTRODUCTION
- JOB DEFINITION
- JOB PROCESSING
- PREPARING COBOL PROGRAMS FOR PROCESSING
- LIBRARIAN FUNCTIONS
- PROGRAM CHECKOUT
- INTERPRETING OUTPUT
- CALLING AND CALLED PROGRAMS
- USING THE SEGMENTATION FEATURE
- USING THE SORT FEATURE

In the years since 1959, COBOL has undergone considerable refinement and standardization. Now, an extensive subset for a standard COBOL has been specified by the American National Standards Institute, an industry-wide association of computer manufacturers and users. This standard is called American National Standard COBOL. IBM American National Standard COBOL is compatible with American National Standard COBOL and includes a number of extensions to it as well.

An IBM American National Standard COBOL program may be processed by the IBM System/360 Disk Operating System. Under control of the operating system, a set of IBM American National Standard COBOL source statements is translated to form a module. In order to be executed, the module in turn must be processed to form a phase. The reasons for this will become clear later. For now it is sufficient to note that the flow of an IBM American National Standard COBOL (herein, simply termed COBOL) program through the operating system is from source statements to module to phase.

The Disk Operating System consists essentially of a control program and a number of processing programs.

CONTROL PROGRAM

The components of the control program are: the Supervisor, Job Control Processor, and the Initial Program Loader.

SUPERVISOR

The main function of the Supervisor is to provide an orderly and efficient flow of jobs through the operating system. (A job is some specified unit of work, such as the processing of a COBOL program.) The Supervisor loads into the computer the phases that are to be executed. During execution of the program, control usually alternates between the Supervisor and the processing program. The Supervisor, for example, handles all requests for input/output operations.

JOB CONTROL PROCESSOR

The primary function of the Job Control Processor is the processing of job control statements. Job control statements describe the jobs to be performed and specify the programmer's requirements for each job. Job control statements are written by the programmer using the job control language. The use of job control statements and the rules for specifying them are discussed later.

INITIAL PROGRAM LOADER

The Initial Program Loader (IPL) routine loads the Supervisor into main storage when system operation is initiated. Detailed information about the Initial Program Loader need not concern the COBOL programmer. Anyone interested in this material, however, can find it in the publication IBM System/360 Disk Operating System: System Control and System Service Programs.

PROCESSING PROGRAMS

The processing programs include the COBOL compiler, service programs, and application programs.

SYSTEM SERVICE PROGRAMS

The system service programs provide the functions of generating the system, creating and maintaining the library sections, and editing programs into disk residence before execution. The system service programs are:

1. Linkage Editor. The Linkage Editor processes modules and incorporates them into phases. A single module can be edited to form a single phase, or several modules can be edited or linked together to form one executable phase. Moreover, a module to be processed by the Linkage Editor may be one that was just created (during the same job) or one that was created in a previous job and saved.

The programmer instructs the Linkage Editor to perform these functions through job control statements. In addition, there are several linkage editor control statements. Information on their use is given later.

2. Librarian. The Librarian consists of a group of programs used for generating the system, maintaining and reorganizing the disk library areas, and providing printed and punched output from the libraries. The three system libraries are: the core image library, the relocatable library, and the source statement library. In addition, the Librarian supports private relocatable and source statement libraries. Detailed information on the Librarian is given later.

APPLICATION PROGRAMS

Application programs are usually programs written in a higher-level programming language (e.g., COBOL). All application programs within the Disk Operating System are executed under the supervision of the control program.

IBM-SUPPLIED PROCESSING PROGRAMS

The following are examples of IBM-supplied processing programs:

1. Language translators, e.g., COBOL compiler
2. Sort/Merge
3. Utilities
4. Autotest

MULTIPROGRAMMING

For those systems with main storage equal to or in excess of 24K bytes, the Disk Operating System offers multiprogramming support. In addition to at least 24K bytes of main storage, multiprogramming support requires the storage protection feature.

Multiprogramming refers to the ability of the system to control more than one program concurrently by interleaving their execution. This support is referred to as fixed partitioned multiprogramming, since programs are assigned to fixed locations when they are cataloged to the system. Each program occupies a contiguous area of main storage. The amount of main storage allocated to programs to be executed may be determined when the system is generated, or it may be determined by the operator when the program is loaded into main storage for execution.

BACKGROUND VS. FOREGROUND PROGRAMS

There are two types of problem programs in multiprogramming: background and foreground. Background programs are initiated by the Job Control Processor from batched-job input streams. Foreground programs may operate in either the batched-job mode or in the single-program mode. Single-program foreground programs are initiated by the operator from the printer-keyboard. When one program is completed, the operator must explicitly initiate the next program.

Background and foreground programs initiate and terminate independently of one another. Neither is aware of the other's status or existence.

The system is capable of concurrently operating one background program and one or two foreground programs. Priority for CPU processing is controlled by the Supervisor with foreground programs having priority over background programs. Control is taken away from a high priority program when that program encounters a condition that prevents continuation of processing, until a specified event has occurred. Control is taken away from a lower priority program when an event for which a higher priority program was waiting has been completed. Interruptions are received and processed by the Supervisor.

COBOL source modules must be compiled as background programs. COBOL program phases can be executed as either background or foreground programs.

A job is a specified unit of work to be performed under control of the operating system. A typical job might be the processing of a COBOL program -- compiling source statements, editing the module produced to form a phase, and then executing the phase. Job definition -- the process of specifying the work to be done during a single job -- allows the programmer considerable flexibility. A job can include as many or as few job steps as the programmer desires.

JOB STEPS

A job step is exactly what the name implies -- one step in the processing of a job. Thus, in the job mentioned above, one job step is the compilation of source statements; another is the linkage editing of a module; another is the execution of a phase. In contrast to a job definition, the definition of a job step is fixed. Each job step involves the execution of a program, whether it be a program that is part of the Disk Operating System or a program that is written by the user. A compilation requires the execution of the COBOL compiler. Similarly, an editing implies the execution of the Linkage Editor Finally, the execution of a phase is the execution of the problem program itself.

Compilation Job Steps

The compilation of a COBOL program may necessitate more than one job step (more than one execution of the COBOL compiler). In some cases, a COBOL program consists of a main program and one or more subprograms. To compile such a program, a separate job step must be specified for the main program and for each of the subprograms. Thus, the COBOL compiler is executed once for the main program and once for each subprogram. Each execution of the compiler produces a module. The separate modules can then be combined into one phase by a single job step -- the execution of the Linkage Editor.

For a COBOL program that consists of a main program and two subprograms, compilation and execution require five steps: (1) compile (main program), (2) compile (first subprogram), (3) compile (second subprogram), (4) linkage edit (three modules combined into one phase), and (5) execute (phase). Figure 1 shows a sample structure of the job deck for these five job steps. Compilation and execution in three job steps -- compile, linkage edit, and execute -- is applicable only when the COBOL source program is a single main program.

```
// JOB PROG1
.
.
.
// EXEC FCOBOL
   {source deck - main program}
/*
.
.
.
// EXEC FCOBOL
   {source deck - first subprogram}
/*
.
.
.
// EXEC FCOBOL
   {source deck - second subprogram}
/*
.
.
.
// EXEC LNKEDT
.
.
.
// EXEC
```

Figure 1. Sample Structure of Job Deck for Compiling, Linkage Editing, and Executing a Main Program and Two Subprograms

Multiphase Program Execution

The execution of a COBOL program has thus far been referred to as the execution of a phase. It is possible, however, to organize a COBOL program so that it is executed as two or more phases. Such a program is known as a multiphase program.

By definition, a phase is that portion of a program that is loaded into main storage by a single operation of the Supervisor. A COBOL program can be executed as a single phase only if there is an area of main storage available to

accommodate all of it. A program that is too large to be executed as a single phase must be structured as a multiphase program. The technique that enables the programmer to use subprograms that do not fit into main storage (along with the main program) is called overlay.

The number of phases in a COBOL program has no effect on the number of job steps required to process that program. As will be seen, the Linkage Editor can produce one or more phases in a single job step. Similarly, both single-phase and multiphase programs require only one execution job step. Phase execution is the execution of all phases that constitute one COBOL program.

Detailed information on overlay structures, as well as information on using the facilities of the operating system to create multiple phases and to execute them, can be found in the chapter "Calling and Called Programs."

TYPES OF JOBS

A typical job falls into one of several categories. A brief description of these categories follows; a complete discussion is found in the chapter "Preparing COBOL Programs for Processing."

Compile-Only: This type of job involves only the execution of the COBOL compiler. It is useful when checking for errors in COBOL source statements. A compile-only job is also used to produce a module that is to be further processed in a subsequent job.

A compile-only job can consist of one job step or several successive job steps.

Edit-Only: This type of job involves only the execution of the Linkage Editor. It is used primarily to combine modules produced in previous compile-only jobs, and to check that all cross references between modules have been resolved. The programmer can specify that all modules be combined to form one phase; or he can specify that some modules form one phase and that others form additional phases. The phase output produced as the result of an edit-only job can be retained for execution in a subsequent job.

Compile and Edit: This type of job combines the functions of the compile-only and the edit-only jobs. It requires the execution of both the COBOL compiler and the Linkage Editor. The job can include one or more compilations, resulting in one or more modules. The programmer can specify that the Linkage Editor process any or all of the modules just produced; in addition, he can specify that one or more previously produced modules be included in the linkage editor processing.

Execute-Only: This type of job involves the execution of a phase (or multiple phases) produced in a previous job. Once a COBOL program has been compiled and edited successfully, it can be retained as one or more phases and executed whenever needed. This eliminates the need for recompiling and re-editing every time a COBOL program is to be executed.

Edit and Execute: This type of job combines the functions of the edit-only and the execute-only jobs. It requires the execution of both the Linkage Editor and the resulting phase(s).

Compile, Edit, and Execute: This type of job combines the functions of the compile and edit and the execute-only jobs. It calls for the execution of the COBOL compiler, the Linkage Editor, and the problem program; that is, the COBOL program is to be completely processed.

When considering the definition of his job, the programmer should be aware of the following: if a job step is cancelled during execution, the entire job is terminated; any remaining job steps are skipped. Thus, in a compile-edit-and execute job, a failure in compilation precludes the editing of the module(s) and phase execution. Similarly, a failure in editing precludes phase execution.

For this reason, a job usually should (but need not) consist of related job steps only. For example, if two independent single-phase executions are included in one job, the failure of the first phase execution precludes the execution of the second phase. Defining each phase execution as a separate job would prevent this from happening. If successful execution of both phases can be guaranteed before the job is run, however, the programmer may prefer to include both executions in a single job.

JOB DEFINITION STATEMENTS

Once the programmer has decided the work to be done within his job and how many job steps are required to perform the job, he can then define his job by writing job control statements. Since these statements are usually punched in cards, the set of job control statememts is referred to as a job deck. In addition to job control statements, the job deck can include input data for a program that is executed during a job step. For example, input data for the COBOL compiler -- the COBOL program to be compiled -- can be placed in the job deck.

The inclusion of input data in the job deck depends upon the manner in which the installation has assigned input/output devices. Job control statements are read from the unit named SYSRDR (system reader), which can be either a card reader, a magnetic tape unit, or a disk extent. Input to the processing programs is read from the unit named SYSIPT (system input), which also can be either a card reader, a magnetic tape unit, or a disk extent. The installation has the option of assigning either two separate devices for these units (one device for SYSRDR, a second device for SYSIPT) or one device to serve as both SYSRDR and SYSIPT. If two devices have been assigned, the job deck must consist of only job control statements; input data must be kept separate. If only one device has been assigned, input data must be included within the job deck.

There are four job control statements that are used for job definition: the JOB statement, the EXEC statement, the end-of-data statement (/*), and the end-of-job statement (/&). In this chapter, the discussion of these job control statements is limited to the function and use of each statement. The rules for writing each statement are given in the chapter "Preparing COBOL Programs for Processing."

The JOB statement defines the start of a job. One JOB statement is required for every job; it must be the first statement in the job deck. The programmer must name his job on the JOB statement.

The EXEC statement requests the execution of a program. Therefore, one EXEC statement is required for each job step within a job. The EXEC statement indicates the program that is to be executed (for example, the COBOL compiler, the Linkage Editor). As soon as the EXEC statement has been processed, the program indicated by the statement begins execution.

The end-of-data statement, also referred to as the /* (slash asterisk) statement, defines the end of a program's input data. When the data is included within the job deck (that is, SYSIPT and SYSRDR are the same device), the /* statement immediately follows the input data. For example, COBOL source statements would be placed immediately after the EXEC statement for the COBOL compiler; a /* statement would follow the last COBOL source statement.

When input data is kept separate (that is, SYSIPT and SYSRDR are separate devices), the /* statement immediately follows each set of input data on SYSIPT. For example, if a job consists of two compilation job steps, an editing job step, and an execution job step, SYSIPT would contain the source statements for the first compilation followed by a /* statement, the source statements for the second compilation followed by a /* statement, any input data for the Linkage Editor followed by a /* statement, and perhaps some input data for the problem program followed by a /* statement.

The end-of-job statement, also referred to as the /& (slash ampersand) statement, defines the end of the job. A /& statement must appear as the last statement in the job deck.

OTHER JOB CONTROL STATEMENTS

The four job definition statements form the framework of the job deck. There are a number of other job control statements in the job control language; however, not all of them must appear in the job deck. The job control statements are summarized briefly in Table 1.

The double slash preceding each statement name identifies the statement as a job control statement. Most of the statements are used for data management -- creating, manipulating, and keeping track of data files. (Data files are externally stored collections of data from which data is read and into which data is written.)

Table 1. Job Control Statements

Statement	Function
// ASSGN	Input/output assignments.
// CLOSE	Closes a logical unit assigned to magnetic tape.
// DATE	Provides a date for the Communication Region.
// DLAB	Disk file label information.
// DLBL	Disk file label information.
// EXEC	Execute program.
// EXTENT	Disk file extent.
// JOB	Beginning of control information for a job.
// LBLTYP	Reserves storage for label information.
// LISTIO	Lists input/output assignments.
// MTC	Controls operations on magnetic tape.
// OPTION	Specifies one or more job control options.
// PAUSE	Creates a pause for operator intervention.
// RESET	Resets input/output assignments to standard assignments.
// RSTRT	Restarts a checkpointed program.
// TLBL	Tape label information.
// TPLAB	Tape label information.
// UPSI	Sets user-program switches.
// VOL	Disk/tape label information.
// XTENT	Disk file extent.
/*	End-of-data-file or end-of-job-step.
/&	End-of-job.
*	Comments.

This chapter describes in greater detail the three types of job steps involved in processing a COBOL program. Once the reader becomes familiar with the information presented here, he should be able to write control statements by referring only to the next chapter, "Preparing COBOL Programs for Processing."

COMPILATION

Compilation is the execution of the COBOL compiler. The programmer requests compilation by placing in the job deck an EXEC statement that contains the program name FCOBOL, the name of the COBOL compiler. This is the EXEC FCOBOL statement.

Input to the compiler is a set of COBOL source statements, consisting of either a main program or a subprogram. Source statements must be punched in Extended Binary-Coded-Decimal Interchange Code (EBCDIC). The COBOL source statements are read from SYSIPT. The job deck is read from SYSRDR. If SYSRDR and SYSIPT are assigned to the same unit, the COBOL source statements should be placed after the EXEC FCOBOL statement in the job deck.

Output from the COBOL compiler is dependent upon the options specified when the system is generated. This output may include a listing of source statements exactly as they appear in the input deck. The source listing is produced on SYSLST. In addition, the module produced by the compiler may be written on SYSLNK, the linkage editor input unit, and punched on SYSPCH. Separate Data and/or Procedure Division maps, a symbolic cross-reference list, and diagnostic messages can also be produced. The format of compiler output is discussed and illustrated in the chapter "Interpreting Output."

The programmer can override any of the compiler options specified when the system was generated, or include some not previously specified, by using the OPTION control statement in the compile job step. Compiler options are discussed in detail in the chapter "Preparing COBOL Programs for Processing."

EDITING

Editing is the execution of the Linkage Editor. The programmer requests editing by placing in the job deck an EXEC statement that contains the program name LNKEDT, the name of the Linkage Editor. This is the EXEC LNKEDT statement.

Input to the Linkage Editor consists of a set of linkage editor control statements and one or more modules to be edited. These modules include any of the following:

1. Modules that were compiled previously in the job and placed at that time on the linkage editor input unit, SYSLNK.

2. Modules that were compiled in a previous job and saved as module decks. The module decks must be placed on SYSIPT. Linkage editor control statements are read from SYSRDR.

3. Modules that were compiled in a previous job step and cataloged in the relocatable library. The relocatable library is a collection of frequently used routines in the form of modules, that can be included in a program phase via the INCLUDE control statement in the linkage editor job step.

Output from the Linkage Editor consists of one or more phases. A phase may be an entire program or it may be part of an overlay structure (multiple phases).

A phase produced by the Linkage Editor can be executed immediately after it is produced (that is, in the job step immediately following the linkage editor job step), or it can be executed later, either in a subsequent job step of the same job or in a subsequent job. In either of the latter cases, the phase to be executed must be cataloged in the core image libary. Such a phase can be retrieved in the execute job step by specifying the phase name in the EXEC statement, where phase name is the name under which it was cataloged. Otherwise, the phase output is retained only for the duration of one job step following the linkage editor job step. That is, if the module that was just linkage edited is to be executed in the next job step, it need not have been cataloged. An EXEC statement will cause the phase to be brought in from the

temporary part of the core image library and will begin execution. However, the next time such a module is to be executed, the linkage editor job step is required since the phase was not cataloged in the core image library.

In addition to the phase, the Linkage Editor produces a phase map on SYSLST. Linkage editor diagnostic messages are also printed on SYSLST. If the NOMAP option of the linkage editor ACTION control statement is specified, no phase map is produced and linkage editor diagnostic messages are listed on SYSLST, if assigned. Otherwise, the diagnostic messages are listed on SYSLOG. The contents of the phase map are discussed and illustrated in the chapter "Interpreting Output."

Linkage editor control statements direct the execution of the Linkage Editor. Together with any module decks to be processed, they form the linkage editor input deck, which is read by the Job Control Processor from SYSIPT and written on SYSLNK.

There are four linkage editor control statements: the ACTION statement, the PHASE statement, the ENTRY statement, and the INCLUDE statement. These statements are discussed in the next chapter.

PHASE EXECUTION

Phase execution is the execution of the problem program, for example, the program written by the COBOL programmer. If the program is an overlay structure (multiple phase), the execution job step actually involves the execution of all the phases in the program.

The phase(s) to be executed must be contained in the core image library. The core image library is a collection of executable phases from which programs are loaded by the Supervisor. A phase is written in the temporary part of the core image library by the Linkage Editor at the time the phase is produced. It is permanently retained (cataloged) in the core image library, if the programmer has so requested, via the CATAL option in the OPTION control statement.

The programmer requests the execution of a phase by placing in the job deck an EXEC statement that specifies the name of the phase. However, if the phase to be executed was produced in the immediately preceding job step, it is not necessary to specify its name in the EXEC statement.

MULTIPHASE PROGRAMS

A COBOL program can be executed as a single phase as long as there is an area of main storage available to accommodate it. This area, known as the problem program area, must be large enough to contain the main program and all called subprograms. When a program is too large to be executed as a single phase, it must be structured as a multiphase program.

The overlay structure available to the COBOL programmer for multiphase programs is known as root phase overlay, and is used primarily for programs of three or more phases. One phase of the program is designated as the root phase (main program) and, as such, remains in the problem program area throughout the execution of the entire program. The other phases in the program -- subordinate phases -- are loaded into the problem program area as they are needed. A subordinate phase may overlay any previously loaded subordinate phase, but no subordinate phase may overlay the root phase. One or more subordinate phases can reside simultaneously in main storage with the root phase.

Use of the linkage editor control statements needed to effect overlay are discussed in the chapter "Calling and Called Programs."

This chapter provides information about preparing COBOL source programs for compilation, linkage editing, and execution.

ASSIGNMENT OF INPUT/OUTPUT DEVICES

Almost all COBOL programs include input/output statements calling for data to be read from or written into data files stored on external devices. COBOL programs do not reference input/output devices by their actual physical address, but rather by their symbolic names. Thus, a COBOL program is dependent on the device type and not on the actual device address. The COBOL programmer need only select the symbolic name of a device from a fixed set of symbolic names. At execution time, as a job control function, the symbolic name is associated with an actual physical device. The standard assignment of physical addresses to symbolic names may be made at system generation time. However, job control statements and operator commands can alter the standard device assignment before program execution. This is discussed later in this chapter.

To simulate an installation environment, all the examples in this publication assume that the symbolic units and their physical and logical assignments are as shown in Figure 2.

The symbolic names are divided into two classes: system logical units and programmer logical units.

The system logical units (SYSIPT, SYSLNK, SYSLOG, SYSLST, SYSPCH, SYSRES, SYSSLB, SYSRLB, and SYSRDR) are used by the control program and by IBM-supplied processing programs. SYSIPT, SYSLST, SYSPCH, and SYSLOG can be implicitly referenced by certain COBOL procedural statements. Two additional names, SYSIN and SYSOUT, are defined for background program assignments. The names are valid only to the Job Control Processor, and cannot be referenced in the COBOL program. SYSIN can be used when SYSRDR and SYSIPT are the same device; SYSOUT must be used when SYSLST and SYSPCH are assigned to the same magnetic tape unit.

Programmer logical units are those in the range SYS000 through SYS221 and may be referenced in the COBOL source language ASSIGN clause.

Logical Unit	Physical Unit	Device Type
SYSRES	X'190'	2311 Disk unit
SYSLNK	X'191'	2311 Disk unit
SYSRDR, SYSIPT	X'00C'	2540 Card reader
SYSLST	X'00E'	1403 Printer
SYSPCH	X'00D'	2540 Card punch
SYSLOG	X'01F'	1052 Printer keyboard
SYSSLB	X'191'	2311 Disk unit
SYSRLB	X'191'	2311 Disk unit
SYS001	X'191'	2311 Disk system work file
SYS002	X'191'	2311 Disk system work file
SYS003	X'190'	2311 Disk system work file
SYS004	X'281'	2400 Tape work file
SYS005	X'00E'	1403 Printer
SYS006	X'191'	2311 Disk unit
SYS007	X'191'	2311 Disk unit
SYS008	X'282'	2400 Tape unit
SYS009	X'283'	2400 Tape unit
SYS010	X'284'	2400 Tape unit
SYS011	X'285'	2400 Tape unit
SYS012	X'00E'	1403 Printer
SYS013	X'00C'	2540 Card reader
SYS014	X'01F'	1052 Printer keyboard
SYS015	X'192'	2314 Disk unit
SYS016	X'192'	2314 Disk unit
SYS017 through SYS221	Unassigned	

Figure 2. Sample Logical Unit Assignments

A COBOL programmer uses the source language ASSIGN clause to assign a file used by his problem program to the appropriate symbolic name. Although symbolic names may be assigned to physical devices at system generation time, the programmer may alter these assignments at execution time by means of the ASSGN control statement. However, if the programmer wishes to use the assignments made at system generation time for his own data files in the COBOL program, ASSGN control statements are unnecessary.

Table 2 is a complete list of symbolic names and their usage.

Table 2. Symbolic Names, Functions, and Permissible Device Types

Symbolic Name	Function	Permissible Device Types
SYSRDR	Input unit for control statements.	Card reader Magnetic tape unit Disk extent
SYSIPT	Input unit for programs.	Card reader Magnetic tape unit Disk extent
SYSPCH	Main unit for punched output.	Card punch Magnetic tape unit Disk extent
SYSLST	Main unit for printed output.	Printer Magnetic tape unit Disk extent
SYSLOG	Receives operator messages and logs in job control statements.	Printer keyboard Printer
SYSLNK	Input to the Linkage Editor.	Disk extent
SYSRES	Contains the operating system, the core image library, relocatable library, and source statement library.	Disk extent
SYSSLB	A private source statement library.	Disk extent
SYSRLB	A private relocatable library.	Disk extent
SYSIN	Must be used when SYSRDR and SYSIPT are assigned to the same disk extent. May be used when they are assigned to the same card reader or magnetic tape.	Disk Magnetic tape unit Card reader
SYSOUT	This name must be used when SYSPCH and SYSLST are assigned to the same magnetic tape unit. It must be assigned by the operator ASSGN command.	Magnetic tape unit
SYSmax	These units are available to the programmer as work files or for storing data files. They are called programmer logical units as opposed to the above-mentioned names which are always referred to as system logical units. The largest number of programmer logical units available in the system is 222 (SYS000 through SYS221). The value of SYSmax is determined by the distribution of the programmer logical units among the partitions.	Any unit

JOB CONTROL

The Job Control Processor for the Disk Operating System prepares the system for execution of programs in a batched job environment. Input to the Job Control Processor is in the form of job control statements and job control commands.

JOB CONTROL STATEMENTS

Job control statements are designed for an 80-column punched card format. Although certain restrictions must be observed, the statements are essentially free form. Job control statements conform to these rules:

1. Name. Two slashes (//) identify the statement as a job control statement. They must be in columns 1 and 2. At least one blank immediately follows the second slash.

   Exceptions: The end-of-job statement contains /& in columns 1 and 2; the end-of-data file statement contains /* in columns 1 and 2; the comment statement contains * in column 1 and a blank in column 2.

2. Operation. This identifies the operation to be performed. It can be up to eight characters long. At least one blank follows its last character.

3. Operand. This may be blank or may contain one or more entries separated by commas. The last term must be followed by a blank, unless its last character is in column 71.

4. Comments. Optional user comments must be separated from the operand by at least one space.

Continuation cards are not recognized by the Job Control Processor. For the exception to this rule, see the descriptions of the DLAB and TPLAB statements.

All job control statements are read from the device identified by the symbolic name SYSRDR.

Comments in Job Control Statements

Comment statements (i.e., statements preceded by an asterisk in column 1 followed by a blank) may be placed anywhere in the job deck. The remainder of the card may contain any character from the EBCDIC set. Comment statements are designed for communication with the operator; accordingly, they are written on the console printer-keyboard, SYSLOG, in addition to being written on SYSLST. If followed by a PAUSE control statement, the comment statement can be used to request operator action.

Statement Formats

The following notation is used in the statement formats:

1. All upper-case letters represent specifications that are to appear in the actual statement exactly as shown in the statement format. For example, JOB in the operation field of the JOB statement should be punched exactly as shown.

2. All lower-case letters represent generic terms that are to be replaced in the actual statement. For example, jobname is a generic term that should be replaced by the name that the programmer is giving his job.

3. Hyphens are used to join two or more words in order to form a single generic term. For example, device-address is one generic term.

4. Brackets are used to indicate that a specification is optional and is not always required in the statement. For example, [type] indicates that the programmer's replacement for the generic term, type, may or may not appear in the statement, depending on the programmer's requirements.

5. Braces enclosing stacked items indicate that a choice of one item must be made by the programmer. For example:

   { SYS
PROG
ALL
SYSxxx }

   indicates that either SYS, PROG, ALL, or SYSxxx must appear in the actual statement.

6. Brackets enclosing stacked items indicate that a choice of one item may, but need not, be made by the programmer. For example:

```
[,X'ss']
[,ALT  ]
```

indicates that either ,X'ss' or ,ALT but not both, may appear in the actual statement, or the specification can be omitted entirely.

7. All punctuation marks shown in the statement formats other than hyphens, brackets, and braces must be punched as shown. This includes periods, commas, and parentheses. For example, ,[date] means that the specification, if present in the statement, should consist of the programmer's replacement for the generic term date preceded by the comma with no intervening space. Even if the date is omitted, the comma must be punched as shown.

8. The ellipsis (...) indicates where repetition may occur at the user's option. The portion of the format that may be repeated is determined as follows:

   a. Scanning right to left, determine the bracket or brace delimiter immediately to the left of the ellipsis.

   b. Continue scanning right to left and determine the logically matching bracket or brace delimiter.

   c. The ellipsis applies to the words and punctuation between the pair of delimiters.

Sequence of Job Control Statements

The job deck for a specific job always begins with a JOB statement and ends with a /& (end-of-job) statement. A specific job consists of one or more job steps. The beginning of a job step is indicated by the appearance of an EXEC statement. When an EXEC statement is encountered, it initiates the execution of the job step, which includes all preceding control statements up to, but not including, a previous EXEC statement.

The only limitation on the sequence of statements within a job step is that which is discussed here for the label information statements.

The label statements must be in the order:

```
VOL
TPLAB
```

or

```
VOL
DLAB
XTENT (one for each area or file in
       the volume)
```

or

```
DLBL
EXTENT (one for each area or file in
        the volume)
```

or

```
TLBL
```

and must immediately precede the EXEC statement to which they apply.

DESCRIPTION AND FORMATS OF JOB CONTROL STATEMENTS

This section contains descriptions and formats of job control statements.

Job control statements, with the exception of /*, /&, and *, contain two slashes in columns 1 and 2 to identify them.

ASSGN Statement

The ASSGN control statement assigns a logical input/output unit to a physical device. An ASSGN control statement must be present in the job deck for each data file assigned to an external storage device in the COBOL program where these assignments differ from those established at system generation time. Data files are assigned to programmer logical units in COBOL by means of the source language ASSIGN clause. The ASSGN control statement may also be used to change a system standard assignment for the duration of the job. The format of the ASSGN control statement is as follows:

```
// ASSGN SYSxxx,device-address [,X'ss']
                               [,ALT  ]
```

SYSxxx
is one of the logical devices listed in Table 2.

Exception: SYSOUT must be assigned using the ASSGN job control command. Job control commands are described in detail in the publication IBM System/360 Disk Operating System: System Control and System Service Programs.

device-address
allows three different formats:

X'cuu'
where c is the channel number and uu the unit number in hexadecimal notation. The values of 'cuu' are determined by each installation.

c = 0 for multiplexor channel, 1 through 6 for selector channels 1 through 6.

uu = 00 to FE (0 to 254) in hexadecimal.

UA
indicates that the logical unit is to be unassigned. Any source language input/output operation attempted on this device causes cancellation of the job.

IGN
indicates that the logical unit is to be unassigned. References to this logical unit are ignored during program execution. However, if OPTIONAL has been specified in the SELECT sentence for an input file, the first READ statement for that file causes control to be transferred to the imperative-statement following the AT END option of the READ statement. The IGN option is not valid for SYSRDR, SYSIPT, and SYSIN. This option is useful in program debugging since source language input or output references to files residing on symbolic units for which IGN has been specified are ignored.

X'ss'
is the device specification. It is used for specifying mode settings for 7-track and dual density 9-track tapes. If X'ss' is not specified, the system assumes X'90' for 7-track tapes and X'C0' for 9-track tapes. The possible specifications for X'ss' are shown in Figure 3.

ALT
must be specified in the control statement that assigns an alternate magnetic tape unit which is used when the capacity of the original assignment is reached. The specifications for the alternate unit must be the same as those of the orginal unit, since X'ss' cannot be specified. The characteristics of the alternate unit must be the same as those of the original unit. Multiple alternates can be assigned to a symbolic unit.

Device assignments made by the ASSGN control statement are considered temporary. They are in effect until another ASSGN control statement or a RESET statement for that logical unit, or the next /& or JOB statement is read, whichever occurs first. If a RESET, /&, or JOB statement is encountered, the assignment reverts to the standard assignment established at system generation time plus any modification by an ASSGN command.

The COBOL programmer may assign only the programmer logical units (SYS000 through SYS221) to data files used in his program. For example, if the following ASSIGN clause is used,

```
SELECT IN-FILE ASSIGN TO SYS004-UR-2540R-S
```

an ASSGN control statement must appear in the job deck which assigns SYS004 to a physical device if the physical device differs from the permanent assignment. In this case, the physical device must be a 2540 card reader. An example of such a control statement is:

```
// ASSGN SYS004,X'00C'
```

Physical unit X'00C' was permanently assigned to a 2540 Card Reader at system generation time.

Note: The ASSGN control statement is necessary only when the symbolic unit assignment is being made to a physical device address which differs from that established at system generation time.

"Appendix I: Sample Job Decks" contains illustrations of ASSGN statement usage.

ss	Bytes per Inch	7-Track Tape: Parity	Translate Feature	Convert Feature
10	200	odd	off	on
20	200	even	off	off
28	200	even	on	off
30	200	odd	off	off
38	200	odd	on	off
50	556	odd	off	on
60	556	even	off	off
68	556	even	on	off
70	556	odd	off	off
78	556	odd	on	off
90	800	odd	off	on
A0	800	even	off	off
A8	800	even	on	off
B0	800	odd	off	off
B8	800	odd	on	off
		9-Track Tape		
C0	800	single density 9-track		
C0	1600	single density 9-track		
C0	1600	dual density 9-track		
C8	800	dual density 9-track		

Figure 3. Possible Specifications for X'ss' in the ASSGN Control Statement

CLOSE Statement

The CLOSE control statement is used to close either a system or programmer logical unit assigned to tape. As a result of the CLOSE control statement, a standard end-of-volume label set is written and the tape is rewound and unloaded. The CLOSE statement applies only to a temporarily assigned logical unit, that is, a logical unit for which an ASSGN control statement has been specified within the same job. The format of the CLOSE control statement is as follows:

```
                  [,X'cuu' [,X'ss']]
                  [,UA              ]
// CLOSE SYSxxx   [,IGN             ]
                  [,ALT             ]
```

The logical unit can optionally be reassigned to another device, unassigned, or switched to an alternate unit.

Note that when SYSxxx is a system logical unit, one of the optional parameters must be specified. When closing a programmer logical unit, no optional parameter need be specified.

SYSxxx
: may only be used for magnetic tape and may be specified as SYSPCH, SYSLST, SYSOUT, or SYS000 through SYS221.

X'cuu'
: specifies that after the logical unit is closed, it will be assigned to the channel and unit specified. (See "ASSGN Control Statement" for an explanation of 'cuu'.) When reassigning a system logical unit, the new unit will be opened if it is either a mass storage device or a magnetic tape at load point.

X'ss'
: represents device specification for mode settings on 7-track and 9-track tape. (See "ASSGN Control Statement" for an explanation of 'ss'.) If X'ss' is not specified, the mode settings remain unchanged.

UA
: specifies that the logical unit is to be closed and unassigned.

IGN
: specifies that the logical unit is to be closed and unassigned with the ignore option. This operand is invalid for SYSRDR, SYSIPT, or SYSIN.

ALT
: specifies that the logical unit is to be closed and an alternate unit is to be opened and used. This operand is valid only for system logical output units (SYSPCH, SYSLST, or SYSOUT) currently assigned to a magnetic tape unit.

DATE Statement

The DATE control statement contains a date that is put in the Communication Region of the Supervisor. A complete description of the fields of the Communication Region is given in "Appendix H: Communication Region." The DATE statement is in one of the following formats:

```
// DATE mm/dd/yy
// DATE dd/mm/yy
```

where:
- mm = month (01 to 12)
- dd = day (01 to 31)
- yy = year (00 to 99)

The format to be used is the format selected when the system was generated.

When the DATE statement is used, it applies only to the current job being executed. The Job Control Processor does not check the operand except to ensure that its length is eight characters. If no DATE statement is specified in the current job, the Job Control Processor supplies the date given in the last SET command. The SET command is discussed in detail in the publication IBM System/360 Disk Operating System: System Control and System Service Programs.

A DATE statement should be included in every job deck that has as one of its job steps the execution of a COBOL program that utilizes the special register CURRENT-DATE, if the date desired is other than that designated in the previous SET command.

TLBL Statement

The TLBL control statement replaces the VOL and TPLAB combination used in previous versions of the system. However, the current system will continue to support these statements. The TLBL control statement contains file label information for tape label checking and writing. Its format follows:

```
// TLBL filename,
     ['file-identifier'],[date],
     [file-serial-number],
     [volume-sequence-number],
     [file-sequence-number],
     [generation-number],
     [version-number]
```

filename
: identifies the file to the control program. It can be from one to seven characters in length. If the following SELECT sentence appears in a COBOL program:

  ```
  SELECT NEWFILE ASSIGN TO
  SYS003-UT-2400-S-OUTFILE
  ```

  the filename operand on control statements for this file must be OUTFILE. If the SELECT clause were coded:

  ```
  SELECT NEWFILE ASSIGN TO
  SYS003-UT-2400-S
  ```

  the filename operand on the control statement for the file must be SYS003.

'file-identifier'
: consists of from 1 to 17 characters, contained within apostrophes, indicating the name associated with the file on the volume. This operand may contain embedded blanks. If this operand is omitted on output files, the filename will be used. If this operand is omitted on input files, no checking will be done.

date
: consists of from one to six characters, in the format yy/ddd, indicating the expiration date of the file for output or the creation date for input. (The day of the year may consist of from one to three characters.) For output files, a one to four character retention period (d-dddd) may be specified. If this operand is omitted, a 0-day retention period will be assumed for output files. For input files, no checking will be done if this operand is omitted or if a retention period is specified.

file-serial-number
: consists of from one to six characters indicating the volume serial number of the first (or only) reel of the file. If fewer than six characters are specified, the field will be right-justified and padded with zeros. If this operand is omitted on output files, the volume serial number of the first (or only) reel of the file will be used. If the operand is omitted on input files, no checking will be done.

volume-sequence-number
: consists of from one to four characters in ascending order for each volume of a multivolume file. This number is incremented automatically by OPEN and CLOSE routines as required. If this operand is omitted on output files, BCD 0001 will be used. If omitted on input files, no checking is done.

file-sequence-number
: consists of from one to four characters in ascending order for each file of a multivolume file. This number is incremented automatically by OPEN and CLOSE routines as required. If this operand is omitted on output files, BCD 0001 will be used. If it is omitted on input files, no checking will be done.

generation-number
: consists of from one to four numeric characters that modify the file-identifier. If this operand is omitted on output files, BCD 0001 is

used. If it is omitted on input files, no checking will be done.

version-number
consists of from one to two numeric characters that modify the generation number. If this operand is omitted on output files, BCD 01 will be used. If it is omitted on input files, no checking will be done.

Note: If a tape file with standard labels is opened two different ways in the same COBOL program, and that file resides on a multifile volume, the programmer should use two separate TLBL cards with different filenames specified on each.

DLBL Statement

The DLBL control statement, in conjunction with the EXTENT statement, replaces the VOL, DLAB, and XTENT combination used in previous versions of the Disk Operating System. However, the current system will continue to support the VOL, DLAB, and XTENT statements. The DLBL statement has the following format:

```
// DLBL filename
   ,['file-identifer'],[date],[codes]
```

filename
identifies the file to the control program. It can be from one to seven characters in length. If the following SELECT sentence appears in a COBOL program:

```
SELECT INFILE ASSIGN TO
SYS005-DA-2311-A-INPUTA
```

the filename operand on control statements for this file must be INPUTA. If the SELECT sentence is coded:

```
SELECT INFILE ASSIGN TO
SYS005-DA-2311-A
```

the filename operand on control statements for the file must be SYS005.

'file-identifier'
is the name associated with the file on the volume. This can consist of from 1 to 44 alphanumeric characters contained within apostrophes, including the file-identifier and, if used, generation-number and version-number of generation. If fewer than 44 characters are used, the field is left-justified and padded with blanks. If this operand is omitted, filename will be used.

date
consists of from one to six characters indicating either the retention period of the file in the format d through dddd (1-9999), or the absolute expiration date of the file in the format yy/ddd. When the d through dddd format is used, the file is retained for the number of days specified as dddd. For example, if date is specified as 31, the file will be retained a month from the day of creation. When the yy/ddd format is used, the file is retained until the day (ddd) in the year (yy) specified. For example, if date is specified as 69/200, the file will be retained through the 200th day of the year 1969.

If date is omitted when the file is created, a 7-day retention period is assumed. If this operand is present for a file opened as INPUT or I-O, it is ignored.

codes
is a 2- or 3-character field indicating the type of file label, as follows:

```
 SD = Sequential Disk
 DA = Direct Access
ISC = Indexed Sequential using Load
      Create
ISE = Indexed Sequential using Load
      Extension, Add, or Retrieve
```

If code is omitted, SD is assumed.

"Appendix I: Sample Job Decks" contains illustrations of DLBL statement usage.

EXTENT Statement

The EXTENT control statement defines each area (or extent) of a DASD file -- a file assigned to a mass storage device. One or more EXTENT control statements must follow each DLBL statement.

The EXTENT control statement replaces the XTENT statement used in previous versions of the Disk Operating System. However, XTENT will continue to be supported in the current system.

The format of the EXTENT control statement is:

```
// EXTENT [symbolic-unit],[serial-number]
    ,[type],[sequence-number]
    ,[relative-track],[number-of-tracks]
    ,[split-cylinder-track],[B=bins]
```

symbolic-unit
is a 6-character field indicating the symbolic unit (SYSxxx) of the volume for which this extent is effective. If this operand is omitted, the symbolic unit of the preceding EXTENT statement will be used. When specified, symbolic-unit may be any SYSxxx assigned to the device type indicated in the SELECT sentence for the file. For example, if the following coding appears in a COBOL program:

```
SELECT OUTFILE ASSIGN TO
SYS004-DA-2311-A
```

the symbolic unit in the EXTENT control statement can by any SYSxxx assigned to a 2311 disk pack. The symbolic unit operand is not required for an IJSYSxx filename, where xx is IN, PH, LS, RS, SL, or RL. If SYSRDR or SYSIPT is assigned, this operand must be included.

serial-number
consists of from one to six characters indicating the volume serial number of the volume for which this extent is effective. If fewer than six characters are used, the field will be right-justified and padded with zeros. If this operand is omitted, the volume serial number of the preceding EXTENT control statement will be used. If no serial number was provided in the EXTENT control statement, the serial number will not be checked and it will be the user's responsibility if files are destroyed as a result of mounting the incorrect volume.

type
consists of one character indicating the type of the extent, as follows:

1 -- Data area (no split cylinder)
2 -- Overflow area (for an indexed file)
4 -- Index area (for an indexed file)
8 -- Data area (split cylinder)

If this operand is omitted, 1 is assumed.

sequence-number
consists of from one to three characters containing a decimal number from 0 to 255 indicating the sequence number of this extent within a multi-extent file. Extent sequence 0 is used for the master index of an indexed file. If the master index is not used, the first extent of an indexed file has the sequence number 1. The extent sequence number for all other types of files begins with 0. If this operand is omitted for the first extent of ISFMS files, the extent will not be accepted. For SD or DA files, this operand is not required. Direct files can have up to five extents. Indexed files can have up to eleven data extents (nine prime, one cylinder index, one separate overflow).

relative-track
consists of from one to five characters indicating the sequential number of the track, relative to zero, where the data extent is to begin. If this field is omitted on an ISFMS file, the extent will not be accepted. This field is not required for DA input or for SD input files (the extents from the file labels will be used).

Formulas for converting actual to relative track addresses (RT) and relative track to actual for the DASD devices follow.

Actual to Relative:

2311 10 x cylinder number + track number = RT

2314 20 x cylinder number + track number = RT

2321 1000 x subcell number + 100 x strip number + 20 x block number + track number = RT

Relative to Actual:

2311 $\frac{RT}{10}$ = quotient is cylinder, remainder is track

2314 $\frac{RT}{20}$ = quotient is cylinder, remainder is track

2321 $\frac{RT}{1000}$ = quotient is subcell, remainder1

$\frac{remainder1}{100}$ = quotient is strip, remainder2

$\frac{remainder2}{20}$ = quotient is block, remainder is track

Example: Track 5, cylinder 150 on a 2311 = 1505 in relative track.

number-of-tracks
consists of from one to five characters indicating the number of tracks to be allocated to the file. For SD input files, this field may be omitted. The number of tracks for a split cylinder file must be a multiple of the number of cylinders specified for the file and the number of tracks specified for each cylinder.

split-cylinder-track
consists of from one to two characters, with a value of 0 through 19, indicating the upper tack number for the split cylinder in SD files.

bins
consists of from one to two characters identifying the 2321 bin that the extent was created for, or on which the extent is currently located. If the field is one character, the creating bin is assumed to be zero. There is no need to specify a creating bin for SD or ISFMS files. If this operand is omitted, bin 0 is assumed for both bins. If the operand is included and positional operands are omitted, only one comma is required preceding the keyword operand. If any operands preceding the bin specification are omitted, one comma for each operand is acceptable, but unnecessary.

Figure 4 shows examples of using the DLBL statement in conjunction with the EXTENT statement. "Applendix I: Sample Job Decks" contains illustrations of EXTENT statement usage.

VOL Statement

The VOL control statement is used when standard labels for a DASD or tape file are checked. It is used in conjunction with TPLAB or DLAB and XTENT statements. The VOL and TPLAB or VOL, DLAB and XTENT statements must appear in that order and must immediately precede the EXEC statement to which they apply. The format of the VOL control statement is:

```
// VOL SYSxxx,filename
```

SYSxxx
is the symbolic unit name. The symbolic unit name is the same name that appears in the XTENT statement for the file.

filename
identifies the file to the control program. It can consist of from one to seven characters. The appearance of two identical operands is characteristic of COBOL object modules, since filename might be the logical unit which is assigned to a device.

Note that filename, as used in this context, does not refer to the COBOL file-name, but to filename as it is used by the system.

For example, if the following COBOL coding appeared as part of a complete program, MASTERX is the name by which the file is known to the control program.

Direct file:

The following DLBL and EXTENT statements describe a direct file occupying 840 tracks, beginning on relative track 10.

```
// DLBL MASTER,,75/001,DA
// EXTENT SYS015,111111,1,0,10,840
```

Indexed file:

The following DLBL and EXTENT statements describe an indexed file occupying 80 tracks, beginning on relative track 1106. The first EXTENT allocates a 4-track cylinder index. The second EXTENT allocates a 76-track data area.

```
// DLBL MASTER,,75/001,ISC
// EXTENT SYS015,111111,4,1,1106,4
// EXTENT SYS015,111111,1,2,1110,76
```

Figure 4. Sample Label and File Extent Information for Mass Storage Files

```
ENVIRONMENT DIVISION.
FILE-CONTROL.
     SELECT MASTER-FILE ASSIGN TO
     SYS004-UT-2400-S-MASTERX
     .
     .
     .
DATA DIVISION.
FILE SECTION.
FD MASTER-FILE
     .
     .
     .
```

The VOL control statement for the file could be coded as follows:

```
// VOL SYS004,MASTERX
```

If the COBOL SELECT sentence had been coded as:

```
     SELECT MASTER-FILE ASSIGN TO
     SYS004-UT-2400-S
```

SYS004 would be the name by which the file is known to the control program and the VOL statement could be coded as follows:

```
// VOL SYS004,SYS004
```

The filename, as used in the VOL control statement format, is identical to the symbolic name of the program DTF that identifies the file. Although, in COBOL, displacement is from the symbolic name MASTER-FILE when referencing the DTF, the system interprets this to be MASTERX in the first case, and SYS004 in the second case.

When coding the VOL control statement for files assigned to mass storage devices, there is an additional consideration. If the following SELECT sentence appears in a COBOL program:

```
     SELECT INFILE ASSIGN TO
     SYS001-DA-2311-A-INPUTA
```

the symbolic unit name on the control statements for the file can be any SYSxxx assigned to a 2311 disk pack. The filename on control statements for the file must be INPUTA.

For example, the VOL control statement might be:

```
// VOL SYS021,INPUTA
```

If the SELECT sentence were coded:

```
  SELECT INFILE ASSIGN TO
  SYS004-DA-2311-A
```

the symbolic unit name on control statements for the file can be any SYSxxx assigned to a 2311 disk pack. The filename on control statements for the file must be SYS004. Both of the following VOL control statements are acceptable:

```
// VOL SYS004,SYS004
// VOL SYS005,SYS004
```

DLAB Statement

The DLAB control statement contains information for label checking and creation of files assigned to mass storage devices. This statement must immediately follow a VOL control statement. (Disk label formats are given in "Appendix C: Standard Mass Storage Device Labels.") The format of the DLAB control statement is:

```
// DLAB 'label fields 1-3',
    xxxx,yyddd,yyddd,'systemcode'[,type]
```

'label fields 1-3'
: The first three fields of the disk-file label are contained just as they appear in the label. This is a 51-character string contained within apostrophes and followed by a comma.

  The DLAB statement requires two cards for completion; therefore, column 72 of the first card requires a character punch other than a blank. The columns between the comma and the continuation character must be blank.

xxxx
: is the volume-sequence-number in field 4 of the Format 1 label and must begin in card column 16 of the second card.

yyddd,yyddd
: is the file creation date followed by the file expiration date. It is recommended that this field be left blank.

'systemcode'
: is ignored by the Disk Operating System. The dummy field specified must be 13 characters long.

type
: indicates the type of file label:

  SD = Sequential Disk
  DA = Direct Access
  ISC = Indexed Sequential (used when creating the file)
  ISE = Indexed Sequential (used when updating or retrieving the file)

  SD is assumed if this entry is omitted.

TPLAB Statement

The TPLAB control statement contains file label information for tape label checking and creation. It must immediately follow a VOL control statement. The TPLAB control statement contains an image of a portion of the standard tape file label. The format and contents of a standard tape label are given in "Appendix B: Standard Tape File Labels." The format of the TPLAB control statement is as follows:

```
// TPLAB  {'label fields 3-10'}
          {'label fields 3-13'}
```

'label fields 3-10'
is a 49-byte character string contained within apostrophes, identical to positions 5 through 53 of the tape file label. These fields can be included in one line and are the only ones used for label checking.

'label fields 3-13'
is a 69-byte character string contained within apostrophes, identical to positions 5 through 73 of the tape file label. These fields are too long to be included on a single line. The character string must extend into column 71, a continuation character (any character) must be placed in column 72, and the character string is completed on the next line. The continuation line starts in column 16. Fields 3 through 13 are written in the corresponding fields when the output label is created. When specified for an input file, fields 11 through 13 are ignored. However, even for output files, fields 11 through 13 are never used by the Disk Operating System label processing routines.

XTENT Statement

The XTENT control statement is used to define an area of a file on a mass storage device. Each DASD file (file assigned to a mass storage device) requires one or more XTENT control statements. The format of the XTENT control statement is:

```
// XTENT type,sequence,lower,upper
       'serial no',SYSxxx[,B2]
```

type
Each XTENT type identifies the funtion of the defined area.

Extent Type -- occupies one or three columns containing:

1 = Data area (no split cylinder)
2 = Overflow area (for an indexed file)
4 = Index area (for an indexed file)
128 = Data area (split cylinder). If type 128 is specified, the lower head is assumed to be H^1 H^2 H^2 in lower, and the upper head is assumed to be H_1 H_2 H_2 in upper. (See the discussion of the lower and upper fields.)

sequence
Extent Sequence Number -- indicates the sequence number of this extent within a multi-extent file. The sequence number occupies one to three columns and contains a decimal number from 0 to 255. Extent sequence 0 is used for the master index of an indexed file. If the master index is not used, the first extent of an indexed file contains sequence number 1. The extent sequence for all other types of files begins with 0. Direct files can have up to five extents. Indexed files can have up to eleven data extents (nine prime, one cylinder index, one separate overflow).

lower
Lower Limit of Extent -- occupies nine columns and contains the lowest address of the extent in the form $B_1C_1C_1C_2C_2C_2H_1H_2H_2$

where:

B_1 is the initially assigned cell number. It is equal to:

0 for 2311 and 2314
0 to 9 for 2321

C_1C_1 is the subcell number. It is equal to:

00 for 2311 and 2314
00 to 19 for 2321

$C_2C_2C_2$ is the cylinder number. It can be:

000 to 199 for 2311 and 2314

or strip number:

000 to 009 for 2321

H_1 is the head block position. It is equal to:

0 for 2311 and 2314
0 to 4 for 2321

H_2H_2 is the head number. It can be:

00 to 09 for 2311
00 to 19 for 2321 and 2314

A lower extent of all zeros is invalid.

Note: For 2321, the last five strips of subcell 19 are reserved for alternate tracks.

upper — Upper Limit of Extent -- occupies nine columns containing the highest address of the extent in the same form as the lower limit.

'serial no' — Volume Serial Number -- This is a 6-byte alphanumeric character string, contained within apostrophes. The number is the same as in the volume label (volume serial number) and the Format 1 label (file serial number).

SYSxxx — This is the symbolic address of the DASD drive. If more than one symbolic address is to be specified on separate XTENT cards for the same file, the symbolic addresses must be in consecutive order. See "EXTENT Statement" for details on SYSxxx assignments.

B_2 — Currently assigned cell number. Its value is:

0 for 2311 or 2314
0 to 9 for 2321

This field is optional. If missing, the Job Control Processor assigns $B_2 = B_1$.

JOB Statement

The JOB control statement indicates the beginning of control information for a job. The JOB control statement is in the following format:

```
// JOB jobname
```

jobname
is a user-defined name consisting of from one to eight alphanumeric characters. Any user comments can appear on the JOB control statement following the jobname (through column 72). If the timer feature is present, the time of day appears in columns 73 to 80 when the JOB statement is printed on SYSLST. The time of day is also printed in columns 1 through 8 on the next line of SYSLOG.

If a job is restarted, the jobname must be identical to that used when the checkpoint was taken.

Note: The JOB statement resets the effect of all previously issued OPTION and ASSGN control statements.

LBLTYP Statement

The LBLTYP control statement defines the amount of storage to be reserved at linkage edit time in the problem program area of main storage in order to process tape and nonsequential DASD file labels. It applies to both background and foreground object programs, and is required if the file contains standard labels.

The LBLTYP control statement immediately precedes the // EXEC LNKEDT statement in the job deck, with the exception of self-relocating programs for which it is instead submitted immediately preceding the // EXEC statement for the program. The format of the LBLTYP control statement is:

```
// LBLTYP { TAPE[(nn)] }
          { NSD(nn)     }
```

TAPE[(nn)]
is used only if tape files requiring label information are to be processed and if no nonsequential DASD files are to be processed. nn is optional and is present only for future expansion. It is ignored by the Job Control Processor.

NSD(nn)
is used if any nonsequential DASD files are to be processed, regardless of other type files that are used. nn specifies the largest number of extents to be used for a single file.

LISTIO Statement

The LISTIO control statement causes the system to print a list of input/output assignments on SYSLST. The format of the LISTIO control statement is:

```
                 SYS
                 PROG
                 F1
                 F2
// LISTIO        ALL
                 SYSxxx
                 UNITS
                 DOWN
                 UA
                 X'cuu'
```

SYS
: causes the physical units assigned to all system logical units to be listed.

PROG
: causes the physical units assigned to all background programmer logical units to be listed.

F1
: causes the physical units assigned to all foreground-one logical units to be listed.

F2
: causes the physical units assigned to all foreground-two logical units to be listed.

ALL
: causes the physical units assigned to all logical units to be listed.

SYSxxx
: causes the physical units assigned to the logical unit specified to be listed.

UNITS
: causes the logical units assigned to all physical units to be listed.

DOWN
: causes all physical units specified as inoperative to be listed.

UA
: causes all physical units not currently assigned to a logical unit to be listed.

X'cuu'
: causes the logical units assigned to the physical unit specified to be listed.

MTC Statement

The MTC control statement controls 2400 series magnetic tape operations. The format is as follows:

```
// MTC opcode,SYSxxx[,nn]
```

opcode
: specifies the operation to be performed. opcode can be chosen from the following:

  BSF -- Backspace to tapemark

  BSR -- Backspace to interrecord gap

  ERG -- Erase gap (write blank tape)

  FSF -- Forward space to tapemark

  FSR -- Forward space to interrecord gap

  RUN -- Rewind and unload

  REW -- Rewind

  WTM -- Write tapemark

SYSxxx
: represents any logical unit assigned to magnetic tape upon which the MTC control statement is to operate.

[,nn]
: is the decimal number (01 through 99) which, if specified, represents the number of times the operation is to be performed. If nn is omitted, the operation is performed once.

OPTION Statement

The OPTION control statement is used to specify one or more of the options of the Job Control Processor. The format of the OPTION statement is:

```
// OPTION option1[,option2]...
```

The order in which the selected options appear in the operand field is arbitrary. Options are reset to the standard established at system generation time upon encountering the next JOB statement or the /& statement.

The options are:

LOG
causes the listing of columns 1 through 80 of all control statements on SYSLST. If LOG is not the standard established at system generation time, control statements are not listed until a LOG option is encountered. Once a LOG option statement is read, logging continues from job step to job step until a NOLOG option is encountered or until either the JOB or /& control statement is encountered.

NOLOG
suppresses the listing of all control statements on SYSLST until a LOG option is encountered, or until either the JOB or /& control statement is encountered.

DUMP
causes a dump of the registers and main storage to be printed on SYSLST in the case of an abnormal program termination (such as a program check).

NODUMP
suppresses the DUMP option.

LINK
indicates that the object module is to be linkage edited. When the LINK option is used, the output of the COBOL compiler is written on SYSLNK. The LINK option must always precede an EXEC LNKEDT statement in the job deck. (CATAL also causes the LINK option to be set.) LINK is not acceptable to the Job Control Processor operating in the foreground.

NOLINK
suppresses the LINK option. The COBOL compiler can also suppress the LINK option if the program contains an error that would preclude the successful execution of the program.

DECK
causes the COBOL compiler to punch an object module on SYSPCH. If both DECK and LINK are specified, the output of the compiler is written on both SYSPCH and SYSLNK.

NODECK
suppresses the DECK option.

LIST
causes the compiler to write the COBOL source statements on SYSLST.

NOLIST
suppresses the LIST option.

LISTX
causes the COBOL compiler to write a Procedure Division map on SYSLST.

NOLISTX
suppresses the LISTX option.

XREF
causes the COBOL compiler to write a symbolic cross-reference list on SYSLST.

NOXREF
suppresses the XREF option.

SYM
causes the COBOL compiler to write a Data Division map on SYSLST.

NOSYM
suppresses the SYM option.

ERRS
causes the COBOL compiler to write the diagnostic messages related to the source program on SYSLST.

NOERRS
suppresses the ERRS option.

CATAL
causes the cataloging of a phase or program in the core image library upon completion of a linkage editor job step. CATAL also causes the LINK option to be set. CATAL is not accepted by the Job Control Processor operating in a batched-job foreground environment.

STDLABEL
causes the standard label track to be cleared and all DASD or tape labels submitted after this point to be written on the standard label track. This option is reset to the USRLABEL option at end-of-job or end-of-job step. All file definition statements submitted after the STDLABEL option are available to any program in any area until another set of standard file definition statements is submitted. STDLABEL is not accepted by the Job Control Processor operating in a batched-job foreground environment. All file definition statements following OPTION STDLABEL are included in the standard file definition set until one of the following occurs:

- End-of-job step
- End-of-job
- OPTION USRLABEL is specified
- OPTION PARSTD is specified

USRLABEL
causes all DASD or tape labels submitted after this point to be written at the beginning of the user label track.

PARSTD
causes all DASD or tape labels submitted after this point to be written at the beginning of the partition standard label track. The PARSTD option is reset to the USRLABEL option at end-of-job or end-of-job step. All file definition statements submitted after the PARSTD option will be available to any program in the current partition until another set of partition standard file definition statements is submitted. All file definition statements submitted after OPTION PARSTD will be included in the standard file definition set until one of the following occurs:

- End-of-job step
- End-of-job
- OPTION USRLABEL is specified
- OPTION STDLABEL is specified

For a given filename, the sequence of search for label information during an OPEN is the USRLABEL area, followed by the PARSTD area, followed by the STDLABEL area.

The options specified in the OPTION statement remain in effect until a contradictory option is encountered or until a JOB control statement is read. In the latter case, the options are reset to the standard that was established at system generation time.

Any assignment for SYSLNK, after the occurrence of the OPTION statement, cancels the LINK and CATAL options. These two options are also canceled after each occurrence of an EXEC statement with a blank operand.

PAUSE Statement

The PAUSE control statement allows for operator intervention between job steps. The format of the PAUSE control statement is:

```
// PAUSE [comments]
```

The PAUSE control statement is effective just before the next input control statement in the job deck is read. The PAUSE control statement always prints on SYSLOG and SYSLST.

An example of this statement is:

```
// PAUSE SAVE SYS004, SYS005, MOUNT
    NEW TAPES
```

This sample statement instructs the operator to save the output tapes and mount two new tapes.

When the PAUSE statement is encountered by the Job Control Processor, the printer keyboard (IBM 1052) is unlocked for operator-message input. The end-of-communication indicator, B, causes processing to continue. If an IBM 1052 Printer is not available, the PAUSE control statement is ignored.

RESET Statement

The RESET control statement resets input/output assignments to the standard assignments. The standard assignments are those specified at system generation time plus any modifications made by the operator by means of the ASSGN command without the TEMP option. The RESET command is discussed in detail in the publication IBM System/360 Disk Operating System: System Control and System Service Programs. The format of the RESET statement is:

```
// RESET  {SYS   }
          {PROG  }
          {ALL   }
          {SYSxxx}
```

SYS
resets all system logical units to their standard assignments.

PROG
resets all programmer logical units to their standard assignments.

ALL
resets all system and programmer logical units to their standard assignments.

SYSxxx
resets the logical unit specified to its standard assignment.

RSTRT Statement

A restart facility is available for checkpoint programs. A programmer can use the source language RERUN clause in his program to cause checkpoint records to be written. This allows sufficient information to be stored so that program execution can be restarted at a specified point. The checkpoint information includes the registers, tape positioning information, a dump of main storage, and a restart address.

The restart facility allows the programmer to continue execution of an interrupted job at a point other than the beginning. The procedure is to submit a group of job control statements including a RSTRT control statement. The format is as follows:

```
// RSTRT SYSxxx,nnnn[,filename]
```

SYSxxx
is the symbolic unit name of the device on which the checkpoint records are stored. This unit must have been assigned previously.

nnnn
is the identification of the checkpoint record to be used for restarting. This serial number consists of four characters. It corresponds to the checkpoint identification used when the checkpoint was taken. The serial number is supplied by the checkpoint routine.

filename
is the symbolic name of the 2311 or 2314 disk checkpoint file used for restarting. It must be identical to the SYSxxx of the system-name specified in the RERUN clause. This operand applies only when specifying a 2311 or 2314 disk as the checkpoint file.

When a checkpoint is taken, the completed checkpoint is noted on SYSLOG. Restarting can be done from any checkpoint record, not just the last. The jobname specified in the JOB statement must be identical to the jobname used when the checkpoint was taken. The proper input/output device assignments must precede the RSTRT control statement.

Assignment of input/output devices to symbolic unit names may vary from the initial assignment. Assignments are made for restarting jobs in the same manner as assignments are made for normal jobs.

See the chapter "Program Checkout" for further details on taking checkpoints and restarting a program for which checkpoints have been taken.

UPSI Statement

The UPSI control statement allows the user to set program switches that can be tested by problem programs at execution time. The UPSI control statement has the following format:

```
// UPSI nnnnnnnn
```

nnnnnnnn
consists of from one to eight characters of 0, 1, or X. Positions containing 1 are set to 1; positions containing X are unchanged. Unspecified rightmost positions are assumed to be X.

The UPSI byte is the 24th byte in the Communication Region of the Supervisor. A complete description of the fields of the Communication Region is given in "Appendix H: Communication Region." The Job Control Processor clears the UPSI byte to binary zeros before reading control statements for each job. When the UPSI control statement is read, the Job Control Processor sets these bits to the user's specifications. Any combination of the eight bits can be tested in the COBOL source program at execution time by means of the source language switches UPSI-0 through UPSI-7.

CBL STATEMENT -- COBOL OPTION CONTROL CARD

Although most options for compilation are specified either at system generation time or in the OPTION control statement,

the COBOL compiler provides an additional statement, the CBL statement, for the specification of compile-time options unique to COBOL.

The CBL card must be placed between the EXEC FCOBOL statement and the first statement in the COBOL program. The CBL card cannot be continued. However, if specification of options will continue past column 71, multiple CBL cards may be used.

The options shown in the following format may appear in any order. No blanks may appear in the operand field. Underscoring indicates the default case.

```
CBL [BUF=nnnnn] [,SEQ  ] [,FLAGW]
                [,NOSEQ] [,FLAGE]

  [,SUPMAP] [,SPACEn] [,CLIST] [,STXIT]

    [,QUOTE] [,LIBR  ] [,TRUNC  ]
    [,APOST] [,NOLIBR] [,NOTRUNC]
```

CBL
must begin in column 2 and be preceded and followed by at least one blank.

BUF=nnnnn
the BUF option specifies the amount of storage to be assigned to each compiler work file buffer. nnnnn is a decimal number from 256 to 32,767. If this option is not specified, 256 is assumed.

SEQ
NOSEQ
indicates whether the compiler will sequence-check source statements.

FLAGW
FLAGE
FLAGW indicates that both warning and error diagnostic messages are to be listed; FLAGE indicates that only error diagnostic messages are to be listed.

SUPMAP
causes the LINK, DECK, CLIST, and LISTX options to be suppressed if an E-level diagnostic message is produced by the compiler.

SPACEn
indicates the type of spacing to be used on the output listing. n can be specified as either 1 (single spacing), 2 (double spacing), or 3 (triple spacing). If the SPACEn option is omitted, single spacing is provided.

CLIST
indicates that a condensed listing is to be produced. The Procedure Division portion of the object listing will contain the address of the first generated instruction for each verb. The CLIST option overrides the LISTX or NOLISTX options. The LISTX or NOLISTX options are either established at system generation time or specified in the OPTION control statement.

STXIT
enables a user error declarative to get control when an input/output error occurs on a unit record device.

QUOTE
APOST
QUOTE indicates to the compiler that the double quotation marks (") should be accepted as the character to delineate literals; APOST indicates that the apostrophe (') should be accepted. The compiler will generate the specified character for the figurative constant QUOTE(S).

LIBR
NOLIBR
LIBR specifies that the BASIS card and/or the COPY statement are used in the source program. NOLIBR allows additional table space to be available during compilation.

NOTRUNC
TRUNC
NOTRUNC specifies nonstandard truncation of COMPUTATIONAL items. With nonstandard truncation, an item is truncated on the basis of the amount of storage it occupies, rather than on the basis of its PICTURE clause.

For example, suppose that the programmer using the NOTRUNC option describes two Data Division items as follows:

```
A PICTURE S9999 USAGE
   COMPUTATIONAL.
B PICTURE S9 USAGE COMPUTATIONAL.
```

After the following Procedure Division statement is executed, B contains the value of A, since each item occupies one halfword of storage.

```
MOVE A TO B
```

With the TRUNC option, standard truncation of COMPUTATIONAL items occurs. Standard truncation is based

on the PICTURE clause of the item being moved.

For example, with standard truncation, after the MOVE described above is executed, B contains only the low-order digit of A. According to the rules of standard truncation, COMPUTATIONAL items are converted to internal decimal, moved with decimal alignment, and truncated.

JOB CONTROL COMMANDS

Job control commands are distinguished from job control statements by the absence of // blank in positions 1 through 3 of each command. They permit the operator to adjust the system according to day-to-day operating conditions. This is particularly true in the area of device assignment, where the operator may need to (1) communicate to the system that a device is unavailable, or (2) designate a different device as the standard for a given symbolic unit. Therefore, these commands normally are not a part of the regular job deck for a job. Job control commands tend to be effective across jobs, whereas job control statements are confined within a job.

Job control commands are discussed in detail in the publication IBM System/360 Disk Operating System: System Control and System Service Programs.

LINKAGE EDITOR CONTROL STATEMENTS

Object modules used as input to the Linkage Editor must include linkage editor control statements. There are four linkage editor control statements: PHASE, INCLUDE, ENTRY, and ACTION.

The discussion of these statements in this publication applies only to background programs. For foreground programs, see the publication IBM System/360 Disk Operating System: System Control and System Service Programs.

Linkage editor control statements initially enter the system through the device assigned to SYSRDR as part of the input job stream. PHASE and INCLUDE statements may also be present on SYSIPT or in the relocatable library. All four statements are verified for operation (INCLUDE, ACTION, ENTRY, or PHASE) and are copied to SYSLNK to become input when the Linkage Editor is executed.

Linkage editor control statements must be blank in position 1 of the statement. The operand field is terminated by the first blank position. It cannot extend beyond column 72.

The Linkage Editor is executed as a distinct job step. Figure 5 shows how the linkage editor function is performed as a job step in three kinds of operations.

1. Catalog Programs in Core Image Library. The linkage editor function is performed immediately preceding the operation that catalogs programs into the core image library. When the CATAL option is specified, programs edited by the Linkage Editor are cataloged in the core image library by the Librarian after the editing function is performed. The sequence of this operation is shown in Part Ⓐ of Figure 5. Note that the input for the LNKEDT function could contain modules from the relocatable library instead of, or in addition to, those modules from the card reader, tape unit, or mass storage unit extent assigned to SYSIPT. This is accomplished by naming the module(s) to be copied from the relocatable library in an INCLUDE statement.

2. Load-and-Execute. The sequence of this operation is shown in Part Ⓑ of Figure 5. Specifying OPTION LINK causes the Job Control Processor to open SYSLNK, and allows the Job Control Processor to place the object module(s) and linkage editor control statements on SYSLNK. As with the catalog operation, the input can consist of object modules from the relocatable library instead of, or in addition to, those modules from the card reader, tape unit, or disk extent assigned to SYSIPT. This is accomplished by specifying the name of the module to be included in the operand of an INCLUDE statement. After the object modules have been edited and placed in the core image library, the program is executed. The blank operand in the EXEC control statement indicates that the program that has just been linkage edited and temporarily stored in the core image library is to be executed.

3. Compile-and-Execute. Source modules can be compiled and then executed in a single sequence of job steps. In order to do this, the COBOL compiler is directed to write the object module directly on SYSLNK. This is done by using the LINK option in the OPTION control statement. Upon completion of this output operation, the linkage

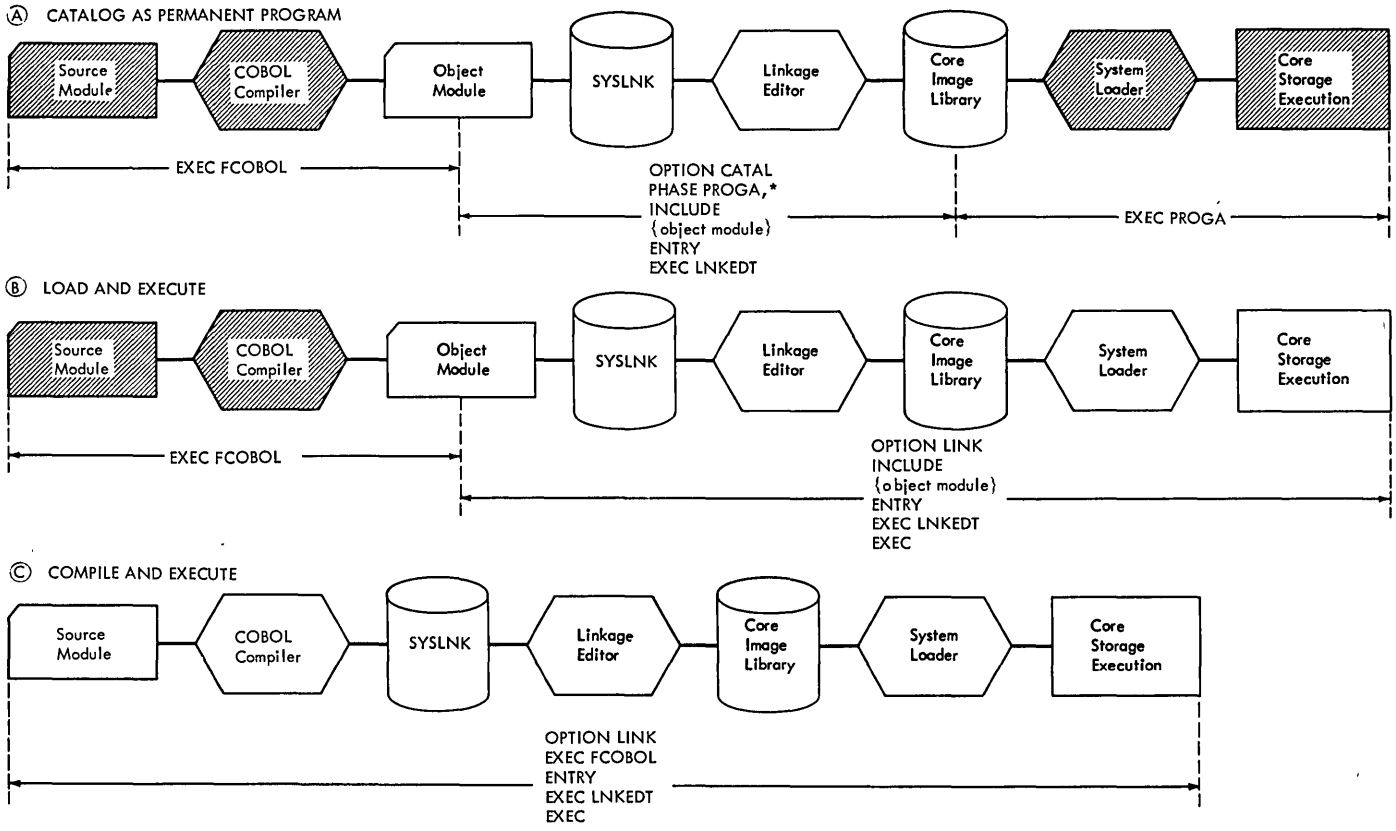


Figure 5. Job Definition -- Use of the Librarian

editor function is performed. The program is linkage edited and temporarily stored in the core image library. The sequence of this operation is shown in Part Ⓒ of Figure 5.

Control Statement Placement

The placement of linkage editor control statements is subject to the following rules:

1. The ACTION statement must be the first linkage editor control statement encountered in the input stream; otherwise, it is ignored.
2. The PHASE statement must precede each object module that is to begin a phase.
3. The INCLUDE statement must be specified for each object module that is to be included in a program phase.
4. A single ENTRY statement should follow the last object module when multiple object modules are processed in a single linkage editor run.

ACTION and ENTRY statements, when present, must be on SYSRDR. PHASE and INCLUDE statements may be present on SYSRDR, SYSIPT, or in the relocatable library.

PHASE Statement

The PHASE statement must be specified if the output of the Linkage Editor is to consist of more than one phase or if the program phase is to be cataloged in the core image library. Each object module that begins a phase must be preceded by a PHASE statement. Any object module not preceded by a PHASE statement will be included in the current phase.

The statement provides the Linkage Editor with a phase name and an origin point for the phase. The PHASE statement is in the following format:

```
PHASE name,origin[,NOAUTO]
```

name
: is the symbolic name of the phase. It is the name under which the program phase is to be cataloged. This name does not have to be the name specified in the PROGRAM-ID paragraph in the Identification Division of the source program and, in the case of overlay and sort, it should not be the same. It must consist of from one to eight alphanumeric characters. Phases that are to be executed in an overlay structure should have phase names of from five to eight alphanumeric characters, the first four of which should be the same. An asterisk cannot be used as the first character of a phase name.

origin
: indicates to the Linkage Editor the starting address of this specific phase. An asterisk may be used as an origin specification to indicate that this phase is to follow the previous phase. This origin specification format of the PHASE statement covers all applications that do not include setting up overlay structures. See the chapter "Calling and Called Programs" for information on the PHASE statement for overlay applications.

NOAUTO
: indicates that the Automatic Library Look-Up (AUTOLINK) feature is suppressed for both the private relocatable library and the system relocatable library. (The use of NOAUTO causes the AUTOLINK process to be suppressed for that phase only.) The AUTOLINK feature is discussed later in this chapter.

INCLUDE Statement

The INCLUDE statement must be specified for each object module deck or object module in the relocatable library that is to be included in a program phase. The format of the INCLUDE statement is as follows:

```
INCLUDE [module-name][,(namelist)]
```

The INCLUDE statement has two optional operands. When both operands are used, they must be in the prescribed order. When the first operand is omitted and the second operand is used, a comma must precede the second operand.

module-name
: must be specified when the object module is in the relocatable library. It is not specified when the module to be included is in the form of a card deck being entered from SYSIPT. module-name is the name under which the module was cataloged in the library, and must consist of from one to eight alphanumeric characters.

(namelist)
: causes the Linkage Editor to construct a phase from the control sections specified in the list. Since control sections are of no interest to the COBOL programmer, users interested in this option should refer to the description of the INCLUDE statement in the publication IBM System/360 Disk Operating System: System Control and System Service Programs.

ENTRY Statement

The ENTRY statement is required only if the user wishes to provide a specific entry point in the first phase produced by the Linkage Editor. When no ENTRY statement is provided, the Job Control Processor writes an ENTRY statement with a blank operand on SYSLNK to ensure that an ENTRY statement will be present to halt linkage editing. The transfer address will be the load address of the first phase. The ENTRY statement is described further in the publication IBM System/360 Disk Operating System: System Control and System Service Programs.

ACTION Statement

The ACTION statement is used to indicate linkage editor options. When used, the statement must be the first linkage editor statement in the input stream. The format of the ACTION statement is as follows:

```
        ( CLEAR  )
        ( MAP    )
        ( NOMAP  )
ACTION  ( NOAUTO )
        ( CANCEL )
        ( F1     )
        ( F2     )
```

CLEAR
indicates that the entire temporary portion of the core image library will be set to binary zero before the beginning of the linkage editor function. CLEAR is a time-consuming function and should be used only when necessary.

MAP
indicates that SYSLST is available for diagnostic messages. In addition, a main storage map is output on SYSLST.

NOMAP
indicates that SYSLST is unavailable when performing the linkage-edit function. The mapping of main storage is not performed, and all linkage editor diagnostic messages are listed on the printer-keyboard (SYSLOG).

NOAUTO
suppresses the AUTOLINK function for both the private and system relocatable libraries during the linkage editing of the entire program. AUTOLINK is discussed later in this chapter.

CANCEL
causes an automatic cancellation of the job if any of the linkage editor errors 2100I through 2170I occur. These diagnostic messages can be found in the publication IBM System/360 Disk Operating System: System Control and System Service Programs.

F1 and F2
are options used in conjunction with programs executed in the foreground area. See the publication IBM System/360 Disk Operating System: System Control and System Service Programs.

AUTOLINK FEATURE

If any references to external-names are still unresolved after all modules have been read from SYSLNK, SYSIPT, and/or the relocatable library, AUTOLINK collects each unresolved external reference from the phase. It then searches the private relocatable library (if SYSRLB has been assigned) and the system relocatable library for module names identical to the unresolved names and includes these modules in the program phase. This feature should not be suppressed (via PHASE or ACTION statements) in linkage editor job steps which include COBOL subroutines cataloged in the relocatable library. See the chapter "Calling and Called Programs" for additional details.

The system residence device (SYSRES) for the Disk Operating System can contain three libraries: the core image library, the relocatable library, and the source statement library. Executable programs (core image format) are stored in the core image library; relocatable object modules are stored in the relocatable library; and source language routines are stored in the source statement library.

The core image library is required for each disk resident system. The relocatable library and the source statement library are not required.

In addition to the three system libraries located on SYSRES, the user may also request creation of private source statement and relocatable libraries. These libraries are discussed under "Private Libraries" in this chapter.

LIBRARIAN

The Librarian is a group of programs that perform three major functions:

1. Maintenance
2. Service
3. Copy

Maintenance functions are used to catalog (that is, add), delete, or rename components of the three libraries, condense libraries and directories, set a condense limit for an automatic condense function, reallocate directory and library extents, and update the source statement library.

The copy function is used either to completely or selectively copy the disk on which the system resides. Service functions are used to translate information from a particular library to printed (displayed) or punched output.

Only the catalog maintenance function of the Librarian is discussed in this publication for the three system libraries. In addition, the update function of the source statement library is discussed. A complete description of librarian functions can be found in the publication IBM System/360 Disk Operating System: System Control and System Service Programs.

CORE IMAGE LIBRARY

The core image library may contain any number of programs. Each program consists of one or more separate phases. Associated with the core image library is a core image directory which contains a unique descriptive entry for each phase in the core image library. These entries in the core image directory are used to locate and retrieve phases from the core image library.

Cataloging and Retieving Program Phases -- Core Image Library

If a program is to be cataloged in the core image library, the job control statement // OPTION with the CATAL option must be specified prior to the first linkage editor control card, and must precede the first PHASE card of the program to be cataloged. Upon successful completion of the linkage editor job step, output from the linkage editor is placed in the core image library as a permanent member. The program phase is cataloged under the name specified in the PHASE statement.

If a phase in the core image library is to be replaced by a new phase having the same name, only the catalog function need be used. The previously cataloged phase of the same name is implicitly deleted from the core image directory by the catalog function, and the space it occupies in the library can later be released by the condense function.

Note: The necessary ASSGN control statements must follow the // JOB control statement if the current assignments are not the following:

1. SYSRDR -- Card reader, tape unit, or disk extent
2. SYSIPT -- Card reader, tape unit, or disk extent
3. SYSLST -- Printer, tape unit, or disk extent
4. SYSLOG -- Printer keyboard
5. SYSLNK -- Disk extent

The following is an example of cataloging a single phase, FOURA, into the core image library. (The program phase FOURA can be executed in the next job step by specifying the // EXEC statement with a blank name field.)

```
// JOB CATALOG
// OPTION CATAL
   PHASE FOURA,*
   INCLUDE

     {object deck}
/*
// LBLTYP TAPE
// EXEC LNKEDT
// EXEC
/&
```

To compile, linkage edit, and catalog the phase FOURA into the core image library in the same job, the following job deck could be used:

```
// JOB CATALOG
// OPTION CATAL
   PHASE FOURA,*
// EXEC FCOBOL

     {source deck}
/*
// EXEC LNKEDT
/*
/&
```

When the phase is executed in a subsequent job, the EXEC statement that calls for execution must specify FOURA, i.e., the name by which the phase has been cataloged.

```
// JOB EXJOB
// EXEC FOURA
/&
```

RELOCATABLE LIBRARY

The relocatable library contains any number of modules. Each module is a complete object deck in relocatable format. The purpose of the relocatable library is to allow the user to maintain frequently used routines in residence and combine them with other modules without recompiling.

Associated with the relocatable library is the relocatable directory. The directory contains a unique, descriptive entry for each module in the relocatable library. The entries in the relocatable directory are used to locate and retrieve modules in the relocatable library.

MAINTENANCE FUNCTIONS

To request a maintenance function for the relocatable library, the following control statement is used:

```
// EXEC MAINT
```

Cataloging a Module -- Relocatable Library

The catalog function adds a module to the relocatable library. A module in the relocatable library is the output of a complete COBOL compilation.

The catalog function implies a delete function. Thus, if a module exists in the relocatable library with the same name as a module to be cataloged, the module in the library is deleted by deleting reference to it in the relocatable directory.

The CATALR control statement is required to add a module to the relocatable library. The format of the CATALR control statement is:

```
CATALR module-name [,v.m]
```

module-name
: is the name by which the module is known to the control program. The module-name consists of from one to eight characters , the first of which must not be an asterisk.

v.m
: specifies the change level at which the module is to be cataloged. v may be any decimal number from 0 through 127. m may be any decimal number from 0 through 255. If this operand is omitted, a change level of 0.0 is assumed. A change level can be assigned only when a module is cataloged.

All control statements required to catalog an object module must be read from SYSIPT. For the catalog function, device assignments must be as follows:

1. SYSRDR -- Card reader, tape unit, or disk extent
2. SYSIPT -- Card reader, tape unit, or disk extent
3. SYSLST -- Printer, tape unit, or disk extent
4. SYSLOG -- Printer keyboard

Note: If SYSRDR and/or SYSIPT are assigned to a tape unit, the MAINT program assumes that the tape is positioned to the first input record. The tape is not rewound at the end of the job.

The following is an example of compiling a source program and cataloging the resultant module in the relocatable library. The job deck is read from SYSIPT.

```
// JOB NINE
// OPTION DECK
// EXEC FCOBOL

     {source deck}
/*
// PAUSE PLACE DECK AFTER CATALR CARD
// EXEC MAINT
   CATALR MOD9

     (punched deck goes here)
/*
/&
```

In the above example, as a result of the compile step, the object module is written on SYSPCH. The next job step catalogs the object module (MOD9) into the relocatable library. Since the object module must be cataloged from SYSIPT, a message to the operator instructs him to place the object module on SYSIPT behind the CATALR statement.

The following is an example of cataloging two previously created object modules in the relocatable library:

```
// JOB EIGHT
// EXEC MAINT
   CATALR MOD8A

     {object deck}
   CATALR MOD8B

     {object deck}
/*
/&
```

SOURCE STATEMENT LIBRARY

The source statement library contains any number of books. Each book in the source statement library is composed of a sequence of source language statements. The purpose of the source statement library is to allow the COBOL programmer to initiate the compilation of a book into the source program by using the COPY statement or BASIS card.

Each book in the source statement library is classified as belonging to a specific sublibrary. Sublibraries are defined for two programming languages: Assembler and COBOL. Individual books are classified by sublibrary names. Therefore, books written in each of these languages may have the same name.

Associated with the source statement library is a source statement directory. The directory contains a unique descriptive entry for each book in the source statement library. The entries in the source statement directory are used to locate and retrieve books in the source statement library.

MAINTENANCE FUNCTIONS

To request a maintenance function for the source statement library, the following control statement must be used:

```
// EXEC MAINT
```

Cataloging a Book -- Source Statement Library

The CATALS control statement is required to add a book to a sublibrary of the source statement library.

A book added to a sublibrary of the source statement library is removed by using the delete function. When a book exists in a sublibrary with the same name as a book to be cataloged in that sublibrary, the existing book in the sublibrary is deleted. The following is the format of the CATALS control statement:

```
┌──────────┬──────────────────────────────────┐
│ CATALS sublib.library-name[,v.m[,C]]        │
└──────────┴──────────────────────────────────┘
```

The operation field contains CATALS.

sublib
: represents the sublibrary to which a book is to be cataloged and can be:

  Any alphanumeric character (0-9, A-Z, #, $, and @) representing source statement libraries. The characters A and C have special uses:

  A is used for the Assembler sublibrary

  C is used for the COBOL sublibrary

The sublib qualifier is required. If omitted, the operand will be flagged as invalid and no processing will be done on the book.

library-name
: represents the name of the book to be cataloged. The library-name consists of from one to eight alphanumeric characters, the first of which must be alphabetic. It is the name the programmer uses to retrieve the book when using the source language COPY statement or BASIS card.

v.m
: specifies the change level at which the book is to be cataloged. v may be any decimal number from 0 through 127; m may be any decimal number from 0 through 255. If this operand is omitted, a change level of 0.0 is assumed. The v.m operand becomes part of the entry in the directory for the specified book. Its value is incremented each time an update is performed on the book.

C
: indicates that change level verification is required before updates are accepted for this book.

See the UPDATE control statement, discussed later in this chapter, for its relationship to the v.m and C operands of the CATALS control statement.

In addition to the CATALS control statement, a control statement of the following form must precede and follow the book to be cataloged:

```
BKEND [sublib.library-name],[SEQNCE],
   [count],[CMPRSD]
```

All operand entries are optional. When used, the entries must be in the prescribed order and need appear only in the BKEND statement preceding the book to be cataloged.

The first entry in the operand field is identical to the operand of the CATALS control statement.

SEQNCE
: specifies that columns 76 to 80 of the card images constituting the book are to be checked for ascending sequence numbers. If an error is detected in the sequence checking, an error message is printed. The error can be corrected, and the book can be recataloged.

count
: specifies the number of card images in the book. When the count operand is used, the card input is counted, beginning with the preceding BKEND statement and including the subsequent BKEND statement. If an error is detected in the card count, an error message is printed. The error can be corrected, and the book can be recataloged.

CMPRSD
: indicates that the book is to be cataloged in the library in compressed format.

Card input for the catalog function is from the device assigned to SYSIPT. The CATALS control statement is also read from the device assigned to SYSIPT. For the catalog function, device assignments must be as follows:

1. SYSRDR -- Card reader, tape unit, or disk extent
2. SYSIPT -- Card reader, tape unit, or disk extent
3. SYSLST -- Printer, tape unit, or disk extent
4. SYSLOG -- Printer keyboard

Frequently used Environment Division, Data Division, and Procedure Division entries can be cataloged in the COBOL sublibrary of the source statement library. A book in the source statement library might consist, for example, of a file description of the Data Division or a paragraph of the Procedure Division.

The following is an example of cataloging a file description in the COBOL sublibrary of the source statement library.

```
// JOB ANYNAME
// EXEC MAINT
   CATALS C.FILEA
   BKEND C.FILEA
          BLOCK CONTAINS 13 RECORDS
          RECORD CONTAINS 120 CHARACTERS
          LABEL RECORDS ARE STANDARD
          DATA RECORD IS RECA.
   BKEND
/*
/&
```

Retrieving a Cataloged Book -- COBOL COPY Statement: The preceding file description can be included in a COBOL source program by writing the following statement:

```
FD FILEB COPY FILEA.
```

Note that the library entry does not include FD or the file-name. It begins with the first clause that is actually to follow the file-name. This is true for all options of the COPY statement. However, data entries in the library may have a level number (01 or 77) identical to the level number of the data-name that precedes the COPY statement. In this case, all information about the library data-name is copied from the library and all references to the library data-name are replaced by the data-name in the program if the REPLACING option is specified. For example, assume the following data entry is cataloged under the library-name DATAR,

```
01  PAYFILE USAGE IS DISPLAY.
    02  CALC PICTURE 99.
    02  GRADE PICTURE 9
        OCCURS 1 DEPENDING ON CALC OF
        PAYFILE.
```

and the following statement is written in a COBOL source module:

```
01  GROSS COPY DATAR REPLACING PAYFILE
     BY GROSS.
```

The compiler interprets this as:

```
01  GROSS USAGE IS DISPLAY.
    02  CALC PICTURE 99.
    02  GRADE PICTURE 9
        OCCURS 1 DEPENDING ON CALC OF
        GROSS.
```

Note also that the library-name is used to identify the book in the library. It has no other use in the COBOL program.

Text cataloged in the source statement library must conform to COBOL margin restrictions.

The COBOL COPY statement is discussed in detail in the section "Extended Source Program Library Facility."

Updating Books -- Source Statement Library

The update function is used to make changes to properly identified statements within a book in the source statement library. Statements are identified in the identification field, columns 73 through 80, which is fixed in format as follows:

Columns 73-76	Program identification which must be constant throughout the book.
Columns 77-80	Sequence number of the statement within the book.

One or more source statements may be added to, deleted from, or replaced in a book in the library without the necessity of replacing the entire book. The update function also provides these facilities:

1. Resequencing statements within a book in the source statement library
2. Changing the change level (v.m) of the book
3. Adding or removing the change level requirement
4. Copying a book with optional retention of the old book with a new name (for backup purposes)

The UPDATE control statement is used for the update function and has the following format:

```
UPDATE sublib.library-name,[s.book1],
   [v.m],[nn]
```

The operation field contains UPDATE.

sublib
: represents the sublibrary that contains the book to be updated. It may be any of the characters 0 through 9, A through Z, #, $, or @.

s.book1
: provides a temporary update option. The old book is renamed s.book1 and the updated book is named sublib.library-name. s indicates the sublibrary that contains the old, renamed book. It may be one of the characters 0 through 9, A through Z, #, $, or @. If this operand is not specified, the old book is deleted.

v.m
: represents the change level of the book to be updated. v may be any decimal number from 0 through 127; m may be any decimal number from 0 through 255. This operand must be present if change level verification is to be performed. Use of the optional entry C in the CATALS control statement at the time the book is cataloged in the library determines whether change level verification is required before updating. If the directory entry specifies that change level verification is not required before updating, the change level operand in the UPDATE control statement is ignored.

If the change level is verified, the change level in the book's directory entry is increased by 1 by the system for verification of the next update. If m is at its maximum value and an update is processed, m is reset to 0 and the value of v is increased by 1. If both v and m are at their maximum values and an update is processed, both v and m are reset to 0.

nn

represents the resequencing status required for the update. nn may be a 1- or 2-character decimal number from 1 through 10, or it may be the word NO. If nn is a decimal number, it represents the increment that will be used in resequencing the statements in the book. If nn is NO, the statements will not be resequenced. If nn is not specified, the statements will be resequenced with an increment of 1. When a book is resequenced, the sequence number of thefirst statement is 0000. For example, if a book is cataloged in the source statement library with sequence numbers ranging from 0010 through 1000 with increments of 5 for each statement:

and nn is not specified when the update function is performed, the book is resequenced with numbers 0000, 0001, 0002, ... etc.

and NO is specified, insertions, deletions, and/or replacements are made with no effect on the original sequence numbers.

and nn is specified as 2, the book is resequenced with numbers 0000, 0002, 0004, ... etc., regardless of the original sequencing of the book in the library or the sequence numbers of the added or replacement cards.

The UPDATE control statement is followed by ADD, DEL (delete), and/or REP (replace) control statements as required, followed by the terminating END statement. The ADD, DEL, REP, and END statements are identified as update control statements by a right parenthesis in the first position (column 1 in card format). This is a variation from the general librarian control statement format; thus, it clearly identifies these control statements as part of the update function.

ADD Statement: The ADD statement is used for the addition of source statements to a book. The format is:

```
) ADD seq-no
```

ADD indicates that source statements following this statement are to be added to the book.

seq-no

represents the sequence number of the statement in the book after which the new statements are to be added. It may be any decimal number consisting of from one to four characters.

DEL Statement: The DEL statement causes the deletion of source statements from the book. The format is:

```
) DEL first-seq-no[,last-seq-no]
```

DEL indicates that statements are to be deleted from the book.

first-seq-no
last-seq-no

represent the sequence numbers of the first and last statements of a section to be deleted. Each number may be a decimal number consisting of from one to four characters. If last-seq-no is not specified, the statement represented by first-seq-no is the only statement deleted.

REP Statement: The REP statement is used when replacement of source statements is required in a book. The format is:

```
) REP first-seq-no[,last-seq-no]
```

REP indicates that source statements following this statement are to replace existing statements in a book.

first-seq-no
last-seq-no

represent the sequence numbers of the first and last statements of a section to be replaced. Each number may be a decimal number consisting of from one to four characters. Any number of new statements can be added to a book when a section is replaced. (The number of statements added need not equal the number of statements being replaced.)

Sequence number 9999 is the highest number acceptable for a statement to be updated. If the book is so large that statement sequence numbers have "wrapped around" (progressed from 9998, 9999, to 0000,0001), it will not be possible to update statements 0000 and 0001.

END Statement: This statement indicates the end of updates for a given book. The format is:

```
| ) END [v.m[,C]]                                    |
```

v.m

represents the change level to be assigned to the book after it is updated; v may be any decimal number from 0 through 127. m may be any decimal number from 0 through 255. This operand provides an additional means of specifying the change level of a book in the library. (The other method is through the use of the v.m operand in the CATALS statement.)

C

indicates that change level verification is required before any subsequent updates for a given book.

If v.m is specified and C is omitted, the book does not require change level verification before a subsequent update. This feature removes a previously specified verification requirement for a particular book.

If both optional operands are omitted, the change level in the book's directory entry is increased as a result of the update, and the verification requirement remains unchanged.

Logical Unit Assignment and Control Statement Placement:

For the update function, SYSIN must be assigned to a card reader, a tape unit, or a disk unit. SYSLST must be assigned to a printer, a tape unit, or a disk extent; SYSLOG must be assigned to the printer keyboard.

Control statement input for the update function, read from the device assigned to SYSIN, must be in the following order:

1. The JOB control statement.
2. The ASSGN control statements, if the current assignments are not those required. The ASSGN control statements that can be used are SYSIN, SYSLST, and SYSLOG.
3. The EXEC MAINT control statement.
4. The UPDATE control statement.
5.) ADD,) DEL, or) REP statements with appropriate source statements.
6.) END statement.
7. The /* control statement.
8. The /& control statement, which is the last control statement of the job.

The source statement library can also be updated by using the DELETE and INSERT cards. These are discussed in "Extended Source Program Library Facility" in this chapter, and in the publication IBM System/360 Disk Operating System: American National Standard COBOL.

UPDATE Function -- Invalid Operand Defaults

UPDATE Statement:

1. If the first or second operand is invalid, the statement is flagged, the book is not updated, and the remaining control statements are checked to determine their validity.
2. If change level verification is required and the incorrect change level is specified, the statement is flagged, the book is not updated, and the remaining control statements are checked to determine their validity.
3. If the resequencing operand is invalid, resequencing is done in increments of 1.

ADD, DEL, or REP Statements:

1. If there is an invalid operation or operand in an ADD, DEL, or REP statement, the statement is flagged, the book is not updated, and the remaining control statements are checked to determine their validity. All options of the UPDATE and END statements are ignored.
2. The second operand must be greater than the first operand in a DEL or REP statement. If not, the statement is considered invalid and is flagged, the book is not updated, and the remaining control statements are checked to determine their validity. All options of the UPDATE and END statements are ignored.
3. All updates to a book between an UPDATE statement and an END statement must be in ascending sequential order of statement sequence numbers. The

first operand of a DEL or REP statement must be greater than the last operand of the preceding control statement. The operand of an ADD statement must be equal to or greater than the last operand of the preceding control statement. Consecutive ADD statements must not have the same operand. If these conditions are not met, the default is the same as for items 1 and 2.

END Statement: If the first operand of the END statement is invalid, the statement is flagged, both operands are ignored, and the book is updated as though no operands were specified. If the second operand is invalid, the statement is flagged, the operand is ignored, and the book is updated as though the second operand were not specified.

Out-of-Sequence Updates: If the source statements to be added to a book are not in sequence or do not contain sequence numbers, the book is updated, and a message indicating the error appears following the END statement. If the resequencing option has been specified in the UPDATE statement, the book is sequenced by the specified value, and subsequent updating is possible. If the resequencing option is not specified, the book is resequenced in increments of 1, and subsequent updating will be possible. If the resequencing option NO is specified, the book will be out of sequence, and subsequent updating may not be possible.

PRIVATE LIBRARIES

Private libraries are desirable in the system to permit some libraries to be located on a disk pack other than the one used by SYSRES.

Private libraries are supported for the relocatable library and for the source statement library on both the 2311 and 2314 mass storage devices. However, the following restrictions apply:

1. The private library must be on the same type of disk device as SYSRES.

2. Reference may be made to a private relocatable library only if SYSRLB is assigned. If SYSRLB is assigned, the system relocatable library cannot be changed.

3. Reference may be made to a private source statement library only if SYSSLB is assigned. If SYSSLB is assigned, the system source statement library cannot be changed.

4. Private libraries cannot be reallocated.

5. The COPY function is not effective for private libraries except when they are being created.

An unlimited number of private libraries is possible. However, each must be distinguished by a unique file identification in the DLBL statement for the library. No more than one private relocatable library and one private source statement library may be assigned in a given job.

The creation and maintenance of private libraries is discussed in the publication IBM System/360 Disk Operating System: System Control and System Service Programs.

SOURCE LANGUAGE CONSIDERATIONS

To use the private source statement library for COPY, BASIS, INSERT, and DELETE (see "Extended Source Program Library Facility" for further details), the ASSGN, DLBL, and EXTENT control statements that define this private library must be present in the job deck for compilation. When present, a search for the book is made in the private library. If it is not there, the system library is searched. If the statements for the private library are not present, the system library is searched. A programmer may create several private libraries, but only one private library can be used in a given job.

EXTENDED SOURCE PROGRAM LIBRARY FACILITY

A complete program may be included as an entry in the source statement library by using the catalog function. This program can then be retrieved by a BASIS card and compiled in a subsequent job.

The following control statements would be used to catalog the program SAMPLE as a book in the COBOL sublibrary of the source statement library:

```
// JOB CATALOG
// EXEC MAINT
   CATALS C.SAMPLE
   BKEND C.SAMPLE

     {source program}

   BKEND
/*
/&
```

When compiling a program that has been cataloged in the COBOL sublibrary of the source statement library, a BASIS card brings in an entire source program. The following control statements could be used to compile the cataloged program SAMPLE:

```
// JOB PGM1
// OPTION LOG,DECK,LIST,LISTX,ERRS
// EXEC FCOBOL
BASIS SAMPLE
/*
/&
```

INSERT or DELETE cards may follow the BASIS card if the user wishes to modify the book SAMPLE before it is processed by the compiler. The original source program must have been coded with sequence numbers in columns 1 through 6 of each source card.

The INSERT statement will add new source statements after the specified sequence numbers. The DELETE statement will delete the statements indicated by the sequence numbers, or will delete more than one statement when the first and last sequence numbers to be deleted are specified, separated by a hyphen. Source program cards may follow a DELETE card for insertion before the card following the last one deleted. The sequence numbers in columns 1 through 6 are used to update COBOL source statements at compilation time, and are in effect for the one run only.

Assume that a company runs its payroll program each week as a source program taken from the source statement library. The name of the program is PAYROLL. During the year, social security tax (FICA) is deducted at the rate of 4-2/5% each week for all personnel until earnings exceed $7800. The coding to accomplish this is shown in Figure 6.

At the beginning of the year, the test for earnings over $7800 is taken out of the program until a more appropriate time later in the year. In addition, at the beginning of the year, a union contract dictates that all draftsmen receive a 5% pay increase. Assume that records for all personnel contain an occupation code. The code identifying draftsmen is DR. The programmer can program these changes as shown in Figure 7.

The altered program will contain the coding shown in Figure 8.

```
000730                IF ANNUAL-PAY GREATER THAN 7800 GO TO PAY-WRITE.
000735                IF ANNUAL-PAY GREATER THAN 7800 - BASE-PAY GO TO LAST-FICA.
000740   FICA-PAYR.   COMPUTE FICA-PAY = BASE-PAY * .044
000745                      MOVE FICA-PAY TO OUTPUT-FICA.
000750   PAY-WRITE.         MOVE BASE-PAY TO OUTPUT-BASE.
000755                      ADD BASE-PAY TO ANNUAL-PAY.
  .                   .
  .                   .
  .                   .
000850                STOP RUN.
```

Figure 6. Sample Coding to Calculate FICA

```
// JOB PGM2
// OPTION LOG,DECK,LIST,LISTX,ERRS
// EXEC FCOBOL
 CBL QUOTE
BASIS PAYROLL
DELETE 000730, 000735
            IF OCCUPATION-CODE = "DR" PERFORM PAY-INCREASE THRU EX1.
INSERT 000850
       PAY-INCREASE. MULTIPLY 1.05 BY BASE-PAY.
       EX1.          EXIT.
/&
```

Figure 7. Altering a Program from the Source Statement Library Using INSERT and DELETE Cards

```
            IF OCCUPATION-CODE = "DR" PERFORM PAY-INCREASE THRU EX1.
000740 FICA-PAYR. COMPUTE FICA-PAY = BASE-PAY * .044
000745           MOVE FICA-PAY TO OUTPUT-FICA.
000750 PAY-WRITE. MOVE BASE-PAY TO OUTPUT-BASE.
000755           ADD BASE-PAY TO ANNUAL-PAY.
.                .
.                .
.                .
000850           STOP RUN.
       PAY-INCREASE. MULTIPLY 1.05 BY BASE-PAY.
       EX1.          EXIT.
```

Figure 8. Effect of INSERT and DELETE Cards

A programmer using the American National Standard COBOL compiler under the IBM System/360 Disk Operating System has several methods available to him for testing, debugging, and revising his programs for increased operating efficiency.

The COBOL debugging language can be used by itself or in conjunction with other COBOL statements. A dump can also be used for program checkout.

DEBUG LANGUAGE

The COBOL debugging language is designed to assist the COBOL programmer in producing an error-free program in the shortest possible time. The following sections discuss the use of the debug language and other methods of program checkout.

The three debug language statements are TRACE, EXHIBIT, and ON. Any one of these statements can be used as often as necessary. They can be interspersed throughout a COBOL source program, or they can be contained in a packet in the input stream to the compiler.

Program checkout may not be desired after testing is completed. A debug packet can be removed after testing to eliminate the extra object program coding generated for the debug statements.

The output produced by the TRACE and EXHIBIT statements is listed on the system logical output device (SYSLST).

The following discussions describe methods of using the debug language.

FLOW OF CONTROL

The READY TRACE statement causes the compiler-generated card numbers for each section-name and paragraph-name to be displayed. These card numbers are listed on SYSLST at execution time when control passes to these sections and paragraphs. Hence, the output of the READY TRACE statement appears as a list of card numbers.

To reduce the length of the list and the time taken to generate it, a trace can be stopped with a RESET TRACE statement. The READY TRACE/RESET TRACE combination is helpful in examining a particular area of the program where the flow of control is difficult to determine, e.g., code consists of a series of PERFORM statements or nested conditional statements. The READY TRACE statement can be coded so that the trace begins before control passes to that area. The RESET TRACE statement can be coded so that the trace stops when the program has passed beyond the area.

Use of the ON statement with the TRACE statement allows conditional control of the tracing. When the COBOL compiler encounters an ON statement, it creates a counter which is incremented during execution, whenever control passes through the ON statement. For example, if an error occurs when a specific record is processed, the ON statement can be used to isolate the problem record. The statement should be placed where control passes through it only once for each record that is read. When the contents of the counter equal the number of the record (as specified in the ON statement), a trace can be taken on that record. The following example shows a method in which the 200th record could be selected for a TRACE statement.

```
Col.
1       Area A
--------+---------------------------------------
        RD-REC.
        .
        .
        .
DEBUG   RD-REC
        PARA-NM-1.      ON 200 READY TRACE.
                        ON 201 RESET TRACE.
```

If the TRACE statement were used without the ON statement, every record would be traced.

An example of a common program error is failing to break a loop or unintentionally creating a loop in the program. If many iterations of the loop are required before it can be determined that a program error exists, the ON statement can be used to initiate a trace after the expected number of iterations has been completed.

Note: If an error occurs in an ON statement, the diagnostic message may refer to the previous statement number.

DISPLAYING DATA VALUES DURING EXECUTION

A programmer can display the value of a data item during program execution by using the EXHIBIT statement. The EXHIBIT statement has three options:

1. EXHIBIT NAMED -- Displays the names and values of the data-names listed in the statement.

2. EXHIBIT CHANGED -- Displays the value of the data-names listed in the statement only if the value has changed since the last execution of the statement.

3. EXHIBIT CHANGED NAMED -- Displays the names and the values of the data-names only if the values have changed since the last execution of the statement.

Data values can be used to check the accuracy of the program. For example, using EXHIBIT NAMED, the programmer can display specified fields from records, compute the calculations himself, and compare his calculations with the output from his program. The coding for a payroll problem might be:

```
Col.
1       Area A
--------------------------------------------
          .
          .
          .
        GROSS-PAY-CALC.
            COMPUTE GROSS-PAY =
            RATE-PER-HOUR * (HRSWKD
            + 1.5 * OVERTIMEHRS).
        NET-PAY-CALC.
          .
          .
DEBUG   NET-PAY-CALC
        SAMPLE-1. ON 10 AND
            EVERY 10 EXHIBIT NAMED
            RATE-PER-HOUR, HRSWKD,
            OVERTIMEHRS, GROSS-PAY.
```

This coding will cause the values of the four fields to be listed for every tenth data record before net pay calculations are made. The output could appear as:

```
RATE-PER-HOUR = 4.00 HRSWKD = 40.0
     OVERTIMEHRS = 0.0 GROSS-PAY = 160.00

RATE-PER-HOUR = 4.10 HRSWKD = 40.0
     OVERTIMEHRS = 1.5 GROSS-PAY = 173.23

RATE-PER-HOUR = 3.35 HRSWKD = 40.0
     OVERTIMEHRS = 0.0 GROSS-PAY = 134.00
```

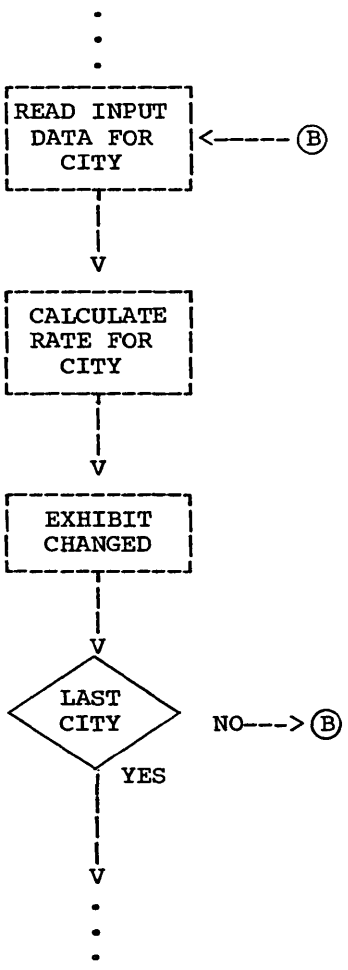
Note: Decimal points are included in this example for clarity, but actual printouts depend on the data description in the program.

The preceding was an example of checking at regular intervals (every tenth record). A check of any unusual conditions can be made by using various combinations of COBOL statements in the debug packet. For example:

```
IF OVERTIMEHRS GREATER THAN 2.0
  EXHIBIT NAMED PAYRCDHRS...
```

In connection with the previous example, this statement could cause the entire pay record to be displayed whenever an unusual condition (overtime exceeding two hours) is encountered.

The EXHIBIT statement with the CHANGED option also can be used to monitor conditions that do not occur at regular intervals. The values of data-names are listed only if the value has changed since the last execution of the statement. For example, suppose the program calculates postage rates to various cities. The flow of the program might be:



The EXHIBIT statement with the CHANGED option in the program might be:

```
EXHIBIT CHANGED STATE CITY RATE
```

The output from the EXHIBIT statement with the CHANGED option could appear as:

```
01  01  10
    02  15
    03
    04  10
02  01
    02  20
    03  15
    04
03  01  10
     .
     .
     .
```

The first column contains the code for a state, the second column contains the code for a city, and the third column contains the code for the postage rate. The value of a data-name is listed only if it has changed since the previous execution. For example, since the postage rate to city 02 and city 03 in state 01 are the same, the rate is not printed for city 03.

The EXHIBIT statement with the CHANGED NAMED option lists the data-name if the value has changed. For example, the program might calculate the cost of various methods of shipping to different cities. After the calculations are made, the following statement could appear in the program:

```
EXHIBIT CHANGED NAMED STATE CITY RAIL
  BUS TRUCK AIR
```

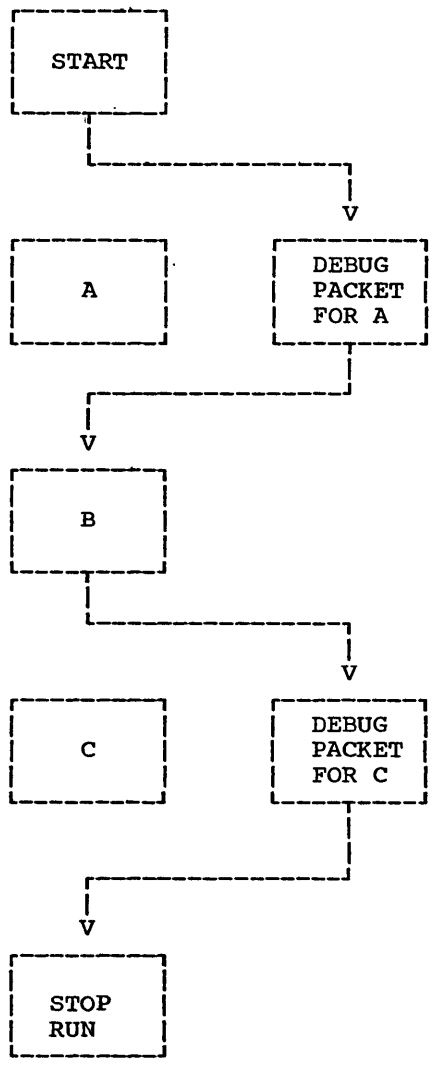
The output from this statement could appear as shown in Figure 9. Note that a data-name and its value are listed only if the value has changed since the previous execution.

```
STATE = 01 CITY = 01 RAIL = 10 BUS = 14 TRUCK = 12 AIR = 20

CITY = 02

CITY = 03 BUS = 06 AIR = 15

CITY = 04 RAIL = 30 BUS = 25 TRUCK = 28 AIR = 34

STATE = 02 CITY = 01 TRUCK = 25

CITY = 02 TRUCK = 20 AIR = 30

     .
     .
     .
```

Figure 9. Sample Output of EXHIBIT Statement with the CHANGED NAMED Option

TESTING A PROGRAM SELECTIVELY

A debug packet allows the programmer to select a portion of the program for testing. The packet can include test data and can specify operations the programmer wants to be performed. When the testing is completed, the packet can be removed. The flow of control can be selectively altered by the inclusion of debug packets, as illustrated in the following example of selective testing of B:



In this program, A creates data, B processes it, and C prints it. The debug packet for A simulates test data. It is first in the program to be executed. In the packet, the last statement is GO TO B, which permits A to be bypassed. After B is executed with the test data, control passes to the debug packet for C, which contains a GO TO statement that transfers control to the end of the program, bypassing C.

TESTING CHANGES AND ADDITIONS TO PROGRAMS

If a program runs correctly, and changes or additions might improve its efficiency, a debug packet can be used to test changes without modifying the original source program.

If the changes to be incorporated are in the middle of a paragraph, the entire paragraph with the changes included must be written in the debug packet. The last statement in the packet should be a GO TO statement that transfers control to the next procedure to be executed.

There are usually several ways to perform an operation. Alternative methods can be tested by putting them in debug packets.

The source program library facility can be used for program checkout by placing a source program in a library (see the chapter "Librarian Functions"). Changes or additions to the program can be tested by using the BASIS card and any number of INSERT and DELETE cards. Such changes or additions remain in effect only for the duration of the run.

A debug packet can also be used in conjunction with the BASIS card to debug a program or to test deletions or additions to it. The debug packet is inserted in the input stream immediately following the BASIS card and any INSERT or DELETE cards.

DUMPS

If a serious error occurs during execution of the problem program, the job is abnormally terminated; any remaining steps are bypassed; and a program phase dump is generated. The programmer can use the dump for program checkout. (However, any pending transfers to an external device may not be completed. For example, if a READY TRACE statement is in effect when the job is abnormally terminated, the last card number may not appear on the external device.) In cases where a serious error occurs in other than the problem program (e.g., Supervisor), a dump is not produced. Note that program phase dumps can be suppressed if the NODUMP option of the OPTION control statement has been specified for the job, or if NODUMP was specified at system generation time and is not overridden by the DUMP option for the current job.

HOW TO USE A DUMP

When a job is abnormally terminated due to a serious error in the problem program, a message is written on SYSLST which indicates the:

1. Type of interrupt (e.g., program check)
2. Hexadecimal address of the instruction that caused the interrupt
3. Condition code
4. Reason for the interrupt (e.g., data exception)

The instruction address can be compared to the Procedure Division map. The contents of LISTX provide a relative address for each statement. The load address of the module (which can be obtained from the map of main storage generated by the Linkage Editor) must be subtracted from the instruction address to obtain the relative instruction address as shown in the Procedure Division map. If the interrupt occurred within the COBOL program, the programmer can use the error address and LISTX to locate the specific statement in the program which caused a dump to be taken. Examination of the statement and the fields associated with it may produce information as to the specific nature of the error.

Figure 10 is a sample dump which was caused by a data exception. Invalid data (i.e., data which did not correspond to its usage) was placed in the numeric field B as a result of redefinition. The following discussion illustrates the method of finding the specific statement in the program which caused the dump. Letters identifying the text correspond to letters in the program listing.

Ⓐ The program interrupt occurred at HEX LOCATION 00373A. This is indicated in the SYSLST message printed just before the dump.

Ⓑ The linkage editor map indicates that the program was loaded into address 003000. This is determined by examining the load point of the control section TESTRUN. TESTRUN is the name assigned to the program module by the source coding:

```
PROGRAM-ID.  TESTRUN.
```

Ⓒ The specific instruction which caused the dump is located by subtracting the load address from the interrupt address (i.e., subtracting 3000 from 373A). The result, 73A, is the relative interrupt address and can be found in the object code listing. In this case the instruction in question is an AP (add decimal).

Ⓓ The left-hand column of the object code listing gives the compiler-generated card number associated with the instruction. It is card 69. As seen in the source listing, card 69 contains the COMPUTE statement.

Additional details about reading a dump are found in the chapter "Interpreting Output."

ERRORS THAT CAN CAUSE A DUMP

A dump can be caused by one of many errors. Several of these errors may occur at the COBOL language level while others can occur at the job control level.

The following are examples of COBOL language errors that can cause a dump:

1. A GO TO statement with no procedure-name following it may have been improperly initialized with an ALTER statement. The execution of this statement will cause an invalid branch.
2. Arithmetic calculations or moves on numeric fields that have not been properly initialized.

   For example, neglecting to initialize the object of an OCCURS clause with the DEPENDING ON option, or referencing data fields prior to the first READ statement may cause a program interrupt and a dump.
3. Invalid data placed in a numeric field as a result of redefinition.
4. Input/output errors that are nonrecoverable.

5. Items with subscripts whose values exceed the defined maximum value can destroy machine instructions when moved.

6. Attempting to execute an invalid operation code through a system or program error.

7. Generating an invalid address for an area that has address protection.

8. Subprogram linkage declarations that are not defined exactly as they are stated in the calling program.

9. Data or instructions can be modified by entering a subprogram and manipulating data incorrectly. A COBOL subprogram can acquire invalid information from the main program, e.g., a CALL statement using a procedure-name and an ENTRY statement using a data-name.

10. An input file contains invalid data such as a blank numeric field or data incorrectly specified by its data description.

    The compiler does not generate a test to check the sign position for a valid configuration before the item is used as an operand. The programmer can test for valid data by means of the numeric class test and, by using the TRANSFORM statement, convert it to valid data under certain conditions.

    For example, if the units position of a numeric data item described as USAGE IS DISPLAY contained a blank, the blank could be transformed to a zero, thus forcing a valid sign.

LOCATING A DTF

One or more DTF's are generated by the compiler for each file opened in the COBOL program. All information about that file is found within the DTF or in the fields preceding the DTF. See the chapter "Advanced Processing Capabilities" for the type of information available and its location.

A particular DTF may be located in an execution-time dump as follows:

1. Determine the order of the DTF address cells in the TGT from the DTF numbers shown for each file-name in the glossary.

   Note: Since the order is the same as the FD's in the Data Division, the order can be determined from the source program if the SYM option was not used (i.e., no glossary was printed).

2. Find the relative starting address of the block of DTF cells from the TGT listing in the Memory Map.

3. Calculate the absolute starting address of the block by adding the hexadecimal relocation factor for the beginning of the object module as given in the linkage editor MAP.

4. Allowing one fullword per DTF cell, count off the cells from the starting address found in step 3, using the order determined in step 1 to locate the desired DTF cell.

5. If more than one DTF is generated for a file, the above procedure should be followed using the PGT and the SUBDTF cells rather than the TGT and the DTFADR cells. The order of multiple DTF's in core is dependent on the OPEN option as follows:

   a. INPUT

   b. OUTPUT

   c. I-O or INPUT REVERSED

The following discussion illustrates the method of finding the DTF's in the sample program in Figure 10. Letters identifying the text refer to letters in the program listing.

(E) The DTF for FILE-1 precedes the DTF for FILE-2.

(F) DTFADR CELLS begin at relative location 5C0.

(G) Since the relocation factor is 3000, the DTFADR CELLS begin at location 35C0 in the dump.

(H) The DTF for FILE-1 begins at location 3158, and the DTF for FILE-2 begins at location 31E0.

LOCATING DATA

The location assigned to a given data-name may similarly be found by using the BL number and displacement given for that entry in the glossary, and then locating the appropriate one fullword BL cell in the TGT. The hexadecimal sum of the glossary displacement and the contents of the cell should give the relative address of the desired area. This can then be converted to an absolute address as described above.

Since the problem program in Figure 10 interrupted because of a data exception, the programmer should locate the contents of field B at the time of the interrupt. This can be done as follows:

(J) Locate data-name B in the glossary. It appears under the column headed SOURCE-NAME. Source-name B has been assigned to base locator 3 (i.e., BL =3) with a displacement of 050. The sum of the value of base locator 3 and the displacement value 50 is the address of data-name B.

(K) The Register Assignment table lists the registers assigned to each base locator. Register 6 has been assigned to BL =3.

(L) The contents of the 16 general registers at the time of the interrupt are displayed at the beginning of the dump. Register 6 contains the address 000030E8.

(M) The location of data-name B can now be determined by adding the contents of register 6 and the displacement value 50. The result, 3138, is the address of the leftmost byte of the 4-byte field B.

Note: Field B contains F1F2F3C4. This is external decimal representation and does not correspond to the USAGE COMPUTATIONAL-3 defined in the source listing.

(N) The location assigned to a given data-name may also be found by using the BL CELLS pointer in the TGT Memory Map. Figure 10 indicates that the BL cells begin at location 35B4 (add 5B4 to the load point address, 3000, of the object module). The first four bytes are the first BL cell, the second four bytes are the second BL cell, etc. Note that the third BL cell contains the value 30E8. This is the same value as that contained in register 6.

Note: Some program errors may destroy the contents of the general registers or the BL cells. In such cases, alternate methods of locating the DTF's are useful.

```
C0CC1   000010 IDENTIFICATION DIVISION.
00002   000020 PROGRAM-ID. TESTRUN.
00003   000030    AUTHOR. PROGRAMMER NAME.
00004   000040    INSTALLATION. NEW YORK PROGRAMMING CENTER.
00005   000050    DATE-WRITTEN. SEPTEMBER 10, 1968.
00006   000060 DATE-COMPILED. 06/20/69
00007   000070    REMARKS. THIS PROGRAM HAS BEEN WRITTEN AS A SAMPLE PROGRAM FOR
00008   000080      COBOL USERS. IT CREATES AN OUTPUT FILE AND READS IT BACK AS
00009   000090      INPUT.
00010   000100
00011   000110 ENVIRONMENT DIVISION.
00012   00012C CONFIGURATION SECTION.
00013   000130 SOURCE-COMPUTER. IBM-360-H50.
00014   00014C OBJECT-COMPUTER. IBM-360-H50.
00015   00015C INPUT-OUTPUT SECTION.
00016   000160 FILE-CONTROL.
00017   000170      SELECT FILE-1 ASSIGN TO SYS008-UT-2400-S.
00018   000180      SELECT FILE-2 ASSIGN TO SYS008-UT-2400-S.
00019   000190
0002C   C002CC DATA DIVISION.
00021   000210 FILE SECTION.
00022   000220 FD  FILE-1
00023   00023C      LABEL RECORDS ARE STANDARD
00024   000240      BLOCK CONTAINS 5 RECORDS
00025   000250      RECORDING MODE IS F
00026   000255      RECORD CONTAINS 20 CHARACTERS
00027   000260      DATA RECORD IS RECORD-1.
00028   000270 01  RECORD-1.
00029   00028C      05 FIELD-A PIC X(20).
00030   000290 FD  FILE-2
00031   000300      LABEL RECORDS ARE STANDARD
00032   00031C     BLOCK CONTAINS 5 RECORDS
00033   000320     RECORD CONTAINS 20 CHARACTERS
00034   00033C     RECORDING MODE IS F
00035   000340     DATA RECORD IS RECORD-2.
00036   00035C 01  RECORD-2.
00037   000360     05 FIELD-A PIC X(20).
```

(E)

Figure 10. Sample Dump Resulting from Abnormal Termination (Part 1 of 6)

```
00038    000370 WORKING-STORAGE SECTION.
00039    000380 01  FILLER.
00040    000390     02 COUNT PIC S99 COMP SYNC.
00041    000400     02 ALPHABET PIC X(26) VALUE IS "ABCDEFGHIJKLMNOPQRSTUVWXYZ".
00042    000410     02 ALPHA REDEFINES ALPHABET PIC X OCCURS 26 TIMES.
00043    000420     02 NUMBR PIC S99 COMP SYNC.
00044    000430     02 DEPENDENTS PIC X(26) VALUE "01234012340123401234012340".
00045    000440     02 DEPEND REDEFINES DEPENDENTS PIC X OCCURS 26 TIMES.
00046    000450 01  WORK-RECORD.
00047    000460     05 NAME-FIELD PIC X.
00048    000470     05 FILLER PIC X.
00049    000480     05 RECORD-NO PIC 9999.
00050    000490     05 FILLER PIC X VALUE IS SPACE.
00051    000500     05 LOCATION PIC AAA VALUE IS "NYC".
00052    000510     05 FILLER PIC X VALUE IS SPACE.
00053    000520     05 NO-OF-DEPENDENTS PIC XX.
00054    000530     05 FILLER PIC X(7) VALUE IS SPACES.
00055    000534 01  RECORDA.
00056    000535     02 A PICTURE S9(4) VALUE 1234.
00057    000536     02 B REDEFINES A PICTURE S9(7) COMPUTATIONAL-3.
00058    000540
00059    000550 PROCEDURE DIVISION.
0006C    000560 BEGIN. READY TRACE.
00061    000570     NOTE THAT THE FOLLOWING OPENS THE OUTPUT FILE TO BE CREATED
00062    000580     AND INITIALIZES COUNTERS.
00063    000590 STEP-1. OPEN OUTPUT FILE-1. MOVE ZERO TO COUNT, NUMBR.
00064    000600     NOTE THAT THE FOLLOWING CREATES INTERNALLY THE RECORDS TO BE
00065    000610     CONTAINED IN THE FILE, WRITES THEM ON TAPE, AND DISPLAYS
00066    000620     THEM ON THE CONSOLE.
00067    000630 STEP-2. ADD 1 TO COUNT, NUMBR. MOVE ALPHA (COUNT) TO
00068    000640     NAME-FIELD.
00069    000645         COMPUTE B = B + 1.  <--(D)
0007C    000650     MOVE DEPEND (COUNT) TO NO-OF-DEPENDENTS.
00071    000660     MOVE NUMBR TO RECORD-NO.
00072    000670 STEP-3. DISPLAY WORK-RECORD UPON CONSOLE. WRITE RECORD-1 FROM
00073    000680     WORK-RECORD.
00074    000690 STEP-4. PERFORM STEP-2 THRU STEP-3 UNTIL COUNT IS EQUAL TO 26.
00075    000700     NOTE THAT THE FOLLOWING CLOSES THE OUTPUT FILE AND REOPENS
00076    000710     IT AS INPUT.
00077    000720 STEP-5. CLOSE FILE-1. OPEN INPUT FILE-2.
00078    000730     NOTE THAT THE FOLLOWING READS BACK THE FILE AND SINGLES
00079    000740     OUT EMPLOYEES WITH NO DEPENDENTS.
00080    000750 STEP-6. READ FILE-2 RECORD INTO WORK-RECORD AT END GO TO STEP-8.
00081    000760 STEP-7. IF NO-OF-DEPENDENTS IS EQUAL TO "0" MOVE "Z" TO
00082    000770     NO-OF-DEPENDENTS. EXHIBIT NAMED WORK-RECORD. GO TO STEP-6.
00083    000780 STEP-8. CLOSE FILE-2.
00084    000790     STOP RUN.
```

Figure 10. Sample Dump Resulting from Abnormal Termination (Part 2 of 6)

INTRNL NAME	LVL	SOURCE NAME	BASE	DISPL	INTRNL NAME	DEFINITION	USAGE	R	O	Q	M
DNM=1-148	FD	FILE-1	DTF=01		DNM=1-148		DTFMT				F
DNM=1-178	C1	RECORD-1	BL=1	000	DNM=1-178	DS 0CL20	GROUP				
DNM=1-199	02	FIELD-A	BL=1	000	DNM=1-199	DS 20C	DISP				
DNM=1-216	FD	FILE-2	DTF=02		DNM=1-216		DTFMT				F
DNM=1-246	01	RECORD-2	BL=2	000	DNM=1-246	DS 0CL20	GROUP				
DNM=1-267	02	FIELD-A	BL=2	000	DNM=1-267	DS 20C	DISP				
DNM=1-287	01	FILLER	BL=3	000	DNM=1-287	DS 0CL56	GROUP				
DNM=1-306	02	COUNT	BL=3	000	DNM=1-306	DS 1H	COMP				
DNM=1-321	C2	ALPHABET	BL=3	002	DNM=1-321	DS 26C	DISP				
DNM=1-339	C2	ALPHA	BL=3	002	DNM=1-339	DS 1C	DISP	R	O		
DNM=1-357	02	NUMBR	BL=3	01C	DNM=1-357	DS 1H	COMP				
DNM=1-372	C2	DEPENDENTS	BL=3	01E	DNM=1-372	DS 26C	DISP				
DNM=1-392	02	DEPEND	BL=3	01E	DNM=1-392	DS 1C	DISP	R	O		
DNM=1-408	01	WORK-RECORD	BL=3	038	DNM=1-408	DS 0CL20	GROUP				
DNM=1-432	02	NAME-FIELD	BL=3	038	DNM=1-432	DS 1C	DISP				
DNM=1-452	02	FILLER	BL=3	039	DNM=1-452	DS 1C	DISP				
DNM=1-471	C2	RECORD-NO	BL=3	03A	DNM=1-471	DS 4C	DISP-NM				
DNM=1-490	C2	FILLER	BL=3	03E	DNM=1-490	DS 1C	DISP				
DNM=2-000	02	LOCATION	BL=3	03F	DNM=2-000	DS 3C	DISP				
DNM=2-018	C2	FILLER	BL=3	042	DNM=2-018	DS 1C	DISP				
DNM=2-037	02	NO-OF-DEPENDENTS	BL=3	043	DNM=2-037	DS 2C	DISP				
DNM=2-063	02	FILLER	BL=3	045	DNM=2-063	DS 7C	DISP				
DNM=2-082	01	RECORDA	BL=3	050	DNM=2-082	DS 0CL4	GROUP				
DNM=2-102	02	A	BL=3	050	DNM=2-102	DS 4C	DISP-NM				
DNM=2-113	02	B ←(J)	BL=3	050	DNM=2-113	DS 4P	COMP-3	R			

MEMORY MAP

TGT	003F8
SAVE AREA	003F8
SWITCH	00440
TALLY	00444
SORT SAVE	00448
ENTRY-SAVE	0044C
SORT CORE SIZE	00450
NSTD-REELS	00454
SORT RET	00456
WORKING CELLS	00458
SORT FILE SIZE	00588
SORT MODE SIZE	0058C
PGT-VN TBL	00590
TGT-VN TBL	00594
SORTAB ADDRESS	00598
LENGTH OF VN TBL	0059C
LNGTH OF SORTAB	0059E
PGM ID	005A0
A(INITI)	005A8
UPSI SWITCHES	005AC
OVERFLOW CELLS	005B4
BL CELLS ←(N)	005B4
DTFADR CELLS	005C0 ←(F)
TEMP STORAGE	005C8
TEMP STORAGE-2	005D0
TEMP STORAGE-3	005D0
TEMP STORAGE-4	005D0
BLL CELLS	005D0
VLC CELLS	005D4
SBL CELLS	005D4
INDEX CELLS	005D4
SUBADR CELLS	005D4
CNCTL CELLS	005DC
PFMCTL CELLS	005DC
PFMSAV CELLS	005DC
VN CELLS	005E0
SAVE AREA =2	005E4
XSASW CELLS	005E4
XSA CELLS	005E4
PARAM CELLS	005E4
RPTSAV AREA	005E8
CHECKPT CTR	005E8
IOPTR CELLS	005EA

REGISTER ASSIGNMENT

REG 6 BL =3 ←(K)
REG 7 BL =1
REG 8 BL =2

Figure 10. Sample Dump Resulting from Abnormal Termination (Part 3 of 6)

```
                000708  54 CF D 1D6              NI    1D6(13),X'0F'          TS=01+6
                00070C  4F 30 D 1D0              CVB   3,1D0(0,13)            TS=01
                000710  40 30 6 01C              STH   3,01C(0,6)             DNM=1-357
67              000714  41 40 6 002              LA    4,002(0,6)             DNM=1-339
                000718  48 20 6 000              LH    2,000(0,6)             DNM=1-306
                00071C  4C 2C C 042              MH    2,042(0,12)            LIT+2
                000720  1A 42                    AR    4,2
                000722  5B 4C C 040              S     4,040(0,12)            LIT+0
                000726  50 40 D 1DC              ST    4,1DC(0,13)            SBS=1
                00072A  58 E0 D 1DC              L     14,1DC(0,13)           SBS=1
                00072E  D2 CC 6 038 E 000        MVC   038(1,6),000(14)       DNM=1-432        DNM=1-339
69              000734  F8 70 D 1D0 C 044        ZAP   1D0(8,13),044(1,12)    TS=01            LIT+4
                00073A  FA 43 D 1D3 6 050     (C)→AP   1D3(5,13),050(4,6)     TS=04            DNM=2-113
                000740  F8 33 6 050 D 1D4        ZAP   050(4,6),1D4(4,13)     DNM=2-113        TS=04+1
7C              000746  41 4C 6 01E              LA    4,01E(0,6)             DNM=1-392
                00074A  48 2C 6 000              LH    2,000(0,6)             DNM=1-306
                00074E  4C 20 C 042              MH    2,042(0,12)            LIT+2
                000752  1A 42                    AR    4,2
                000754  5B 4C C 040              S     4,040(0,12)            LIT+0
                000758  50 40 D 1E0              ST    4,1E0(0,13)            SBS=2
                00075C  58 EC D 1E0              L     14,1E0(0,13)           SBS=2
                000760  D2 0C 6 043 E 000        MVC   043(1,6),000(14)       DNM=2-37         DNM=1-392
                000766  92 40 6 044              MVI   044(6),X'40'           DNM=2-37+1
71              00076A  48 3C 6 01C              LH    3,01C(0,6)             DNM=1-357
                00076E  4E 3C D 1D0              CVD   3,1D0(0,13)            TS=01
                000772  F3 31 6 03A D 1D6        UNPK  03A(4,6),1D6(2,13)     DNM=1-471        TS=07
                000778  96 FC 6 03D              OI    03D(6),X'F0'           DNM=1-471+3
72              00077C  58 FC C 004              L     15,004(0,12)           V(ILBDDSP0)
```

```
   PHASE  XFR-AD  LOCORE  HICORE  DSK-AD    ESD TYPE  LABEL     LOADED  REL-FR
TEST      003000  003000  0048E3  50 07 2   CSECT     TESTRUN   003000  003000←(B)

                                            CSECT     IJFFBZZN  0039D8  0039D8
                                            *  ENTRY  IJFFZZZN  0039D8
                                            *  ENTRY  IJFFBZZZ  0039D8
                                            *  ENTRY  IJFFZZZZ  0039D8

                                            CSECT     ILBDSAE0  0047F0  0047F0
                                               ENTRY  ILBDSAE1  004810

                                            CSECT     ILBDMNS0  0047E8  0047E8

                                            CSECT     ILBDDSP0  003FA8  003FA8
                                            *  ENTRY  ILBDDSP1  0044F8
                                            *  ENTRY  ILBDDSP2  004590
                                            *  ENTRY  ILBDDSP3  004748

                                            CSECT     ILBDIML0  004780  004780

                                            CSECT     IJJCPD1   003DE0  003DE0
                                               ENTRY  IJJCPD1N  003DE0
                                            *  ENTRY  IJJCPD3   003DE0
```

Figure 10. Sample Dump Resulting from Abnormal Termination (Part 4 of 6)

```
OS03I PROGRAM CHECK INTERRUPTION - HEX LOCATION 00373A - CONDITION CODE 0 - DATA EXCEPTION (A)
OS00I JOB SAMPLE   CANCELED


        SAMPLE              06/20/69

                                                                                    (L)
GR C-7  000035C0 000036D8 00CCCC01 00000001   000030EA 5000399A 000030E8 000032C0
GR 8-F  00003328 0CC0396A 00CC3CCC 00003000   0Q0035F0 000033F8 000030EA 00003FA8
FP REG  00000000 00000000 00CCCCC0 C0000000   0000000C 0C000000 0C000000 00000000
COMREG  BG ADDR IS 0001F0


--BG--
002F80                    D5D64CD5 C1D4C540   FF1500C7 CC003740 0000396A 00003000
002FA0  00CC300C 000035F0 000033F8 C00030EA   000C3FA8 0CC035C0 000C36D8 00000001
002FC0  0CC0CCC1 CCC030EA 50CC399A 000030E8   0000320 00003328 00CC0CC0 0C0E607D
002FE0  CCCC0000 --SAME--
003000  00E2E8E2 F0F0F840 00E3C1D7 C5C6C9D3   C54C404C 4C4C4040 404CF1F1 F1F1F1F1
00302C  F0F0F0F1 F0F0F0F1 F0F0F0F1 F0F140F6   F9F1F7F1 40F6F9F3 F6F5F0F0 F0F0F0F0
003040  FCC4D6E2 61E3D6E2 61F3F6FC 40400000   F1F240C1 C3C3C5D7 E3C5C461 61D5D640
003C60  E5C1D3C9 C440D6D7 C5D540C6 D6D940C6   C9D3C54B 4CC6C9D3 C57E02C2 D3D6C3D2
003C80  E2C9E9C5 58C0F0C6 58E0CCCC 58D0F0CA   9500E000 4770F0A2 961CD048 92FFE00C
0030A0  47F0F0A0 98CEF03A 90ECDCCC 185D989F   F0BA9110 D048C719 07FF0700 0000396A
0C3CCC  0CC030C0 0CCC3000 000035FC 000033F8   00003664 00003950 C3D6C2C6 F0F0F0F0
0030E0  E3C5E2E3 D9E4D54C CC01C1C2 C3C4C5C6   C7C8C9D1 D2D3D4D5 D6D7D8D9 E2E3E4E5
003100  E6E7E8E9 0001F0F1 F2F3F4FC F1F2F3F4   (M) F0F1F2F3 F4FCF1F2 F3F4F0F1 F2F3F4FC
00312C  C1C3C5D7 E3C540D5 E8C340E3 C5404040   404C4040 D6D560D9 F1F2F3C4 CC000000
00314C  C1C1C014 0CCCCCCC 00CCCCCC 00000000   (H) 13CC00CC CC000000 000C9200 00000108
003160  00003190 00C00000 10C039D8 1260E2E8   E2FCFCF8 40400166 900C0000 04000000
0C318C  CCC00000 86BCF018 41ECECC1 58201044   01003258 2C000064 00CC32C0 000C32C0
0031AC  00000C14 00003323 00E4CC63 C0000000   0000F1F1 F1F1F1F1 F0F0F0F1 F0F0F0F1
(H)→0031C0  010047F0 0CC00000 0101C014 00000000   000C000C 0CC00000 0C0C0C00 0C000000
0031EC  CC0C82C0 C0C0C1C8 00CC3218 C0000000   100039D8 1468E2E8 E2FCFCF8 40400276
0032CC  800C0000 20C00CC0 00CCC000 86BCF018   41E0E0C1 58201044 02003328 00000064
003220  000C339C 0CC0C0C0 00000014 00000000   00640063 0C000000 00004810 0C0047FC
003240  C00C0C0C 0CCCCCCC 0CCCC000 00000000   0000C3C5 40C6C9C5 D3C440C9 D3D3C5C7
003260  C1D34B7E 15E2E3C5 D9D3C9D5 C740D5D6   D56CD9C5 D7D6D9E3 4CD7C9C3 E3E4D9C5
003280  4C6C40F6 40D6D940 F740C9D5 40C9D3D3   C5C7C1D3 40D7D6E2 C9E3C9D6 D54B7E15
0032A0  E2E3C5D9 D3C9D5C7 40D5D6D5 60D9C5D7   D6D9E340 D7C9C3E3 E4D9C540 6040E4E2
0032CC  C1C7C540 D5D6E340 C4C9E2D7 D3C1E860   E2E34B7E 16E2E3C5 D9D3C9D5 C740D5D6
0C32E0  D56CD9C5 D7D6D9E3 40D7C9C3 E3E4D9C5   406040E5 40C9D540 C9D3D3C5 C7C1D340
003300  D7D6E2C9 E3C9D6D5 4B7E15E2 E3C5D9D3   C9D5C740 D5D6D560 D9C5D7D6 D9E340D7
003320  C9C3E3E4 D9C54060 40E240C9 D540C9D3   D3C5C7C1 D340D7D6 E2C9E3C9 D6D54B7E
003340  15CC4770 946A45EC E2E3C5D9 C3C9D5C7   40D5D6D5 60D9C5D7 D6D9E340 D7C9C3E3
003360  E4D9C540 604CC4C9 C7C9E340 D3C5D5C7   E3C840C7 E340F1F8 4B7E15E2 E3C5D9D3
003380  C9D5C740 D5D6D560 D9C5D7D6 D9E340D7   C9C3E3E4 D9C54060 40E2C8C9 D3D3C9D5
0033A0  C74CC6C9 C5D3C440 C7E34CF2 4B7E15E2   E3C5D9D3 C9D5C740 D5D6D560 D9C5D7D6
0033CC  D9E340D7 C9C3E3E4 D9C5406C 40D7C5D5   C3C540C6 C9C5D3C4 4CC7E340 F24B7E15
0033E0  E2E3C5D9 D3C9D5C7 40D5D6D5 60D9C5D7   D6D9E340 D7C9C3E3 E4D9C540 6040D5D6
```

Figure 10. Sample Dump Resulting from Abnormal Termination (Part 5 of 6)

```
003400  40D7D6E4 D5C44CE2 C5D7C1D9 C1E3D6D9    4B7E15D6 D5D3E840 E3C8C540 D9C5D5C1
003420  D4C5E240 C3D3C1E4 E2C54CD4 C1E840C2    C54CE2D7 C5C3C9C6 0000004B 0C000000
003440  7C00004B 0CC000C0 00CCC000 00003664    00000000 00000000 000C3158 0C0C32CC
003460  0CCC300C 00003158 500C399A 40C9D3D3    C5C7C1D3 40D7D6E2 C9E3C9D6 D54B7E16
003480  D5E4D4C5 D9C9C340 D7C9C3E3 E4D9C540    604CD740 C9D540C9 D3D3C5C7 C1D340D7
0034A0  D6E2C9E3 C9D6D54B 7E16D5E4 D4C5D9C9    C34CD7C9 C3E3E4D9 C54C6C40 E540C9D5
0034CC  40C9D3D3 C5C7C1D3 40D7D6E2 C9E3C9D6    D54B7E16 D5E4D4C5 D9C9C340 D7C9C3E3
0034E0  E4D9C540 6040D5D6 4CF940C9 D540D7C9    C3E3E4D9 CCCC36D0 000C32C0 0100C340
CC35C0  700C36AC 0CCC03FA8 000035C0 000036D8    000C32C0 00003000 00003158 5000399A
0C3520  CC0030E8 000032C0 000C3328 0000396A    00003000 0C003000 000C35F0 000036D8
003540  00003FA8 000035C0 000C44F8 000032C0    00C1C5C7 000036D0 E4D4C5D9 C9C340D7
C03560  C9C3E3E4 D9C5406C 4CC4C9C7 C9E340D3    C5D5C7E3 C840C7E3 40F1F84B 7E16D5E4
003580  CC000000 00000000 00CCC0C0 C0000000    C4C9C7C9 E340D3C5 D5C7E3C8 404E40E2
0C35A0  00003000 E2C3C1D3 00003000 C0000250    000CC320 000032C0 00003328 0C003CE8
0035C0  00003158 0CCC31EC 0CCC00C0 0000001C    00000000 0C0030EA C540D5D6 E340C4C9
0035E0  000037C2 E84B7E2D 000047D0 00003F90    000C47E8 CCC03FA8 000C4780 000036C8
003600  000037C2 00C03864 000038B0 00003910    000037BC 000037E2 000C37F6 000038AA
0C3620  000C3EEC CCCC38D6 000C37C2 C01A8B5B    000C00C1 1CC0001A 5B5BC2D6 D7C5D54C
003640  5B5BC2C3 C3D6E2C5 5B5BC2C6 C3D4E4D3    F0E900C0 CCC0C0C0 E6D6D9D2 60D9C5C3
003660  D6D9C4F7 58FCCCC4 051F00C1 4004F6F0    404040C1 9640D048 58F0C004 051F0001
003680  4004F6F3 404040F0 411CC048 5800D1C8    184C05F0 5C00F008 4500F00C 00003158
0036A0  0AC24100 D1C858F0 C00805EF 5810D1C8    961C1020 5C20D1BC 587CD1BC D2016000
0036C0  C04CD201 6C1CC040 58F0CC04 051F0001    4004F6F7 4C4040D0 4830C042 4A3060C0
0036E0  4E3CD1DC D7C5D1D0 D1DC94CF D1D64F30    D1DC403C 6CCC4830 CC424A30 601C4E3C
0037C0  D1DCD7C5 D1D0D1D0 940FD1D6 4F30D1D0    4030601C 41406002 48206000 4C20C042
003720  1A425B4C CC4C5C40 D1DC58EC D1DCD200    6038E000 F87CD1D0 C044FA43 D1D36050
003740  F8336050 D1D44140 601E4820 60004C20    C0421A42 5B40C040 504CD1E0 58E0D1E0
003760  D20C6C43 ECCC9240 6C444830 601C4E30    D1D0F331 6C3AD1D6 96F0603D 58F0C004
003780  051F0001 4004F7F2 40404CC8 58F0C004    051F00C2 0C000014 CD0001C4 0038FFFF
0037A0  C2137000 60385810 D1C81841 58F01010    45E0F0CC 5020D1BC 587CD1BC 5810D1E8
0037C0  07F158FC C0C4C51F CCC140C4 F7F44040    408F5800 D1E85000 D1E45800 CC245000
0037E0  D1E84830 600C4930 CC4658F0 C028078F    581CC00C 07F15800 D1E45000 D1E858F0
CC3800  C004051F CCC140C4 F7F74C4C 40005810    D1C894EF 10201801 184C4110 CC5005F0
003820  500CF008 4500FCCC 0CCCCCCC 0A025800    D1C84110 CC580A02 4110C048 5800D1CC
003840  184005F0 5000F008 4500FCCC 00000000    0A0241CC D1CC58F0 CCC805EF 5810D1CC
003860  961C1020 58F0CC04 051FCCC1 4004F8F0    404C40F0 5810D1CC 58F0C02C 91201010
0C3880  071F1841 41F0CC2C D2C21025 F00158F0    101C45E0 FC085020 D1CC5880 D1C0D213
0038A0  6C388000 58F0C018 07FF5810 C01C07F1    58FCCC04 051FC001 4004F8F1 4040405F
0038C0  581CCC34 5E2CCC30 D5C0CC60 60430772    95406044 C772D200 6043C061 92406044
0038E0  5810CC64 5010D1EC 4120D1EC 58F0C004    051F80C1 1C000000 0C000068 00000000
CC39C0  00140D0C C1C40038 FFFF5810 C01407F1    58FCC004 C51FC0C1 4CC4F8F3 404040CC
003920  581CD1CC 94EF1020 18011840 411CC050    070005F0 5000F008 4500F00C 00000000
003940  0A02580C D1CC4110 CC580AC2 CA0E0A0E    50D050C8 5C50D0C4 582CC000 95002000
003960  C77992FF 200C9610 D04850E0 D05405F0    9120D048 47E0F016 580CBC48 982DB050
003980  58ECDC54 C7FE962C DC48416C C0044110    C00C4170 CC400670 055C5840 10001E4B
0039A0  50401000 871650C0 4180D1BC 4170D1CF    051058CC 8CC01E0B 5CC28000 87861000
0C39C0  D2C3D1E8 C0385860 D1C45870 D1BC5880    D1CC58E0 D05407FE 47F0F07C 47F0F0A0
0039E0  47F0F046 47F0F02A 47F0F272 47F0F052    47F0F0C6 47F0F150 C9D1C6C6 C2E9E9D5
003AC0  F3F490AE F3D09620 103C45A0 F0B898AE    F3D044C0 1C349101 1C15078E 0A0990AE
003A20  F3DC45AC FCB847FC F03690AE F3D058A0    10445BA0 1C404780 F03640A0 1C3E45AC
003A40  F32E45A0 FCC645A0 F32ED2C1 103E1050    47F0F036 5CACF3D0 45ACF32E 42001038
003A60  CA0C45AC F32ED2C0 1038101E D7011000    100058A0 F3D007FE 91041015 4710F0BC
003A80  D7031C4C 1C4C07FE D7C31044 104407FE    98BD1044 4400102C 50B01044 07FA9110
003AA0  1003471C F37C9140 1005471C F33A9180    1015478C F37C9120 1C15471C F0E80A00
003AC0  50A0F400 45A0F32E 58A0F400 91011004    4710F282 58E01028 440C1030 50E01028
0C3AE0  48BC1000 41E01058 12BB47EC F1349640    100512BB 41E01058 4740F1A2 948F1005
003B00  58E01048 10EE1BBE 47F0F110 91401005    4710F1A2 91101003 41E0105C 4710F1A2
0C3B20  91201C02 4710F1A2 58B01C38 948F1005    D2021039 1C4141BB 000050B0 10409104
```

Figure 10. Sample Dump Resulting from Abnormal Termination (Part 6 of 6)

DIAGNOSTIC MESSAGES

Diagnostic messages are generated by the compiler and listed on SYSLST when errors are found in the source program.

Note: Diagnostic messages are suppressed when the NOERRS option is in effect.

WORKING WITH DIAGNOSTIC MESSAGES

1. Approach the diagnostic messages in the order in which they appear on the source listing. It is possible to get compound diagnostic messages. Frequently, an earlier diagnostic message indicates the reason for a later diagnostic message. For example, a missing quotation mark for an alphabetic or alphanumeric literal could involve the inclusion of some clauses not intended for that particular literal. This could cause an apparently valid clause to be diagnosed as invalid because it is not complete, or because it is in conflict with something that preceded it.

2. Check for missing or superfluous punctuation, or other errors of this type.

3. Frequently, a seemingly meaningless message is clarified when the valid syntax or format of the clause or statement in question is referenced.

GENERATION OF DIAGNOSTIC MESSAGES

The compiler scans the statement, element by element, to determine whether the words are combined in a meaningful manner. Based upon the elements that have already been scanned, there are only certain words or elements that can be correctly encountered.

If the anticipated elements are not encountered, a diagnostic message is produced. Some errors may not be uncovered until information from various sections of the program is combined and the inconsistency is noted. Errors uncovered in this manner can produce a slightly different message format than those uncovered when the actual source text is still available. The message that is made unique through that particular error may not contain, for example, the actual source statement that produced the error.

Errors that appear to be identical are diagnosed in a slightly different manner, depending on where they were encountered by the compiler and how they fit within the context of valid syntax. For example, a period missing from the end of the Working-Storage section header is diagnosed specifically as a period required. There is no other information that can appear at that point. However, if at the end of a data item description entry, an element is encountered that is not valid at that point, such as the digits 02, it is diagnosed as invalid. Any clauses associated with the 02 entry which conflict with the clauses in the previous entry (the one that contained the missing period), are diagnosed. Thus, a missing period produces a different type of diagnostic message in one situation than in the other.

LINKAGE EDITOR OUTPUT

The Linkage Editor produces diagnostic messages, console messages, and a storage map. For a complete description of output and error messages from the Linkage Editor, see the publication IBM System/360 Disk Operating System: System Control and System Service Programs. Output resulting from the linkage editing of a COBOL program is discussed in the chapter "Interpreting Output."

EXECUTION TIME MESSAGES

When an error condition that is recognized by compiler-generated code occurs during execution, an error message is written on SYSLST and SYSLOG. No message is written on SYSLST when an error occurs in the foreground and SYSLST is assigned to a disk.

Messages that normally appear on SYSLOG are provided with a code indicating whether the message originated in a foreground or background program. These messages are listed in "Appendix F: Diagnostic Messages."

RECORDING PROGRAM STATUS

When a program is expected to run for an extended period of time, provision should be made for taking checkpoint information periodically during the run. A checkpoint is the recording of the status of a problem program and main storage (including

input/output status and the contents of the general registers). Thus, it provides a means of restarting the job at an intermediate checkpoint position rather than at the beginning, if for any reason processing is terminated before the normal end of the program. For example, a job of higher priority may require immediate processing, or some malfunction (such as a power failure) may occur and cause an interruption. Checkpoints are taken using the COBOL RERUN clause.

Restart is a means of resuming the execution of the program from one of the checkpoints rather than from the beginning. The ability to restart is provided through the RSTRT job control statement.

RERUN CLAUSE

The presence of the RERUN clause in the source program causes the CHKPT macro instruction to be issued at the specified interval. When the CHKPT macro instruction is issued, the following information is saved:

1. Information for the Restart and other supervisor or job control routines.
2. The general registers.
3. Bytes 8 through 10, and 12 through 45 of the Communication Region.
4. The problem program area.
5. All file protection extents for files assigned to mass storage devices if the extents are attached to logical units contained in the program for which checkpoints are taken.

Since the COBOL RERUN clause provides a linkage to the system CHKPT macro instruction, any warnings and restrictions on the use of this macro instruction also apply to the use of the RERUN clause. See the publication IBM System/360 Disk Operating System: Supervisor and Input/Output Macros for a complete description of the CHKPT macro instruction.

TAKING A CHECKPOINT

In order to take a checkpoint, the programmer must specify the source language RERUN clause and must define the file upon which checkpoint records are to be written (e.g., ASSGN, EXTENT, etc.) Checkpoint information must be written on a 2311 or 2314 mass storage device or on a magnetic tape -- either 7- or 9-track. Checkpoint records cannot be imbedded in one of the problem program's output files, i.e., the program must establish a separate file exclusively for checkpoint records.

In designing a program for which checkpoints are to be taken, the user should consider the fact that, upon restarting, the program must be able to continue as though it had just reached that point in the program at which termination occurred. Hence, the user should ensure that:

1. File handling is such as to permit easy reconstruction of the status of the system as it existed at the time of checkpoint was taken. For example, when multifile reels are used, the operator should be informed (by message) as to which file is in use at the time a checkpoint is to be taken. He requires this information at restart time.

2. The contents of files are not altered between the time of the checkpoint and the time of the restart. For sequential files, all records written on the file at the time the checkpoint is taken should be unaltered at restart time. For nonsequential files, care must be taken to design the program so that a restart will not duplicate work that has been completed between checkpoint time and restart time. For example, suppose that checkpoint 5 is taken. By adding an amount representing the interest due, account XYZ is updated on a direct-access file that was opened with the I-O option. If the program is restarted from checkpoint 5 and if the interest is recalculated and again added to account XYZ, incorrect results will be produced.

If the program is modular in design, RERUN statements must be included in all modules that handle files for which checkpoints are to be taken. (When an entry point of a module containing a RERUN statement is encountered, a COBOL subroutine, ILBDCKP0, is called. ILBDCKP0 enters the files of the module into the list of files to be repositioned.) Repositioning to the proper record will not occur for any files that were defined in modules other than those containing RERUN statements. Moreover, a restart from any given checkpoint may not reposition other tapes on which checkpoints are stored. Note, too, that only one disk checkpoint file can be used.

RESTARTING A PROGRAM

If the programmer requests checkpoints in his job by means of the COBOL RERUN clause, the following message is given each time a checkpoint is taken:

```
0C001 CHKPT nnnn HAS BEEN TAKEN ON
      SYSxxx
```

nnnn
is the 4-character identification of the checkpoint record.

To restart a job from a checkpoint, the following steps are required:

1. Replace the // EXEC statement with a // RSTRT statement. The format of the RSTRT statement is discussed in the chapter "Preparing COBOL Programs For Processing." All other job control statements applicable to the job step should be the same as when the job was originally run. If necessary, the channel and unit addresses for the // ASSGN control statements may be changed.

2. Rewind all tapes used by the program being restarted, and mount them on devices assigned to the symbolic units required by the program. If multivolume files are used, mount (on the primary unit) the reel being used at the time that the checkpoint was taken, and rewind it. If multifile volumes are used, position the reel to the start of the file referenced at the time the checkpoint is being taken.

3. Reposition any card file so that only cards not yet read when the checkpoint was taken are in the card reader.

4. Execute the job.

The American National Standard COBOL compiler, COBOL object module, Linkage Editor, and other system components can produce output in the form of printed listings, punched card decks, diagnostic or informative messages, and data files directed to tape or to mass storage devices. This chapter gives the format of and describes this output. The same COBOL program is used for each example. "Appendix A: Sample Program Output" shows the output formats in the context of a complete listing generated by the sample program.

COMPILER OUTPUT

The output of the compilation job step may include:

- A printed listing of the job control statements
- A printed listing of the statements contained in the source program
- A glossary of compiler-generated information about data
- A printed listing of the object code
- A condensed listing containing only the first generated instruction for each verb
- Compiler diagnostic messages
- A cross-reference listing
- System messages
- An object module

The presence or absence of the above-mentioned types of compiler output is determined by options specified at system generation time. These options can be overridden or additional options specified at compilation time by using the OPTION control statement and the CBL card.

The level of diagnostic message printed depends upon the FLAGW or FLAGE option of the CBL card.

All output to be listed is written on the device assigned to SYSLST. Line spacing of the source listing is controlled by the SPACEn option of the CBL card and by SKIP 1/2/3 and EJECT in the COBOL source program. The number of lines per page can be specified in the SET command. In addition, a listing of input/output assignments can be printed on SYSLST by using the LISTIO control statement.

Figure 11 contains the compiler output listing shown in "Appendix A: Sample Program Output." Each type of output is numbered, and each format within each type is lettered. The text following the figure is an explanation of the figure.

(1) The listing of the job control statements associated with this job step. These statements are listed because the LOG option was specified at system generation time.

(2) Compiler options. The CBL card, if specified, is printed on SYSLST unless the LIST option is suppressed.

(3) The source module listing. The statements in the source program are listed exactly as submitted except that a compiler-generated card number is listed to the left of each line. This is the number referenced in diagnostic messages and in the object code listing. It is also the number printed on SYSLST as a result of the source language TRACE statement. The source module is not listed when the NOLIST option is specified.

The following notations may appear on the listing:

C Denotes that the statement was inserted with a COPY statement.

** Denotes that the card is out of sequence. NOSEQ should be specified on the CBL card if the sequence check is to be suppressed.

I Denotes that the card was inserted with an INSERT or BASIS card.

If DATE-COMPILED is specified in the Identification Division, any sentences in that paragraph are replaced in the listing by the date of compilation. It is printed in one of the following formats depending upon the format chosen at system generation time.

DATE-COMPILED. month/day/year or

DATE-COMPILED. day/month/year

```
// JOB SAMPLE
// OPTION NODECK,LINK,LIST,LISTX,SYM,ERRS                    (1)
 PHASE TEST,*
// EXEC FCOBCL

  CBL QUOTE    (2)


00001   000010 IDENTIFICATION DIVISION.
00002   000020 PROGRAM-ID. TESTRUN.
00003   000030    AUTHOR. PROGRAMMER NAME.
00004   000040    INSTALLATION. NEW YORK PROGRAMMING CENTER.
00005   000050    DATE-WRITTEN. SEPTEMBER 10, 1968.
00006   000060 DATE-COMPILED. 06/20/69
00007   000070    REMARKS. THIS PROGRAM HAS BEEN WRITTEN AS A SAMPLE PROGRAM FOR
00008   000080      COBOL USERS. IT CREATES AN OUTPUT FILE AND READS IT BACK AS
00009   000090      INPUT.
00010   000100
00011   000110 ENVIRONMENT DIVISION.
00012   000120 CONFIGURATION SECTION.
00013   000130 SOURCE-COMPUTER. IBM-360-H50.
00014   000140 OBJECT-COMPUTER. IBM-360-H50.
00015   000150 INPUT-OUTPUT SECTION.
00016   000160 FILE-CONTROL.
00017   000170      SELECT FILE-1 ASSIGN TO SYS008-UT-2400-S.
00018   000180      SELECT FILE-2 ASSIGN TO SYS008-UT-2400-S.
00019   000190

~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~

00056   000550 PROCEDURE DIVISION.
00057   000560 BEGIN. READY TRACE.
00058   000570     NOTE THAT THE FOLLOWING OPENS THE OUTPUT FILE TO BE CREATED
00059   000580     AND INITIALIZES COUNTERS.
00060   000590 STEP-1. OPEN OUTPUT FILE-1. MOVE ZERO TO COUNT, NUMBR.

~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~

00073   000720 STEP-5. CLOSE FILE-1. OPEN INPUT FILE-2.
00074   000730     NOTE THAT THE FOLLOWING READS BACK THE FILE AND SINGLES
00075   000740     OUT EMPLOYEES WITH NO DEPENDENTS.
00076   000750 STEP-6. READ FILE-2 RECORD INTO WORK-RECORD AT END GO TO STEP-8.
00077   000760 STEP-7. IF NO-OF-DEPENDENTS IS EQUAL TO "0" MOVE "Z" TO
00078   000770     NO-OF-DEPENDENTS. EXHIBIT NAMED WORK-RECORD. GO TO STEP-6.
00079   000780 STEP-8. CLOSE FILE-2.
00080   000790     STOP RUN.
```

(3)

Figure 11. Examples of Compiler Output (Part 1 of 4)

```
      (A)          (B)  (C)              (D)      (E)      (F)          (G)          (H)          (J)
INTRNL NAME        LVL  SOURCE NAME       BASE     DISPL    INTRNL NAME  DEFINITION   USAGE        R  O  Q  M
DNM=1-148          FD   FILE-1           DTF=01              DNM=1-148               DTFMT           F
DNM=1-178          01   RECORD-1          BL=1     000      DNM=1-178    DS 0CL20     GROUP
DNM=1-199          02   FIELD-A           BL=1     000      DNM=1-199    DS 20C       DISP
DNM=1-216          FD   FILE-2           DTF=02              DNM=1-216               DTFMT           F
DNM=1-246          01   RECORD-2          BL=2     000      DNM=1-246    DS 0CL20     GROUP
DNM=1-267          C2   FIELD-A           BL=2     000      DNM=1-267    DS 20C       DISP
DNM=1-287          01   FILLER            BL=3     000      DNM=1-287    DS 0CL56     GROUP
DNM=1-306          C2   COUNT             BL=3     000      DNM=1-306    DS 1H        COMP
DNM=1-321          C2   ALPHABET          BL=3     002      DNM=1-321    DS 26C       DISP
DNM=1-339          C2   ALPHA             BL=3     002      DNM=1-339    DS 1C        DISP        R  O
DNM=1-357          C2   NUMBR             BL=3     01C      DNM=1-357    DS 1H        COMP
```

(4)

```
                    MEMORY MAP

                TGT  (A)                 003F0

            SAVE AREA                    003F0
            SWITCH                       00438
            TALLY                        0043C
            SORT SAVE                    00440
            ENTRY-SAVE                   00444
            SORT CORE SIZE               00448
            NSTD-REELS                   0044C

LITERAL POOL (HEX)  (B)

00628 (LIT+0)     C0000001  001A5B5B  C2D6D7C5  D54C5B5B  C2C3D3D6  E2C55B5B
0064C (LIT+24)    C2C6C3D4  E4D3F0E9  C0000000

     DISPLAY LITERALS (BCD)

0064C (LTL+36)   'WCRK-RECORD'

                PGT  (C)                 005E8

            OVERFLOW CELLS               005E8
            VIRTUAL CELLS                005E8
            PROCEDURE NAME CELLS         005F4
            GENERATED NAME CELLS         00608
            SUBCTF ADDRESS CELLS         00620
            VNI CELLS                    00620
            LITERALS                     00628
            DISPLAY LITERALS             0064C
```

(5)

Figure 11. Examples of Compiler Output (Part 2 of 4)

```
REGISTER ASSIGNMENT

  REG 6    BL =3
  REG 7    BL =1
  REG 8    BL =2
```

⑥

Ⓐ	Ⓑ	Ⓒ		Ⓔ		Ⓕ	
57	000658		START	EQU	*		
	0CC658	58 FC C 004		L	15,004(0,12)	V(ILBDDSP0)	
	00065C	05 1F		BALR	1,15		
	CCC65E	CC0140		DC	X'000140'		
	000661	04F5F7404040		DC	X'04F5F7404040'		
57	000668	96 40 D 048		OI	048(13),X'40'	SWT+0	
60	0C066C	58 FC C 004		L	15,004(0,12)	V(ILBDDSP0)	
	000670	05 1F		BALR	1,15		
	00C672	0C0140		DC	X'000140'		
	000675	C4F6FC4C4040		DC	X'04F6F0404040'		
60	00067C	41 10 C 046		LA	1,046(0,12)	LIT+6	
	00068C	58 CC D 1C8		L	0,1C8(0,13)	DTF=1	
	000684	18 40		LR	4,0		
	000686	C5 F0		BALR	15,0		
	C00688	50 CC F 008		ST	0,008(0,15)		
	00068C	45 00 F 00C		BAL	0,00C(0,15)		
	000690	0CCCC000		DC	X'00000000'		
	00C694	0A 02		SVC	2		
	000696	41 0C D 1C8		LA	0,1C8(0,13)	DTF=1	
	0CC69A	58 FC C 008		L	15,008(0,12)	V(ILBDIML0)	
	00069E	05 EF		BALR	14,15		
	0006A0	58 10 D 1C8		L	1,1C8(0,13)	DTF=1	
	0C06A4	96 10 1 020		OI	020(1),X'10'		
	0006A8	50 20 D 1BC		ST	2,1BC(0,13)	BL =1	
	0006AC	58 7C D 1BC		L	7,1BC(0,13)	BL =1	
60	0C06B0	D2 01 6 000 C 040	Ⓓ	MVC	000(2,6),040(12)	DNM=1-306	LIT+0
	0006B6	D2 01 6 01C C 040		MVC	01C(2,6),040(12)	DNM=1-357	LIT+0
64	0006BC		PN=01	EQU	*		
	0C06BC	58 F0 C 004		L	15,004(0,12)	V(ILBDDSP0)	
	0006C0	C5 1F		BALR	1,15		
	0006C2	C0014C		DC	X'000140'		
	0006C5	C4F6F4404040		DC	X'04F6F4404040'		
64	0006CC	48 30 C 042		LH	3,042(0,12)	LIT+2	
	0C06D0	4A 30 6 000		AH	3,000(0,6)	DNM=1-306	
	0006D4	4E 30 D 1D0		CVD	3,1D0(0,13)	TS=01	
	0CC6D8	D7 05 D 1D0 D 1D0		XC	1D0(6,13),1D0(13)	TS=01	TS=01
	0006DE	94 0F D 1D6		NI	1D6(13),X'0F'	TS=01+6	
	0006E2	4F 30 D 1D0		CVB	3,1D0(0,13)	TS=01	
	0C06E6	40 3C 6 000		STH	3,000(0,6)	DNM=1-306	
	0006EA	48 30 C 042		LH	3,042(0,12)	LIT+2	
	0C06EE	4A 30 6 01C		AH	3,01C(0,6)	DNM=1-357	
	0006F2	4E 30 D 1D0		CVD	3,1D0(0,13)	TS=01	
	0006F6	D7 05 D 1D0 D 1D0		XC	1D0(6,13),1D0(13)	TS=01	TS=01

⑦

Figure 11. Examples of Compiler Output (Part 3 of 4)

```
                                CROSS-REFERENCE DICTIONARY

    (A)
DATA NAMES                      DEFN      REFERENCE

FILE-1                          00017     00060  C006C  00068  0C073
RECCRD-1                        00028     00068
FILE-2                          00018     00073  00073  00076  00076  00079
RECORD-2                        00036     00076
COUNT                           00040     00060  00064  00064  00064  00066  00070
ALPHA                           00042     00064  CC064
NUMBR                           00043     0006C  CC064  00064  0C067
DEPEND                          00045     00066  00066
WORK-RECORD                     CC046     00068  00068  C0076  00078
NAME-FIELD                      00047     00064
RECORD-NO                       00049     00067  00067
NC-OF-DEPENDENTS                00053     00066  00066  00077  0C077  00077  00077
    (B)
PROCEDURE NAMES                 DEFN      REFERENCE

STEP-2                          00064     0007C
STEP-6                          0007b     00078
STEP-8                          00079     00076

                                                                              (8)

  CARD    ERROR MESSAGE
(A)       (B)          (C)                    (D)
  64      ILA5011I-W     HIGH ORDER TRUNCATION MIGHT OCCUR.
  64      ILA5011I-W     HIGH ORDER TRUNCATION MIGHT OCCUR.                    (9)
```

Figure 11. Examples of Compiler Output (Part 4 of 4)

④ Glossary. The glossary is listed when the SYM option is specified. The glossary contains information about names in the COBOL source program.

Ⓐ and Ⓕ The internal-name generated by the compiler. This name is used in the compiler object code listing to represent the name used in the source program. It is repeated in column F for readability.

Ⓑ A normalized level number. This level number is determined by the compiler as follows: the first level number of any hierarchy is always 01, and increments for other levels are always by one. Only level numbers 03 through 49 are affected; level numbers 66, 77, and 88, and FD, SD, and RD indicators are not changed.

Ⓒ The data-name that is used in the source module.

Note: The following Report Writer internally-generated data-names can appear under the SOURCE NAME column:

CTL.LVL	Used to coordinate control break activities.
GRP.IND	Used by coding for GROUP INDICATE clause.
TER.COD	Used by coding for TERMINATE clause.
FRS.GEN	Used by coding for GENERATE clause.
-nnnn	Generated report record associated with the file on which the report is to be printed.
RPT.RCD	Build area for print record.
CTL.CHR	First or second position of RPT.RCD. Used for carriage control character.
RPT.LIN	Beginning of actual information which will be displayed. Second or third position of RPT.RCD.
CODE-CELL	Used to hold code specified.
E.nnnn	Name generated from COLUMN clause in 02-level statement.
S.nnnn	Used for elementary level with SUM clause, but not with data-name.
N.nnnn	Used to save the total number of lines used by a report group when relative line numbering is specified.

Ⓓ and Ⓔ For data-names, these columns contain information about the address in the form of a base and displacement. For file-names, the column contains information about the associated DTF, if any.

Ⓖ This column defines storage for each data item. It is represented in assembler-like terminology. Table 3 refers to information in this column.

Ⓗ Usage of the data-name. For FD entries, the DTF type is identified (e.g., DTFDA). For group items containing a USAGE clause, the usage type is printed. For group items that do not contain a USAGE clause, GROUP is printed. For elementary items, the information in the USAGE clause is printed.

Ⓙ A letter under column:

R - Indicates that the data-name redefines another data-name.

O - Indicates that an OCCURS clause has been specified for that data-name.

Q - Indicates that the data-name is or contains the DEPENDING ON object of the OCCURS clause.

M - Indicates the record format. The letters which may appear under column M are:

F - fixed-length records

U - undefined records

V - variable-length records

S - spanned records

Table 3. Glossary Definition and Usage

Type	Definition	Usage
Group Fixed-Length	DS 0CLN	GROUP
Alphabetic	DS NC	DISP
Alphanumeric	DS NC	DISP
Alphanumeric Edited	DS NC	AN-EDIT
Numeric Edited	DS NC	NM-EDIT
Index-Name	DS 1H	INDEX-NM
Group Variable-Length	DS VLI=N	GROUP
Sterling Report	DS NC	RPT-ST
External Decimal	DS NC	DISP-NM
External Floating-Point	DS NC	DISP-FP
Internal Floating-Point	DS 1F	COMP-1
	DS 1D	COMP-2
Binary	DS 1H, 1F, OR 2F	COMP
Internal Decimal	DS NP	COMP-3
Sterling Non-Report	DS NC	DISP-ST
Index-Name	BLANK	INDEX-NAME
File (FD)	BLANK	DTF TYPE
Condition (88)	BLANK	BLANK
Report Definition (RD)	BLANK	BLANK
Sort Definition (SD)	BLANK	BLANK

Note: Under the definition column, N = size in bytes, except in group variable-length where it is a variable cell number.

⑤ Global tables and literal pool: Global tables are listed when the LISTX option is specified, unless SUPMAP is also specified and an E-level error is encountered. A global table contains easily addressable information needed by the object program for execution. For example, in the Procedure Division output coding (3), the address of the first instruction under STEP-1 (OPEN OUTPUT FILE-1) is found in the PROCEDURE NAME CELLS portion of the Program Global Table (PGT).

Ⓐ The Task Global Table (TGT). This table is used to record and save information needed during the execution of the object program. This information includes switches, addresses, and work areas.

Ⓑ The Literal Pool. This lists all literals used in the program, with duplications removed. These literals include those specified by the programmer (e.g., MOVE "ABC" TO DATA-NAME) and those generated by the compiler (e.g., to align decimal points in arithmetic computations). The literals are divided into two groups: those that are referenced by instructions (marked "LITERAL POOL") and those that are parameters to the display object time subroutine (marked "DISPLAY LITERALS").

Ⓒ The Program Global Table (PGT). This table contains literals and the addresses of procedure-names referenced by Procedure Division instructions.

⑥ Register assignment: This lists the register assigned to each base locator in the object program.

⑦ Object code listing. The object code listing is produced when the LISTX option is specified, unless SUPMAP is also specified and an E-level error is encountered. The actual object code listing contains:

Ⓐ The compiler-generated card number. This number identifies the COBOL statement in the source deck which contains the verb that generates the object code found in column C.

Ⓑ The relative location, in hexadecimal notation, of the object code instruction in the module.

Ⓒ The actual object code instruction in hexadecimal notation.

Ⓓ The procedure-name number. A number is assigned only to procedure-names referred to in other Procedure Division statements.

Ⓔ The object code instruction in the form that closely resembles assembler language. (Displacements are in hexadecimal notation.)

Ⓕ Compiler-generated information about the operands of the generated instruction. This includes names and relative locations of literals. Table 4 refers to information in this column.

Table 4. Symbols Used in the Listing and Glossary to Define Compiler-Generated Information

Symbol	Meaning
DNM	SOURCE DATA NAME
SAV	SAVE AREA CELL
SWT	SWITCH CELL
TLY	TALLY CELL
WC	WORKING CELL
TS	TEMPORARY STORAGE CELL
VLC	VARIABLE LENGTH CELL
SBL	SECONDARY BASE LOCATOR
BL	BASE LOCATOR
BLL	BASE LOCATOR FOR LINKAGE SECTION
ON	ON COUNTER
PFM	PERFORM COUNTER
PSV	PERFORM SAVE
VN	VARIABLE PROCEDURE NAME
SBS	SUBSCRIPT ADDRESS
XSW	EXHIBIT SWITCH
XSA	EXHIBIT SAVE AREA
PRM	PARAMETER
PN	SOURCE PROCEDURE NAME
GN	GENERATED PROCEDURE NAME
DTF	DTF ADDRESS
VN	VARIABLE NAME INITIALIZATION
LIT	LITERAL
TS2	TEMPORARY STORAGE (NON-ARITHMETIC)
RSV	REPORT SAVE AREA
TS3	TEMPORARY STORAGE (SYNCHRONIZATION)
TS4	TEMPORARY STORAGE (SYNCHRONIZATION)
INX	INDEX CELL
V(BCDNAME)	VIRTUAL
VIR	VIRTUAL

⑧ Cross-reference Dictionary: The cross reference dictionary is produced when the XREF option is specified. It consists of two parts:

Ⓐ The XREF dictionary for data-names consists of data-names followed by the generated card number of the statement which defines each data-name, and the generated card number of statements where each data-name is referenced.

Ⓑ The XREF dictionary for procedure-names consists of the procedure-names followed by the generated card number of the statement where each procedure-name is used as a section-name or paragraph-name, and the generated card number of statements where each procedure-name is referenced.

The names appear in the order in which they appear in the source program. The number of references appearing in the cross-reference dictionary for a given name is based upon the number of times the name is referenced in the code generated by the compiler.

⑨ Diagnostic messages: The diagnostic messages associated with the compilation are always listed. The format of the diagnostic message is:

Ⓐ Compiler-generated card number. This is the number of a line in the source program related to the error.

Ⓑ Message identification. The message identification for the Disk Operating System American National Standard COBOL compiler always begins with the symbols ILA.

Ⓒ The severity level. There are four severity levels as follows:

(W) Warning
This level indicates that an error was made in the source program. However, it is not serious enough to interfere with the execution of the program. These warning messages are listed only if the FLAGW option is specified in the CBL card or chosen at system generation time.

(C) Conditional
This level indicates that an error was made but the compiler usually makes a

corrective assumption. The statement containing the error is retained. Execution can be attempted.

(E) Error
This level indicates that a serious error was made. Usually the compiler makes no corrective assumption. The statement or operand containing the error is dropped. Compilation is completed, but execution of the program should not be attempted.

(D) Disaster
This error indicates that a serious error was made. Compilation is not completed. Results are unpredictable.

Ⓓ The message text. The text identifies the condition that caused the error and indicates the action taken by the compiler.

Since Report Writer generates a number of internal data items and procedural statements, some error messages may reflect internal names. In cases where the error occurs mainly in these generated routines, the error messages may indicate the card number of the RD entry for the report under consideration. In addition, there are errors that may indicate the number of the card upon which the statement containing the error ends rather than the card upon which the error occurs. Internal name formats for Report Writer are discussed under "Glossary" (heading 4, item C).

"Appendix F: Diagnostic Messages" gives a complete list of compiler diagnostic messages.

OBJECT MODULE

The object module contains the external symbol dictionary, the text of the program, and the relocation dictionary. It is followed by an END statement that marks the end of the module. For additional information about the external symbol dictionary and the relocation dictionary, see the publication IBM System/360 Disk Operating System: System Control and System Service Programs.

An object deck is punched if the DECK option is specified, unless SUPMAP option was specified and an E-level diagnostic message was generated. The object module is written on SYSLNK if the LINK option is specified, unless SUPMAP was specified and an E-level diagnostic message is generated.

LINKAGE EDITOR OUTPUT

The output of the linkage edit step may include:

- A printed listing of the job control statements
- A map of the phase after it has been processed by the Linkage Editor
- Diagnostic messages
- A listing of the linkage editor control statements
- A phase which may be assigned to the core image library

Any diagnostic messages associated with the Linkage Editor are automatically generated as output. The other forms of output may be requested by the OPTION control statement. All output to be listed is printed on the device assigned to SYSLST.

Figure 12 is an example of a linkage editor output listing. It shows the job control statements and the phase map. The different types of output are numbered and each type to be explained is lettered. The text following the figure is an explanation of the figure.

```
 ENTRY
// EXEC LNKEDT
```
(1)

```
JOB  SAMPLE                DISK LINKAGE EDITOR DIAGNOSTIC OF INPUT
```
(2)

```
ACTION TAKEN    MAP
LIST    PHASE TEST,*
LIST    AUTOLINK    IJFFBZZN
LIST    AUTOLINK    ILBDDSP0
LIST    INCLUDE IJJCPD1
LIST    AUTOLINK    ILBDIML0
LIST    AUTOLINK    ILBDMNS0
LIST    AUTOLINK    ILBDSAE0
LIST    ENTRY
```

```
      (A)      (B)      (C)      (D)      (E)      (F)        (G)        (H)      (J)
     PHASE   XFR-AD   LOCORE   HICORE   DSK-AD   ESD TYPE   LABEL      LOADED   REL-FR

TEST         003000   003000   0048CB   50 07 2  CSECT      TESTRUN    003000   003000

                                                 CSECT      IJFFBZZN   0039C0   0039C0
                                                 *  ENTRY   IJFFZZZN   0039C0
                                                 *  ENTRY   IJFFBZZZ   0039C0
                                                 *  ENTRY   IJFFZZZZ   0039C0

                                                 CSECT      ILBDSAE0   0047D8   0047D8
                                                    ENTRY   ILBDSAE1   0047F8

                                                 CSECT      ILBDMNS0   0047D0   0047D0

                                                 CSECT      ILBDDSP0   003F90   003F90
                                                 *  ENTRY   ILBDDSP1   0044E0
                                                 *  ENTRY   ILBDDSP2   004578
                                                 *  ENTRY   ILBDDSP3   004730

                                                 CSECT      ILBDIML0   004768   004768

                                                 CSECT      IJJCPD1    003DC8   003DC8
                                                    ENTRY   IJJCPD1N   003DC8
                                                 *  ENTRY   IJJCPD3    003DC8
```
(3)

Figure 12. Linkage Editor Output

① The job control statements. These statements are listed since the LOG option is specified.

② Disk linkage editor diagnostic message of input. The ACTION statement is not required. If the MAP option is specified, SYSLST must be assigned. If the statement is not used and SYSLST is assigned, MAP is assumed and a map of main storage and any error diagnostic messages are considered output on SYSLST.

③ Map of main storage. A phase map is printed when MAP is specified (or assumed) during linkage editor processing. The following information is contained in the map of main storage:

Ⓐ The name of each phase. This is the name specified in the phase statement.

Ⓑ The transfer address of each phase.

Ⓒ The lowest main storage location of each phase.

Ⓓ The highest main storage location of each phase.

Ⓔ The hexadecimal disk address where the phase begins in the core image library.

Ⓕ The names of all CSECT's belonging to a phase.

Ⓖ All defined entry points within a CSECT. If an entry point is not referenced, it is flagged with an asterisk (*).

Ⓗ The address where each CSECT is loaded.

Ⓙ The relocation factor of each CSECT.

Comments on the Phase Map

The severity of linkage editor diagnostic messages may affect the production of the phase map. Since various processing options affect the structure of the phase, the text of the phase map will sometimes provide additional information. For example, the phase may contain an overlay structure. In this case, a map will be listed for each segment in the overlay structure.

Linkage Editor Messages

The Linkage Editor may generate informative or diagnostic messages. A complete list of these messages is included in the publication IBM System/360 Disk Operating System: System Control and System Service Programs.

COBOL PHASE EXECUTION OUTPUT

The output generated by program execution (in addition to data written on output files) may include:

- Data displayed on the console or on the printer
- Messages to the operator
- System informative messages
- System diagnostic messages
- A system dump

A dump and system diagnostic messages are generated automatically during program execution only if the program contains errors that cause abnormal termination.

Figure 13 is an example of output from the execution job step. The following text is an explanation of the illustration.

① Job control statements. These statements are listed because the LOG option is specified.

② Program output on printer. The results of execution of the TRACE and EXHIBIT NAMED statements appear on the program listing.

③ Console output. Data is printed on the console as a result of the execution of DISPLAY UPON CONSOLE.

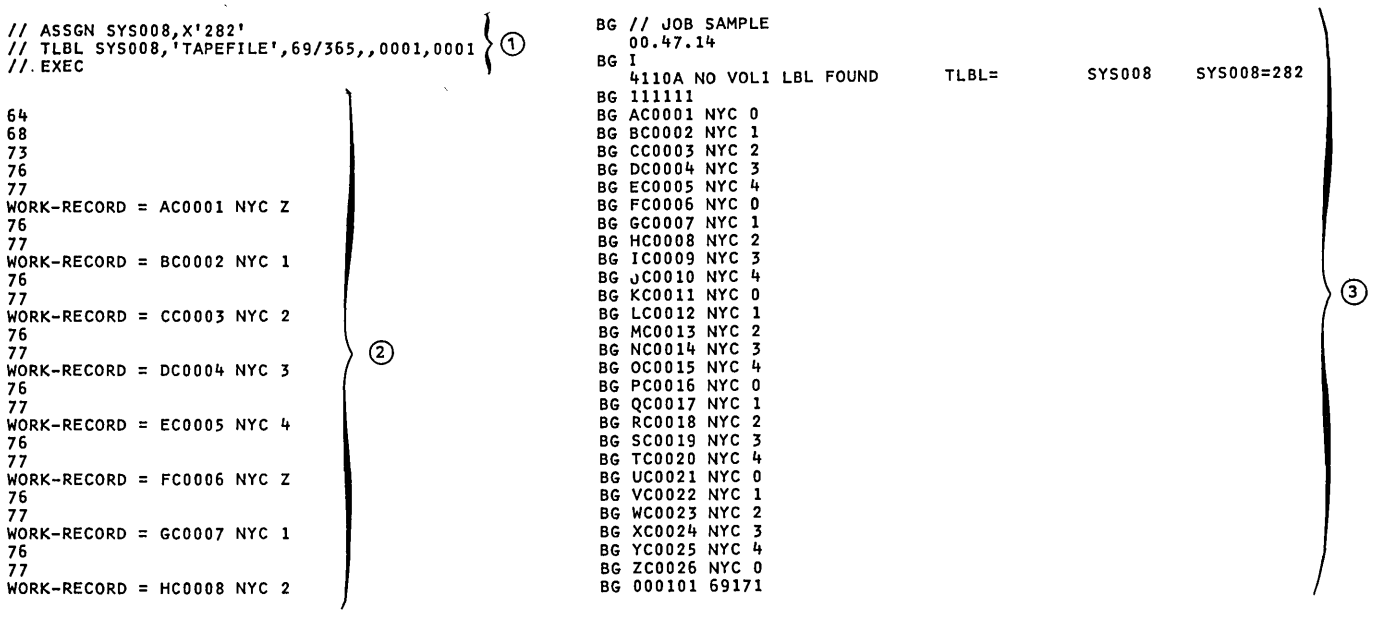


Figure 13. Output from Execution Job Steps

OPERATOR MESSAGES

The COBOL phase may issue operator messages. In the message, XX denotes a system-generated 2-character numeric field that is used to identify the program issuing the message.

STOP Statement

The following message is generated by the STOP statement with the literal option:

XX C110A STOP 'literal'

Explanation: This message is issued at the programmer's discretion to indicate possible alternative action to be taken by the operator.

Operator Response: Follows the instructions given both by the message and on the job request form supplied by the programmer. If the job is to be resumed, hit end-of-block.

ACCEPT Statement

The following message is generated by an ACCEPT statement with the FROM CONSOLE option:

XX C111A "AWAITING REPLY"

Explanation: This message is issued by the object program when operator intervention is required.

Operator Response: Enter the reply and hit end-of-block. To send message, hit end-of-block again. (The contents of the text field should be supplied by the programmer on the job request form.)

SYSTEM OUTPUT

Informative and diagnostic messages may appear in the listing during the execution of the object program.

Each of these messages contains an identification code in the first column of the message to indicate the portion of the operating system that generated the message. Table 5 lists these codes, together with identification for each.

Table 5. System Message Identification Codes

Code	Identification
0	An on-line console message from the Supervisor
1	A message from the Job Control Processor
2	A message from the Linkage Editor
3	A message from the Librarian
4	A message from LIOCS
7	A message from the Sort program
C	A message from COBOL object time subroutines

This chapter describes the accepted linkage conventions for calling and called programs and discusses linkage methods when using an assembler language program. In addition, this chapter contains a description of the overlay facility which enables different called programs to occupy the same area in main storage at different times. It also contains a suggested assembler language program to be used in conjunction with the overlay feature.

A COBOL source program that passes control to another program is a calling program. The program that receives control from the calling program is referred to as a called program. Both programs must be compiled (or assembled) in separate job steps, but the resulting object modules must be linkage edited together in the same phase.

A called program can also be a calling program; that is, a called program can, in turn, call another program. In Figure 14 for instance, program A calls program B; program B calls program C. Therefore:

1. A is considered a calling program by B
2. B is considered a called program by A
3. B is considered a calling program by C
4. C is considered a called program by B

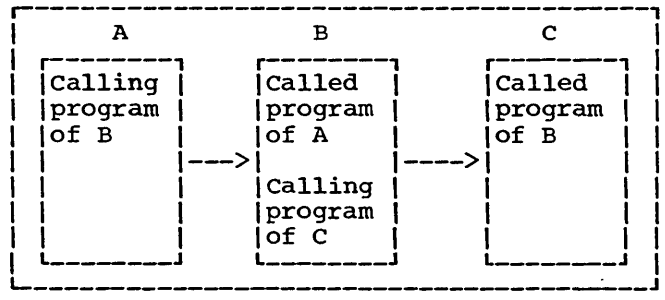


Figure 14. Calling and Called Programs

By convention, a called program may call to an entry point in any other program, except one on a higher level in the "path" of that program. That is, A may call to an entry point in B or C, and B may call C; however, C should not call A or B. Instead, C transfers control only to B by issuing the EXIT PROGRAM or GOBACK statements in COBOL (or its equivalent in another language). B then returns to A.

Compiler generated switches, e.g., ON and ALTER, are not reinitialized upon each entrance to the called program, that is, the program is in its last executed state.

LINKAGE

Whenever a program calls another program, linkage must be established between the two. The calling program must state the entry point of the called program and must specify any arguments to be passed. The called program must have an entry point and must be able to accept the arguments. Further, the called program must establish the linkage for the return of control to the calling program.

LINKAGE IN A CALLING PROGRAM

A calling COBOL program must contain the following statement at the point where another program is to be called:

```
CALL literal-1 [USING identifier-1
          [identifier-2]...]
```

literal-1
: is the name specified as the program-name in the PROGRAM-ID paragraph of the called program, or the name of the entry point in the called program. When the called program is to be entered at the beginning of the Procedure Division, literal-1 is the name of the program being called. When the called program is to be entered at some point other than the beginning of the Procedure Division, literal-1 should not be the same as the name specified in the PROGRAM-ID paragraph of the called program. Since the program-name in the PROGRAM-ID paragraph produces an external reference defining an entry point, this entry point name would not be uniquely defined as an external reference.

identifier-1 [identifier-2]...
: are the arguments being passed to the called program. Each identifier represents a data item defined in the Linkage Section of the called program

and must contain a level number 01 or 77. If the called program is an assembler language program, the arguments may represent file-names and procedure-names. If no arguments are to be passed, the USING option is omitted.

LINKAGE IN A CALLED PROGRAM

A called COBOL program must contain two sets of statements:

1. One of the following statements must appear at the point where the program is entered.

   If the called program is entered at the first instruction in the Procedure Division and arguments are passed by the calling program:

```
PROCEDURE DIVISION [USING
  identifier-1 [identifier-2]...].
```

   If the entry point of the called program is not the first statement of the Procedure Division:

```
ENTRY literal-1 [USING identifier-1
  [identifier-2]...]
```

   literal-1
   is the name of the entry point in the called program. It is the same name that appears in the CALL statement of the program that calls this program. literal-1 must not be the name of any other entry point or program-name in the run unit.

   identifier-1 [identifier-2]...]
   are the data items representing parameters. They correspond to the arguments of the CALL statement of the calling program. Each data item in this parameter list must be defined in the Linkage Section of the called program and must contain a level number of 01 or 77.

2. Either of the following statements must be inserted where control is to be returned to the calling program.

```
EXIT PROGRAM.
GOBACK.
```

   Both the EXIT PROGRAM and GOBACK statements cause the restoration of the necessary registers, and return control to the point in the calling program immediately following the calling sequence.

ENTRY POINTS

Each time an entry point is specified in a called program, an external-name is defined. An external-name is a name that can be referenced by another program that has been separately compiled or assembled. Each time an entry name is specified in a calling program, an external reference is defined. An external reference is a symbol that is defined as an external-name in another separately compiled or assembled program. The Linkage Editor resolves external-names and external references, and combines calling and called programs into a format suitable for execution together, i.e., as a single phase.

Note: Several different entry points may be defined in one COBOL source module. Different CALL statements in any module of the phase may specify the same entry point, but each definition of an entry point must be unique in the same phase.

CORRESPONDENCE OF ARGUMENTS AND PARAMETERS

The number of identifiers in the parameter list of the called program must be the same as the number of identifiers in the argument list of the calling program. There is a one-for-one correspondence. The correspondence is positional and not by name. An identifier must not appear more than once in the same USING clause.

Only the address of an argument is passed. Consequently, both the identifier that is an argument and the identifier that is the corresponding parameter refer to the same location in main storage. The pair of identifiers need not be identical, but the data descriptions must be equivalent. For example, if an argument is a level-77 data-name representing a 30-character string, its corresponding parameter could also be a level-77 data-name representing a character string of length 30, or the parameter could be a level-01 data item

with subordinate items representing character strings whose combined length is 30.

Although all parameters in the ENTRY statement must be described with level numbers 01 or 77, there is no such restriction made for arguments in the CALL statement. An argument may be a qualified name or a subscripted name. When a group item with a level number other than 01 is specified as an argument, proper boundary word alignment is required if subordinate items are described as COMPUTATIONAL, COMPUTATIONAL-1, or COMPUTATIONAL-2. If the argument corresponds to an 01-level parameter, doubleword alignment is required.

LINKAGE EDITING WITHOUT OVERLAY

Assume that a COBOL main program (COBMAIN), at one or more points in its logic executes CALL statements to COBOL programs SUBPRGA, SUBPRGB, SUBPRGC, and SUBPRGD. Also assume that the module sizes for the main program and subprograms are:

Program	Module Size (in bytes)
COBMAIN	20,000
SUBPRGA	4,000
SUBPRGB	5,000
SUBPRGC	6,000
SUBPRGD	3,000

Through the linkage mechanism, all called programs plus COBMAIN must be linkage edited together to form one module of 38,000 bytes. Therefore, COBMAIN would require 38,000 bytes of storage in order to be executed. No overlay structure need be specified at linkage edit time if 38,000 bytes of core storage are available.

The following is an example of the job control statements needed to linkage edit these calling and called programs without specifying an overlay structure. The source decks for COBMAIN and SUBPRGA are included in the job deck, whereas SUBPRGB, SUBPRGC, and SUBPRGD are in the relocatable library.

```
// JOB NOVERLAY
// OPTION LINK,LIST,DUMP
   ACTION MAP
   PHASE EXAMP1,*
   INCLUDE

   {object module COBMAIN}
/*
   INCLUDE SUBPRGB
   INCLUDE SUBPRGC
   INCLUDE SUBPRGD
   INCLUDE

   {object module SUBPRGA}
/*
   ENTRY
// EXEC LNKEDT
// EXEC

   {data for program}
/*
/&
```

Figure 15 is an example of the data flow logic of this call structure where all the programs fit into main storage.

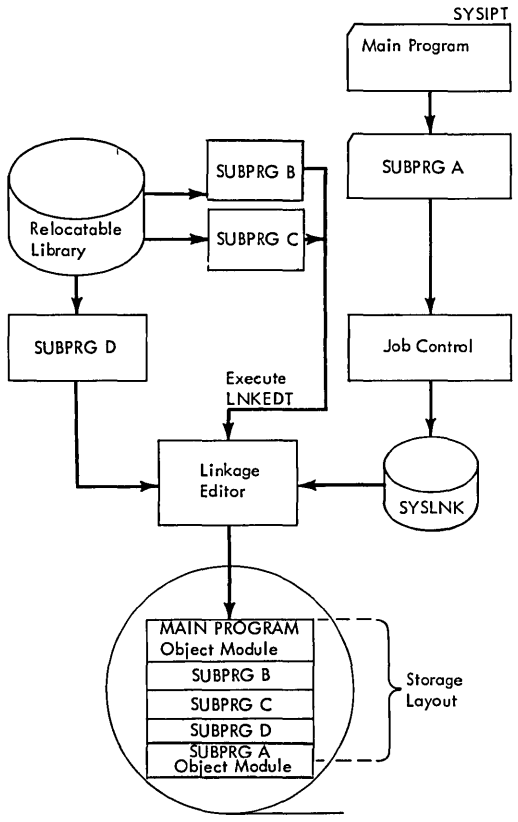


Figure 15. Example of Data Flow Logic in a Call Structure

Note: For the example given, it is assumed that SYSLNK is a standard assignment. The flow diagram illustrates how the various

program segments are linkage edited into storage in a sequential arrangement.

ASSEMBLER LANGUAGE SUBPROGRAMS

A main program written in COBOL can call programs written in other languages that use the same linkage conventions. Whenever a COBOL program calls an assembler language program, certain conventions and techniques must be used.

There are three basic ways to use assembler-written called programs with a main program written in COBOL:

1. A COBOL main program or called program calling an assembler-writtem program.
2. An assembler-written program calling a COBOL program.
3. An assembler-written program calling another assembler-written program.

From these combinations, more complicated structures can be formed.

In a COBOL program, the expansions of the CALL and GOBACK or EXIT PROGRAM statements provide the save and return coding that is necessary to establish linkage between the calling and called programs in accordance with the linkage conventions of the system. Assembler language programs must be prepared in accordance with the same linkage conventions. These conventions include:

1. Using the proper registers to establish linkage.
2. Reserving, in the calling program, a storage area for items contained in the argument list. This storage area can be referenced by the called program.
3. Reserving, in the calling program, a save area in which the contents of the registers can be saved.

REGISTER USE

The Disk Operating System has assigned functions to certain registers used in linkages. Table 6 shows the conventions for using general registers as linkage registers. The calling program must load the address of the return point into register 14, and it must load the address of the entry point of the called program into register 15.

Table 6. Conventional Use of Linkage Registers

Reg. No.	Reg. Name	Function
1	Argument list register	Address of the argument list passed to the called program.
13	Save area register	Address of the area reserved by the calling program in which the contents of certain registers are stored by the called program.
14	Return register	Address of the location in the calling program to which control is returned after execution of the called program.
15	Entry point register	Address of the entry point in the called program.

SAVE AREA

A calling assembler language program must reserve a save area of 18 words, beginning on a fullword boundary, to be used by the called program for saving registers; it must load the address of this area into register 13. Table 7 shows the layout of the save area and the contents of each word.

A called COBOL program does not save floating-point registers. The programmer is responsible for saving and restoring the contents of these registers in the calling program.

ARGUMENT LIST

The argument list is a group of contiguous fullwords, beginning on a fullword boundary, each of which is an address of a data item to be passed to the called program. If the program is to pass arguments, an argument list must be prepared and its address loaded into register 1. The high-order bit of the last argument, by convention, is set to 1 to indicate the end of the list.

Table 7. Save Area Layout and Word Contents

AREA (word 1)	This word is a part of the standard linkage convention established under the Disk Operating System. The word must be reserved for proper addressing of the subsequent entries. However, an assembler subprogram may use the word for any desired purpose.
AREA+4 (word 2)	The address of the previous save area, that is, the save area of the subprogram that called this one.
AREA+8 (word 3)	The address of the next save area, that is, the save area of the subprogram to which this subprogram refers.
AREA+12 (word 4)	The contents of register 14, that is, the return address.
AREA+16 (word 15)	The contents of register 15, that is, the entry address.
AREA+20 (word 6)	The contents of register 0.
AREA+24 (word 7)	The contents of register 1.
.	.
.	.
.	.
AREA+68 (word 18)	The contents of register 12.

Any assembler-written program must be coded with a detailed knowledge of the data formats of the arguments being passed. Most coding errors occur because of the data format discrepancies of the arguments.

If one programmer writes both the calling program and the called program, the data format of the arguments should not present a problem when passed as parameters. However, when the programs are written by different programmers, the data format specifications for the arguments must be clearly defined for the user.

The linkage conventions used by an assembler program that calls another program are illustrated in Figure 16. The linkage should include:

1. The calling sequence.
2. The save and return routines.
3. The out-of-line parameter list. (An in-line parameter list may be used.)
4. A save area on a fullword boundary.

In-Line Parameter List

The assembler programmer may establish an in-line parameter list instead of an out-of-line list. In this case, he may substitute the calling sequence and parameter list illustrated in Figure 17 for that shown in Figure 16.

```
deckname  START  0                 INITIATES PROGRAM ASSEMBLAGE AT FIRST
*                                  AVAILABLE LOCATION.  ENTRY POINT TO THE
*                                  PROGRAM.
          ENTRY  name1
          EXTRN  name2
          USING  name1,15
*  SAVE ROUTINE
name1     STM    14,r1,12(13)      THE CONTENTS OF REGISTERS 14, 15, AND
*                                  0 THROUGH r1 ARE STORED IN THE SAVE
*                                  AREA OF THE CALLING PROGRAM (PREVIOUS
*                                  SAVE AREA). r1 IS ANY NUMBER FROM 0 THROUGH 12.
          LR     r3,15
          DROP   15
          USING  name1,r3          WHERE r3 AND r2 HAVE BEEN SAVED
          LR     r2,13             LOADS REGISTER 13, WHICH POINTS TO THE
*                                  SAVE AREA OF THE CALLING PROGRAM, INTO
*                                  ANY GENERAL REGISTER, r2, EXCEPT 0 AND 13.
          LA     13,AREA           LOADS THE ADDRESS OF THIS PROGRAM'S
*                                  SAVE AREA INTO REGISTER 13.
          ST     13,8(r2)          STORES THE ADDRESS OF THIS PROGRAM'S SAVE
*                                  AREA INTO WORD 3 OF THE SAVE AREA OF THE
*                                  CALLING PROGRAM.
          ST     r2,4(13)          STORES THE ADDRESS OF THE PREVIOUS SAVE
*                                  AREA (I.E., THE SAME AREA OF THE CALLING
*                                  PROGRAM) INTO WORD 2 OF THIS PROGRAM'S
*                                  SAVE AREA.
          BC     15,prob1
AREA      DS     18F               RESERVES 18 WORDS FOR THE SAVE AREA
*                                  THIS IS LAST STATEMENT OF SAVE ROUTINE.
prob1     {User-written program statements}
*  CALLING SEQUENCE
          LA     1,ARGLST
          L      15,ADCON
          BALR   14,15
          {Remainder of user-written program statements}
*  RETURN ROUTINE
          L      13,4(13)          LOADS THE ADDRESS OF THE PREVIOUS SAVE
*                                  AREA BACK INTO REGISTER 13.
          LM     2,r1,28(13)       THE CONTENTS OF REGISTER 2 THROUGH r1 ARE
*                                  RESTORED FROM THE PREVIOUS SAVE AREA.
          L      14,12(13)         LOADS THE RETURN ADDRESS, WHICH IS IN
*                                  WORD 4 OF THE CALLING PROGRAM'S SAVE AREA,
*                                  INTO REGISTER 14.
          MVI    12(13),X'FF'      SETS FLAG FF IN THE SAVE AREA OF THE
*                                  CALLING PROGRAM TO INDICATE THAT CONTROL
*                                  HAS RETURNED TO THE CALLING PROGRAM.
          BCR    15,14             LAST STATEMENT IN RETURN ROUTINE
ADCON     DC     A(name2)          CONTAINS THE ADDRESS OF SUBPROGRAM name2.
*  PARAMETER LIST
ARGLST    DC     AL4(arg1)         FIRST STATEMENT IN PARAMETER AREA SETUP
          DC     AL4(arg2)
          DC     X'80'             FIRST BYTE OF LAST ARGUMENT SETS BIT 0 TO 1
          DC     AL3(argn)         LAST STATEMENT IN PARAMETER AREA SETUP
```

Figure 16. Sample Linkage Routines Used with a Calling Subprogram

```
ADCON    DC      A(prob1)
           .
           .
           .
         LA      14,RETURN
         L       15,ADCON
         CNOP    2,4
         BALR    1,15
         DC      AL4(arg1)
         DC      AL4(arg2)
           .
           .
           .
         DC      X'80'
         DC      AL3(argn)
RETURN   EQU     *
```

Figure 17. Sample In-line Parameter List

LOWEST LEVEL PROGRAM

If an assembler called program does not call any other program (i.e., if it is at the lowest level), the programmer should omit the save routine, calling sequence, and parameter list shown in Figure 16. If the assembler called program uses any registers, it must save them. Figure 18 illustrates the appropriate linkage conventions used by an assembler program at the lowest level.

```
deckname  START  0
          ENTRY  name

          USING  *,15
name      STM    14,r1,12(13)
            .
            .
            .
User-written program statements
            .
            .
            .
          LM     2,r1,28(13)
          MVI    12(13),X'FF'
          BCR    15,14
```

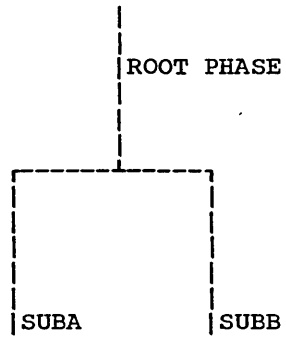
Note: If registers 13 and/or 14 are used in the called subprogram, their contents should be saved and restored by the called subprogram.

Figure 18. Sample Linkage Routines Used with a Lowest Level Subprogram

OVERLAYS

If a program is too large to be contained in the number of bytes available in main storage, it can still be executed by means of an overlay structure. An overlay structure permits the re-use of storage locations previously occupied by another program. In order to use an overlay structure, the programmer must plan his program so that one or more called programs need not be in main storage at the same time as the rest of the program phase.

The following is a diagram of the basic form of a program to be overlaid:



The root phase consists of the COBOL main program and an assembler language subroutine which handles the overlay structures. SUBA and SUBB are the called programs that are to be overlaid in core storage.

In using the overlay technique, the programmer specifies to the Linkage Editor which programs are to overlay each other. These programs are processed by the Linkage Editor so they can be placed automatically in main storage for execution when called by the main program. The resulting output of the Linkage Editor is called an overlay structure.

SPECIAL CONSIDERATIONS WHEN USING OVERLAY STRUCTURES

There are three areas of special concern to the programmer who decides to use the overlay feature. These problems concern the use of the assembler language subroutine, proper linkage editing, and job control statements.

ASSEMBLER LANGUAGE SUBROUTINE FOR ACCOMPLISHING OVERLAY

The CALL statement is used for "direct" linkage; that is, the assistance of the Supervisor is not required (as it is when loading or fetching a phase). There are no COBOL statements that will generate the equivalent of the LOAD or FETCH assembler macro instructions. For this reason, one must call an assembler program to effect an overlay of a COBOL program. This routine must be linkage edited as part of either a root phase or permanently resident phase.

The sample overlay subroutine shown in Figure 19 is governed by the following restrictions:

1. The example is a suggested technique, and is not the only technique.
2. It can be used for assembler overlays if the user has a desired entry point in his END card and the first statement at that entry point is 'STM 14,12,12(13)'(90ECD00C).
3. The subroutine cannot be used for entry points other than at the first instruction of the Procedure Division. A suggested technique for diverse entry points is a table look-up using V-type constants.

Note: Care should be taken with the techniques used in statements 0019 and 0020. Only when the COBOL program is

```
STMNT       SOURCE STATEMENT

0001  OVERLAY   START 0
0002            ENTRY OVRLAY
0003  * AT ENTRY TIME
0004  *     R1=POINTER TO ADCON LIST OF USING ARGUMENTS
0005  *     FIRST ARGUMENT IS PHASE OR SUBROUTINE NAME
0006  *     MUST BE 8 BYTES
0007  *     R13=ADDRESS OF SAVE AREA
0008  *     R14=RETURN POINT OF CALLING PROGRAM
0009  *     R15=ENTRY POINT OF OVERLAY PROGRAM
0010  * AT EXIT
0011  *     R1=POINTER TO SECOND ARGUMENT OF ADCON LIST
0012  *        OF USING ARGUMENTS
0013  *     R14=RETURN POINT OF CALLING PROGRAM--NOT THIS PROG
0014  *     R15=ENTRY POINT OF PHASE OR SUBPROGRAM
0015  *
0016            USING *,15
0017            STM   0,1,20(13)      SAVE REG 0 AND 1
0018            L     1,0(1)          R1=ADDRESS OF PHASE NAME
0019            CLC   0(8,1),CORSUB   IS IT IN CORE
0020            BE    SUBIN           YES
0021            MVC   CORSUB(8),0(1)  NO,CORSUB=PHASE NAME
0022            SR    0,0             R0=0
0023  *     LOAD REQUIRES R0=0 IF LOAD ISN'T SPECIFIED,
0024  *        R1=ADDRESS OF PHASE ENTRY
0025  *            UPON RETURN
0026  *
0027  *
0028            SVC   4               LOAD PHASE
0029            SH    1,HWTWO
0030  ENTRY     LA    1,2(1)          STEP SEARCH POINTER
0031            CLC   0(4,1),STMINS   IS THIS THE ENTRY POINT
0032            BNE   ENTRY           NO,LOOP BACK
0033            ST    1,ASUB          YES, SAVE IT
0034  SUBIN     LM    0,1,20(13)      RESTORE REG0 AND 1
0035            LA    1,4(1)          STEP PAST PHASE NAME ADCON
0036            L     15,ASUB         LOAD ENTRY POINT ADDRESS
0037            BR    15
0038  ASUB      DS    1F
0039  CORSUB    DC    8X'FF'
0040  STMINS    DC    X'90ECD00C'
0041  HWTWC     DC    H'2'
0042            END
```

Figure 19. Example of an Assembler Language Subroutine for Accomplishing Overlay

loaded are altered GO TO statements reinitialized. A better technique would be to load the called programs each time they are required.

LINKAGE EDITING WITH OVERLAY

In a linkage editor job step, the programmer specifies the overlay points in a program by using PHASE statements. In the Working-Storage Section, a level-77 constant must be created for each phase to be called at execution time. These constants have a PICTURE of X(8) and a VALUE clause containing the same name as that appearing on the PHASE card for that segment in the linkage edit run.

In addition, each argument to be passed to the called program must have an entry in the Linkage Section. Remember, also, that the ENTRY statement should not refer to the program-name but rather to the paragraph-name in the Procedure Division where control is to be passed on entering the called program. (Use of the program-name will result in incorrect execution.)

When preparing the control cards for the Linkage Editor, the programmer should be certain to include the assembler language subroutine with the main (root) phase. Also, to achieve maximum overlay, the phase names for the called programs should be different from the names of the called programs specified in the PROGRAM-ID paragraphs.

Figure 20 is a flow diagram of the overlay logic. The PHASE cards indicate the beginning address of each phase. The phases OVERLAYC and OVERLAYD will have the same beginning address as OVERLAYB. The sequence of events is:

1. The main program calls the overlay routine.
2. The overlay routine fetches the particular COBOL subprogram and places it in the overlay area.
3. The overlay routine transfers control to the first instruction of the called program.
4. The called program returns to the COBOL calling program (not to the assembler language overlay routine).

If OVERLAYB were known to be in storage, the CALL statement would be:

```
CALL "OVERLAYB" USING PARAM-1, PARAM-2.
```

But when using the assembler language overlay routine (OVRLAY), it becomes:

```
CALL "OVRLAY" USING PROCESS-LABEL,
  PARM-1, PARM-2.
```

where PROCESS-LABEL contains the external-name OVERLAYB of the called program.

However, the ENTRY statement of the called program is the same for both cases, i.e., ENTRY "OVERLAYB" USING PARAM-1, PARAM-2, whether it is called indirectly by the main program through the overlay program or called directly by the main program.

Note: An ENTRY which is to be called by OVRLAY must precede the first executable statement in the called program.

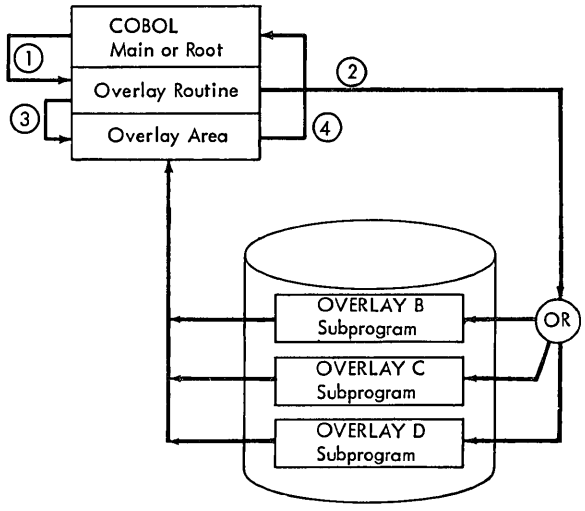


Figure 20. Flow Diagram of Overlay Logic

JOB CONTROL FOR ACCOMPLISHING OVERLAY

The job control statements required to accomplish the overlay illustrated in Figure 20 are shown in Figure 21. The PHASE statements specify to the Linkage Editor that the overlay structure to be established is one in which the called programs OVERLAYB, OVERLAYC, and OVERLAYD overlay each other when called during execution.

Note: The phase name specified in the PHASE card must be the same as the value contained in the first argument for CALL "OVRLAY", i.e., PROCESS-LABEL, COMPUTE-TAX, etc., contain OVERLAYB, OVERLAYC, respectively, which are the names given in the PHASE card.

It is the programmer's responsibility to write the entire overlay, i.e., the COBOL main (or calling) program and an assembler language subroutine (for which a sample program is given in this chapter) that fetches and overlays the called programs. A calling sequence to obtain an overlay structure between three COBOL subprograms is illustrated in Figure 22.

```
// JOB OVERLAYS
// OPTION LINK
   PHASE OVERLAY,ROOT
// EXEC FCOBOL
   {COBOL Source for Main Program OVERLAY}
/*
// EXEC ASSEMBLY
    Source deck for Assembler Language Routine OVRLAY
/*
   PHASE OVERLAYB,*
// EXEC FCOBOL
    {COBOL Source for Called Program OVERLAYB}
/*
   PHASE OVERLAYC,OVERLAYB
// EXEC FCOBOL
    {COBOL Source for Called Program OVERLAYC}
/*
   PHASE OVERLAYD,OVERLAYC
    {COBOL Source for Called Program OVERLAYD}
/*
// EXEC LNKEDT
// EXEC
/*
/&
```

Figure 21. Job Control for Accomplishing Overlay

```
COBOL Program Main (Root or Main Program)

IDENTIFICATION DIVISION.
PROGRAM-ID. OVERLAY.
.
.
.
ENVIRONMENT DIVISION.
.
.
.
DATA DIVISION.
.
.
.
WORKING-STORAGE SECTION.
77  PROCESS-LABEL PICTURE IS X(8) VALUE IS "OVERLAYB".
77  PARAM-1 PICTURE IS X.
77  PARAM-2 PICTURE IS XX.
77  COMPUTE-TAX PICTURE IS X(8) VALUE IS "OVERLAYC".

01  NAMET.
    02 EMPLY-NUMB PICTURE IS 9(5).
    02 SALARY PICTURE IS 9(4)V99.
    02 RATE PICTURE IS 9(3)V99.
    02 HOURS-REG PICTURE IS 9(3)V99.
    02 HOURS-OT PICTURE IS 9(2)V99.
01  COMPUTE-SALARY PICTURE IS X(8) VALUE IS "OVERLAYD".
01  NAMES.
    02 RATES PICTURE IS 9(6).
    02 HOURS PICTURE IS 9(3)V99.
    02 SALARYX PICTURE IS 9(2)V99.
.
.
.
PROCEDURE DIVISION.
.
.
.
    CALL "OVRLAY" USING PROCESS-LABEL, PARAM-1, PARAM-2.
.
.
.
    CALL "OVRLAY" USING COMPUTE-TAX, NAMET.
.
.
.
    CALL "OVRLAY" USING COMPUTE-SALARY, NAMES.
.
.
.
```

Figure 22. Calling Sequence to Obtain Overlay Between Three COBOL Subprograms (Part 1 of 3)

```
COBOL Subprogram B

IDENTIFICATION DIVISION.
PROGRAM-ID. OVERLAY1.
.
.
.
ENVIRONMENT DIVISION.
.
.
.
DATA DIVISION.
.
.
LINKAGE SECTION.

01  PARAM-10 PICTURE X.
01  PARAM-20 PICTURE XX.
.
.
.
PROCEDURE DIVISION.
PARA-NAME.  ENTRY "OVERLAYB" USING PARAM-10, PARAM-20.
.
.
.
             GOBACK.


COBOL Subprogram C

IDENTIFICATION DIVISION.
PROGRAM-ID. OVERLAY2.
.
.
.
 ENVIRONMENT DIVISION.
.
.
.
.
DATA DIVISION.
.
.
LINKAGE SECTION.

01  NAMEX.
    02 EMPLY-NUMBX PICTURE IS 9(5).
    02 SALARYX PICTURE IS 9(4) V99.
    02 RATEX PICTURE IS 9(3)V99.
    02 HOURS-REGX   PICTURE IS 9(3)V99.
    02 HOURS-OTX PICTURE IS 9(2)V99.

PROCEDURE DIVISION.
PARA-NAME. ENTRY "OVERLAYC" USING NAMEX.
.
.
.
             GOBACK.
```

Figure 22. Calling Sequence to Obtain Overlay Between Three COBOL Subprograms (Part 2 of 3)

```
COBOL Subprogram D

IDENTIFICATION DIVISION.
PROGRAM-ID. OVERLAY3.
.
.
.
ENVIRONMENT DIVISION.
.
.
.
DATA DIVISION.
.
.
LINKAGE SECTION
01  NAMES.
    02 RATES PICTURE IS 9(6).
    02 HOURS PICTURE IS 9(3)V99.
    02 SALARYX PICTURE IS 9(2)V99.
.
PROCEDURE DIVISION.
PARA-NAME. ENTRY "OVERLAYD" USING NAMES.
.
.
.
           GOBACK.
```

Figure 22. Calling Sequence to Obtain Overlay Between Three COBOL Subprograms (Part 3 of 3)

In order to use the System/360 Disk Operating System Sort/Merge program, Sort Feature statements are written in the COBOL source program. These statements are described in the publication IBM System/360 Disk Operating System: American National Standard COBOL. The Sort/Merge program itself is described in the publication IBM System/360 Disk Operating System: Tape and Disk Sort/Merge, Form GC28-6676. "Appendix G: Machine Considerations" in this publication contains information about system requirements when the Sort Feature is used.

Additional job control statements must be included in the execution step of the job to describe the files used by the sort program. These statements are described below in "Sort Job Control Requirements."

Note: The Checkpoint/Restart Feature can be activated during a sorting operation by specifying the RERUN statement.

SORT JOB CONTROL REQUIREMENTS

Three types of files can be defined for the Sort program in the execution job step: input, output, and work.

SORT INPUT AND OUTPUT CONTROL STATEMENTS

When the USING and/or GIVING options are specified, the compiler generates dummy Input and/or Output Procedures. Hence, the job control requirements for files named as operands of USING and GIVING are the same as those for files used as input to or output from the sorting operation in these procedures.

The following job control statements are required for files used as input to or output from the sorting operation:

```
ASSGN

  followed by

    VOL
    TPLAB

or

    VOL
    DLAB
    XTENT

or

    DLBL
    EXTENT

or

    TLBL
```

The symbolic unit to which each sort input or output file is assigned in the source language ASSIGN clause is specified in an ASSGN control statement.

Note: ASSGN control statements are required only if the input/output devices used in an application have not been previously assigned the appropriate symbolic names.

If an input file contains standard labels, a TLBL or DLBL (or VOL and TPLAB or VOL and DLAB) statement(s) is required. The symbolic name of the device from which the input file is to be read must also be included on this statement.

One EXTENT (XTENT) control statement is required to define the limits of each area of a mass storage device from which an input file will be read. EXTENT (XTENT) statements must include the symbolic unit name of the device containing the extent.

If the output file is to use standard labels, a TLBL or DLBL (or VOL and TPLAB or VOL and DLAB) statement(s) is required.

One EXTENT (XTENT) control statement must be used to define the limits of each area of a mass storage device onto which the output file is written. The symbolic name of the output unit must appear on this card.

SORT WORK FILE CONTROL STATEMENTS

The Sort program requires at least one mass storage unit or three tape units as an intermediate sort work file. The symbolic units to which this file is assigned must be consecutively numbered beginning with SYS001. Intermediate storage may be assigned on the following devices:

- IBM 2400 Magnetic Tape Units
- IBM 2311 Disk Storage Drive
- IBM 2314 Direct-Access Storage Device

Note: When variable-length or redefined-length records are being sorted, sort work files must not be assigned to 7-track tapes. 7-track tape work files can only be used to sort records whose keys are packed decimal or binary.

Device types may not be mixed; i.e., work units for a particular sort operation must all be of the same type.

If spanned records are being sorted and mass storage devices are being used as sort work files, it is the user's responsibility to assign these work files to devices whose track sizes are larger than the logical record sizes of the records being sorted. A spanned record that is larger than the available track size can be sorted by assigning the work files to magnetic tape.

If a work unit is to use standard labels, a TLBL or DLBL (or VOL and TPLAB or VOL and DLAB) control statement(s) is required. The filename entry on these statements must be SORTWK1 through SORTWKn. The symbolic unit names assigned to the work areas to be allocated (SYS001, SYS002, etc.) must appear on these cards.

One EXTENT (XTENT) control statement must be included to define each work area on a mass storage device. The total work area required may be divided into as many as eight extents, which would require eight EXTENT (XTENT) control statements. Symbolic unit names on these statements must be in consecutive order, (SYS001, SYS002, etc.).

Amount of Intermediate Storage Required

When intermediate storage is assigned on a mass storage unit, at least twice the amount required to hold all input records should be assigned. This area may consist of from one to eight extents, and the extents may be assigned on no more than eight devices.

If tape intermediate storage is used, at least the minumum number of units (three) must be assigned. The input file can be as large as the number of records that can be written on one full reel of tape. Assigning more than three intermediate storage tape drives does not increase the maximum input file size, but does improve performance.

Improving Performance

Performance increases significantly if 50K is available for execution of the Sort program. At the 100K level, the performance is very high. If no core is available, the Sort/Merge program will issue a message:

7054A "INSUFFICIENT CORE"

SORT DIAGNOSTIC MESSAGES

The messages generated by the Sort Feature are listed in the publication IBM System/360 Disk Operating System: Tape and Disk Sort/Merge, Form GC28-6676.

LINKAGE WITH THE SORT FEATURE

To initiate a sort operation, the COBOL object program includes the object time subroutine ILBDSRT0 and transfers control to it.

If the INPUT PROCEDURE option of the SORT statement is specified in the source program, exit E15 of the Sort/Merge program is used. At this exit, the record released by the user is passed to the Sort/Merge program. Since a dummy Input Procedure will be generated by the compiler when the USING option is specified, records in the USING file are also passed to the Sort/Merge program at exit E15.

If the OUTPUT PROCEDURE option of the SORT statement is specified, exit E35 of the Sort/Merge program is used. At this exit, the record returned by the Sort/Merge program is passed to the user. Since a dummy Output Procedure is generated by the compiler when the GIVING option is specified, records are also returned at exit E35 and written on this file.

Completion Codes

The Sort/Merge program returns a completion code upon termination and this code is stored in the COBOL special register SORT-RETURN. The codes are:

0 -- Successful completion of Sort/Merge

16 -- Unsuccessful completion of Sort/Merge

Successful Completion: When a Sort/Merge application has been successfully executed, a completion code of zero is returned and the sort operation terminates.

Unsuccessful Completion: If the Sort program encounters an error during execution that will not allow it to complete successfully, it returns a completion code of 16 and terminates. (A possible error is an uncorrectable input/output error.) The publication IBM System/360 Disk Operating System: Tape and Disk Sort/Merge, Form GC28-6676 contains a detailed description of the conditions under which this termination will occur.

The user may test the SORT-RETURN register for successful termination of the sort operation, as shown in the following example:

```
SORT SALES-RECORDS ON ASCENDING KEY,
CUSTOMER-NUMBER, DESCENDING KEY DATE,
USING FN-1, GIVING FN-2.  IF
SORT-RETURN NOT EQUAL TO ZERO, DISPLAY
"SORT UNSUCCESSFUL" UPON CONSOLE, STOP
RUN.
```

CHECKPOINT/RESTART DURING A SORT

The Checkpoint/Restart Feature is available to the programmer using the COBOL SORT statement. The programmer uses the RERUN clause to specify that checkpoints should be taken during program execution. The control statement requirements for taking a checkpoint are discussed in the chapter "Program Checkout."

The system-name specified in the RERUN clause as the sort checkpoint device must not be the same as any system-name used in the source language ASSIGN clause, but follows the same rules of formation.

The RERUN clause is fully described in the publication IBM System/360 Disk Operating System: American National Standard COBOL.

COBOL segmentation is a facility that provides a means of accomplishing object time overlay as a result of specifications made at the source language level. Segmentation will allow the programmer to divide the Procedure Division of a source program into sections. Through the use of a system of priority numbers, certain sections are designated as permanently resident in core storage and other sections as overlayable fixed segments and/or independent segments. Thus, a large program can be executed in a defined area of core storage by limiting the number of segments in the program that are permanently resident in core.

If there is a limit on the amount of core available, the program SAVECORE could be segmented as illustrated in Figure 23.

```
IDENTIFICATION DIVISION.

PROGRAM-ID. SAVECORE.
.
.
ENVIRONMENT DIVISION.

OBJECT-COMPUTER. IBM-360-50
     SEGMENT-LIMIT IS 15.
.
.
DATA DIVISION.
.
.
PROCEDURE DIVISION.
SECTION-1 SECTION 8.
.
.
SECTION-2 SECTION 8.
.
.
SECTION-3 SECTION 16.
.
.
SECTION-4 SECTION 8.
.
.
SECTION-5 SECTION 50.
.
.
SECTION-6 SECTION 16.
.
.
SECTION-7 SECTION 50.
.
.
```

Figure 23. Segmenting the Program SAVECORE

Assuming that 12K is available for the program SAVECORE, Figure 24 shows the manner in which core storage would be utilized. It is apparent from the illustration that SECTION-3, SECTION-6, and SECTION-7 cannot be in core at the same time, nor can SECTION-3, SECTION-5 and SECTION-7 be in core simultaneously.

Sections in the permanent segment (SECTION-1, SECTION-2, and SECTION-4) are those which must be available for reference at all times, or which are referenced frequently. They are distinguished here by the fact that they have been assigned priority numbers less than the segment limit.

Sections in the overlayable fixed segment are sections which are less frequently used. They are always made available in the state they were in when last used. They are distinguishable here by the fact that they have been assigned priority numbers greater than the segment limit but less than 49.

Sections in the independent segment can overlay, and be overlaid by, either an overlayable fixed segment or another independent segment. Independent segments are those assigned priority numbers greater than 49 and less than 100, and they are always given control in their initial state.

OPERATION

Execution of the object program begins in the root segment. The first segment in the permanent segment is considered the root segment. If the program does not contain a permanent segment, the compiler generates a dummy segment which will initiate the execution of the first overlayable or independent segment. All global tables, literals, and data areas are part of the root segment. Called object time subroutines are also part of the root segment. When CALL statements appear in a segmented program, subprograms are loaded with the fixed portion of the main program as if they had a priority of zero.

Segmented programs must not be called by another program (segmented or not segmented).

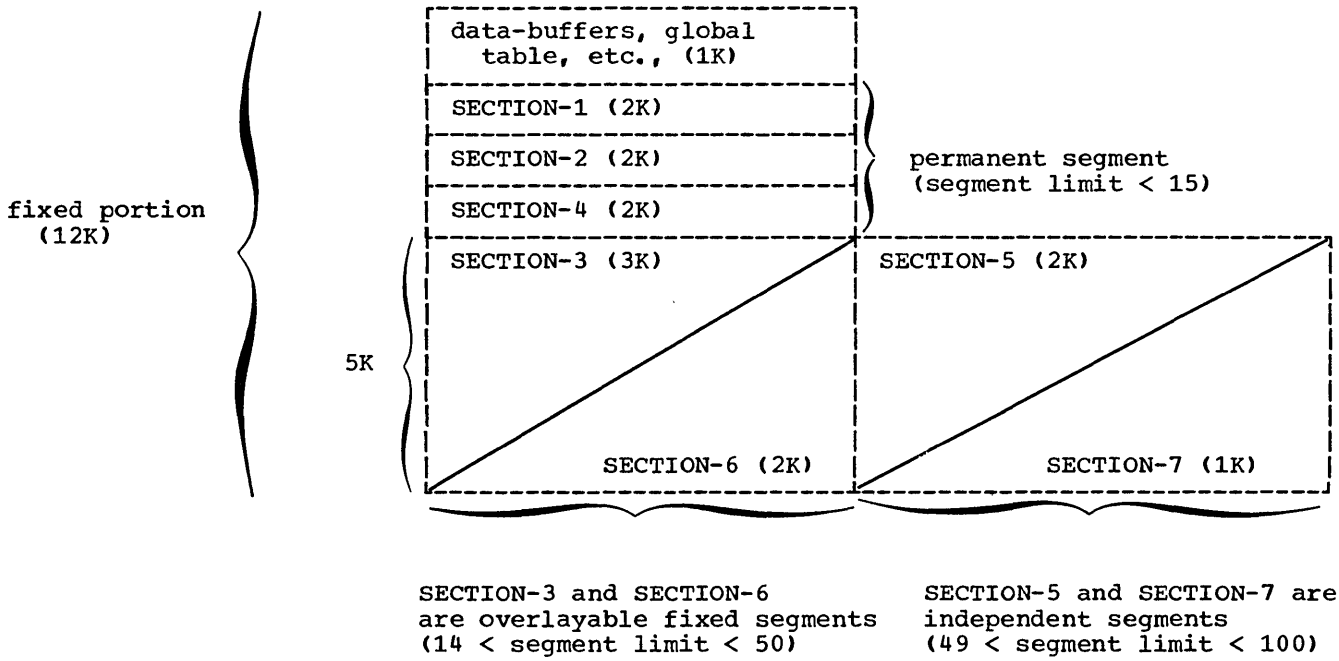


Figure 24. Storage Layout for SAVECORE

OUTPUT FROM A SEGMENTED PROGRAM

COMPILER OUTPUT

The output produced by the compiler is an overlay structure consisting of multiple object modules preceded by linkage editor control statements. Segments whose priority is greater than the segment limit (or 49, if no SEGMENT-LIMIT clause is specified) consist of executable instructions only.

The compiler generates each segment as a separate object module preceded by a PHASE card. The names appearing on these PHASE cards (segment-names) conform to the following naming conventions:

1. The name of the root segment is the same as the program-name specified in the PROGRAM-ID clause.

2. The name of each overlayable and independent segment is a combination of the program-name and the priority number of the segment. These names are formed according to the following rules:

   a. If the program-name is 6, 7, or 8 characters in length, the segment-name will consist of the first 6 characters of program-name plus the 2-character priority number.

   b. If the program-name is less than 6 characters in length, the priority-number is appended after the program-name.

   Note: Single digit priority-numbers are preceded by a zero.

The compiler generates an ENTRY card to the root segment as the last card input to the Linkage Editor.

Figure 25 is an illustration of the compiler output for the skeleton program shown in Figure 23.

```
PHASE SAVECORE,ROOT

 {object module for the root segment
  (sections with priority-numbers less
  than the segment limit) including any
  programs called by SAVECORE}


PHASE SAVEC016,*

 {object module for segments with a
  priority of 16 (two sections)}


PHASE SAVEC050,SAVEC016

 {object module for segments with a
  priority of 50 (two sections)}


ENTRY SAVECORE
```

Figure 25. Compiler Output for SAVECORE

LINKAGE EDITOR OUTPUT

Figure 26 is an illustration of the input to the Linkage Editor and the phase map produced by the Linkage Editor resulting from the compilation and editing of the segmented program BIGJOB. The following text is an explanation of the figure.

(1) PHASE card generated by the compiler for the root segment BIGJOB.

(2) AUTOLINK card for the Segmentation subroutine.

(3) PHASE cards generated by the compiler for segments of priority 10, 47-50, 60, 62, and 63.

(4) Control cards generated for the Sort Feature. These cards are explained in "Sort in a Segmented Program".

(5) ENTRY card generated by the compiler for the root phase.

(6) Location of the entry point CURSEGM. Item 6 is explained in "Determining the Priority of the Last Segment Loaded into the Transient Area".

(7) Load address of phase ILBDDUM0. Item 7 is explained in "Sort in a Segmented Program."

Cataloging a Segmented Program

When the CATAL option is used to catalog a segmented program, the following points should be observed:

1. To avoid duplicate names, the user must be aware of the naming conventions used by the compiler (see "Compiler Output") because a segment-name may be the same as a phase-name already existing in the core image library.

2. Since the PHASE card is generated by the compiler, the user must not specify a PHASE card for the program.

To invoke a previously cataloged segmented program, the user must use the following control statement:

```
// EXEC name
```

where name is the program-name specified in the PROGRAM-ID clause.

Determining the Priority of the Last Segment Loaded into the Transient Area

If a segmented program is abnormally terminated during execution, the priority of the last segment loaded into the transient area can be determined as follows:

1. In the map of main storage generated by the Linkage Editor, under the column LABEL, look for the name 'CURSEGM' (see item 6 in Figure 26).

2. Associated with this label, in the column LOADED, is an address.

3. At this location is stored the priority (one byte) of the segment current in the transient area. If this byte is X'00', no segment has been loaded into the transient area. This indicates that the error causing the dump occurred in the root segment.

SORT IN A SEGMENTED PROGRAM

If a segmented program contains a SORT statement, the sort program will be loaded above the largest overlayable or independent segment as shown in Figure 27.

The compiler accomplishes this by providing the following two control statements at the end of the overlay structure:

```
PHASE ILBDDUM0,transient area + L

INCLUDE ILBDDUM0
```

These cards are illustrated in Figure 26, item 4. The value of "L" in the figure is X'2F2' which is the length of the longest segment, BIGJOB47, rounded to the next halfword boundary. Note that Linkage Editor relocates the phase ILBDDUM0 to the next doubleword boundary (see Figure 26, item 7).

```
JOB  BIGJ                  DISK LINKAGE EDITOR DIAGNOSTIC OF INPUT


ACTION TAKEN      MAP

LIST   PHASE  BIGJOB,ROOT      (1)
.
.
.
LIST   AUTOLINK     ILBDSEM0  (2)
LIST   AUTOLINK     ILBDSRT0
.
.
LIST   PHASE BIGJOB10,*          )
LIST   PHASE BIGJOB47,BIGJOB10   )
LIST   PHASE BIGJOB48,BIGJOB47   )
LIST   PHASE BIGJOB49,BIGJOB48   ) (3)
LIST   PHASE BIGJOB50,BIGJOB49   )
LIST   PHASE BIGJOB60,BIGJOB50   )
LIST   PHASE BIGJOB62,BIGJOB60   )
LIST   PHASE BIGJOB63,BIGJOB62   )
LIST   PHASE ILBDDUM0,BIGJOB63+X'002F2' } (4)
LIST   INCLUDE ILBDDUM0
LIST   ENTRY BIGJOB     (5)
```

```
       PHASE    XFR-AD  LOCORE  HICORE  DSK-AD    ESD TYPE  LABEL     LOADED  REL-FR

ROOT   BIGJOB   003000  003000  0075A3  64 04 1   CSECT     BIGJOB    003000  003000
                                                    .
                                                    .
                                                    .
                                                  CSECT     ILBDSEM0  006268  006268
                                                  * ENTRY   CURSEGM   00637D          (6)
                                                  CSECT     ILBDSRT0  006B38  006B38
                                                    .
                                                    .
                                                    .
       BIGJOB10 0075A8  0075A8  0075E9  64 09 2   CSECT     BIGJOB10  0075A8  0075A8
       BIGJOB47 0075A8  0075A8  007899  65 00 1   CSECT     BIGJOB47  0075A8  0075A8
       BIGJOB48 0075A8  0075A8  0075DB  65 00 2   CSECT     BIGJOB48  0075A8  0075A8
       BIGJOB49 0075A8  0075A8  0075D3  65 01 1   CSECT     BIGJOB49  0075A8  0075A8
       BIGJOB50 0075A8  0075A8  0075F1  65 01 2   CSECT     BIGJOB50  0075A8  0075A8
       BIGJOB60 0075A8  0075A8  0076ED  65 02 1   CSECT     BIGJOB60  0075A8  0075A8
       BIGJOB62 0075A8  0075A8  0075D1  65 02 2   CSECT     BIGJOB62  0075A8  0075A8
       BIGJOB63 0075A8  0075A8  007621  65 03 1   CSECT     BIGJOB63  0075A8  0075A8
       ILBDDUM0 0078A0  0078A0  0078A1  65 03 2   CSECT     ILBDDUM0  0078A0  0078A0 (7)
```

Figure 26. Linkage Editing a Segmented Program

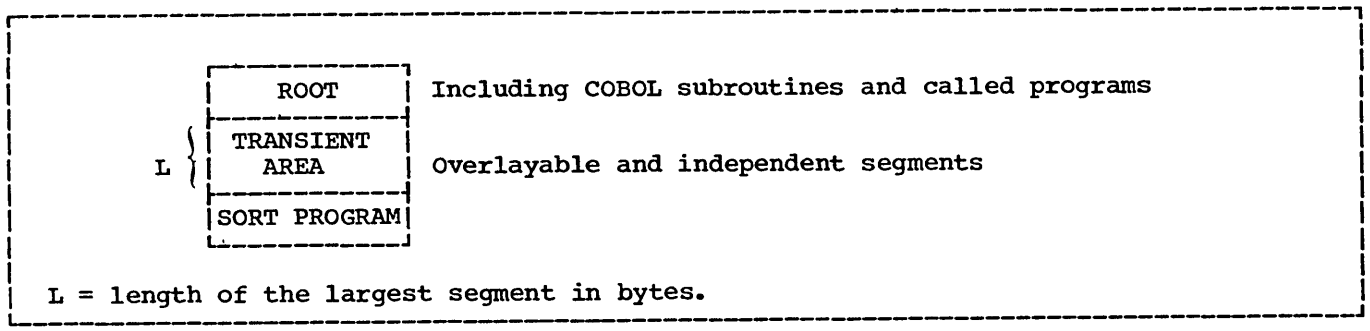


Figure 27. Location of Sort Program in a Segmentation Structure

- PROCESSING COBOL FILES ON MASS STORAGE DEVICES
- ADVANCED PROCESSING CAPABILITIES
- RECORD FORMATS
- PROGRAMMING TECHNIQUES

A mass storage device is one on which records can be stored in such a way that the location of any one record can be determined without extensive searching. Records can be accessed directly rather than serially.

The recording surface of a mass storage device is divided into many tracks. A track is defined as a circumference of the recording surface. The number of tracks per recording surface and the capacity of a track for each device are shown in Table 8.

Table 8. Recording Capacities of Mass Storage Devices

Device	Capacity
2311	200 tracks per surface; 3625 bytes per track.
2314	200 tracks per surface; 7294 bytes per track.
2321	100 tracks per strip; 2000 bytes per track.

Each device has some type of access mechanism through which data is transferred to and from the device. The mechanisms are different for each device, but each mechanism contains a number of read/write heads that transfer data as the recording surfaces rotate past them. Only one head can transfer data (either reading or writing) at a time.

FILE ORGANIZATION

Records in a file must be logically organized so that they can be retrieved efficiently for processing. Three methods of organization for mass storage devices are supported by the Disk Operating System American National Standard COBOL compiler: sequential, direct, and indexed.

SEQUENTIAL ORGANIZATION

In a sequential file, records are organized solely on the basis of their successive physical location in the file. The records are read or updated in the same order in which they appear.

Individual records cannot be located quickly. Records usually cannot be deleted or added unless the entire file is rewritten. This organization is used when most of the records in the file are processed each time the file is used.

DIRECT ORGANIZATION

A file with direct organization is characterized by some predictable relationship between the key of a record and the address of that record on a mass storage device. This relationship is established by the user.

Direct organization is generally used for files where the time required to locate individual records must be kept to an absolute minimum, or for files whose characteristics do not permit the use of sequential or indexed organization.

This organization method has considerable flexibility. The accompanying disadvantage is that although the Disk Operating System provides the routines to read or write a file of this type, the user is largely responsible for the logic and programming required to locate the key of a record and its address on a mass storage device.

INDEXED ORGANIZATION

An indexed file is similar to a sequential file in that rapid sequential processing is possible. The indexes associated with an indexed file also allow quick retrieval of individual records through random access. Moreover, a separate area of the file is set aside for additions; this eliminates the need to rewrite the entire file when adding records, a process that would usually be necessary with a sequentially organized file. Although the added records are not physically in key sequence, the indexes are constructed in such a way that the added records can be quickly retrieved in key sequence, thus making rapid sequential access possible.

In this method of organization, the Disk Operating System has control over the location of the individual records. Since the characteristics of the file are known, most of the mechanics of locating a particular record are handled by the system.

DATA MANAGEMENT CONCEPTS

The data management facilities of the Disk Operating System are provided by a group of routines that are collectively referred to as the Input/Output Control System (IOCS). A distinction is made between two types of routines:

1. Physical IOCS (PIOCS) -- the physical input/output routines included in the Supervisor. PIOCS is used by all programs run within the system. It includes facilities for scheduling input/output operations, checking for and handling error conditions related to input/output devices, and handling input/output interruptions to maintain maximum input/output speeds without burdening the user's problem program.

2. Logical IOCS (LIOCS) -- the logical input/output routines linked with the user's problem program. These routines provide an interface between the user's file processing routines and the PIOCS routines.

   LIOCS performs those functions that a programmer needs to locate and access a logical record for processing. A logical record is one unit of information in a file of similar units, for example, one employee's record in a master payroll file, one part-number record in an inventory file, or one customer account record in an account file. One or more logical records may be included in one physical record. LIOCS refers to the routines that perform the following functions:

   a. Blocking and deblocking records

   b. Switching between input/output areas when two areas are specified for a file

   c. Handling end-of-file and end-of-volume conditions

   d. Checking and writing labels

A brief description of functions performed by LIOCS and their relationship to a COBOL program follows.

Whenever COBOL imperative-statements (READ, WRITE, REWRITE, etc.) are used in a program to control the input/output of records in a file, that file must be defined by a DTF (Define The File). A DTF is created for each file opened in a COBOL program from information specified in the Environment Division, FD entry, and input/output statements in the source program. The DTF for each file is part of the object module that is generated by the compiler. It describes the characteristics of the logical file, indicates the type of processing to be used for the file, and specifies the main storage areas and routines used for the file.

One of the constants in the DTF table is the address of a logic module that is to be used at execution time to process that file. A logic module contains the coding necessary to perform data management functions required by the file such as blocking and deblocking, initiating label checking, etc.

Generally, these logic modules are assembled separately and cataloged in the relocatable library under a standard name. At linkage editing time, the Linkage Editor searches the relocatable library using the virtual reference to locate the logic module. The logic module is then included as part of the program phase. Note that since the Autolink feature of the Linkage Editor is responsible for including the logic modules, the COBOL programmer need not specify any INCLUDE statements.

The type of DTF table prepared by the compiler depends on the organization of the file and the device to which it is assigned. The DTF's used for processing files assigned to mass storage devices are as follows:

DTFSD -- Sequential organization, sequential access

DTFDA -- Direct organization, sequential or random access

DTFIS -- Indexed organization, sequential or random access

The remainder of this chapter provides information about preparing programs which process files assigned to mass storage devices. Included are general descriptions of the organization, the COBOL statements that must be specified in order to build the correct DTF tables, and coding examples.

SEQUENTIAL ORGANIZATION (DTFSD)

In a sequential file on a mass storage device, records are written one after another -- track by track, cylinder by cylinder -- at successively higher addresses.

Records may be fixed-length, spanned, or variable-length, blocked or unblocked, or undefined. Since the file is always accessed sequentially, it is not formatted with keys.

Processing a sequentially organized file for selected records is inefficient. If it is done infrequently, the time spent in locating the records is not significant. The slowest way is to read the records sequentially until the desired one is located. On the average, half of the file must be read to locate one record.

Additions and deletions require a complete rewrite of a sequentially organized file on a mass storage device. Sequential organization is used on mass storage devices primarily for tables and intermediate storage rather than for master files.

Sequentially organized files formatted with keys cannot be created using DTFSD. DTFDA may be used to create and access (sequentially or randomly) such files.

PROCESSING A SEQUENTIALLY ORGANIZED FILE

To create, retrieve, or update a DTFSD file, the following specifications should be made in the source program:

ENVIRONMENT DIVISION

Required clauses:

SELECT [OPTIONAL] file-name

ASSIGN TO SYSnnn- {UT / DA} - {2311 / 2314 / 2321} -S

Optional clauses:

RESERVE Clause
FILE-LIMIT Clause
ACCESS MODE IS SEQUENTIAL
PROCESSING MODE IS SEQUENTIAL
RERUN Clause
SAME Clause
APPLY WRITE-ONLY Clause (create only)
APPLY WRITE-VERIFY Clause (create or update only)

Invalid clauses:

ACCESS MODE IS RANDOM
ACTUAL KEY Clause
NOMINAL KEY Clause
RECORD KEY Clause
TRACK AREA Clause
MULTIPLE FILE TAPE Clause
APPLY EXTENDED-SEARCH Clause
APPLY CYL-OVERFLOW Clause

APPLY {MASTER-INDEX / CYL-INDEX} Clause

APPLY CORE-INDEX Clause

DTFSD files may be opened as INPUT, OUTPUT, or I-O. When creating such a file, an INVALID KEY condition occurs when the file limit has been reached and an attempt is made to place another record on the mass storage device. The file limit is determined from the XTENT or EXTENT control statements.

When a DTFSD file is opened as OUTPUT, each WRITE statement signifies the creation of a new record. When opened as I-O, each WRITE statement signifies that the record just read is to be rewritten.

DIRECT ORGANIZATION (DTFDA)

With direct organization, there is a definite relationship beteween the key of a record and its address. This relationship permits rapid access to any record if the file is carefully organized. The user develops a record address that ranges from zero to some maximum by converting a particular field in each record to a track address. Each byte in the address is a binary number. To reference a particular record, the user must supply both the track address and the identifier that makes each record unique on its track. Both the track address and the identifier are supplied by the user in the ACTUAL KEY clause. This will be discussed in detail later in this chapter.

With direct organization, records may be fixed length, spanned or undefined. The records must be unblocked. R0 (record zero) of each track is used as a capacity record. It contains the address of the last record written on the track, and is used by the system to determine whether a new record will fit on the track. The capacity records are updated by the system as records are added to the file. The capacity records do not account for deletions: as far as the system is

concerned, once a track is full it remains full (even if the user deletes records) until the file is reorganized.

Often, more records are converted to a given track address than will actually fit on the track. These surplus records are known as overflow records and are usually written into a separate area known as an overflow area.

As already noted, the user has an unlimited choice in deciding where records are to be located in a directly organized file. The logic and programming are his responsibility.

When creating or making additions to the file, the user must specify the location for a record (track address) and the identifier that makes each record on the track unique. If there is space on the track, the system writes the record and updates the capacity record. If the specified track is full, a standard error condition occurs, and the user may specify another track address in his USE AFTER STANDARD ERROR declarative routine.

In the case of one maximum size record per track (when spanned records are not specified), the data length plus the length of the symbolic key cannot exceed the following values:

2311 -- 3605 bytes
2314 -- 7249 bytes
2321 -- 1984 bytes

When reading or updating the file, the user must supply the track address and the unique identifier on the track for the specific record being sought. The system locates the track and searches that track for the record with the specified identifier. If the record is not found, COBOL indicates this to the user by raising an INVALID KEY condition. Only the track specified by the user is searched. If, however, the APPLY EXTENDED-SEARCH clause has been specified for the file, the entire cylinder is searched for the desired record. In this case, the INVALID KEY condition arises only if the record cannot be found on the cylinder. To ensure file integrity, the upper limit of each extent of a file using EXTENDED-SEARCH must be the last track of a cylinder.

Error recovery from a DTFDA file is described in detail in the chapter "Advanced Processing Capabilities."

ACCESSING A DIRECTLY ORGANIZED FILE

A directly organized file (DTFDA) may be accessed either sequentially or randomly.

ACCESSING A DIRECTLY ORGANIZED FILE SEQUENTIALLY: When reading a direct file sequentially, records are retrieved in logical sequence; this logical sequence corresponds exactly to the physical sequence of the records. To retrieve a DTFDA file sequentially, the following specifications are made in the source program:

ENVIRONMENT DIVISION

Required clauses:

SELECT [OPTIONAL] file-name

ASSIGN TO SYSnnn-DA- {2311 / 2321 / 2314} -{A / D}

Optional clauses:

FILE-LIMIT Clause
ACCESS MODE IS SEQUENTIAL
PROCESSING MODE IS SEQUENTIAL
ACTUAL KEY Clause
RERUN Clause
SAME Clause

Invalid clauses:

RESERVE Clause
ACCESS MODE IS RANDOM
NOMINAL KEY Clause
RECORD KEY Clause
TRACK-AREA Clause
MULTIPLE FILE TAPE Clause
APPLY WRITE-ONLY Clause
APPLY CYL-OVERFLOW Clause
APPLY EXTENDED- SEARCH Clause
APPLY WRITE-VERIFY Clause

APPLY {MASTER-INDEX / CYL-INDEX} Clause

APPLY CORE-INDEX Clause

When DTFDA records are retrieved sequentially, the file may be opened only as INPUT. The AT END condition occurs when the last record has been read and execution of another READ is attempted.

Note that in the ASSIGN clause, an A must be specified for files with actual track addressing, and a D must be specified for files with relative track addressing.

ACCESSING A DIRECTLY ORGANIZED FILE RANDOMLY: To create a directly organized file randomly, the following

specifications are made in the source program:

ENVIRONMENT DIVISION

Required clauses:

SELECT file-name

ASSIGN TO SYSnnn-DA- {2311 / 2321 / 2314} - {A / D}

ACCESS MODE IS RANDOM
ACTUAL KEY Clause

Optional clauses:

FILE-LIMIT Clause
PROCESSING MODE IS SEQUENTIAL
RERUN Clause
SAME Clause
APPLY WRITE-VERIFY Clause

Invalid clauses:

RESERVE Clause
ACCESS MODE IS SEQUENTIAL
NOMINAL KEY Clause
RECORD KEY Clause
TRACK-AREA Clause
MULTIPLE FILE TAPE Clause
APPLY WRITE-ONLY Clause
APPLY EXTENDED-SEARCH Clause
APPLY WRITE-VERIFY Clause
APPLY CYL-OVERFLOW Clause

APPLY {MASTER-INDEX / CYL-INDEX} Clause

APPLY CORE-INDEX Clause

Note that in the ASSIGN clause, an A must be specified for files with actual track addressing, and a D must be specified for files with relative track addressing.

To retrieve or update a directly organized file randomly, the following specifications must be made in the source program.

ENVIRONMENT DIVISION

Required clauses:

SELECT file-name

ASSIGN TO SYSnnn-DA- {2311 / 2314 / 2321} - {A / D / U / W}

ACCESS MODE IS RANDOM
ACTUAL KEY Clause

Note that in the ASSIGN clause an A must be specified for files with actual track addressing, a D must be specified for files with relative track addressing, a U must be specified for files with actual track addressing when the REWRITE statement is used, and W must be specified for files with relative track addressing when the REWRITE statement is used.

The optional and invalid clauses are the same as those specified previously for creating a directly organized file.

Exception: APPLY EXTENDED-SEARCH is optional when retrieving or updating a directly organized file randomly.

ACTUAL KEY CLAUSE

Note that the ACTUAL KEY clause is required for DTFDA files when ACCESS IS RANDOM, is optional for DTFDA files when ACCESS IS SEQUENTIAL, and is not used for DTFSD files.

The actual key consists of two components. One component expresses the track address at which the record is to be placed for an output operation, or at which the search is to begin for an input operation. The track address can be expressed either as an actual address or as a relative address, depending upon the addressing scheme chosen when the file was created. The other component is associated with the record itself and serves as its unique identifier. The structures of both actual keys are shown in Figure 28.

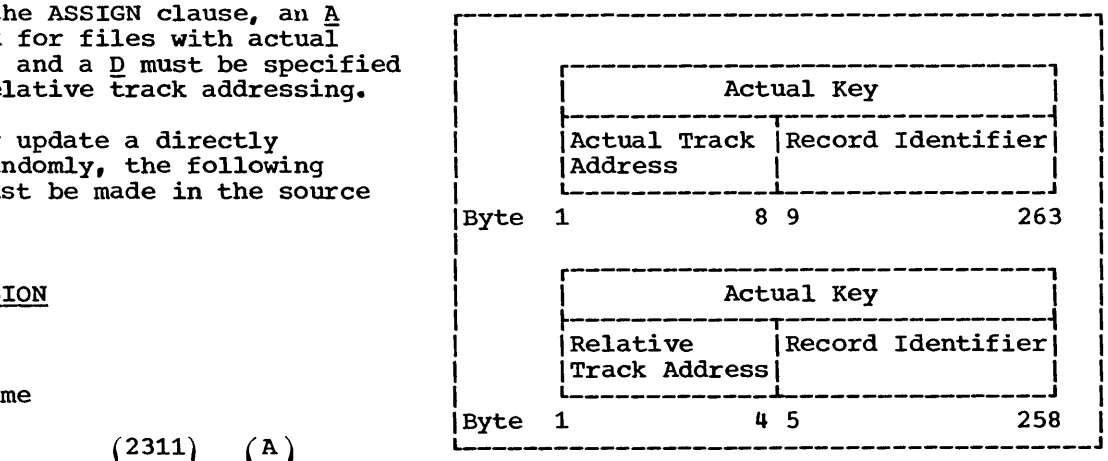


Figure 28. Structures of the Actual Key

The format of the ACTUAL KEY clause is:

ACTUAL KEY IS data-name

	Pack	Cell		Cylinder		Head		Record
	M	B	B	C	C	H	H	R
Byte / Device	0	1	2	3	4	5	6	7
2311	0-221	0	0	0	0-199	0	0-9	0-255
2314	0-221	0	0	0	0-199	0	0-19	0-255
2321	0-221	0	0-9	0-19	0-9	0-4	0-19	0-255

Figure 29. Permissible Specifications for the First Eight Bytes of the Actual Key

When actual track addressing is used, data-name may be any fixed item from 9 through 263 bytes in length. It must be defined in the Working-Storage, File,. or Linkage Section. The first eight bytes are used to specify the actual track address. The structure of these bytes and permissible specifications for the 2311, 2314, and 2321 mass storage devices are shown in Figure 29. The user may select from 1 to 255 bytes for the record identifier portion of the actual key field.

When relative track addressing is used, data-name may be any fixed item from 5 through 258 bytes in length. It must be defined in the File Section, the Working-Storage Section, or the Linkage Section. The first four bytes of data-name are the track identifier. The identifier is used to specify the relative track address for the record and must be defined as an 8-integer binary data item whose maximum value does not exceed 16,777,215. The remainder of data-name, which is 1 through 254 bytes in length, is the record identifier. It represents the symbolic portion of the key field used to identify a particular record on a track.

For a complete discussion of the ACTUAL KEY clause, see the publication IBM System/360 Disk Operating System: American National Standard COBOL.

Randomizing Techniques

One method of determining the value of the track address portion of the field defined in the ACTUAL KEY clause is referred to as indirect addressing. Indirect addressing generally is used when the range of keys for a file includes a high percentage of unused values. For example, employee numbers may range from 000001 to 009999, but only 3000 of the possible 9999 numbers are currently assigned. Indirect addressing is also used for nonnumeric keys. Key, in this discussion, refers to that field of the record being written that will be converted to the track address portion.

Indirect addressing signifies that the key is converted to a value for the actual track address by using some algorithm intended to limit the range of addresses. Such an algorithm is called a randomizing technique. Randomizing techniques need not produce a unique address for every record and, in fact, such techniques usually produce synonyms. Synonyms are records whose keys randomize to the same address.

Two objectives must be considered in selecting a randomizing technique:

1. Every possible key in the file must randomize to an address within the designated range.
2. The addresses should be distributed evenly across the range so that there are as few synonyms as possible.

Note that one way to minimize synonyms is to allocate more space for the file than is actually required to contain all the records. For example, the percentage of locations that are actually used might be 80% to 85% of the allocated space.

When actual track addressing is used, the first eight bytes of the ACTUAL KEY field can be thought of as a "discontinuous binary address." This is significant to the programmer because he must keep two considerations in mind. First, the cylinder and head number must be in binary notation, so the results of the randomizing formula must be in binary format. Second, the address is "discontinuous" since a mathematical overflow from one element (e.g., head number) does not increment the adjacent element (e.g., cylinder number).

DIVISION/REMAINDER METHOD: One of the simplest ways to indirectly address a directly organized file is by using the division/remainder method. (For a discussion of other randomizing techniques, see the publication Introduction to IBM System/360 Direct Access Storage Devices and Organization Methods, Form GC20-1649.)

1. Determine the amount of locations required to contain the data file. Include a packing factor for additional space to eliminate synonyms. The packing factor should be approximately 20% of the total space allocated to contain the data file.

2. Select, from the prime number table, the nearest prime number that is less than the total of step 1. A prime number is a number divisible only by itself and the integer 1. Table 9 is a partial list of prime numbers.

3. Clear any zones from the first eight bytes of the actual key field. This can be accomplished by moving the key to a field described as COMPUTATIONAL.

4. Divide the key by the prime number selected.

5. Ignore the quotient; utilize the remainder as the relative location within the data file.

6. (For actual track addressing only) Locate the beginning of the space available and manipulate the relative address, to the actual device address if necessary.

For example, assume that a company is planning to create an inventory file on a 2311 disk storage device. There are 8000 different inventory parts, each identified by an 8-character part number. Using a 20% packing factor, 10,000 record positions are allocated to store the data file.

Method A: The closest prime number to 10,000, but under 10,000, is 9973. Using one inventory part number as an example, in this case #25DF3514, and clearing the zones we have 25463514. Dividing by 9973 we get a quotient of 2553 and a remainder of 2445. 2445 is the relative location of the record within the data file corresponding to part number 25DF3514. The record address can be determined from the relative location as follows:

1. (For actual track addressing only) Determine the beginning point for the data file (e.g., cylinder 100, track 0).

2. Determine the number of records that can be stored on a track (e.g., twelve per track on a 2311 disk pack, assuming each inventory record is 200 bytes long).

   Because each data record contains non-data components, such as a count area and interrecord gaps, track capacity for data storage will vary with record length. As the number of separate records on a track increases, interrecord gaps occupy additional byte positions so that data capacity is reduced. Track capacity formulas provide the means to determine total byte requirements for records of various sizes on a track. These formulas can be found in the publications IBM System/360 Component Descriptions, Forms GA26-5988 and GA26-3599.

3. Divide the relative number (2445) by the number of records to be stored on each track.

4. (For actual track addressing only) The result, quotient = 203, is now divided into cylinder and head designation. Since the 2311 disk pack has ten heads, the quotient of 203 is divided by 10 to show:

   Cylinder or CC = 20
   Head or HH = 03 (high-order zero added)

4B. (For relative track addressing only) The result, quotient = 203, now becomes the track identifier of the actual key.

Method B: Utilizing the same example, another approach will also provide the relative track address:

1. The number of records that may be contained on one track is twelve. Therefore, if 10,000 record locations are to be provided, 834 tracks must be reserved.

2. The prime number nearest, but less than 834, is 829.

3. Divide the zone-stripped key by the prime value. (In the example, 25463514 divided by 829 provides a quotient of 30715 and a remainder of 779. The remainder is the relative address.)

Table 9. Partial List of Prime Numbers (Part 1 of 2)

A (Number)	B (Nearest Prime Number Less Than A)
500	499
600	599
700	691
800	797
900	887
1000	997
1100	1097
1200	1193
1300	1297
1400	1399
1500	1499
1600	1597
1700	1699
1800	1789
1900	1889
2000	1999
2100	2099
2200	2179
2300	2297
2400	2399
2500	2477
2600	2593
2700	2699
2800	2797
2900	2897
3000	2999
3100	3089
3200	3191
3300	3299
3400	3391
3500	3499
3600	3593
3700	3697
3800	3797
3900	3889
4000	3989
4100	4099
4200	4177
4300	4297
4400	4397
4500	4493
4600	4597
4700	4691
4800	4799
4900	4889
5000	4999
5100	5099
5200	4197
5300	5297
5400	4399
5500	5483

Table 9. Partial List of Prime Numbers (Part 2 of 2)

A (Number)	B (Nearest Prime Number Less Than A)
5600	5591
5700	5693
5800	5791
5900	5897
6000	5987
6100	6091
6200	6199
6300	6299
6400	6397
6500	6491
6600	6599
6700	6691
6800	6793
6900	6899
7000	6997
7100	7079
7200	7193
7300	7297
7400	7393
7500	7499
7600	7591
7700	7699
7800	7793
7900	7883
8000	7993
8100	8093
8200	8191
8300	8297
8400	8389
8500	8467
8600	8599
8700	8699
8800	8793
8900	8899
9000	8899
9100	9091
9200	9199
9300	9293
9400	9397
9500	9497
9600	9587
9700	9697
9800	9791
9900	9887
10,000	9973
10,100	10,099
10,200	10,193
10,300	10,289
10,400	10,399
10,500	10,499
10,600	10,597

4. (For actual track addressing only) To convert the relative address to an actual device address, divide the relative address by the number of tracks in a cylinder. The quotient will provide the cylinder number and the remainder will be the track number. For example, the 2311 disk pack would utilize 779 as:

   Cylinder or CC = 77
   Track or HH = 9

Figure 30 is a sample COBOL program which creates a direct file with actual track addressing using Method B and provides for the possibility of synonym overflow. Synonym overflow will occur if a record randomizes to a track that is already full. The following description highlights the features of the example. Circled numbers on the program listing correspond to the numbers in the text.

(1) The value 10 is added to TRACK-1 to ensure that the user program does not write on cylinder 0. Cylinder 0 must be reserved for the Volume Table of Contents.

- Since the prime number used as a divisor is 829, the largest possible remainder will be 828. Adding 10 to TRACK-1 adjusts the largest possible remainder to 838.

(2) If synonym overflow occurs, control is given to the error procedure declarative specified in the first section of the Procedure Division. The declarative provides that:

- Any record which cannot fit on a track (i.e., tracks 0 through 8 of any cylinder) will be written in the first available position on the following track(s).
- Any record which cannot fit within a single cylinder will be written on cylinder 84 (i.e., the cylinder overflow area).
- If a record cannot fit on either cylinders 1 through 83, or on cylinder 84, the job is terminated.

(3) The standard error condition "no room found" is tested before control is given to the synonym routine. Other standard error conditions as well as invalid key conditions result in job termination.

ERROR-COND is the identifier which specifies the error condition that caused control to be given to the error declarative. ERROR-COND is printed on SYSLST whenever the error declarative section is entered. TRACK-ID and C-REC are also printed on SYSLST. They are printed before the execution of each WRITE statement. This output has been provided in order to facilitate an understanding of the logic involved in the creation of D-FILE.

(4) The first twelve records which randomize to cylinder 002 track 8 are actually written on track 8.

(5) The next twelve records which randomize to cylinder 002 track 8 are adjusted by the SYNONYM-ROUTINE and written on cylinder 002 track 9.

(6) The next twelve records which randomize to cylinder 002 track 8 are adjusted by the SYNONYM-ROUTINE and written on cylinder 84 track 0 (i.e., the overflow cylinder).

(7) The last two records which randomize to cylinder 002 track 8 are adjusted by the SYNONYM-ROUTINE and written on cylinder 84 track 1 (i.e., the overflow cylinder).

```
// JOB  METHODB
// OPTION NODECK,LINK,LIST,LISTX,SYM,ERRS
// EXEC FCOBOL
```

```
       IDENTIFICATION DIVISION.
       PROGRAM-ID. METHOD-B.
       ENVIRONMENT DIVISION.
       CONFIGURATION SECTION.
       SOURCE-COMPUTER. IBM-360.
       OBJECT-COMPUTER. IBM-360.
       INPUT-OUTPUT SECTION.
       FILE-CONTROL.
           SELECT D-FILE ASSIGN SYS015-DA-2311-A-MASTER
           ACCESS IS RANDOM
           ACTUAL KEY IS ACT-KEY.
           SELECT C-FILE ASSIGN TO SYS007-UR-2540R-S.
       DATA DIVISION.
       FILE SECTION.
       FD  D-FILE
           LABEL RECORDS ARE STANDARD.
       01  D-REC.
           02  PART-NUM PIC X(8).
           02  NUM-ON-HAND PIC 9(4).
           02  PRICE PIC 9(5)V99.
           02  FILLER PIC X(181).
       FD  C-FILE
           LABEL RECORDS ARE OMITTED.
       01  C-REC.
           02  PART-NUM PIC X(8).
           02  NUM-ON-HAND  PIC 9(4).
           02  PRICE PIC 9(5)V99.
       WORKING-STORAGE SECTION.
       77  HD    PIC 9 VALUE ZERO.
       77  SAVE    PIC S9(8) COMP SYNC.
       77  QUOTIENT    PIC S9(5) COMP SYNC.
       01  ERROR-COND.
           02  FILLER PIC 99  VALUE ZERO.
           02  ERR    PIC 9  VALUE ZERO.
           02  FILLER PIC 9(5)  VALUE ZERO.
       01  TRACK-1     PIC 9999.
       01  TRACK-ID REDEFINES TRACK-1.
           02  CYL PIC 999.
           02  HEAD PIC 9.
       01  KEY-1.
           02  M   PIC S999 COMP SYNC VALUE ZEROES.
           02  BB  PIC S9  COMP SYNC VALUE ZERO.
           02  CC  PIC S999 COMP SYNC.
           02  HH  PIC S9  COMP SYNC.
           02  R   PIC X   VALUE LOW-VALUE.
           02  REC-ID  PIC X(8).
       01  KEY-2 REDEFINES KEY-1.
           02  FILLER PIC X.
           02  ACT-KEY PIC X(16).
```

Figure 30. Creating a Direct File Using Method B (Part 1 of 4)

```
PROCEDURE DIVISION.
DECLARATIVES.
ERROR-PROCEDURE SECTION. USE AFTER STANDARD ERROR PROCEDURE
    ON D-FILE GIVING ERROR-COND.
ERROR-ROUTINE.
    EXHIBIT NAMED ERROR-COND.
    IF ERR = 1 GO TO SYNONYM-ROUTINE    ELSE
        DISPLAY 'CTHER STANDARD ERROR  ' REC-ID
        GO TO EOJ.
SYNONYM-ROUTINE.
    IF CC = 84 AND HD = 9  DISPLAY 'OVERFLCW AREA FULL'
        GO TO EOJ.
    IF CC = 84 ADD 1 TO HD  GO TO ADJUST-HD.
    IF HH = 9 GO TO END-CYLINDER.
    ADD 1 TO HH.
    GO TO WRITES.
END-CYLINDER.
    MOVE 84 TO CC.
ADJUST-HD.
    MOVE HD TO HH.
    GO TO WRITES.
END DECLARATIVES.
FILE-CREATION SECTION.
    OPEN INPUT C-FILE
        OUTPUT D-FILE.
READS.
    READ C-FILE AT END GO TO EOJ.
    MOVE CORRESPONDING C-REC TO D-REC.
    MOVE PART-NUM OF C-REC TO REC-ID SAVE.
    DIVIDE SAVE BY 829 GIVING QUOTIENT REMAINDER TRACK-1.
    ADD 10 TO TRACK-1.
    MOVE CYL TO CC.
    MOVE HEAD TO HH.
WRITES.
    EXHIBIT NAMED TRACK-ID  C-REC  CC  HH.
    WRITE D-REC INVALID KEY GO TO INVALID-KEY.
    GO TO READS.
INVALID-KEY.
    DISPLAY 'INVALID KEY  '  REC-ID.
EOJ.
    CLOSE C-FILE D-FILE.
    STOP RUN.
```

① ② ③

```
// LBLTYP NSD(01)
// EXEC LNKEDT
```

Figure 30. Creating a Direct File Using Method B (Part 2 of 4)

```
// ASSGN SYSCC7,X'00C'
// ASSGN SYS015,X'192'
// DLBL MASTER,,50/CC1,DA
// EXTENT SYS015,111111,1,0,10,840
'/ EXEC
```

```
TRACK-ID = 0010 C-REC = 829CCCCC          CC = 001 HH = 0
TRACK-ID = 0011 C-REC = 8290C001          CC = 001 HH = 1
TRACK-ID = 0011 C-REC = 82900001X         CC = 001 HH = 1
TRACK-ID = 0028 C-REC = 8290CC18C1        CC = 002 HH = 8
TRACK-ID = 0028 C-REC = 8290C018C2        CC = 002 HH = 8
TRACK-ID = 0028 C-REC = 8290CC18C3        CC = 002 HH = 8
TRACK-ID = 0028 C-REC = 8290001804        CC = 002 HH = 8
TRACK-ID = 0028 C-REC = 8290CC18C5        CC = 002 HH = 8
TRACK-ID = 0028 C-REC = 8290C018C6        CC = C02 HH = 8   (4)
TRACK-ID = 0028 C-REC = 8290CC18C7        CC = 002 HH = 8
TRACK-ID = 0028 C-REC = 8290CC18C8        CC = C02 HH = 8
TRACK-ID = 0028 C-REC = 8290C01809        CC = 002 HH = 8
TRACK-ID = 0028 C-REC = 8290CC1810        CC = CC2 HH = 8
TRACK-ID = 0028 C-REC = 8290CC1811        CC = 002 HH = 8
TRACK-ID = 0028 C-REC = 8290CC1812        CC = C02 HH = 8
TRACK-ID = 0C28 C-REC = 8290CC1813        CC = 002 HH = 8
ERROR-COND = C0100000
TRACK-ID = 0028 C-REC = 8290CC1813        CC = 002 HH = 9
TRACK-ID = 0028 C-REC = 8290CC1814        CC = 002 HH = 8
ERROR-COND = CC1CCCC0
TRACK-ID = C028 C-REC = 8290CC1814        CC = 002 HH = 9
TRACK-ID = 0028 C-REC = 8290001815        CC = C02 HH = 8
ERROR-COND = C01C0000
TRACK-ID = 0028 C-REC = 8290CC1815        CC = CC2 HH = 9
TRACK-ID = 0028 C-REC = 8290001816        CC = 002 HH = 8
ERROR-COND = C01CCCC0
TRACK-ID = 0028 C-REC = 8290CC1816        CC = CC2 HH = 9
TRACK-ID = 0028 C-REC = 8290001817        CC = 002 HH = 8
ERROR-COND = C01CCCC0
TRACK-ID = 0028 C-REC = 8290C01817        CC = C02 HH = 9
TRACK-ID = 0C28 C-REC = 8290CC1818        CC = 002 HH = 8
ERROR-COND = 00100CC0
TRACK-ID = 0028 C-REC = 8290CC1818        CC = 002 HH = 9   (5)
TRACK-ID = 0C28 C-REC = 8290CC1819        CC = C02 HH = 8
ERROR-COND = 00100000
TRACK-ID = 0028 C-REC = 8290CC1819        CC = 002 HH = 9
TRACK-ID = 0028 C-REC = 8290CC1820        CC = CC2 HH = 8
ERROR-COND = C01C00C0
TRACK-ID = 0028 C-REC = 8290CC182C        CC = CC2 HH = 9
TRACK-ID = CC28 C-REC = 8290001821        CC = 002 HH = 8
ERROR-COND = CC1C0CC0
TRACK-ID = 0028 C-REC = 8290001821        CC = CC2 HH = 9
TRACK-ID = 0C28 C-REC = 8290CC1822        CC = C02 HH = 8
ERROR-COND = C0100000
TRACK-ID = CC28 C-REC = 8290CC1822        CC = 0C2 HH = 9
TRACK-ID = 0028 C-REC = 8290CC1823        CC = CC2 HH = 8
ERROR-COND = C01C00C0
TRACK-ID = CC28 C-REC = 8290C01823        CC = 002 HH = 9
TRACK-ID = 0028 C-REC = 8290CC1824        CC = 002 HH = 8
ERROR-COND = C01CCCC0
TRACK-ID = 0028 C-REC = 8290CC1824        CC = C02 HH = 9
```

Figure 30. Creating a Direct File Using Method B (Part 3 of 4)

```
TRACK-ID = 0028 C-REC = 8290CC1825     CC = 002 HH = 8
ERROR-COND = 00100000
TRACK-ID = 0028 C-REC = 829000C1825    CC = C02 HH = 9
ERROR-COND = C0100CC0
TRACK-ID = C028 C-REC = 8290001825     CC = C84 HH = 0
TRACK-ID = 0028 C-REC = 8290CC1826     CC = 002 HH = 8
ERROR-COND = 00100000
TRACK-ID = 0028 C-REC = 8290001826     CC = C02 HH = 9
ERROR-COND = 00100000
TRACK-ID = C028 C-REC = 8290C01826     CC = 084 HH = 0
TRACK-ID = 0028 C-REC = 8290C01827     CC = CC2 HH = 8
ERROR-COND = 001CCCCO
TRACK-ID = 0028 C-REC = 8290C01827     CC = 002 HH = 9
ERROR-COND = C01C0CC0
TRACK-ID = 0028 C-REC = 8290CC1827     CC = C84 HH = C
TRACK-ID = 0028 C-REC = 8290C01828     CC = CC2 HH = 8
ERROR-COND = C01CCCCC
TRACK-ID = 0028 C-REC = 829000C1828    CC = C02 HH = 9
ERROR-COND = C01C0CC0
TRACK-ID = 0028 C-REC = 8290C01828     CC = 084 HH = 0
TRACK-ID = 0028 C-REC = 8290001829     CC = 002 HH = 8
ERROR-COND = C0100CC0
TRACK-ID = CC28 C-REC = 8290001829     CC = 002 HH = 9
ERROR-COND = 001CCCC0
TRACK-ID = 0028 C-REC = 8290001829     CC = 084 HH = 0
TRACK-ID = 0028 C-REC = 8290CC183C     CC = C02 HH = 8
ERROR-COND = CC1CCCCC
TRACK-ID = 0028 C-REC = 8290001830     CC = C02 HH = 9
ERROR-COND = C01C0CC0
TRACK-ID = 0028 C-REC = 829CCC183C     CC = C84 HH = 0
TRACK-ID = 0028 C-REC = 8290C01831     CC = CC2 HH = 8
ERROR-COND = C0100000
TRACK-ID = CC28 C-REC = 8290001831     CC = 002 HH = 9
ERROR-COND = 00100000
TRACK-ID = CC28 C-REC = 8290C01831     CC = 084 HH = 0
TRACK-ID = 0028 C-REC = 8290001832     CC = CC2 HH = 8
ERROR-COND = CC1CC000
TRACK-ID = 0028 C-REC = 8290C01832     CC = 002 HH = 9
ERROR-COND = C01C0000
TRACK-ID = 0028 C-REC = 829CC01832     CC = 084 HH = 0
TRACK-ID = CC28 C-REC = 8290001833     CC = C02 HH = 8
ERROR-COND = CC1CCCC0
TRACK-ID = 0028 C-REC = 8290CC1833     CC = 002 HH = 9
ERROR-COND = C0100000
TRACK-ID = 0028 C-REC = 8290CC1833     CC = 084 HH = 0
TRACK-ID = CC28 C-REC = 8290001834     CC = C02 HH = 8
ERROR-COND = 001C00C0
TRACK-ID = 0028 C-REC = 8290001834     CC = 002 HH = 9
ERROR-COND = CC1CCCCC
TRACK-ID = 0028 C-REC = 8290C01834     CC = C84 HH = 0
TRACK-ID = 0028 C-REC = 8290CC1835     CC = CC2 HH = 8
ERROR-COND = CC1CCCC0
TRACK-ID = 0028 C-REC = 829000C1835    CC = 002 HH = 9
ERROR-COND = 00100000
TRACK-ID = CC28 C-REC = 8290CC1835     CC = 084 HH = 0
TRACK-ID = CC28 C-REC = 8290C01836     CC = C02 HH = 8
ERROR-COND = C0100CC0
TRACK-ID = CC28 C-REC = 8290001836     CC = 002 HH = 9
ERROR-COND = 001CC0C0
TRACK-ID = 0028 C-REC = 8290001836     CC = 084 HH = 0

TRACK-ID = 0028 C-REC = 8290001837     CC = CC2 HH = 8
ERROR-COND = C0100000
TRACK-ID = 0028 C-REC = 8290C01837     CC = 002 HH = 9
ERROR-COND = CC1C00C0
TRACK-ID = CC28 C-REC = 8290001837     CC = 084 HH = 0
ERROR-COND = CC1CC000
TRACK-ID = 0028 C-REC = 829C001837     CC = 084 HH = 1
TRACK-ID = 0028 C-REC = 8290001838     CC = 002 HH = 8
ERROR-COND = C01C0CC0
TRACK-ID = 0028 C-REC = 8290CC1838     CC = 002 HH = 9
ERROR-COND = C0100CC0
TRACK-ID = 0028 C-REC = 8290CC183A     CC = C84 HH = 1
```

Figure 30. Creating a Direct File Using Method B (Part 4 of 4)

Figure 31 is a sample COBOL program which creates a direct file with relative track addressing using Method B. The sample program provides for the possibility of synonym overflow. Synonym overflow will occur if a record randomizes to a track which is already full. The following discussion highlights some basic features. Circled numbers on the program listing correspond to numbers in the text.

(1) Since the prime number used as a divisor is 829, the largest possible remainder will be 828.

(2) If synonym overflow occurs, control is given to the USE AFTER STANDARD ERROR declarative specified in the first section of the Procedure Division. The declarative provides that any record that cannot fit on the track to which it randomizes will be written on the first subsequent track available.

(3) The standard error condition "no room found" is tested before control is given to the SYNONYM-ROUTINE. Other standard error conditions as well as invalid key conditions result in job termination (EOJ).

ERROR-COND is the identifier which specifies the error condition that caused control to be given to the error declarative. ERROR-COND is printed on SYSLST whenever the error declarative section is entered. TRACK-ID and C-REC are also printed on SYSLST before execution of each WRITE statement. This output has been provided in order to facilitate an understanding of the logic involved in the creation of D-FILE.

(4) The first twelve records which randomize to relative track 18 are actually written on relative track 18.

(5) The next twelve records which randomize to relative track 18 are adjusted by the SYNONYM-ROUTINE and are actually written on relative track 19.

(6) The next twelve records which randomize to relative track 18 are adjusted by the SYNONYM-ROUTINE and are actually written on relative track 20.

(7) The last two records which randomize to relative track 18 are adjusted by the SYNONYM-ROUTINE and are actually written on relative track 21.

```
// JOB METHODB
// OPTION NODECK,LINK,LIST,LISTX,SYM,ERRS
// EXEC FCOBOL
```

```
 CBL QUOTE
00001           IDENTIFICATION DIVISION.
00002           PROGRAM-ID. METHODB.
00003           ENVIRONMENT DIVISION.
00004           CONFIGURATION SECTION.
00005           SOURCE-COMPUTER. IBM-360.
00006           OBJECT-COMPUTER. IBM-360.
00007           INPUT-OUTPUT SECTION.
00008           FILE-CONTROL.
00009               SELECT D-FILE ASSIGN TO SYS015-DA-2311-D-MASTER
00010               ACCESS IS RANDOM
00011               ACTUAL KEY IS ACT-KEY.
00012               SELECT C-FILE ASSIGN TO SYS007-UR-2540R-S.
00013           DATA DIVISION.
00014           FILE SECTION.
00015           FD  D-FILE
00016               LABEL RECORDS ARE STANDARD.
00017           01  D-REC.
00018               05 PART-NUM PIC X(8).
00019               05 NUM-ON-HAND PIC 9(4).
00020               05 PRICE PIC 9(5)V99.
00021               05 FILLER PIC X(181).
00022           FD  C-FILE
00023               LABEL RECORDS ARE OMITTED.
00024           01  C-REC.
00025               05 PART-NUM PIC X(8).
00026               05 NUM-ON-HAND PIC 9(4).
00027               05 PRICE PIC 9(5)V99.
00028               05 FILLER PIC X(61).
00029           WORKING-STORAGE SECTION.
00030           77  SAVE PIC S9(8) COMP SYNC.
00031           77  QUOTIENT PIC S9(8) COMP SYNC.
00032           01  ACT-KEY.
00033               02 TRACK-ID PIC S9(8) COMP SYNC.
00034               02 REC-ID PIC X(8).
00035           01  ERROR-COND.
00036               02 FILLER PIC 99 VALUE ZERO.
00037               02 ERR PIC 9 VALUE ZERO.
00038               02 FILLER PIC 9(5) VALUE ZERO.
```

• **Figure 31. Creating a Direct File with Relative Track Addressing Using Method B (Part 1 of 4)**

```
00039          PROCEDURE DIVISION.
00040          DECLARATIVES.
00041          ERROR-PROCEDURE SECTION. USE AFTER STANDARD ERROR PROCEDURE
00042              ON D-FILE GIVING ERROR-COND.
00043          ERROR-ROUTINE.
00044              EXHIBIT NAMED ERROR-COND.
00045              IF ERR = 1 GO TO SYNONYM-ROUTINE ELSE
00046                  DISPLAY "OTHER STANDARD ERROR  " REC-ID
00047                  GO TO EOJ.
00048          SYNONYM-ROUTINE.
00049              IF TRACK-ID IS LESS THAN 834, ADD 1 TO TRACK-ID. GO TO
00050                  WRITES.
00051          END DECLARATIVES.
00052              OPEN INPUT C-FILE
00053                   OUTPUT D-FILE.
00054          READS.
00055              READ C-FILE AT END GO TO EOJ.
00056              MOVE CORRESPONDING C-REC TO D-REC.
00057              MOVE PART-NUM OF C-REC TO REC-ID, SAVE.
00058              DIVIDE SAVE BY 829 GIVING QUOTIENT REMAINDER TRACK-ID.
00059          WRITES.
00060              EXHIBIT NAMED TRACK-ID C-REC.
00061              WRITE D-REC INVALID KEY GO TO INVALID-KEY.
00062              GO TO READS.
00063          INVALID-KEY.
00064              DISPLAY "INVALID KEY  " REC-ID.
00065          EOJ.
00066              CLOSE C-FILE D-FILE.
00067              STOP RUN.
```

(1) (2) (3)

```
// LBLTYP NSD(01)
// EXEC LNKEDT
```

• **Figure 31. Creating a Direct File with Relative Track Addressing Using Method B (Part 2 of 4)**

```
// ASSGN SYS007,X'00C'
// ASSGN SYS015,X'192'
// DLBL MASTER,,70/365,DA
// EXTENT SYS025,111111,1,0,10,850
// EXEC

TRACK-ID = 00000000 C-REC = 82900000
TRACK-ID = 00000001 C-REC = 82900001
TRACK-ID = 00000018 C-REC = 8290001801
TRACK-ID = 00000018 C-REC = 8290001802
TRACK-ID = 00000018 C-REC = 8290001803
TRACK-ID = 00000018 C-REC = 8290001804
TRACK-ID = 00000018 C-REC = 8290001805
TRACK-ID = 00000018 C-REC = 8290001806
TRACK-ID = 00000018 C-REC = 8290001807
GRACK-ID = 00000018 C-REC = 8290001808
TRACK-ID = 00000018 C-REC = 8290001809
TRACK-ID = 00000018 C-REC = 8290001810
TRACK-ID = 00000018 C-REC = 8290001811
TRACK-ID = 00000018 C-REC = 8290001812                    (4)
TRACK-ID = 00000018 C-REC = 8290001813
ERROR-COND = 00100000
TRACK-ID = 00000019 C-REC = 8290001813
TRACK-ID = 00000018 C-REC = 8290001814
ERROR-COND = 00100000
TRACK-ID = 00000019 C-REC = 8290001814
TRACK-ID = 00000018 C-REC = 8290001815
ERROR-COND = 00100000
TRACK-ID = 00000019 C-REC = 8290001815
TRACK-ID = 00000018 C-REC = 8290001816
ERROR-COND = 00100000
TRACK-ID = 00000019 C-REC = 8290001816
TRACK-ID = 00000018 C-REC = 8290001817
ERROR-COND = 00100000
TRACK-ID = 00000019 C-REC = 8290001817
TRACK-ID = 00000018 C-REC = 8290001818
ERROR-COND = 00100000
TRACK-ID = 00000019 C-REC = 8290001818
TRACK-ID = 00000018 C-REC = 8290001819                    (5)
ERROR-COND = 00100000
TRACK-ID = 00000019 C-REC = 8290001819
TRACK-ID = 00000018 C-REC = 8290001820
ERROR-COND = 00100000
TRACK-ID = 00000019 C-REC = 8290001820
TRACK-ID = 00000018 C-REC = 8290001821
ERROR-COND = 00100000
TRACK-ID = 00000019 C-REC = 8290001821
TRACK-ID = 00000018 C-REC = 8290001822
ERROR-COND = 00100000
TRACK-ID = 00000019 C-REC = 8290001822
TRACK-ID = 00000018 C-REC = 8290001823
ERROR-COND = 00100000
TRACK-ID = 00000019 C-REC = 8290001823
TRACK-ID = 00000018 C-REC = 8290001824
ERROR-COND = 00100000
TRACK-ID = 00000019 C-REC = 8290001824
```

• **Figure 31. Creating a Direct File with Relative Track Addressing Using Method B (Part 3 of 4)**

```
TRACK-ID = 00000018 C-REC = 8290001825
ERROR-COND = 00100000
TRACK-ID = 00000019 C-REC = 8290001825
ERROR-COND = 00100000
TRACK-ID = 00000020 C-REC = 8290001825
TRACK-ID = 00000018 C-REC = 8290001826
ERROR-COND = 00100000
TRACK-ID = 00000019 C-REC = 8290001826
ERROR-COND = 00100000
TRACK-ID = 00000020 C-REC = 8290001826
TRACK-ID = 00000018 C-REC = 8290001827
ERROR-COND = 00100000
TRACK-ID = 00000019 C-REC = 8290001827
ERROR-COND = 00100000
TRACK-ID = 00000020 C-REC = 8290001827
TRACK-ID = 00000018 C-REC = 8290001828
ERROR-COND = 00100000
TRACK-ID = 00000019 C-REC = 8290001828
ERROR-COND = 00100000
TRACK-ID = 00000020 C-REC = 8290001828
TRACK-ID = 00000018 C-REC = 8290001829
ERROR-COND = 00100000
TRACK-ID = 00000019 C-REC = 8290001829
ERROR-COND = 00100000
TRACK-ID = 00000020 C-REC = 8290001829
TRACK-ID = 00000018 C-REC = 8290001830
ERROR-COND = 00100000
TRACK-ID = 00000019 C-REC = 8290001830
ERROR-COND = 00100000
TRACK-ID = 00000020 C-REC = 8290001830
TRACK-ID = 00000018 C-REC = 8290001831
ERROR-COND = 00100000
TRACK-ID = 00000019 C-REC = 8290001831
ERROR-COND = 00100000
TRACK-ID = 00000020 C-REC = 8290001831
TRACK-ID = 00000018 C-REC = 8290001832
ERROR-COND = 00100000
TRACK-ID = 00000019 C-REC = 8290001832
ERROR-COND = 00100000
TRACK-ID = 00000020 C-REC = 8290001832
TRACK-ID = 00000018 C-REC = 8290001833
ERROR-COND = 00100000
TRACK-ID = 00000019 C-REC = 8290001833
ERROR-COND = 00100000
TRACK-ID = 00000020 C-REC = 8290001833
TRACK-ID = 00000018 C-REC = 8290001834
ERROR-COND = 00100000
TRACK-ID = 00000019 C-REC = 8290001834
ERROR-COND = 00100000
TRACK-ID = 00000020 C-REC = 8290001834
TRACK-ID = 00000018 C-REC = 8290001835
ERROR-COND = 00100000
TRACK-ID = 00000019 C-REC = 8290001835
ERROR-COND = 00100000
TRACK-ID = 00000020 C-REC = 8290001835
TRACK-ID = 00000018 C-REC = 8290001836
ERROR-COND = 00100000
TRACK-ID = 00000019 C-REC = 8290001836
ERROR-COND = 00100000
TRACK-ID = 00000020 C-REC = 8290001836
TRACK-ID = 00000018 C-REC = 8290001837
ERROR-COND = 00100000
TRACK-ID = 00000019 C-REC = 8290001837
ERROR-COND = 00100000
TRACK-ID = 00000020 C-REC = 8290001837
ERROR-COND = 00100000
TRACK-ID = 00000021 C-REC = 8290001837
TRACK-ID = 00000018 C-REC = 8290001838
ERROR-COND = 00100000
TRACK-ID = 00000019 C-REC = 8290001838
ERROR-COND = 00100000
TRACK-ID = 00000020 C-REC = 8290001838
ERROR-COND = 00100000
TRACK-ID = 00000021 C-REC = 8290001838
```

(6)

(7)

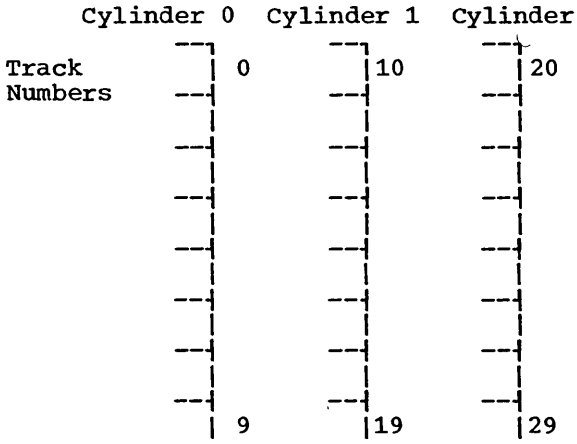
• Figure 31. Creating a Direct File with Relative Track Addressing Using Method B (Part 4 of 4)

ACTUAL TRACK ADDRESSING CONSIDERATIONS FOR SPECIFIC DEVICES

Randomizing for the 2311 Disk Drive

When randomizing for the 2311 Disk Drive, it is possible to circumvent the discontinous binary address by coding the randomizing formula in decimal arithmetic and then converting the results to binary. This can be done by setting aside a decimal field with the low-order byte reserved for the head number, and the high-order bytes reserved for the cylinder number. A mathematical overflow from the head number will now increment the cylinder number and produce a valid address. The low-order byte should then be converted to binary and stored in the HH field, and the high-order bytes converted to binary and stored in the CC field of the actual key field.

Randomizing to the 2311 Disk Drive should present no significant problems if the programmer using direct organization is completely aware that the cylinder and head number give him a unique track number. To illustrate, the 2311 could be thought of as consisting of tracks numbered as follows:

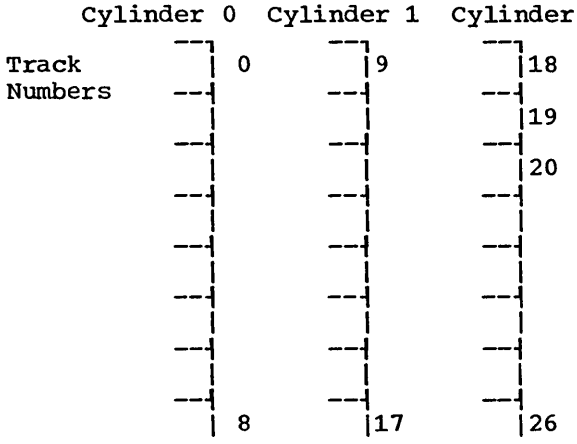


If the randomizing formula resulted in an address of cylinder 001, head 9:

Cylinder Number	Head Number
001	9

this would be a reference to track 19. This fact allows the programmer to ignore the discontinuous cylinder and head number. If his formula resulted in an address of 0020, this would result in accessing cylinder 2, head 0, the location of track 20.

The programmer can make another use of this decimal track address. He may wish to reserve the last track of each cylinder for synonyms. If this is the case, he is in effect redefining the cylinder to consist of nine tracks rather than ten tracks. The 2311 cylinder could then be thought of as consisting of track numbers, as follows:



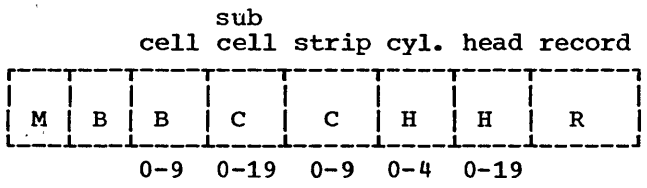
If the programmer randomizes to relative track number 20, he can access it by dividing the track address by the number of tracks (9) in a cylinder. The quotient now becomes the cylinder number, and the remainder becomes the head number.

```
      2 = cylinder number
   ------
9/ 0020
    18
    ---
      2 = head number
```

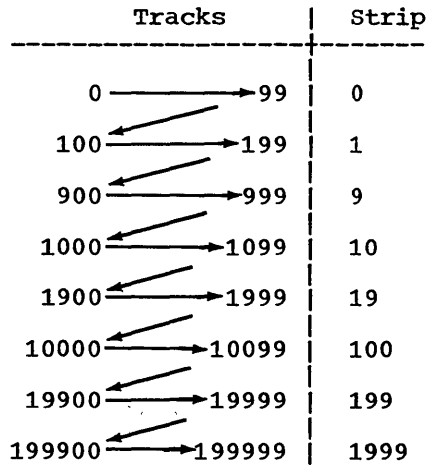
To simplify randomizing, an algorithm must be developed to generate a decimal track address. This track address can then be converted to a binary cylinder number and head number. In addition, tracks can be reserved by dividing the track address by the number of tracks in a cylinder. The same concepts will hold true for devices such as the 2314. For example, an algorithm can be developed using 20 tracks per cylinder and dividing by the closest prime number less than 20.

Randomizing for the 2321 Data Cell

The track reference field for the 2321 Data Cell is composed of the following discontinuous binary address:



At first glance, this presents an almost impossible randomizing task; but since each strip includes 100 tracks that are accessible through cylinder and head number, the 2321 Data Cell can be considered to consist of consecutively numbered tracks.



It can be seen that relative track 20 is located on cylinder 1, head 0 of some particular strip. Its address can be calculated by dividing by 20.

```
      1 = cylinder number
   ---
20 ) 20
     20
    ---
      0 = head number
```

Thus, relative track number 120 will be located on strip 1, cylinder 1, head 0 of some subcell. Note that the strip number is given by the hundreds digit, and the cylinder and head number are derived by dividing the two low-order digits by 20.

The same relationship holds true for relative track number 900. It is located on strip 9, cylinder 0, track 0. Again, the hundreds digit gives the strip number, and dividing the two low-order digits by 20 results in a quotient and remainder of zero.

This relationship holds true through a relative track number of 19999, which is the number of tracks that can be contained on one cell of a data cell array. By applying the foregoing rules, an address of subcell 19, strip 9, cylinder 4, head 19 is derived.

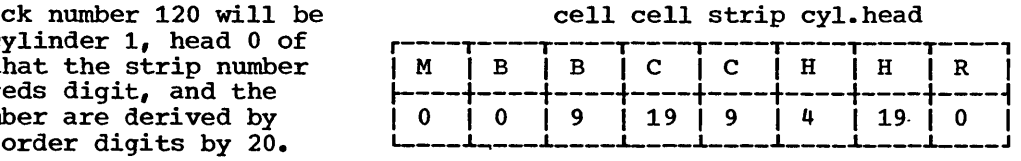
Thus, by randomizing to a 5-digit decimal track number, the programmer will be able to access the 20,000 tracks (40,000,000 characters) contained in a cell.

The thousands digits would represent the subcell number, the hundreds digit the strip number, and the quotient and remainder of the two low-order digits divided by 20 would represent the cylinder and head number. Each one of these resulting decimal digits would then be converted to binary and placed in the appropriate location in the track reference field.

There is a total of 200,000 tracks per data cell array. To derive valid addresses that cross cell boundaries, the user should randomize to a 6-digit decimal track address. The highest address possible should be 199,999. To convert this to a data cell address, similar rules apply. In this case, the user must divide the three high-order digits by 20:

```
      9 = cell
   ---
20 ) 199
     180
    ----
      19 = subcell
```

The quotient becomes the cell number and the remainder becomes the subcell number. The hundreds digit is still the strip number, and the cylinder and head number can be derived as previously illustrated. The resulting address is 0091994190 and would appear in the first eight bytes of the actual key field as follows:



Randomizing to the data cell can be accomplished by developing an algorithm to generate decimal track addresses. The use of the foregoing rules makes it possible to

convert these generated track addresses to the appropriate discontinuous binary address.

INDEXED ORGANIZATION (DTFIS)

An indexed file is a sequential file with indexes that permit rapid access to individual records as well as rapid sequential processing. Error recovery from a DTFIS file is described in detail in the chapter "Advanced Processing Capabilities." An indexed file has three distinct areas: a prime area, indexes, and an overflow area. Each area is described in detail below.

PRIME AREA

When the file is first created, or when it is subsequently reorganized, records are written in the prime area. Until the prime area is full, additions to the file may also be written there. The prime area may span multiple volumes.

The records in the prime area must be formatted with keys, and must be positioned in key sequence. The records may be blocked or unblocked. If records are blocked, each logical record within the block contains its key, and the key area for the block contains the key of the highest record in the block. The Disk Operating System permits fixed-length records only. Figure 32 shows the formats of blocked and unblocked records on a track.

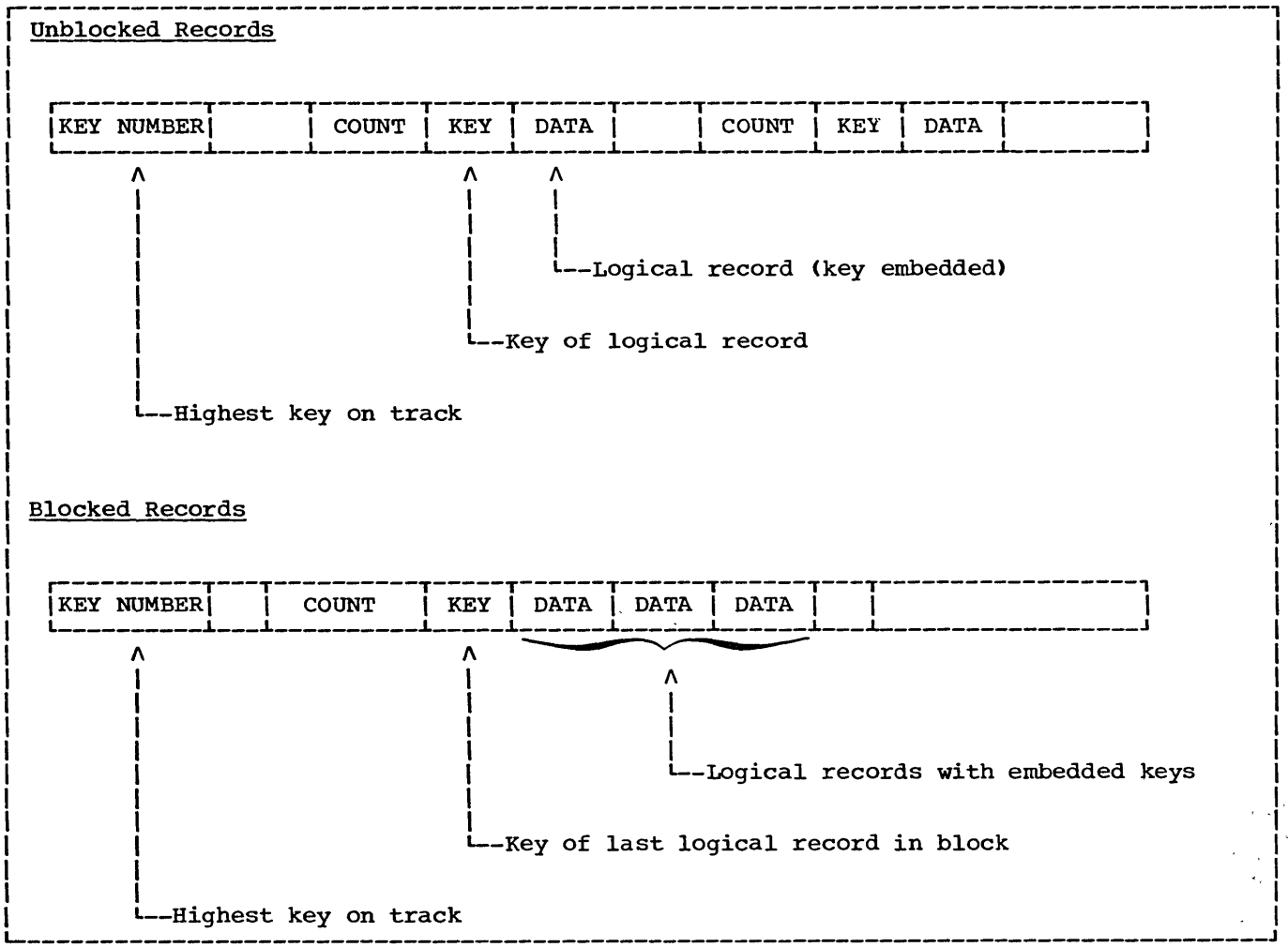


Figure 32. Formats of Blocked and Unblocked Records

INDEXES

There are three possible levels of indexes for a file with indexed organization: a track index, a cylinder index, and a master index. They are created and written by the system when the file is created or reorganized.

Track Index

This is the lowest level of index and is always present. There is one track index for each cylinder in the prime area. It is always written on the first track of the cylinder that it indexes.

The track index contains a pair of entries for each prime data track in the cylinder: a normal entry and an overflow entry. The normal entry contains the home address of the prime track and the key of the highest record on the track. The overflow entry contains the highest key associated with that track and the address of the lowest record in the overflow area. If no overflow entry has yet been made, the address of the lowest record in the overflow area is the dummy entry X'FF'.

Cylinder Index

The cylinder index is a higher level of index and is always present. Its entries point to track indexes. There is one cylinder index for the file. It is written on the device specified in the APPLY CYL-INDEX clause. If this clause is not specified, the cylinder index is written on the same device as the prime area.

Master Index

The master index is the highest level index and is optional. It is used when the cylinder index is so long that searching it is very time consuming. It is suggested that a master index be requested when the cylinder index occupies more than four tracks. (A master index consists of one entry for each track of the cylinder index.)

The Disk Operating System permits one level of master index for the file and requires that it be written immediately before the cylinder index. If a master index is desired, the APPLY MASTER-INDEX clause must be specified in the source program. When this clause is specified, the cylinder index is placed on the same device as the master index.

OVERFLOW AREA

There are two types of overflow areas: a cylinder overflow area and an independent overflow area. Either or both may be specified for an indexed file. Records are written in the overflow area(s) as additions are made to the file.

Cylinder Overflow Area

A certain number of whole tracks are reserved in each cylinder for overflow records from the prime tracks in that cylinder. The user may specify the number of tracks to be reserved by means of the APPLY CYL-OVERFLOW clause. If he specifies 0 as the number of tracks in this clause, no cylinder overflow area is reserved. If the clause is omitted, 20% of each cylinder is reserved for overflow.

Independent Overflow Area

Overflow records from anywhere in the prime area are placed in a certain number of cylinders reserved soley for this purpose. The size and location of the independent overflow area can be specified if the user includes the proper job control XTENT (or EXTENT) cards. The area must, however, be on the same mass storage device type as the prime area.

A suggested approach is to have cylinder overflow areas large enough to contain the average number of overflow records caused by additions and an independent overflow area to be used as the cylinder overflow areas are filled.

Adding Records to an Indexed File

A new record added to an indexed file is placed into a location on a track in the prime area determined by the value of its key field. If records in the file were placed in precise physical sequence, the addition of a new record would require the shifting of all records with keys higher than that of the one inserted. However, indexed organization allows a record to be inserted into its proper position on a track, with the shifting of only the

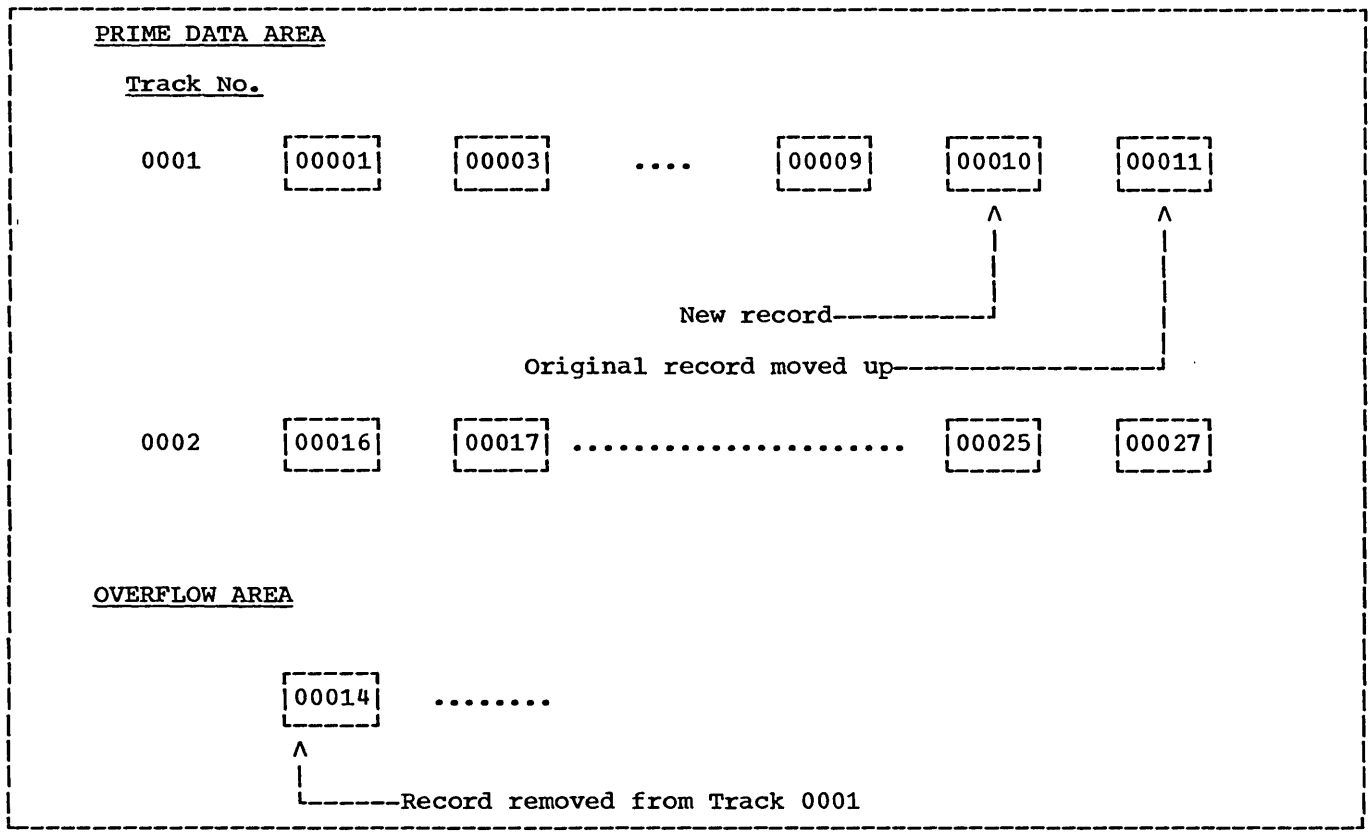


Figure 33. Adding a Record to a Prime Track

records on that track. Any records for which there is no space on that track are then placed in an overflow area, and become overflow records. Overflow records are always fixed-length, unblocked records, formatted with keys.

As records are added to the overflow area, they are no longer in key sequence. The system ensures, however, that they are always in logical sequence.

Figure 33 illustrates the addition of a record to a prime track.

The new record (00010) is written in its proper sequential location on the prime track. The rest of its prime records are moved up one location. The bumped record (00014) is written in the first available location in the overflow area. The record is placed in the cylinder overflow area for that cylinder, if a cylinder overflow area exists and if there is space in it; otherwise, the record is placed in the independent overflow area. The first addition to a track is always handled in this manner. Any record that is higher than the original highest record on the preceding track, but lower than the original highest record on this track, is written on the prime track. Record 00015, for example, would be written as the first record on track 0002, and record 00027 would be bumped into the overflow area.

Subsequent additions are written either on the prime track where they belong or as part of the overflow chain from that track. If the addition belongs between the last prime record on a track and a previous overflow from that track (as is the case with record 00013), it is written in the first available location in the overflow area on an empty track, or on a track whose first record has a numerically lower key.

If the addition belongs on a prime track (as would be the case with record 00005), it is written in its proper sequential location on the prime track. The bumped record (record 00011) is written in the overflow area.

A record with a key higher than the current highest key in the file is placed on the last prime track containing data records. If that track is full, the record is placed in the overflow area.

ACCESSING AN INDEXED FILE (DTFIS)

An indexed file may be accessed both sequentially and randomly.

ACCESSING AN INDEXED FILE SEQUENTIALLY: An indexed file may only be created sequentially. It can also be read and updated in the sequential access mode. The following specifications may be made in the source program.

ENVIRONMENT DIVISION

Required clauses:

SELECT [OPTIONAL] file-name

ASSIGN TO SYSnnn-DA-{2311 / 2314 / 2321}- I

RECORD KEY Clause
NOMINAL KEY Clause (when reading, if the START statement is used)

Optional clauses:

FILE-LIMIT Clause
ACCESS MODE IS SEQUENTIAL
PROCESSING MODE IS SEQUENTIAL
RERUN Clause
SAME Clause
APPLY WRITE-VERIFY Clause (create and update)
APPLY CYL-OVERFLOW Clause (create)

APPLY {MASTER-INDEX / CYL-INDEX} Clause

Invalid clauses:

ACCESS MODE IS RANDOM
ACTUAL KEY Clause
TRACK-AREA Clause
MULTIPLE FILE TAPE Clause
APPLY WRITE-ONLY Clause
APPLY EXTENDED-SEARCH Clause
APPLY CORE-INDEX Clause
RESERVE Clause

ACCESSING AN INDEXED FILE RANDOMLY: A randomly-accessed indexed file may be read, updated, or added to. The following specifications may be made in the source program:

ENVIRONMENT DIVISION

Required clauses:

SELECT [OPTIONAL] file-name

ASSIGN TO SYSnnn-DA-{2311 / 2314 / 2321} -I

ACCESS IS RANDOM
NOMINAL KEY Clause
RECORD KEY Clause

Optional clauses:

FILE LIMIT Clause
PROCESSING MODE IS SEQUENTIAL
TRACK-AREA Clause
RERUN Clause
SAME Clause
APPLY WRITE VERIFY Clause
APPLY CYL-OVERFLOW Clause
APPLY CORE-INDEX Clause

APPLY {MASTER-INDEX / CYL-INDEX} Clause

Invalid clauses:

RESERVE Clause
ACCESS MODE IS SEQUENTIAL
ACTUAL KEY Clause
MULTIPLE FILE TAPE Clause
APPLY EXTENDED-SEARCH Clause

Key Clauses

When creating an indexed file, the only key clause required is the RECORD KEY clause. The data-name specified in this clause is the name of the field within the record that contains the key. Keys must be in ascending numerical order when creating an indexed file.

If a START statement is used when retrieving an indexed file sequentially, the NOMINAL KEY clause is required.

When accessing an indexed file randomly, both the NOMINAL KEY and RECORD KEY clauses are required. When reading the file, the data-name specified in the NOMINAL KEY clause is the key of the record which is being retrieved. The data-name specified in the RECORD KEY clause is the name of the field within the record that contains this key.

When adding records to an indexed file, the data-name specified in the NOMINAL KEY clause is the key for the record being written and is used to determine its physical location. The data-name specified in the RECORD KEY clause specifies the field in the record that contains the key.

Improving Efficiency

When processing an indexed file, the following source language Environment Division clauses may be used to improve efficiency:

TRACK-AREA Clause
APPLY CORE-INDEX Clause

For additional details, see the publication IBM System/360 Disk Operating System: American National Standard COBOL.

The following topics are discussed within this chapter:

DTF Tables

Error Recovery

Volume and File Label Handling

DTF TABLES

Whenever COBOL imperative-statements (READ, WRITE, REWRITE, etc.) are used in a program to control the input and/or output of records in a file, that file must be defined by a DTF. A DTF is created by the compiler for each file opened in a COBOL program from information specified in the Environment Division, FD entry, and input/output statements in the source program. The DTF for each file is part of the object module that is generated by the compiler. It describes the characteristics of the logical file, indicates the type of processing to be used for the file, and specifies the main storage areas and routines used for the file.

The DTF's generated for the permissible combinations of device type and COBOL file processing technique are as follows:

DTFCD Card reader, punch -- organization and access sequential

DTFPR Printer -- organization and access sequential

DTFMT Tape -- organization and access sequential

DTFSD Mass storage device -- organization and access sequential

DTFDA Mass storage device -- organization direct, access sequential or random

DTFIS Mass storage device -- organization indexed, access sequential or random

Because of their limited interest for the COBOL programmer, the contents and location of the fields of each of the DTF types are not discussed in this publication. However, there are certain fields which immediately precede the storage area allocated for the DTF which are pertinent and which are described below.

For magnetic tape files (DTFMT) or sequentially organized files on mass storage devices (DTFSD), a 24-byte Pre-DTF is reserved in front of the DTF. The fields of the Pre-DTF are shown in Table 10. If any option is not specified, the field will contain binary zeros.

When actual track addressing is used for files with direct organization and random access (DTFDA), a variable-length Pre-DTF is reserved. The fields of the Pre-DTF are shown in Table 11. If any option is not specified, the field will contain binary zeros.

When relative track addressing is used for files with direct organization and random access (DTFDA), a variable-length Pre-DTF is reserved. The fields of the Pre-DTF are shown in Table 12. If any option is not specified, the field will contain binary zeros.

• Table 10. Fields Preceding DTFMT and DTFSD

1 byte	Number of reels (as specified in the ASSIGN clause) when file is opened[1]
1 byte	Number of reels remaining (i.e., file not completely read)[1]
2 bytes	Maximum record length if records are variable, blocked and APPLY WRITE-ONLY is not specified.
4 bytes	Address of label declarative with BEGINNING [REEL / UNIT] option
4 bytes	Address of label declarative with ENDING [REEL / UNIT] option
4 bytes	Address of label declarative with ENDING FILE option
4 bytes	Address of label declarative with BEGINNING FILE option
1 byte	Switch -- FF if closed WITH LOCK; otherwise, the switch is used as shown in Table 16
3 bytes	Address of USE AFTER STANDARD ERROR declarative
	DTFMT/DTFSD

[1]For INPUT files with nonstandard labels only.

• Table 11. Fields Preceding DTFDA - ACCESS IS RANDOM - Actual Track Addressing

9-263 bytes	ACTUAL KEY[1]
8 bytes	SEEK Address[2]
2 bytes	Error bytes[3]
4 bytes	Address of file extent information
4 bytes	Address of label declarative with ENDING FILE option
4 bytes	Address of label declarative with BEGINNING FILE option
1 byte	Switch -- FF if closed WITH LOCK; otherwise the switch is used as shown in Table 16
3 bytes	Address of USE AFTER STANDARD ERROR declarative
	DTFDA

[1]ACTUAL KEY specified in last executed WRITE statement
[2]In the form MBBCCHHR
[3]This area is reserved by the Supervisor and assigned the name ERRBYTE. For a complete discussion, refer to the publication IBM System/360 Disk Operating System: Supervisor and Input/Output Macros, Form GC24-5037.

• Table 12. Fields Preceding DTFDA - ACCESS IS RANDOM - Relative Track Addressing

5-258 bytes	ACTUAL KEY[1]
4 bytes	SEEK address[2]
3 bytes	Last extent used[3]
1 byte	Not used
2 bytes	Error bytes[4]
1 byte	Index to last extent used in the Disk Extent Table
3 bytes	Address of Disk Extent Table in the DTF
4 bytes	Address of label declarative with ENDING FILE option
4 bytes	Address of label declarative with BEGINNING FILE option
1 byte	Switch -- FF if closed WITH LOCK; otherwise the switch is used as shown in Table 16
3 bytes	Address of USE AFTER STANDARD ERROR declarative
	DTFDA

[1]ACTUAL KEY specified in the last executed WRITE statement
[2]In the form TTTR
[3]In the form TTT
[4]This area is reserved by the DOS Supervisor and assigned the name ERRBYTE. For a complete discussion, refer to the publication IBM System/360 Disk Operating System: Supervisor and Input/Output Macros.

When actual track addressing is used for files with direct organization and sequential access (DTFDA), a 31-byte Pre-DTF is reserved. The fields of the Pre-DTF are shown in Table 13. If any option is not specified, the field will contain binary zeros.

When relative track addressing is used for files with direct organization and sequential access (DTFDA), a 31-byte Pre-DTF is reserved. The fields of the Pre-DTF are shown in Table 14. If any option is not specified, the field will contain binary zeros.

For files whose organization is indexed, only four bytes are reserved preceding the DTF, as shown in Table 15.

• Table 13. Fields Preceding DTFDA - ACCESS IS SEQUENTIAL - Actual Track Addressing

8 bytes	SEEK address[1]
5 bytes	IDLOC[2]
2 bytes	Error bytes[3]
4 bytes	Address of file extent information
4 bytes	Address of label declarative with ENDING FILE option
4 bytes	Address of label declarative with BEGINNING FILE option
1 byte	Switch -- FF if closed WITH LOCK; otherwise the switch is used as shown in Table 16
3 bytes	Address of USE AFTER STANDARD ERROR declarative
	DTFDA

[1]In the form MBBCCHHR
[2]Address (returned by the system) of next record in the form CCHHR
[3]This area is reserved by the DOS Supervisor and assigned the name ERRBYTE. For a complete discussion, refer to the publication IBM System/360 Disk Operating System: Supervisor and Input/Output Macros.

• Table 14. Fields Preceding DTFDA - ACCESS IS SEQUENTIAL - Relative Track Addressing

4 bytes	SEEK address[1]
3 bytes	Last extent used[2]
1 byte	Not used
4 bytes	IDLOC[3]
1 byte	Not used
2 bytes	Error bytes[4]
1 byte	Index to the last extent used in the Disk Extent Table
3 bytes	Address of Disk Extent Table in the DTF
4 bytes	Address of label declarative with ENDING FILE option
4 bytes	Address of label declarative with BEGINNING FILE option
1 byte	Switch -- FF if closed with LOCK; otherwise the switch is used as shown in Table 16
3 bytes	Address of USE AFTER STANDARD ERROR declarative
DTFDA	

[1]In the form TTTR
[2]In the form TTT
[3]Address (returned by the system) of the next record in the form TTTR
[4]This area is reserved by the DOS Supervisor and assigned the name ERRBYTE. For a complete discussion, refer to the publication IBM System/360 Disk Operating System Supervisor and Input/Output Macros.

• Table 15. Fields Preceding DTFIS

1 byte	Switch -- FF if closed WITH LOCK; otherwise the switch is used as shown in Table 16
3 bytes	Address of USE AFTER STANDARD ERROR declarative
DTFIS	

Some files can be opened several different ways in one COBOL program.

For DTFCD and DTFPR, only one DTF will be generated for each file.

For DTFMT, a maximum of three DTF's may be needed -- one each for OPEN INPUT, OPEN INPUT REVERSED, and OPEN OUTPUT.

For DTFSD, a maximum of three DTF's may be needed -- one each for OPEN INPUT, OPEN OUTPUT, and OPEN I-O statements.

For DTFIS, a maximum of two DTF's may be needed. If access is sequential, one DTF for "load" and one for "retrieve" may be needed. If access is random, only one DTF need be generated at a time.

For DTFDA, only one DTF is needed.

Pre-DTF Switch

When used, this switch provides communication between the executing program and its input/output subroutines at execution time. The entire byte may be set to X'FF' to indicate that the file was closed WITH LOCK and cannot be reopened. Otherwise the switch is used as shown in Table 16.

ERROR RECOVERY

COBOL allows the user to handle input/output errors through 1) the INVALID KEY clause for certain source language statements, and 2) the USE AFTER STANDARD ERROR declarative sentence.

Input/output errors caused by the program can be recovered from directly by the procedure specified in the INVALID KEY clause. That is, when the system determines that an invalid key condition exists, control is returned to the user at the imperative-statement specified in the INVALID KEY clause. An invalid key condition can occur on files with direct or indexed organization and on sequentially organized disk files. The errors that cause an invalid key condition are shown in Table 17.

• Table 16. Meaning of Pre-DTF Switch

Bit	Meaning, if ON
0	Turned ON the first time a DTFSD output file is opened. The entire DTF is saved for subsequent OPEN OUTPUT statements.
1	Turned ON when DTFDA or DTFSD files are opened I-O.
2	This bit is ON to indicate beginning of volume user label processing. The bit is set OFF when a file is opened to indicate to the user label processing subroutine (ILBDUSL0) that beginning-of-file user labels are to be processed. That subroutine sets the bit ON after beginning-of-file processing to indicate that all subsequent calls for this subroutine are for beginning-of-volume user label processing.
3	For output files with variable-length blocked records, this bit is turned OFF when a file is opened and ON for all WRITE's after the first.
4	Turned ON for spanned record processing on a DTFDA file.
5-7	Not used.

Table 17. Errors Causing an Invalid Key Condition

Organization	ACCESS	OPEN	I-O Verb	Condition
Sequential	[SEQUENTIAL]	OUTPUT	WRITE	End of extents reached.
Direct	[SEQUENTIAL]	OUTPUT	WRITE	Track address outside file extents.
Direct	RANDOM	INPUT	READ	No record found.
		OUTPUT	WRITE	Track address outside file extents.
		I-O	READ REWRITE	Track address outside file extents.
Indexed	[SEQUENTIAL]	INPUT I-O	START	No record found.
		OUTPUT	WRITE	Duplicate record; sequence check.
	RANDOM	INPUT	READ	No record found.
		I-O	REWRITE	
		I-O	WRITE	Duplicate record.

Other input/output errors cause the job to be cancelled unless the user has specified a USE AFTER STANDARD ERROR declarative. Control is transferred to this declarative section if the system determines that a "standard" error has occurred during input/output processing. In this declarative section, the user may interrogate the COBOL error bytes if he has specified the GIVING option of the USE AFTER STANDARD ERROR declarative sentence. The meaning of these bytes for a specified combination of device type and file processing technique is shown in Table 18.

Table 18. Meaning of Error Bytes for GIVING Option of Error Declarative (Part 1 of 2)

Device	Organization	ACCESS	OPEN	I/O Verb	Condition	Byte	Result
Unit record	Sequential	[SEQUENTIAL]			Input/output error	1	File must be closed and job must be terminated.
Tape	Sequential	[SEQUENTIAL]	INPUT	READ	Wrong length record	2	Skip block if return is made to non-declarative portion.
					Parity error	1	Skip block if return is made to non-declarative portion.
			OUTPUT	WRITE	All exceptional conditions are handled by the system.		
DASD	Sequential	[SEQUENTIAL]	INPUT I-O	READ	Wrong length record	2	Skip block if return is made to non-declarative portion.
					Parity error	1	Skip block if return is made to non-declarative portion.
			OUTPUT I-O	WRITE	Parity error	1	Bad block written.
					Wrong length record	2	Bad block written.
DASD	Direct	[SEQUENTIAL]	INPUT	READ	Wrong length record	2	Return to statement after READ.
					Data check in count area	1	Return to statement after READ.
					Data check for key and/or data	4	Return to statement after READ.
DASD	Direct	RANDOM	INPUT I-O	READ	Same as ACCESS SEQUENTIAL (above).		
			OUTPUT	WRITE	Wrong length record	2	Return to next statement; bad block written.
					Data check in count area	1	Return to next statement; bad block written.
					Data check for key and/or data	4	Return to next statement; bad block written.
					No room found	3	Return to next statement.

Note: If no USE AFTER STANDARD ERROR routine is specified and one of the above conditions occurs, the user is notified of the condition and the job is cancelled.

Table 18. Meaning of Error Bytes for GIVING Option of Error Declarative (Part 2 of 2)

Device	Organization	ACCESS	OPEN	I/O Verb	Condition	Byte	Result
DASD	Direct	RANDOM	I/O	REWRITE	Wrong length record	2	Return to next statement; bad block written.
					Data check in count area	1	Return to next statement; bad block written.
					Data check in key and/or data	4	Return to next statement; bad block written.
DASD	Indexed	[SEQUENTIAL]	INPUT I-O	READ REWRITE	DASD error	1	Return to next statement; bad block read or written.
					Wrong length record	2	
				START	DASD error	1	Continued processing of file permitted.
			OUTPUT	WRITE	DASD error	1	Return to next statement; bad block written.
					Wrong length record	2	
					Prime data area full	3	File must be closed.
					Cylinder index full	4	File must be closed.
					Master index full	5	File must be closed.
DASD	Indexed	RANDOM	INPUT I-O	READ REWRITE	DASD error	1	Return to next statement; bad block read or written.
					Wrong length record	2	
			I-O	WRITE	DASD error	1	Return to next statement; bad block written.
					Wrong length record	2	
					Overflow area full	6	Files must be closed.

Note: If no USE AFTER STANDARD ERROR routine is specified and one of the above conditions occurs, the user is notified of the condition and the job is cancelled.

If the user includes a USE AFTER STANDARD ERROR routine without specifying the GIVING option, he must call an assembler language routine within the declarative if he wishes to interrogate the error bits -- set either in the DTF (DTFMT, DTFSD, or DTFIS) or in the fields preceding the DTF (DTFDA).

Interrogation of these error bits should be made to the locations shown in Tables 19, 20, 21, and 22.

Note: The byte and bit displacement in Tables 19, 20, 21, and 22 is relative to zero.

Table 19. Location and Meaning of Error Bits for DTFMT

OPEN	Verb	Condition	Byte*	Bit
INPUT	READ	Wrong length record	3	1
		Parity error	2	6
OUTPUT	WRITE	Wrong length record	3	1
		Parity error	2	6

*Within the DTF.

Table 20. Location and Meaning of Error Bits for DTFSD

OPEN	Verb	Condition	Byte*	Bit
INPUT, I-O	READ	Wrong length record	3	1
		Parity error	2	6
OUTPUT, I-O	WRITE	Parity error	2	6

*Within the DTF.

Table 21. Location and Meaning of Error Bits for DTFDA

ACCESS	OPEN	Verb	Condition	Byte*	Bit
[SEQUENTIAL]	INPUT	READ	Wrong length record	0	1
			Data check in count area	1	0
			Data check in key or data	1	3
			No record found	1	2 or 4
RANDOM	INPUT, I-O	READ	Same as sequential		
	OUTPUT	WRITE	Wrong length record	0	1
			No room found	0	4
			Data check in count area	1	0
			Data check in key or data	1	3
	I-O	REWRITE	Wrong length record	0	1
			Data check in count area	1	0
			Data check in key or data	1	3
			No record found	1	2 or 4

*Within error bytes preceding DTF. See the section "DTF Tables" for the location of these bytes.

Table 22. Location and Meaning of Error Bits for DTFIS

ACCESS	OPEN	Verb	Condition	Byte*	Bit
[SEQUENTIAL]	INPUT, I-O	READ	DASD error	30	0
			Wrong length record	30	1
	OUTPUT	WRITE	DASD error	30	0
			Wrong length record	30	1
			Prime data area full	30	2
			Cylinder index full	30	3
			Master index full	30	4
RANDOM	INPUT, I-O	READ REWRITE	DASD error	30	0
			Wrong length record	30	1
	I-O	WRITE	DASD error	30	0
			Wrong length record	30	1
			Overflow area full	30	6
*Within the DTF.					

The following should be considered when processing tape input files:

1. Two types of errors are returned to the user: wrong length record and parity check. The COBOL error bytes, if requested, are set to reflect the error condition and control is transferred to the USE AFTER STANDARD ERROR declarative sentence. The error block is made available at data-name-2 of the GIVING option, if specified.

   If a parity error is detected when a block of records is read, the tape is backspaced and reread 100 times before control is returned to the user. If the error persists, the block is considered an error block and is added to the block count found in the DTF table.

2. Normal return (to the non-declarative portion) from a USE AFTER STANDARD ERROR declarative section is through the invoked IOCS subroutine. Thus, the next sequential block is brought into main storage permitting continued processing of the file. (The error block is bypassed.) A return through the use of a GO TO statement does not bring the next block into main storage; therefore, it is impossible to continue processing the file.

The processing of a sequential disk file opened as input is the same as the previous discussion of tape files, except that the disk block is reread ten times before being considered an error block.

COBOL cannot handle nested errors on sequential files. If errors occur within an error declarative, results are unpredictable.

VOLUME AND FILE LABEL HANDLING

TAPE LABELS

Among the several types of tape labels allowed under the Disk Operating System are: volume labels, standard file labels, user standard labels, and nonstandard labels. Unlabeled files are also permitted. The description of each type of label follows.

Volume Labels

A volume label is used whenever standard file labels are used. Logical IOCS requires a volume label with VOL1 as its first four characters on every standard or user labeled file. VOL2-VOL8 are also allowed, but must be written and checked by the user.

Standard File Labels

A standard file label is an 80-character label created when an output file is opened or closed, in part by IOCS using the VOL and TPLAB or TLBL control statements. The first three characters are HDR (header), EOV (end-of-volume), or EOF (end-of-file). The fourth character is a 1, indicating the first of a possible eight labels. The remainder of the label is formatted into fields describing the file. Labels 2 through 8 in this field are bypassed on input, and are not created on output under the Disk Operating System. The contents of the fields of a standard file label are described in "Appendix B: Standard Tape File Labels." The relationship between the TPLAB statement and a standard file label is shown in Figure 34. The relationship between the TLBL statement and a standard file label is shown in Figures 35 and 36.

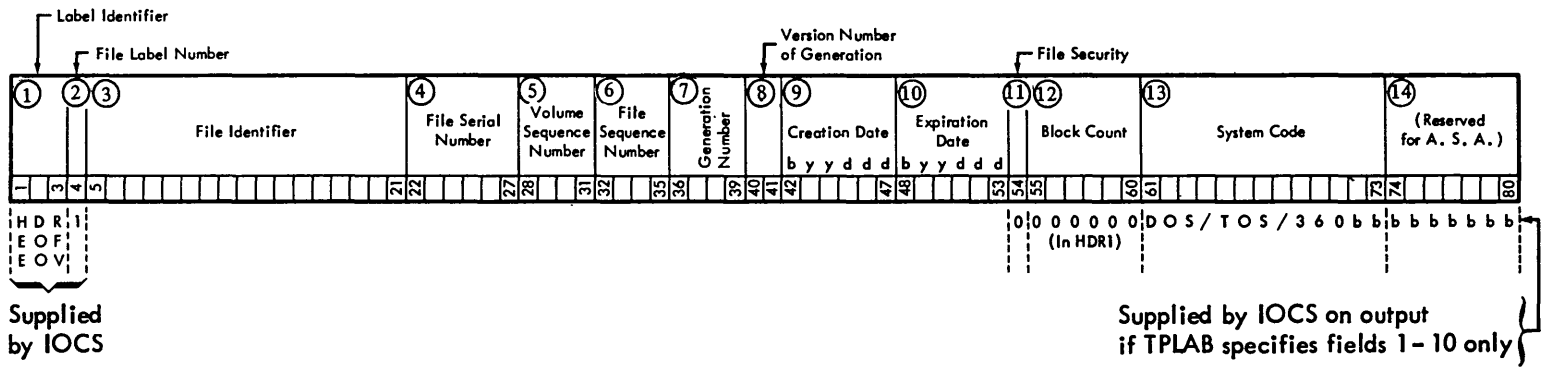
User Labels

A user standard label is an 80-character label having UHL (user header label) or UTL (user trailer label) in the first three positions. The fourth position contains a number 1 through 8 which represents the relative position of the user label within a group of user labels. The contents of the remaining 76 positions are entirely up to the user. User labels, if present, follow HDR, EOV, or EOF standard labels. On multivolume files, they may also appear at beginning-of-volume. User header labels are resequenced starting with one (UHL1) at the beginning of a new volume. Figure 37 shows the positioning of user labels on a file.

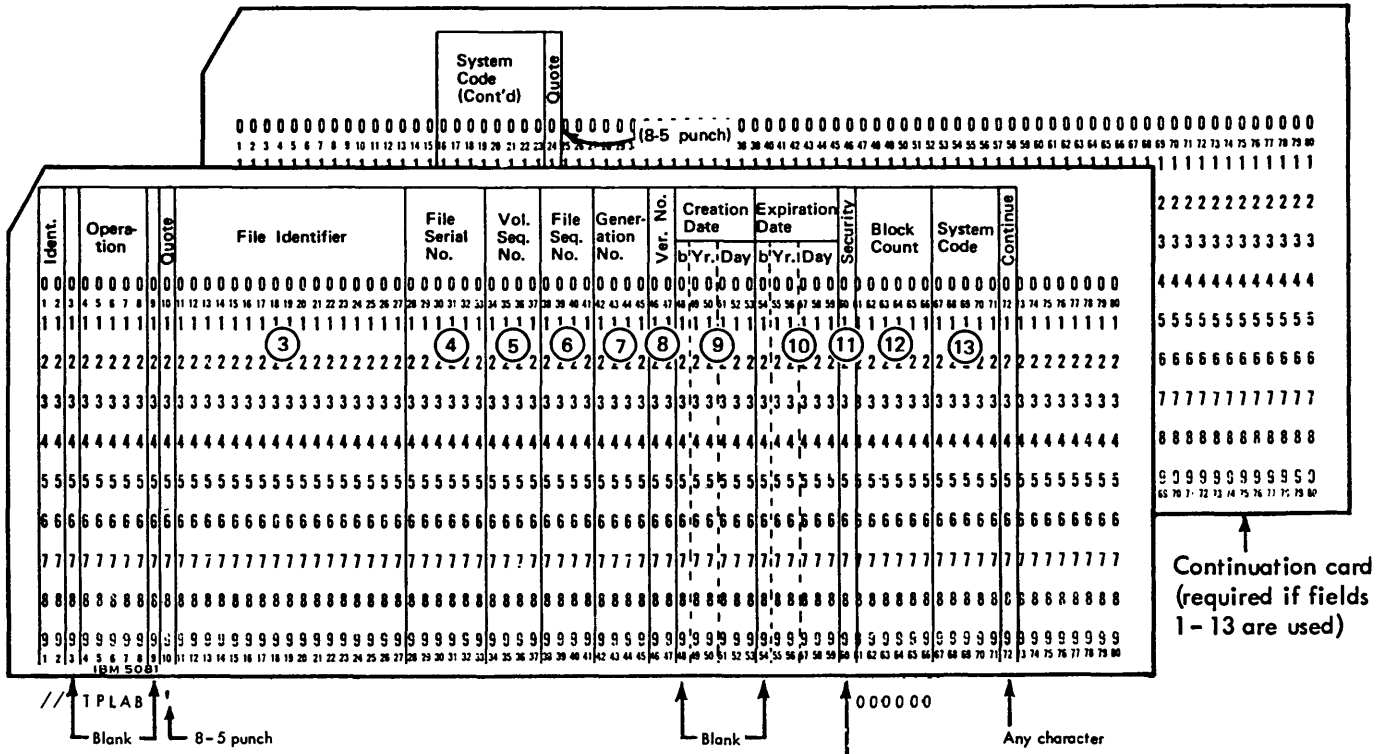
Nonstandard Labels

A nonstandard label may be any length. The contents of a nonstandard label is entirely user-dependent. It is the COBOL user's responsibility either to process or bypass nonstandard labels on input and to create them on output. Figure 38 shows the positioning of nonstandard labels on a file.

Standard Tape File Label



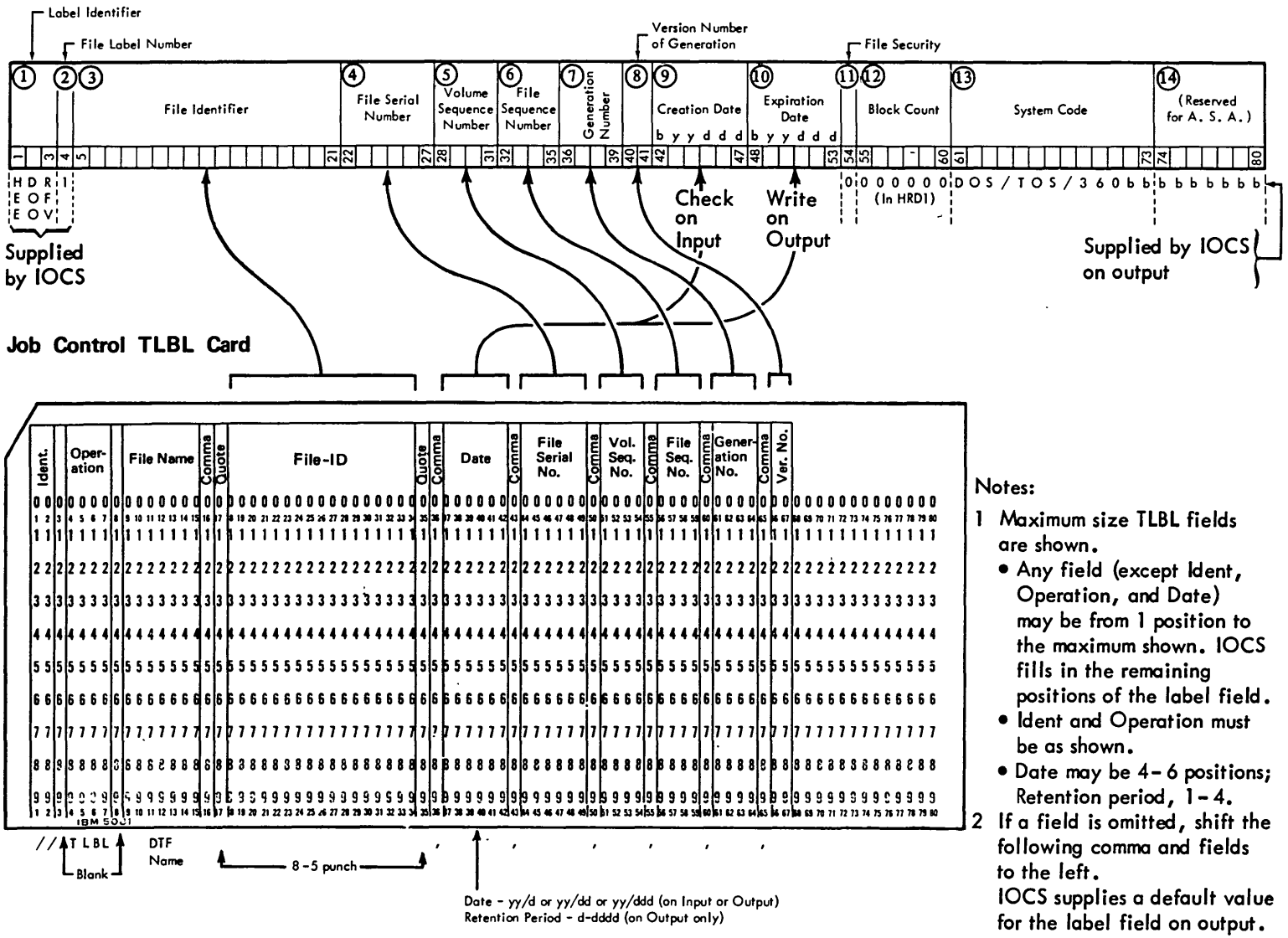
Job Control TPLAB Card(s) (May be punched with fields 1 - 10 or 1 - 13)



The circled numbers on the TPLAB cards correspond to the numbered fields of the tape label above.

Figure 34. Standard Tape File Label and TPLAB Cards

Standard Tape File Label



Notes:

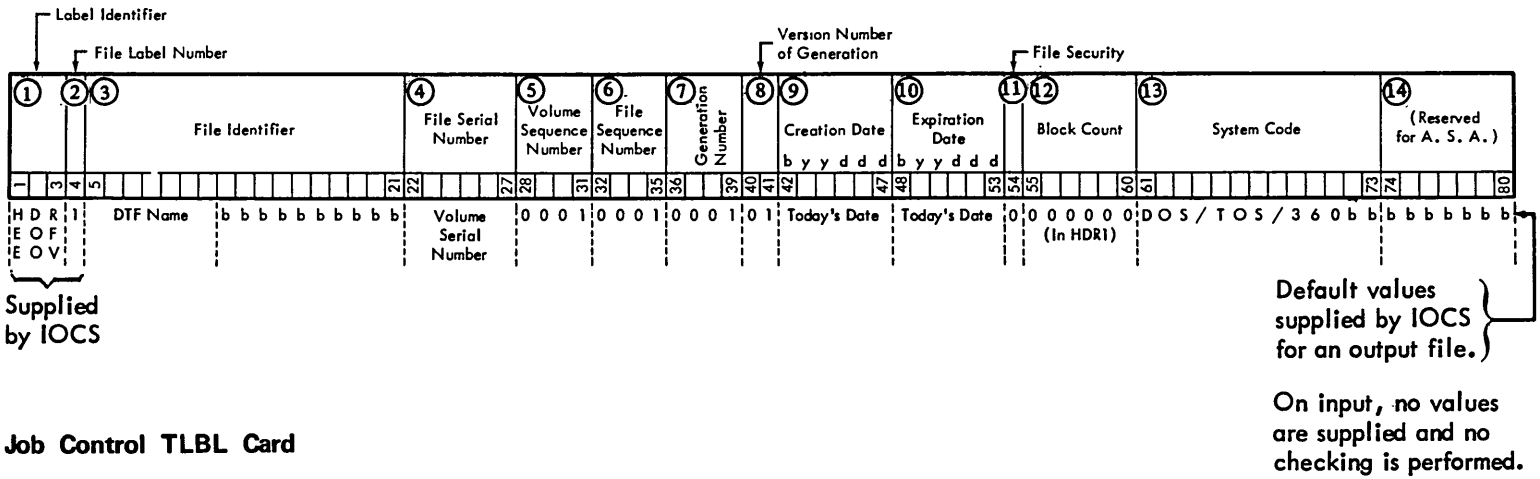
1 Maximum size TLBL fields are shown.
- Any field (except Ident, Operation, and Date) may be from 1 position to the maximum shown. IOCS fills in the remaining positions of the label field.
- Ident and Operation must be as shown.
- Date may be 4-6 positions; Retention period, 1-4.

2 If a field is omitted, shift the following comma and fields to the left. IOCS supplies a default value for the label field on output.

3 No comma follows the last field used.

Figure 35. Standard Tape File Label and TLBL Card (Showing Maximum Specifications)

Standard Tape File Label



Job Control TLBL Card

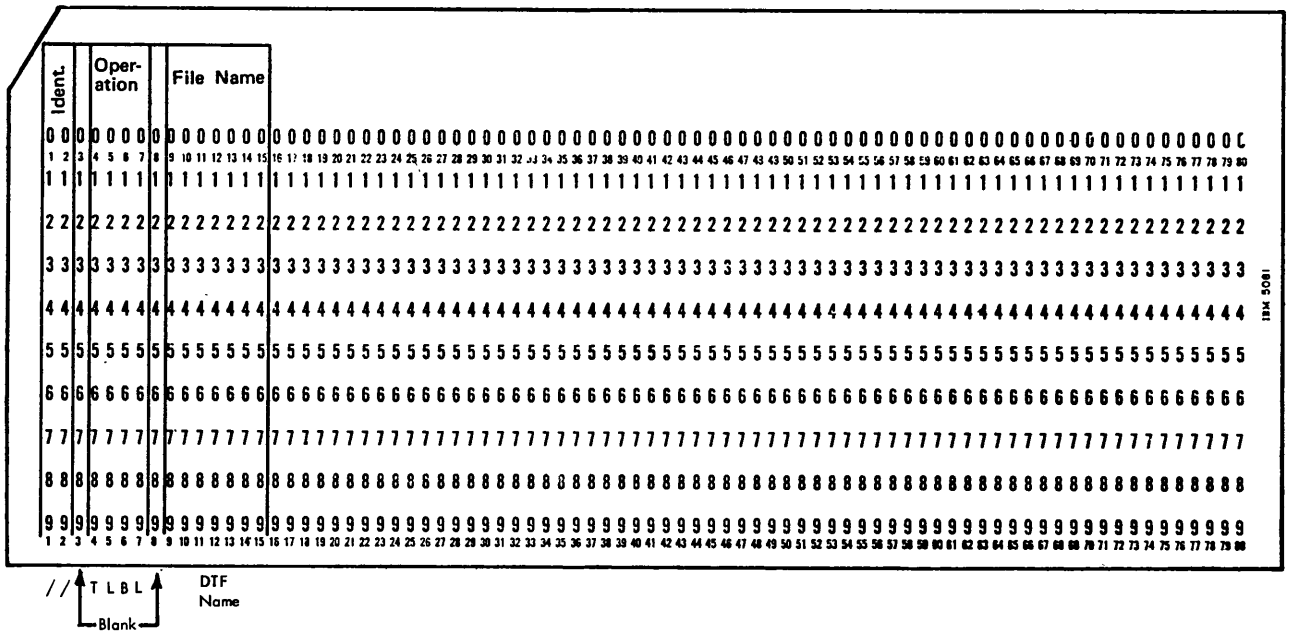


Figure 36. Standard Tape File Label and TLBL Card (Showing Minimum Requirements)

LABEL PROCESSING CONSIDERATIONS

The labels which may appear on tape are shown in Figures 37 and 38. The compiler allows the user to work with all the previously mentioned labels as well as with unlabeled files.

If user labels are to be created or checked in the COBOL program, the USE AFTER BEGINNING/ENDING LABELS declarative sentence and the LABEL RECORDS clause with the data-name option must be specified.

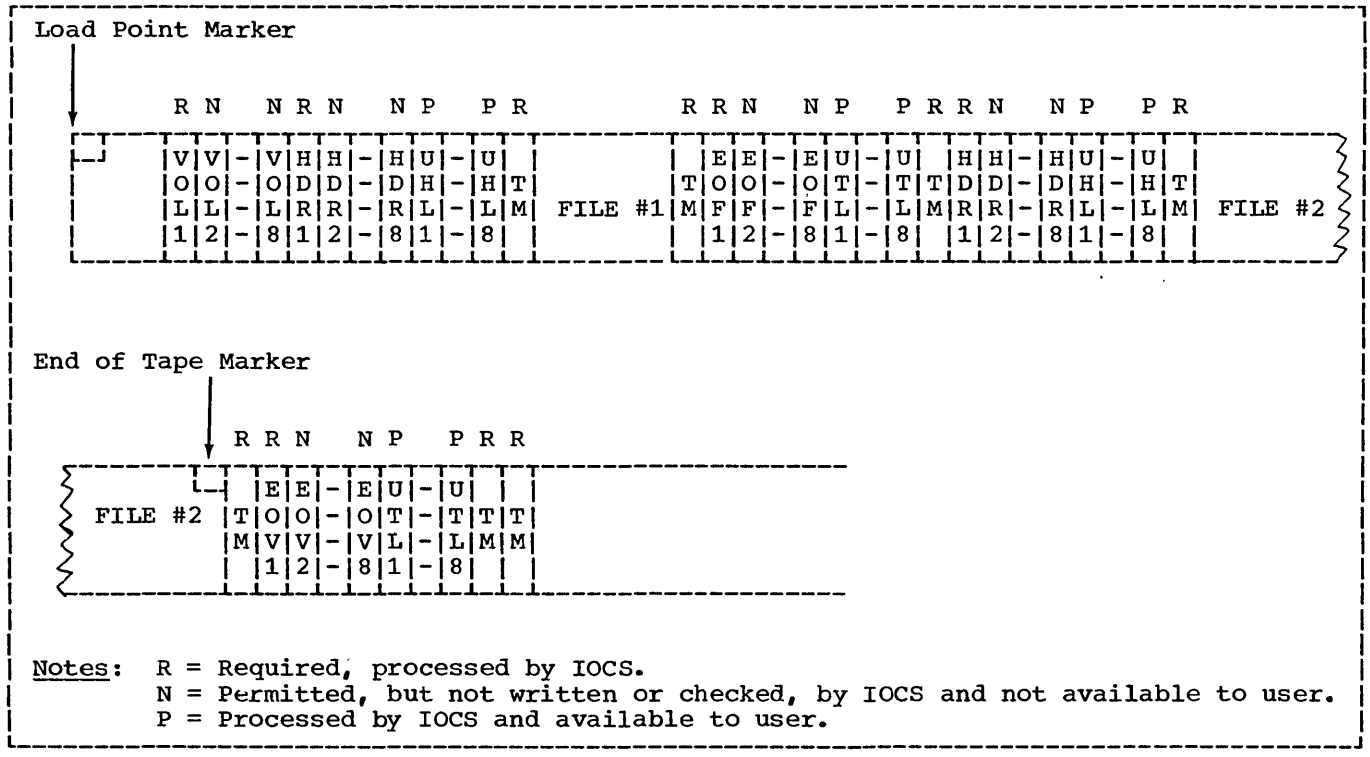


Figure 37. Standard, User, and Volume Labels

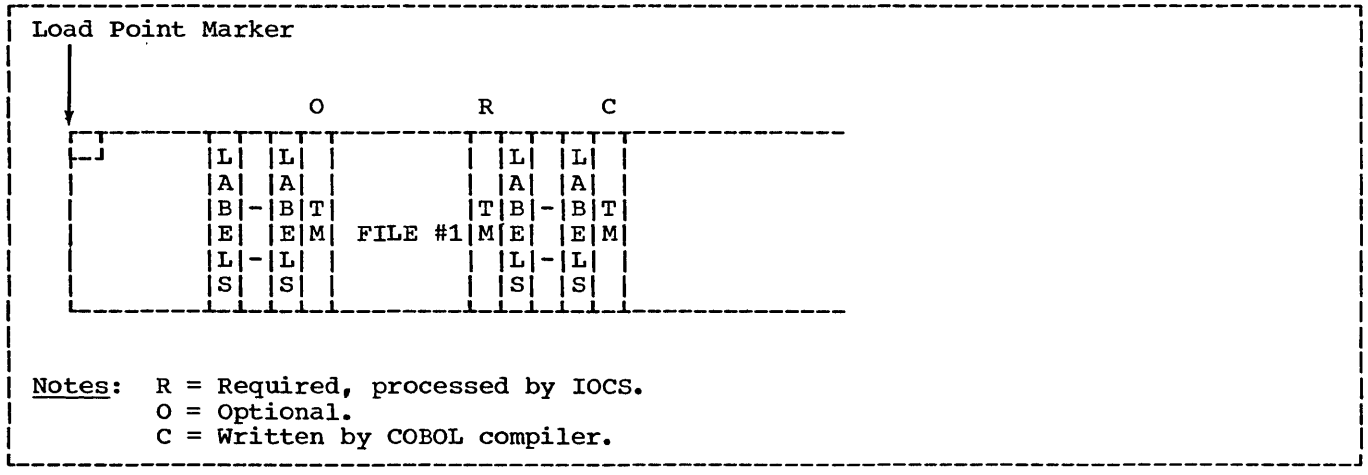


Figure 38. Nonstandard Labels

Header labels are written or read when the file is opened or when a volume switch occurs. Trailer labels are written when the physical end of the reel is reached, or when a CLOSE REEL or CLOSE file-name is issued. Trailer labels are read on each reel except the last when a tapemark is reached. For the last reel (i.e., EOF labels), trailer labels are not read until the file is closed.

For multivolume input files with nonstandard labels, the user must specify the integer-1 option of the source language ASSIGN clause, where integer-1 is the number of reels in the file. This number can be overridden at execution time by storing a nonzero integer in the special register NSTD-REELS before opening the file. Therefore, the number of reels is available to the programmer while the file is opened both in the special register NSTD-REELS and in the field reserved for this purpose which precedes the DTF table for DTFMT (see "DTF Tables" in this chapter). In addition, the number of reels remaining after each volume switch can also be found in the field reserved for this purpose which precedes the DTF table for DTFMT.

When processing a multivolume file with nonstandard labels (i.e., when the data-name option of the LABEL RECORDS clause is specified), if the user wishes to stop reading or writing before the physical end of a reel is reached, he must set a switch in the appropriate declarative section. In the Procedure Division, he can either CLOSE REEL or CLOSE FILE depending on the switch setting. Volume switching is done by LIOCS when CLOSE REEL is executed.

MASS STORAGE FILE LABELS

The IBM System/360 Disk Operating System provides postive identification and protection of all files on mass storage devices by recording labels on each volume. These labels ensure that the correct volume is used for input, and that no current information is destroyed on output.

The mass storage labels always include one volume label for each volume and one or more file labels for each logical file on the volume. There may also be user header labels and user trailer labels.

Volume Labels

The volume label is an 80-byte data field preceded by a 4-byte key field. Both the key field and the first four bytes of the data field contain the label identifier VOL1. IOCS creates a standard volume label for every volume processed by the Disk Operating System. It is always the third record on cylinder 0, track 0. The format and contents of a standard volume label can be found in the publication IBM System/360 Disk Operating System: Data Management Concepts.

Standard File Labels

A standard file label identifies a particular logical file, gives its location(s) on the mass storage device, and contains information to prevent premature destruction of current files. A standard file label for a file located on a mass storage device is a 140-character label created (OPEN/CLOSE OUTPUT) in part by IOCS using the VOL and DLAB, or DLBL control statements. The fields contained within the label follow three standard formats.

1. Format 1 is used for all logical files. The contents of the fields of a Format 1 label is discussed in "Appendix C: Standard Mass Storage Device Labels."

2. Format 2 is required for indexed files. The contents of the fields of a Format 2 label can be found in the publication IBM System/360 Disk Operating System: Data Management Concepts.

3. Format 3 is required if a logical file uses more than three extents of any volume. The contents of the fields of a Format 3 label can be found in the DOS Data Management Concepts publication cited previously.

User Labels

The user can include additional labels to further define his file. The labels are referred to as user standard labels. They cannot be specified for indexed files. A user label is an 80-character label containing UHL (user header label) or UTL (user trailer label) in the first three character positions. The fourth position contains a number 1 through 8 which represents the relative position of the user label with a group of user labels. The contents of the remaining 76 positions is entirely up to the user. User header and trailer labels are written on the first track of the first extent of each volume allocated by the user for the file. User header labels are resequenced starting with one (UHL1) at the beginning of each new volume.

LABEL PROCESSING CONSIDERATIONS

Files on Mass Storage Device Opened as Input

1. Standard labels checked

   a. The volume serial numbers in the volume labels are compared to the file serial numbers in the EXTENT (or XTENT) cards.

   b. Fields 1 through 3 in Format 1 label are compared to the corresponding fields in the DLBL (or DLAB) card. Fields 4 through 6 are then checked against their EBCDIC equivalents in the DLAB continuation card.

   c. Each of the extent definitions in the Format 1 and Format 3 labels is checked against the limit fields supplied in the EXTENT (or XTENT) cards.

2. User labels checked

   a. If user header labels are indicated for directly or sequentially organized files, they are read as each volume of the file is opened. After reading each label, the OPEN routine branches to the user's label routine if the appropriate USE AFTER STANDARD LABEL PROCEDURE declarative is specified in the source program. The LABEL RECORDS clause with the data-name option must be specified in the Data Division. The user's label routine then performs any processing required.

   b. If user trailer labels are indicated on a sequential file, they are read after reaching the end of the last extent on each volume when the file is closed, provided end-of-file has been reached. Trailer labels are processed by the user's label routine if the appropriate USE AFTER STANDARD LABEL PROCEDURE declarative is specified in the source program. The LABEL RECORDS clause with the data-name option must be specified in the Data Division.

Files on Mass Storage Devices Opened as Output

1. Standard labels created

   a. The volume serial numbers in the volume labels are compared to the file serial numbers in the EXTENT (or XTENT) cards.

   b. The extent definitions in all current labels on the volume are checked to determine whether any extend into those defined in the EXTENT (or XTENT) cards. If any overlap, the expiration date is checked against the current date in the Communication Region of the Supervisor. If the expiration date has passed, the old labels are deleted. If not, the operator is notified of the condition.

   c. The new Format 1 label is written with information supplied in the DLBL card (or the DLAB card and the DLAB continuation card). If an indexed file is being processed, the DTFIS routine supplies information for the Format 2 label.

   d. The information in the EXTENT (or XTENT) cards is placed in the Format 1 labels and, if necessary, in the additional Format 3 labels.

2. User header labels created

   a. If user header labels are indicated by the presence of the appropriate USE AFTER STANDARD LABEL PROCEDURE declarative and the LABEL RECORDS clause with the data-name option, the user's label

routine is entered to furnish the labels as each volume of the file is opened. This can be done for as many as eight user header labels per volume. As each label is presented, IOCS writes it out on the first track of the first extent of the volume.

b. If user trailer labels are indicated by the presence of the appropriate USE AFTER STANDARD LABEL PROCEDURE declarative and the LABEL RECORDS clause with the data-name option, the user's label routine is entered to furnish the labels when the end of the last extent on each volume is reached. This can be done for as many as eight user trailer labels. The CLOSE statement must be issued to create trailer labels for the last volume of a sequential file or for a direct file.

UNLABELED FILES

When a multivolume tape file is opened as INPUT and integer as specified in the ASSIGN clause is greater than 1, the compiler will generate the following message to the operator:

```
C126D IS IT EOF?
```

The operator must respond either with N if it is not the last reel, or with Y if it is the last reel. If it is end-of-file, control passes to the imperative-statement specified in the AT END phrase of the READ statement; if it is not end-of-file, processing of the next volume is initiated.

If the integer specified in the ASSIGN clause is not greater than 1, control always passes at end-of-volume to the imperative-statement specified in the AT END phrase of the READ statement.

Logical records may be in one of four formats: fixed-length (format F), variable-length (format V), undefined (format U), or spanned (format S). F-mode files must contain records of equal lengths. Files containing records of unequal lengths must be V-mode, S-mode, or U-mode. Files containing logical records that are longer than physical records must be S-mode.

The record format is specified in the RECORDING MODE clause in the Data Division. If this clause is omitted, the compiler determines the record format from the record descriptions associated with the file. If the file is to be blocked, the BLOCK CONTAINS clause must be specified in the Data Division.

The prime consideration in the selection of a record format is the nature of the file itself. The programmer knows the type of input his program will receive and the type of output it will produce. The selection of a record format is based on this knowledge as well as an understanding of the type of input/output devices on which the file is written and of the access method used to read or write the file.

FIXED-LENGTH (FORMAT F) RECORDS

Format F records are fixed-length records. The programmer specifies format F records by including RECORDING MODE IS F in the file description entry in the Data Division. If the clause is omitted and both of the following are true:

- All records in the file are the same size
- BLOCK CONTAINS [integer-1 TO] integer-2... does not specify integer-2 less than the length of the maximum level-01 record

the compiler determines the recording mode to be F. All records in the file are the same size if there is only one record description associated with the file and it contains no OCCURS clause with the DEPENDING ON option, or if multiple record descriptions are all the same length.

The number of logical records within a block (blocking factor) is normally constant for every block in the file. When fixed-length records are blocked, the programmer specifies the BLOCK CONTAINS clause in the file description entry in the Data Division.

In unblocked format F, the logical record constitutes the block. The BLOCK CONTAINS clause is unnecessary for unblocked records.

Format F records are shown in Figure 39. The optional control character, represented by C in Figure 39, is used for stacker selection and carriage control. When carriage control or stacker selection is desired, the WRITE statement with the ADVANCING or POSITIONING option is used to write records on the output file. In this case one character position must be included as the first character of the record. This position will be automatically filled in with the carriage control or stacker select character. The carriage control character never appears when the file is written on the printer or punched on the card punch.

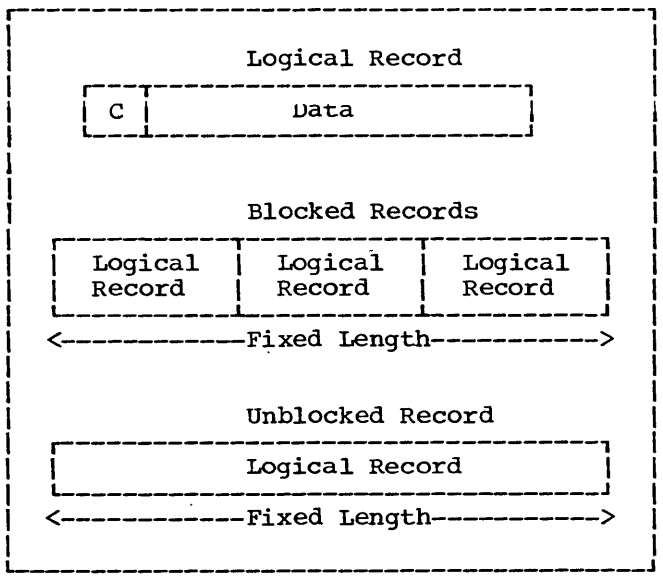


Figure 39. Fixed-Length (Format F) Records

UNDEFINED (FORMAT U) RECORDS

Format U is provided to permit the processing of any blocks that do not conform to F or V formats. Format U records are shown in Figure 40. The optional control character C, as discussed

under "Fixed-Length (Format F) Records," may be used in each logical record.

The programmer specifies format U records by including RECORDING MODE IS U in the file description entry in the Data Division. U-mode records may be specified only for directly organized or standard sequential files.

If the RECORDING MODE clause is omitted, and BLOCK CONTAINS [integer-1 TO] integer-2... does not specify integer-2 less than the maximum level-01 record, the compiler determines the recording mode to be U if the file is directly organized and one of the following conditions exist:

- The FD entry contains two or more level-01 descriptions of different lengths.
- A record description contains an OCCURS clause with the DEPENDING ON option.
- A RECORD CONTAINS clause specifies a range of record lengths.

Each block on the external storage media is treated as a logical record. There are no record-length or block-length fields.

Note: When a READ INTO statement is used for a U-mode file, the size of the longest record for that file is used in the MOVE statement. All other rules of the MOVE statement apply.

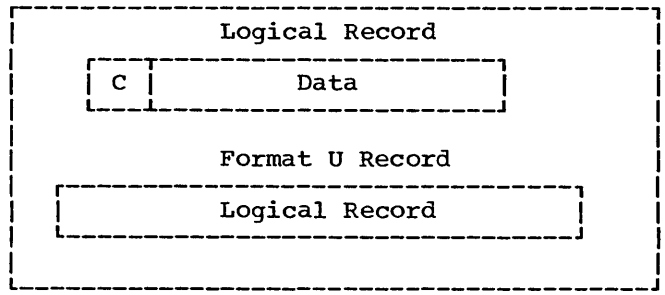


Figure 40. Undefined (Format U) Records

VARIABLE-LENGTH (FORMAT V) RECORDS

The programmer specifies format V records by including RECORDING MODE IS V in the file description entry in the Data Division. V-mode records may only be specified for standard sequential files. If the RECORDING MODE clause is omitted and BLOCK CONTAINS [integer-1] TO integer-2... does not specify integer-2 less than the maximum level-01 record, the compiler determines the recording mode to be V if the file is standard sequential and one of the following conditions exists:

- The FD entry contains two or more level 01 descriptions of different lengths.
- A record description contains an OCCURS clause with the DEPENDING ON option.
- A RECORD CONTAINS clause specifies a range of record lengths.

V-mode records, unlike U-mode or F-mode records, are preceded by fields containing control information. These control fields are illustrated in Figures 41 and 42.

The first four bytes of each block contain control information (CC):

LL -- represents two bytes designating the length of the block (including the 'CC' field).

BB -- represents two bytes reserved for system use.

The first four bytes of each logical record contain control information (cc):

ll -- represents two bytes designating the logical record length (including the 'cc' field).

bb -- represents two bytes reserved for system use.

For unblocked V mode records (see Figure 37) the data portion + CC + cc constitute the block.

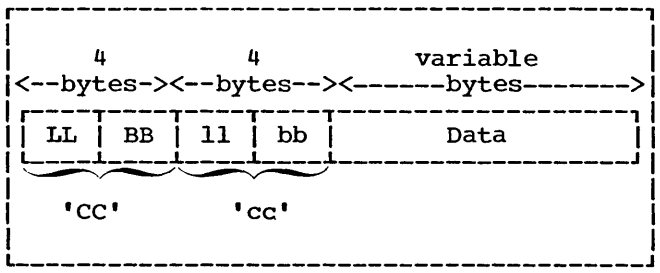


Figure 41. Unblocked V-Mode Records

For blocked V-mode records (see Figure 42) the data portion of each record + the cc of each record + CC constitute the block.

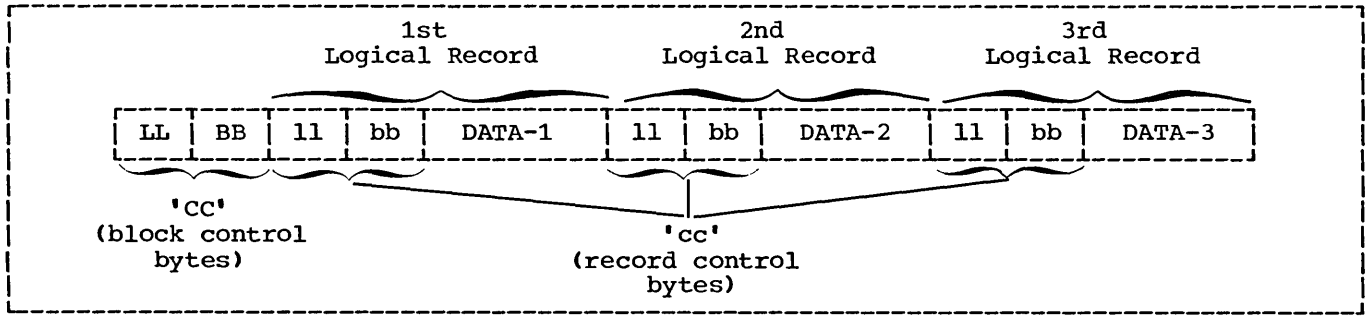


Figure 42. Blocked V-Mode Records

The control bytes are automatically provided when the file is written and are not communicated to the user when the file is read. Although they do not appear in the description of the logical record provided by the user, the compiler will allocate input and output buffers which are large enough to accomodate them. When variable-length records are written on unit record devices, control bytes are neither printed nor punched. They appear, however, on other external storage devices as well as in buffer areas of core storage. V-mode records moved from an input buffer to a working-storage area will be moved without the control bytes.

Note: When a READ INTO statement is used for a V-mode file, the size of the longest record for that file is used in the MOVE statement. All other rules of the MOVE statement apply.

Example 1:

Consider the following standard sequential file consisting of unblocked V-mode records:

```
FD  VARIABLE-FILE-1
    RECORDING MODE IS V
    BLOCK CONTAINS 35 TO 80 CHARACTERS
    RECORD CONTAINS 27 TO 72 CHARACTERS
    DATA RECORD IS VARIABLE-RECORD-1
    LABEL RECORDS ARE STANDARD.

01  VARIABLE-RECORD-1.
    05  FIELD-A      PIC X(20).
    05  FIELD-B      PIC 99.
    05  FIELD-C  OCCURS 1 TO 10 TIMES
        DEPENDING ON
        FIELD-B      PIC 9(5).
```

The LABEL RECORDS clause is always required. The DATA RECORD(S) clause is never required. If the RECORDING MODE clause is omitted, the compiler determines the mode as V since the record associated with VARIABLE-FILE-1 varies in length depending on the contents of FIELD-B. The RECORD CONTAINS clause is never required. The compiler determines record sizes from the record description entries. The BLOCK CONTAINS clause is also unnecessary, since the compiler assumes unblocked records if the clause is omitted. Record length calculations are affected by the following:

- When the BLOCK CONTAINS clause with the RECORDS option is used, the compiler adds four bytes to the logical record length and four more bytes to the block length.

- When the BLOCK CONTAINS clause with the CHARACTERS option is used, the user must include each cc + CC in the length calculation (see Figure 42). In the definition of VARIABLE-FILE-1, the BLOCK CONTAINS clause specifies 8 more bytes than does the record contains clause. Four of these bytes are the logical record control bytes and the other four are the block control bytes.

Assumming that FIELD-B contains the value 02 for the first record of a file and FIELD-B contains the value 03 for the second record of the file, the first two records will appear on an external storage device and in buffer areas of core storage as shown in Figure 43.

If the file described in Example 1 had a blocking factor of 2, the first two records would appear on an external storage medium as shown in Figure 44.

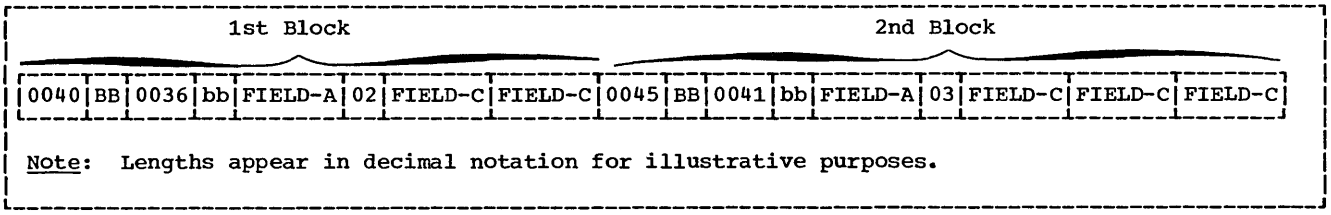


Figure 43. Fields in Unblocked V-Mode Records

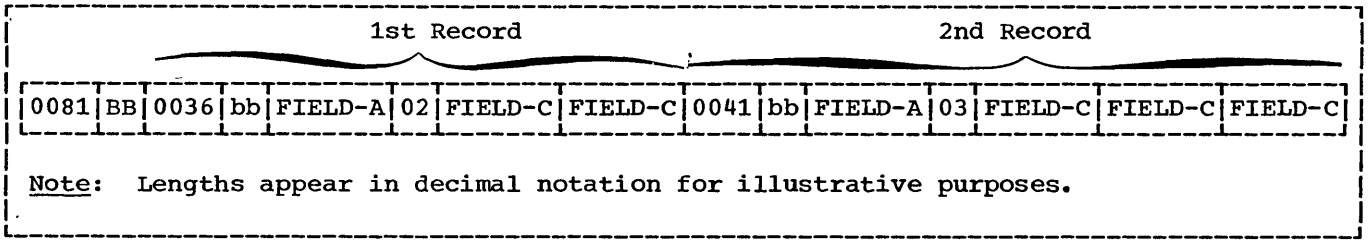


Figure 44. Fields in Blocked V-Mode Records

Example 2:

If VARIABLE-FILE-2 is blocked, with space allocated for three records of maximum size per block, the following FD entry could be used when the file is created:

```
FD  VARIABLE-FILE-2
    RECORDING MODE IS V
    BLOCK CONTAINS 3 RECORDS
    RECORD CONTAINS 20 TO 100 CHARACTERS
    DATA RECORDS ARE VARIABLE-RECORD-1,
    VARIABLE-RECORD-2
    LABEL RECORDS ARE STANDARD.

01  VARIABLE-RECORD-1.
    05  FIELD-A  PIC X(20).
    05  FIELD-B  PIC X(80).

01  VARIABLE-RECORD-2.
    05  FIELD-X  PIC X(20).
```

As mentioned previously, the RECORDING MODE, RECORD CONTAINS, and DATA RECORDS clauses are unnecessary. By specifying that each block contains three records, the programmer allows the compiler to provide space for three records of maximum size plus additional space for the required control bytes. Hence, 316 character positions are reserved by the compiler for each output buffer. If this size is other than that required, the BLOCK CONTAINS clause with the CHARACTERS option should be specified.

Assuming that the first six records written are five 100-character records followed by one 20-character record, the first two blocks of VARIABLE-FILE-2 will appear on the external storage device as shown in Figure 45.

The buffer for the second block is truncated after the sixth WRITE statement is executed since there is not enough space left for a maximum size record. Hence, even if the seventh WRITE to VARIABLE-FILE-2 is a 20-character record, it will appear as the first record in the third block. This situation can be avoided by using the APPLY WRITE-ONLY clause when creating files of variable-length blocked records.

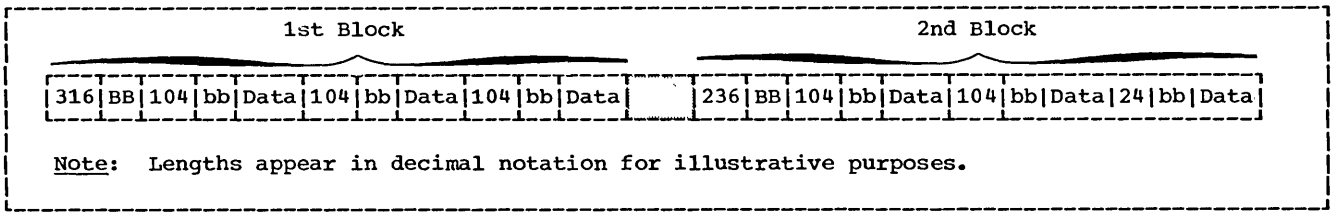


Figure 45. First Two Blocks of VARIABLE-FILE-2

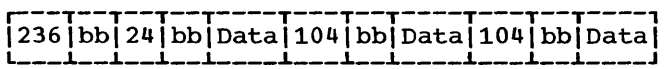
APPLY WRITE-ONLY Clause

The APPLY WRITE-ONLY clause is used to make optimum use of buffer space when creating a standard sequential file with blocked V-mode records.

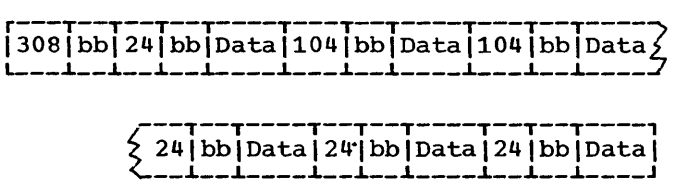
Suppose VARIABLE-FILE-2 is being created with the following FD entry:

```
FD  VARIABLE-FILE-2
    RECORDING MODE IS V
    BLOCK CONTAINS 316 CHARACTERS
    RECORD CONTAINS 20 TO 100 CHARACTERS
    DATA RECORDS ARE VARIABLE-RECORD-1,
    VARIABLE-RECORD-2
    LABEL RECORDS ARE STANDARD.

01  VARIABLE-RECORD-1.
    05  FIELD-A  PIC X(20).
    05  FIELD-B  PIC X(80).

01  VARIABLE-RECORD-2.
    05  FIELD-X  PIC X(20).
```

The first three WRITE statements to the file create one 20-character record followed by two 100-character records. Without the APPLY WRITE-ONLY clause, the buffer is truncated after the third WRITE statement is executed, since the maximum size record no longer fits. The block is written as shown below:



Using the APPLY WRITE-ONLY clause will cause a buffer to be truncated only when the next record does not fit in the buffer. That is, if the next three WRITE statements to the file specify VARIABLE-RECORD-2, the block will be created containing six logical records, as shown below:



Note: When using the APPLY WRITE-ONLY clause, records must not be constructed in buffer areas. An intermediate work area must be used with a WRITE FROM statement.

SPANNED (FORMAT S) RECORDS

A spanned record is a logical record that may be contained in one or more physical blocks. Format S records may be specified for direct files and for standard sequential files assigned to magnetic tape or to mass storage devices.

When creating files with S-mode records, if a record is larger than the remaining space in a block, a segment of the record is written to fill the block. The remainder of the record is stored in the next block or blocks, as required.

When retrieving a file with S-mode records, only complete records are made available to the user.

Spanned records are preceded by fields containing control information. Figure 46 illustrates the control fields.

BDF (Block Descriptor Field):

- LL -- represents 2 bytes designating the length of the physical block (including the block descriptor field itself).
- BB -- represents 2 bytes reserved for system use.

SDF (Segment Descriptor Field):

- ll -- represents 2 bytes designating the length of the record segment (including the segment descriptor field itself).
- bb -- represents 2 bytes reserved for system use.

Note: There is only one block descriptor field at the beginning of each physical block. There is, however, one segment descriptor field for each record segment within the block.

Each segment of a record in a block, even if it is the entire record, is preceded by a segment descriptor field. The segment descriptor field also indicates whether the segment is the first, the last, or an intermediate segment. Each block includes a block descriptor field. These fields are not described in the Data Division; provision is automatically made for them. These fields are not available to the user.

A spanned blocked file may be described as a file composed of physical blocks of fixed length established by the programmer. The logical records may be either fixed or variable in length and that size may be smaller, equal to, or larger than the physical block size. There are no required relationships between logical records and physical block sizes.

A spanned unblocked file may be described as a file composed of physical blocks each containing one logical record or one segment of a logical record. The logical records may be either fixed or variable in length. When the physical block contains one logical record, the length of the block is determined by the logical record size. When a logical record has to be segmented, the system always writes the largest physical block possible. The system segments the logical record when the entire logical record cannot fit on the track.

Figure 47 is an illustration of blocked spanned records of SFILE. SFILE is described in the Data Division with the following file description entry:

```
FD SFILE
   RECORD CONTAINS 250 CHARACTERS
   BLOCK CONTAINS 100 CHARACTERS
   .
   .
   .
```

Figure 47 also illustrates the concept of record segments. Note that the third block contains the last 50 bytes of REC-1 and the first 50 bytes of REC-2. Such portions of logical records are called record segments. It is therefore correct to say that the third block contains the last segment of REC-1 and the first segment of REC-2. The first block contains the first segment of REC-1 and the second block contains an intermediate segment of REC-1.

S-MODE CAPABILITIES

Formatting a file in the S-mode allows the user to make the most efficient use of external storage while organizing data files with logical record lengths most suited to his needs.

1. Physical record lengths can be designated in such a manner as to make the most efficient use of track capacities on mass storage devices.
2. The user is not required to adjust logical record lengths to maximum physical record lengths and their device-dependent variants when designing his data files.
3. The user has greater flexibility in transferring logical records across DASD types.

Spanned record processing will support the 2400 tape series, the 2311 and 2314 disk storage devices, and the 2321 data cell drive.

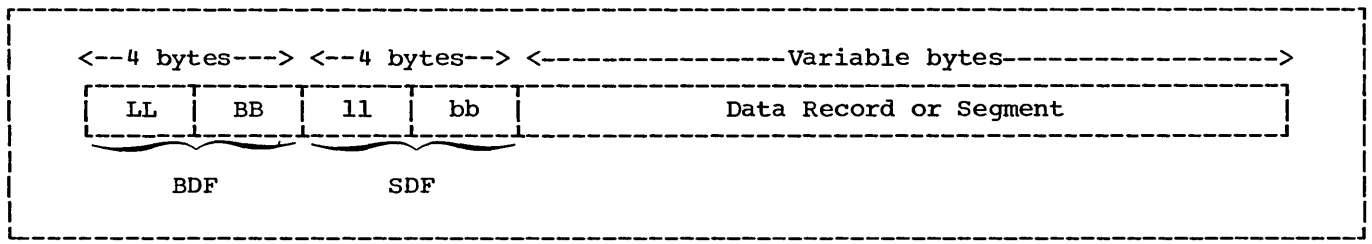
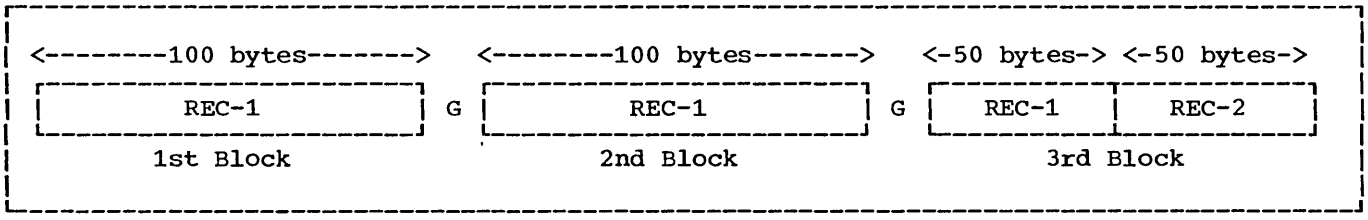


Figure 46. Control Fields of an S-Mode Record



•Figure 47. One Logical Record Spanning Physical Blocks

SEQUENTIALLY ORGANIZED S-MODE FILES ON TAPE OR MASS STORAGE DEVICES

When the spanned format is used for DTFMT or DTFSD files, the logical records may be either fixed or variable in length and are completely independent of physical record length. A logical record may span physical records. A logical record may span physical records. A physical record may contain one or more logical records and/or segments of logical records.

Source Language Considerations

The user specifies S-mode by describing the file with the following clauses in the file description (FD) entry of his COBOL program:

- BLOCK CONTAINS integer-2 CHARACTERS
- RECORD CONTAINS [integer-1 TO] integer-2 CHARACTERS
- RECORDING MODE IS S

The size of the physical record must be specified using the BLOCK CONTAINS clause with the CHARACTERS option. Any block size may be specified. Block size is independent of logical record size.

The size of the logical record may be specified by the RECORD CONTAINS clause. If this clause is omitted, the compiler will determine the maximum record size from the record descriptions under the FD.

Format S may be specified by the RECORDING MODE IS S clause. If this clause is omitted, the compiler will set the recording mode to S if the BLOCK CONTAINS integer-2 CHARACTERS clause was specified and either:

1. integer-2 is less than the largest fixed-length level-01 FD entry
2. integer-2 is less than the maximum length of a variable level-01 FD entry (i.e., an entry containing one or more OCCURS clauses with the DEPENDING ON option).

When the spanned recording mode is being used, each logical record is processed in a work area, not in the buffer. Logical records are always aligned on a double-word boundary. Therefore, the user is not required to add inter-record slack bytes for alignment purposes.

Except for the APPLY WRITE-ONLY clause, all the options for a variable file apply to a spanned file.

Processing Sequentially Organized S-Mode Files

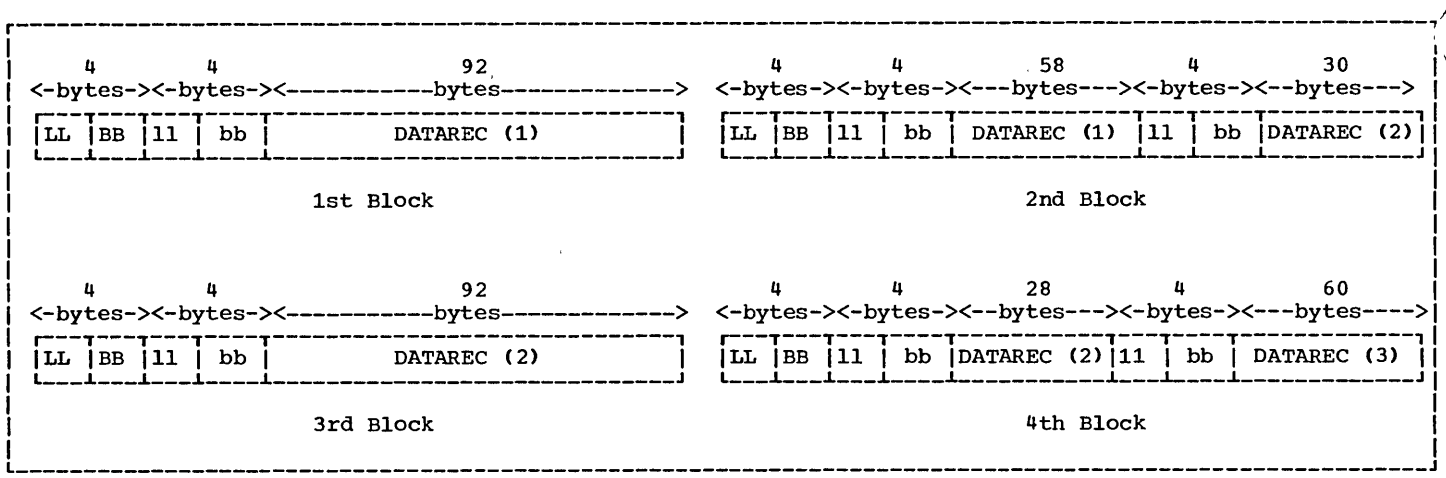
Suppose a file has the following file description entry:

```
FD  SPAN-FILE
    BLOCK CONTAINS 100 CHARACTERS
    LABEL RECORDS ARE STANDARD
    DATA RECORD IS DATAREC.


01  DATAREC.
    05 FIELD-A PIC X(100).
    05 FIELD-B PIC X(50).
```

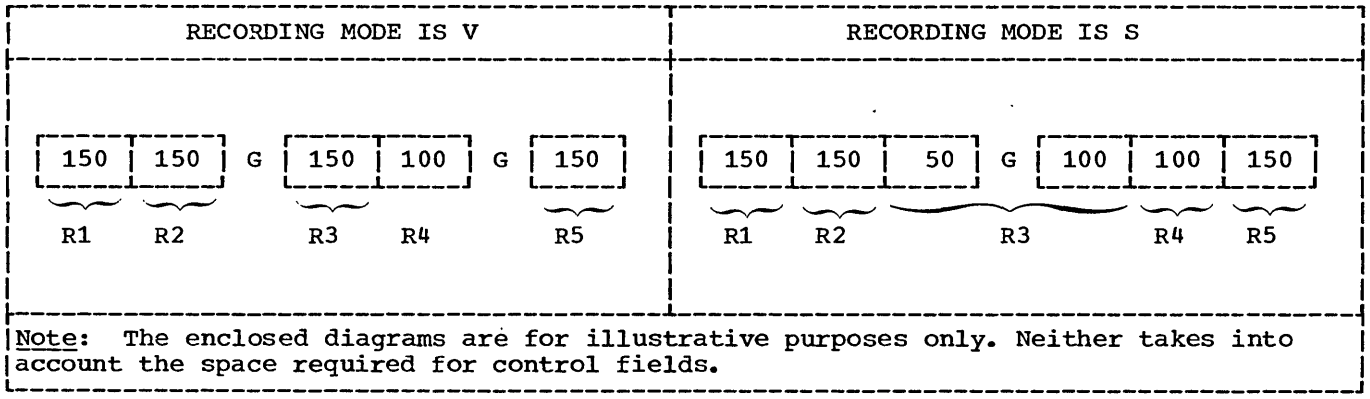
Figure 48 illustrates the first four blocks of SPAN-FILE as they would appear on external storage devices (i.e., tape or mass storage) or in buffer areas of core storage.

Note:

1. The RECORDING MODE clause is not specified. The compiler determines the recording mode to be S since the block size is less than the record size.
2. The length of each physical block is 100 bytes, as specified in the BLOCK CONTAINS clause. All required control fields, as well as data, must be contained within these 100 bytes.
3. No provision is made for the control fields within the level-01 entry DATAREC.



• Figure 48. First Four Blocks of SPAN-FILE



• Figure 49. Advantage of S-Mode Records Over V-Mode Records

The preceding discussion dealt with S-mode records which were larger than the physical blocks that contained them. It is also possible to have S-mode records which are equal to or smaller than the physical blocks that contain them. In such cases, the RECORDING MODE clause must specify S (if so desired) since the compiler cannot determine this by comparing block size and record size.

One advantage of S-mode records over V-mode records is illustrated by a file with the following characteristics:

1. RECORD CONTAINS 50 TO 150 CHARACTERS
2. BLOCK CONTAINS 350 CHARACTERS
3. The first five records written are 150, 150, 150, 100, and 150 characters in length.

For V-mode records, buffers are truncated if the next logical record is too large to be completely contained in the block (see Figure 49). This results in more physical blocks and more inter-record gaps on the external storage device.

Note: For V-mode records, buffer truncation occurs:

1. when the maximum level-01 record is too large
2. if APPLY WRITE-ONLY or SAME RECORD AREA is specified and the actual logical record is too large

For S-mode records, all blocks are 350 bytes long and records that are too large to fit entirely into a block will be segmented. This results in more efficient use of external storage devices since the

number of inter-record gaps are minimized (Figure 49).

With the exception of the last block, the actual physical block size will always fall between the limits of specified block size and four bytes less than the specified block size, depending on whether or not the residual space of an incomplete block in the buffer is sufficient to add a segment length field and at least one byte of data. That is, specified block size - 4 ≤ actual block size ≤ specified block size.

The last block may be short when an incomplete block remains in the buffer at CLOSE time.

A second advantage of S-mode processing over that of V-mode is that the user is no longer limited to a record length that does not exceed the track capacity of the mass storage device selected. Records may span track, cylinders, and extents, but not volumes.

DTFMT and DTFSD spanned records differ from other formats because of an allocation of an area of core know as the "logical record area." If logical records span physical blocks, COBOL will use this logical record area to assemble complete logical records. If logical records do not span blocks (i.e., they are contained within a single physical block) the logical record area is not used. Regardless, it is complete logical records that are made available to the user. Both READ and WRITE statements should be thought of as manipulating complete logical records and not record segments.

DIRECTLY ORGANIZED S-MODE FILES

When S-mode is used for a directly organized file, only unblocked records are permitted. Logical records may be either fixed or variable in length. A logical record will span physical records if, and only if, it spans tracks. A physical record will contain only one logical record or a segment of a logical record, or segments of two logical records and/or whole logical records. Records may span tracks, cylinders, and extents, but not volumes.

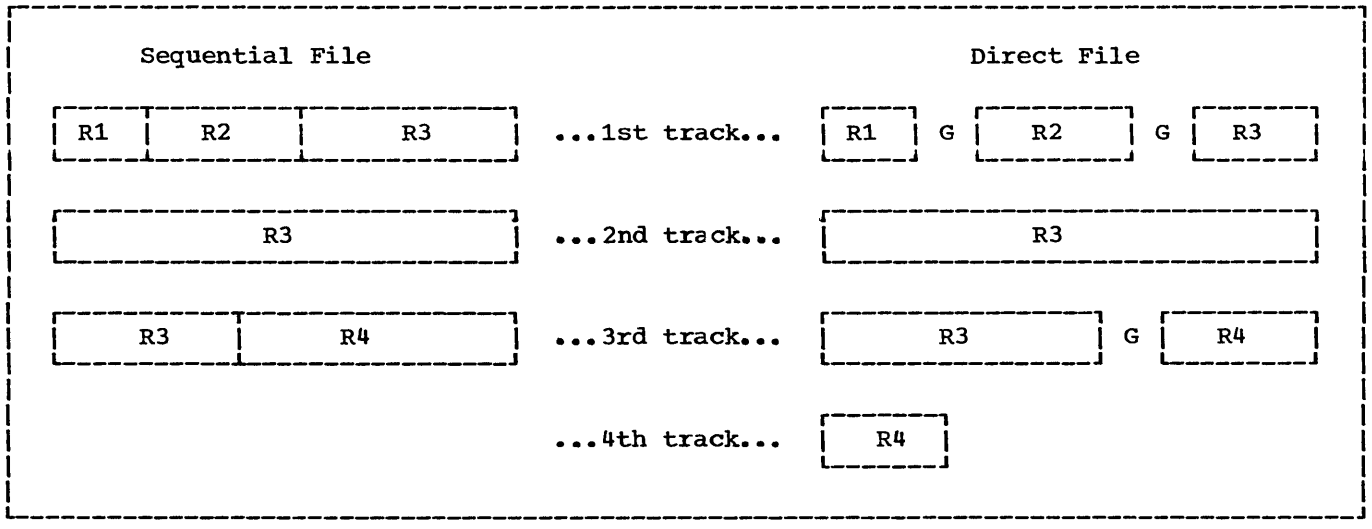
Source Language Considerations

The user specifies S-mode by describing the file with the following clauses in the file description (FD) entry of his COBOL program:

- BLOCK CONTAINS integer-2 CHARACTERS
- RECORD CONTAINS [integer-1 TO] integer-2 CHARACTERS
- RECORDING MODE IS S

The size of a logical record may be specified by the RECORD CONTAINS clause. If this clause is omitted, the compiler will determine the maximum record size from the record descriptions under the FD.

The spanned format may be specified by the RECORDING MODE IS S clause. If this clause is omitted, the compiler will set the recording mode to S if the BLOCK



• Figure 50. Direct and Sequential Spanned Files on a Mass Storage Device

CONTAINS integer-2 CHARACTERS clause was specified and integer-2 is less than the greatest logical record size. This is the only use of the BLOCK CONTAINS clause. It is otherwise treated as comments.

The physical block size is determined by either:

1. the logical record length
2. the track capacity of the device being used

If, for example, the track capacity of a mass storage device is 3,625 characters, any record smaller than 3,625 characters may be written as a single physical block. If a logical record is greater than 3,625 characters, the record is segmented. The first segment may be contained in a physical block of up to 3,625 bytes, and the remaining segments must be contained in succeeding blocks. In other words, a logical record will span physical blocks if, any only if, it spans tracks.

Figure 50 illustrates four variable-length records (R1, R2, R3, and R4) as they would appear in direct and sequential files on a mass storage device. In both cases, control fields have been omitted for illustrative purposes. For both files, assume:

1. BLOCK CONTAINS 3625 CHARACTERS (track capacity = 3,625)
2. RECORD CONTAINS 500 TO 5000 CHARACTERS

In the sequential file, each physical block is 3,625 bytes in length and is completely filled with logical records. The file consists of three physical blocks, occupies three tracks, and contains no inter-record gaps.

In the direct file, the physical blocks vary in length. Each block contains only one logical record or one record segment. Logical record R3 spans physical blocks only because it spans tracks. The file consists of seven physical blocks, occupies more than three tracks, and contains three inter-record gaps.

Processing Directly Organized S-Mode Files

When processing directly organized files, there are two advantages spanned format has over the other record formats:

1. Logical record lengths may exceed the length restriction of the track capacity of the mass storage device.

   If, for example, the track capacity of a mass storage device is 2,000 bytes, the length of each logical record for formats other than spanned is, by necessity, restricted to the track capacity.

   Note: Even when the spanned format is used, the COBOL restriction on the length of logical records (i.e., a maximum length of 32,767 characters) must be adhered to.

2. For formats other than spanned, only complete logical records can be written on any single track. This means that if a track has only 1,000 unoccupied bytes and the user attempts to add a record of 1,100 bytes to this track, an INVALID KEY condition will occur. When the spanned format is used, a 1,000 byte segment will be written on the specified track, and the remainder will be written on the next track. The segmenting is transparent to the user.

OCCURS CLAUSE WITH THE DEPENDING ON OPTION

If a record description contains an OCCURS clause with the DEPENDING ON option, the record length is variable. This is true for records described in an FD as well as in the Working-Storage section. The previous sections discussed four different record formats. Three of them, V-mode, U-mode, and S-mode, may contain one or more OCCURS clauses with the DEPENDING ON option.

This section discusses some factors that affect the manipulation of records containing OCCURS clauses with the DEPENDING ON option. The text indicates whether the factors apply to the File or Working-Storage sections, or both.

The compiler calculates the length of V-mode records containing the OCCURS clause with the DEPENDING ON option at two different times, as follows (the first applies to FD entries only; the second to both FD and working-storage entries):

1. When a file is read and the object of the DEPENDING ON option is within the record.
2. When the object of the DEPENDING ON option is changed as a result of a move to it or any item within its group. (The length is not calculated when a move is made to an item which redefines or renames it.)

Consider the following example:

```
WORKING-STORAGE SECTION.

77  CONTROL-1     PIC 99.
77  WORKAREA-1    PIC 9(6)V99.
        .
        .
        .
01  SALARY-HISTORY.
    05  SALARY OCCURS 0 TO 10 TIMES
        DEPENDING ON
        CONTROL-1  PIC 9(6)V99.
```

The Procedure Division statement MOVE 5 TO CONTROL-1 will cause a recalculation of the length of SALARY-HISTORY. MOVE SALARY (5) TO WORKAREA-1 will not cause the length to be recalculated.

The compiler permits the occurrence of more than one level-01 record, containing the OCCURS clause with the DEPENDING ON option, in the same FD entry (see Figure 51). If the BLOCK CONTAINS clause is omitted, the buffer size is calculated from the longest level-01 record description entry. In Figure 51, the buffer size is determined by the description of RECORD-1 (RECORD-1 need not be the first record description under the FD).

During the execution of a READ statement, the length of each level-01 record description entry in the FD will be calculated (see Figure 51). The length of the variable portion of each record will be the product of the numeric value contained in the object of the DEPENDING ON option and the length of the subject of the OCCURS clause. In Figure 51, the length of FIELD-1 is calculated by multiplying the contents of CONTROL-1 by the length of FIELD-1; the length of FIELD-2, by the product of the contents of CONTROL-2 and the length of FIELD-2; the length of FIELD-3 by the contents of CONTROL-3 and the length of FIELD-3.

Since the execution of a READ statement makes available only one record type (i.e., RECORD-1 type, RECORD-2 type, or RECORD-3 type), two of the three record descriptions in Figure 51 will be inappropriate. In such cases, if the contents of the object of the DEPENDING ON option does not conform to its picture, the length of the corresponding record will not be calculated. For the contents of an item to conform to its picture:

- An item described as USAGE DISPLAY must contain decimal data.
- An item described as USAGE COMPUTATIONAL-3 must contain internal decimal data.
- An item described as USAGE COMPUTATIONAL must contain binary data.

The following example illustrates the length calculations made by the system when a READ statement is executed:

```
FD
    .
    .
    .
01  RECORD-1.
    05  A  PIC  99.
    05  B  PIC  99.
    05  C  PIC  99  OCCURS 5 TIMES
        DEPENDING ON A.

01  RECORD-2.
    05  D  PIC  XX.
    05  E  PIC  99.
    05  F  PIC  99.
    05  G  PIC  99  OCCURS 5 TIMES
        DEPENDING ON F.

WORKING-STORAGE SECTION.
    .
    .
    .
01  TABLE-3.
    05  H OCCURS 10 TIMES DEPENDING ON B.

01  TABLE-4.
    05  I OCCURS 10 TIMES DEPENDING ON E.
```

When a record is read, lengths are determined as follows:

1. The length of RECORD-1 is calculated using the contents of field A.
2. The length of RECORD-2 is calculated using the contents of field F.
3. The length of TABLE-3 is calculated using the contents of field B.
4. The length of TABLE-4 is calculated using the contents of field E.

The user should be aware of several characteristics of the previously cited length calculations. The following example illustrates a group item (i.e., REC-1) whose subordinate items contain an OCCURS clause with the DEPENDING ON option and the object of that DEPENDING ON option.

```
FD  INPUT-FILE
    .
    .
    DATA RECORDS ARE RECORD-1 RECORD-2 RECORD-3.

01  RECORD-1.
    05  CONTROL-1              PIC 99.
    05  FIELD-1 OCCURS 0 TO 10 TIMES DEPENDING ON CONTROL-1  PIC 9(5).

01  RECORD-2.
    05  CONTROL-2              PIC 99.
    05  FIELD-2 OCCURS 1 TO 5 TIMES DEPENDING ON CONTROL-2  PIC 9(4).

01  RECORD-3.
    05  FILLER                 PIC XX.
    05  CONTROL-3              PIC 99.
    05  FIELD-3 OCCURS 0 TO 10 TIMES DEPENDING ON CONTROL-3  PIC X(4).
```

Figure 51. Calculating Record Lengths When Using the OCCURS Clause with the DEPENDING ON Option

```
WORKING-STORAGE SECTION.
01  REC-1.
    05  FIELD-1 PIC 9.
    05  FIELD-2 OCCURS 5 TIMES DEPENDING ON
               FIELD-1  PIC X(5).

01  REC-2.
    05  REC-2-DATA  PIC X(50).
```

The results of executing a MOVE to the group item REC-1 will be affected by the following:

- The length of REC-1 may have been calculated at some time prior to the execution of this MOVE statement.
- The length of REC-1 may never have been calculated at all.
- After the move, since the contents of FIELD-1 have been changed, an attempt will be made to recalculate the length of REC-1. This recalculation, however, will only be made if the new contents of FIELD-1 conform to its picture (i.e., USAGE DISPLAY must contain an external decimal item, USAGE COMPUTATIONAL-3 must contain an internal decimal item and USAGE COMPUTATIONAL must contain a binary item). In the preceding example, if FIELD-1 does not contain an external decimal item, the length of REC-1 will not be calculated.

Note: According to the COBOL description, FIELD-2 can occur a maximum of five times. If, however, FIELD-1 contains an external decimal item whose value exceeds five, the length of REC-1 will still be calculated. One possible consequence of this invalid calculation will be encountered if the user attempts to initialize REC-1 by moving zeros or spaces to it. This initialization would inadvertently delete part of the adjacent data stored in REC-2.

The following discussion applies to updating a record containing an OCCURS clause with the DEPENDING ON option and at least one other subsequent entry. In this case, the subsequent entry is another OCCURS clause with the DEPENDING ON option.

```
WORKING-STORAGE SECTION.
01  VARIABLE-REC.
    05  FIELD-A      PIC X(10).
    05  CONTROL-1    PIC 99.
    05  CONTROL-2    PIC 99.
    05  VARY-FIELD-1 OCCURS 10 TIMES
        DEPENDING ON CONTROL-1 PIC X(5).
    05  VARY-FIELD-2 OCCURS 10 TIMES
        DEPENDING ON CONTROL-2 PIC X(9).

01  STORE-VARY-FIELD-2.
    05  VARY-FLD-2 OCCURS 10 TIMES
        DEPENDING ON CONTROL-2 PIC X(9).
```

Assume that CONTROL-1 contains the value 5 and VARY-FIELD-1 contains 5 entries.

In order to add a sixth field to VARY-FIELD-1 the following steps are required:

```
MOVE VARY-FIELD-2 TO STORE-VARY-FIELD-2.
ADD 1 TO CONTROL-1.
MOVE 'additional field' TO VARY-FIELD-1
  (CONTROL-1).
MOVE STORE-VARY-FIELD-2 TO VARY-FIELD-2.
```

This chapter describes several techniques for increasing the efficiency of a COBOL program. It is divided into six parts. The first four parts deal with the divisions of a COBOL program. The fifth is concerned with the Report Writer Feature, and the sixth with Table Handling Feature.

GENERAL CONSIDERATIONS

Spacing the Source Program Listing

There are four statements that can be coded in any or all of the four divisions of a source program: SKIP1, SKIP2, SKIP3, and EJECT. These statements provide the user with the ability to control the spacing of a source listing and thereby improve its readability.

ENVIRONMENT DIVISION

SELECT Sentence

SELECT sentences for the most active files should appear first, since the COBOL compiler assigns registers to files until it runs out of registers and then reuses the last registers for all subsequent files.

APPLY WRITE-ONLY Clause

To make optimum use of buffer space allocated when creating a standard sequential file with blocked V-mode records, the programmer should use the APPLY WRITE-ONLY clause for the file. Using this clause causes a buffer to be truncated only when the next record does not fit in the buffer. (If APPLY WRITE-ONLY is not specified, the buffer is truncated when the maximum size record will not fit in the space remaining in the buffer.)

DATA DIVISION

OVERALL CONSIDERATIONS

Prefixes

Assign a prefix to each level-01 item in a program, and use this prefix on every subordinate item (except FILLER) to associate a file with its records and work areas. For example, MASTER is the prefix used here:

```
FILE SECTION.
FD  MASTER-INPUT-FILE
    .
    .
    .
01  MASTER-INPUT-RECORD.
    .
    .
    .
WORKING-STORAGE SECTION.
01  MASTER-WORK-AREA.
    05  MASTER-PAYROLL PICTURE 9(3).
    05  MASTER-SSNO PICTURE 9(9).
```

If files or work areas have the same fields, use the prefix to distinguish between them. For example, if three files all have a date field, instead of DATE, DAT, and DA-TE, use MASTER-DATE, DETAIL-DATE, and REPORT-DATE. Using a unique prefix for each level-01 item and all subordinate fields makes it easier for a user unfamiliar with the program to find fields in the program listing, and to know which fields are logically part of the same record or area.

When using the MOVE statement with the CORRESPONDING option and referring to individual fields, redefine or rename "corresponding" names with the prefixed unique names. This technique eliminates excessive qualifying. For example:

```
01  MST-WORK-AREA.
    05  SAME-NAMES.      (***)
        10  LAST-NAME  PIC...
        10  FIRST-NAME  PIC...
        10  PAYROLL  PIC...
        .
        .
        .
    05  DIFF-NAMES REDEFINES SAME-NAMES.
        10  MST-LAST-NAME  PIC...
        10  MST-FIRST-NAME  PIC...
        10  MST-PAYROLL  PIC...
01  RPT-WORK-AREA.
    05  SAME-NAMES.      (***)
        10  PAYROLL  PIC...
        10  FILLER  PIC...
        10  FIRST-NAME  PIC...
        10  FILLER  PIC...
        10  LAST-NAME  PIC...
        .
        .
        .
PROCEDURE DIVISION.
    .
    .
    IF MST-PAYROLL IS EQUAL TO HDQ-PAYROLL
    AND MST-LAST-NAME
    IS NOT EQUAL TO PRRV-LAST-NAME
    MOVE CORRESPONDING
    MST-WORK-AREA
    TO RPT-WORK-AREA.
```

Note: Fields marked *** above must have exactly the same names for their subordinate fields to be considered "corresponding." The same names must not be the redefining ones or they will not be considered to correspond.

Level Numbers

The programmer should use widely incremented level numbers such as 01, 05, 10, 15, etc., instead of 01, 02, 03, 04, etc., in order to allow space for future insertions of group levels. For readability, indent level numbers. Use level number 88 for codes. Thus, if the codes must be changed, the Procedure Division coding for tests need not be changed.

FILE SECTION

RECORD CONTAINS Clause

The programmer should use the RECORD CONTAINS clause with the integer CHARACTERS option in order to save himself, as well as any future programmer, the task of counting the data record description positions. In addition, the compiler can then diagnose errors if the data record description conflicts with the RECORD CONTAINS clause.

WORKING-STORAGE SECTION

Separate Modules

In a large program, the programmer should plan ahead for breaking the programs into separately compiled modules, as follows:

1. When using separate modules, an attempt should be made to combine entries of each Working-Storage Section into a single level-01 record (or a single level-01 record for each 32K bytes). Logical record areas can be indicated by using level-02, -03, etc., entries. A CALL statement with the USING option is more efficient when a single item is passed than when many level-01 and/or -77 items are passed. When this method is employed, mistakes are more easily avoided.

2. Areas which do not contain VALUE clauses should be separated from areas that do contain VALUE clauses. VALUE clauses (except for level-88 items) are invalid in the Linkage Section.

3. When the Working-Storage Section consists of one level-01 item without any VALUE clauses, the COPY statement can easily be used to include the item as the description of a Linkage Section in a separately compiled module.

4. See the chapter "Using the Segmentation Feature" for additional information on how to modularize the Procedure Division of a COBOL program.

Locating the Working-Storage Section in Dumps

A simple method of locating the Working-Storage Section of a program in object-time dumps is to include the two following statements as the first and last Working-Storage statements, respectively, in the program.

```
77  FILLER PICTURE X(44), VALUE "PROGRAM
    XXXXXXXX WORKING-STORAGE BEGINS HERE".

01  FILLER PICTURE X(42), VALUE "PROGRAM
    XXXXXXXX WORKING-STORAGE ENDS HERE".
```

These two nonnumeric literals will appear in all dumps of the program, delimiting the Working-Storage Section. The program-name specified in the PROGRAM-ID clause should replace the XXXXXXXX in the literal.

DATA DESCRIPTION

The Procedure Division operations that most often require adjustment of data items include the MOVE statement, the IF statement when used in a relation test, and arithmetic operations. Efficient use of data description clauses, such as REDEFINES, PICTURE, and USAGE, avoids the generation of extra code.

REDEFINES Clause

REUSING DATA AREAS: The main storage area can be used more efficiently by writing different data descriptions for the same data area. For example, the coding that follows shows how the same area can be used as a work area for the records of several input files that are not processed concurrently.

```
WORKING-STORAGE SECTION.
01  WORK-AREA-FILE1.
     (largest record description for FILE1)
.
.
.
01  WORK-AREA-FILE2 REDEFINES
    WORK-AREA-FILE1.
     (largest record description for FILE2)
.
.
.
```

ALTERNATE GROUPINGS AND DESCRIPTIONS: Program data can often be described more efficiently by providing alternate groupings or data descriptions for the same data. For example, a program references both a field and its subfields, each of which is more efficiently described with a different usage. This can be done by using the REDEFINES clause as follows:

```
01  PAYROLL-RECORD.
    05  EMPLOYEE-RECORD PICTURE X(28).
    05  EMPLOYEE-FIELD REDEFINES
        EMPLOYEE-RECORD.
        10  NAME PICTURE X(24).
        10  NUMBERX PICTURE S9(5) COMP.
    05  DATE-RECORD PICTURE X(10).
```

The following illustrates how a table (TABLEA) can be initialized by having different data descriptions for the same data:

```
    05  VALUE-A.
        10  A1 PICTURE S9(9) COMPUTATIONAL
            VALUE IS ZEROES.
        10  A2 PICTURE S9(9) COMPUTATIONAL
            VALUE IS 1.
        .
        .
        .
        10  A100 PICTURE S9(9) COMPUTATIONA
            VALUE IS 99.
    05  TABLEA REDEFINES VALUE-A
        PICTURE S9(9) COMPUTATIONAL
        OCCURS 100 TIMES.
```

PICTURE Clause

DECIMAL-POINT ALIGNMENT: Procedure Division operations are most efficient when the decimal positions of the data items involved are aligned. If they are not, the compiler generates instructions to align the decimal positions before any operations involving the data items can be executed.

Assume, for example, that a program contains the following instructions:

```
WORKING-STORAGE SECTION.
77  A PICTURE S999V99.
77  B PICTURE S99V9.
.
.
.
PROCEDURE DIVISION.
.
.
.
    ADD A TO B.
```

Time and internal storage space are saved by defining B as:

```
77 B PICTURE S99V99.
```

If it is inefficient to define B differently, a one-time conversion can be done, as explained in "Data Format Conversion" in this chapter.

FIELDS OF UNEQUAL LENGTH: When a data item is moved to another data item of a different length, the following should be considered:

- If the items are external decimal items, the compiler generates instructions to insert zeros in the high-order positions of the receiving field, when it is the larger.
- If the items are nonnumeric, the compiler generates instructions to insert spaces in the low-order positions of the receiving field (or the high-order positions if the JUSTIFIED RIGHT clause is specified). This generation of extra instructions can be avoided if the sending field is described with a length equal to or greater than the receiving field.

SIGN USAGE: The presence or absence of a plus or minus sign in the description of an arithmetic field often can affect the efficiency of a program. The following paragraphs discuss some of the considerations.

Decimal Items: The sign position in an internal or external decimal item can contain:

1. A plus or minus sign. If S is specified in the PICTURE clause, a plus or minus sign is inserted when either of the following conditions prevail:

   a. The item is in the Working-Storage Section and a VALUE clause has been specified.

   b. A value for the item is assigned as a result of an arithmetic operation during execution of the program.

   If an external decimal item is punched, printed, or displayed, an overpunch will appear in the low-order digit. In EBCDIC, the configuration for low-order zeros normally is a nonprintable character. Low-order digits of positive values will be represented by one of the letters A through I (digits 1 through 9); low-order digits of negative values will be represented by one of the letters J through R (digits 1 through 9).

2. A hexadecimal F. If S is not specified in the PICTURE clause, an F is inserted in the sign position when either of the following conditions prevail:

   a. The item is in the Working-Storage Section and a VALUE clause has been specified

   b. A value for the item is developed during the execution of the program.

   An F is treated as positive, but is not an overpunch.

3. An invalid configuration. If an internal or external decimal item contains an invalid configuration in the sign position, and if the item is involved in a Procedure Division operation, the program will be abnormally terminated.

Unsigned items (items for which no S has been specified) are treated as absolute values. Whenever a value (signed or unsigned) is stored in or moved in an elementary move to an unsigned item, a hexadecimal F is stored in the sign position of the unsigned item. For example, if an arithmetic operation involves signed operands and an unsigned result field, compiler-generated code will insert an F in the sign position of the result field when the result is stored.

For internal and external decimal items used as input, it is the user's responsibility to ensure that the input data is valid. The compiler does not generate a test to ensure that the configuration in the sign position is valid.

When a group item is being moved, the data is moved without regard to the level structure of the group items involved. The possibility exists that the configuration in the sign position of a subordinate numeric item may be destroyed. Therefore, caution should be exercised in moving group items with subordinate numeric fields or with other group operations such as READ or ACCEPT.

USAGE Clause

The USAGE clause should be written at the highest level possible.

DATA FORMAT CONVERSION: Operations involving mixed, elementary numeric data formats require conversion to a common format. This usually means that additional storage is used and execution time is increased. The code generated must often move data to an internal work area, perform any necessary conversion, and then execute the indicated operation. Often, too, the result may have to be converted in the same way. Table 23 indicates when data conversion is necessary.

If it is impractical to use the same data formats throughout a program, and if two data items of different formats are frequently used together, a one-time conversion can be effected. For example, if A is defined as a COMPUTATIONAL item and B as a COMPUTATIONAL-3 item, A can be moved to a work area that has been defined as COMPUTATIONAL-3. This move causes the data in A to be converted to COMPUTATIONAL-3. Whenever A and B are used in a Procedure Division operation, reference can be made to the work area rather than to A. When this technique is used, the conversion is performed only once, instead of each time an operation is performed.

Table 23. Data Format Conversion

Usage	Bytes Required	Boundary Alignment Required	Typical Usage	Converted for Arithmetic Operations	Special Characteristics
DISPLAY (external decimal)	1 per digit (except for V)	No	Input from cards, output to cards, listings	Yes	May be used for numeric fields up to 18 digits long. Fields over 15 digits require extra instructions if used in computations.
DISPLAY (external floating point)	1 per character (except for V)	No	Input from cards, output to cards, listings	Yes	Converted to COMP-2 format via COBOL library subroutine.
COMP-3 (internal decimal)	1 per 2 digits plus 1 byte for low-order digit and sign	No	Input to a report item Arithmetic fields Work areas	Sometimes when a small COMP-3 item is used with a small COMP item	Requires less space than DISPLAY. Convenient form for decimal alignment. Can be used in arithmetic computations without conversion. Fields over 15 digits require a subroutine when used in computations.
COMP (binary)	2 if 1≤N≤4 4 if 5≤N≤9 8 if 10≤N≤18 where N is the number of 9's in the picture	Halfword Fullword Fullword	Subscripting Arithmetic fields	Sometimes for both mixed and unmixed usages	Rounding and testing for the ON SIZE ERROR condition are cumbersome if calculated result is greater than 9(9). Extra instructions are generated for binary computations if the SYNCHRONIZED clause is not specified. Fields of over nine digits require additional handling.
COMP-1 (internal floating point)	4 (short-precision)	Fullword	Fractional exponentiation	No	Tends to produce less accurate results if more than 17 significant digits are required and if the exponent is large. Requires floating-point feature.
COMP-2 (internal floating point)	8 (long-precision)	Double-word	Fractional exponentiation when additional precision is required	No	Same as COMP-1.

The following seven cases show how data conversions are handled on mixed elementary items for names, data comparisions, and arithmetic operations. Moves without the CORRESPONDING option to and from group items, as well as comparisons involving group items, are done without conversion.

Numeric DISPLAY to COMPUTATIONAL-3:

To Move Data: Converts DISPLAY data to COMPUTATIONAL-3 data.

To Compare Data: Converts DISPLAY data to COMPUTATIONAL-3 data.

To Perform Arithmetic Operations: Converts DISPLAY data to COMPUTATIONAL-3 data.

Numeric DISPLAY to COMPUTATIONAL:

To Move Data: Converts DISPLAY data to COMPUTATIONAL-3 data and then to COMPUTATIONAL data.

To Compare Data: Converts DISPLAY to COMPUTATIONAL or converts both DISPLAY and COMPUTATIONAL data to COMPUTATIONAL-3 data.

To Perform Arithmetic Operations: Converts DISPLAY data to COMPUTATIONAL-3 or COMPUTATIONAL data.

COMPUTATIONAL-3 to COMPUTATIONAL:

To Move Data: Moves COMPUTATIONAL-3 data to a work area and then converts COMPUTATIONAL-3 data to COMPUTATIONAL data.

To Compare Data: Converts COMPUTATIONAL data to COMPUTATIONAL-3 or vice versa, depending on the size of the field.

To Perform Arithmetic Operations: Converts COMPUTATIONAL data to COMPUTATIONAL-3 or vice versa, depending on the size of the field.

COMPUTATIONAL to COMPUTATIONAL-3:

To Move Data: Converts COMPUTATIONAL data to COMPUTATIONAL-3 data in a work area, and then moves the work area.

To Compare Data: Converts COMPUTATIONAL to COMPUTATIONAL-3 data or vice versa, depending on the size of the field.

To Perform Arithmetic Operations: Converts COMPUTATIONAL to COMPUTATIONAL-3 data or vice versa, depending on the size of the field.

COMPUTATIONAL to Numeric DISPLAY:

To Move Data: Converts COMPUTATIONAL data to COMPUTATIONAL-3 data and then to DISPLAY data.

To Compare Data: Converts DISPLAY to COMPUTATIONAL or both COMPUTATIONAL and DISPLAY data to COMPUTATIONAL-3 data, depending on the size of the field.

To Perform Arithmetic Operations: Depending on the size of the field, converts DISPLAY data to COMPUTATIONAL data, or both DISPLAY and COMPUTATIONAL data to COMPUTATIONAL-3 data in which case the result is generated in a COMPUTATIONAL-3 work area and then converted and moved to the DISPLAY result field.

COMPUTATIONAL-3 to Numeric DISPLAY:

To Move Data: Converts COMPUTATIONAL-3 data to DISPLAY data.

To Compare Data: Converts DISPLAY data to COMPUTATIONAL-3 data. The result is generated in a COMPUTATIONAL-3 work area and is then converted and moved to the DISPLAY result field.

Numeric DISPLAY to Numeric DISPLAY:

To Perform Arithmetic Operations: Converts all DISPLAY data to COMPUTATIONAL-3 data. The result is generated in a COMPUTATIONAL-3 work area and is then converted to DISPLAY and moved to the DISPLAY result field.

Internal Floating-point to Any Other: When an item described as COMPUTATIONAL-1 or COMPUTATIONAL-2 (internal floating-point) is used in an operation with another data format, the item in the other data format is always converted to internal floating-point. If necessary, the internal floating-point result is then converted to the format of the other data item.

SYNCHRONIZED Clause

As illustrated in Table 20, COMPUTATIONAL, COMPUTATIONAL-1 and COMPUTATIONAL-2 items have specific boundary alignment requirements. To ensure correct alignment, either the programmer or the compiler may have to insert slack bytes.

The SYNCHRONIZED clause may be used at the elementary level to specify the automatic alignment of elementary items on their proper boundaries, or at the 01 level to synchronize all elementary items within the group. For COMPUTATIONAL items, if the PICTURE is in the range of S9 through S9(4), the item is aligned on a halfword boundary. If the PICTURE is in the range of S9(5) through S9(18), the item is aligned on a fullword boundary. For COMPUTATIONAL-1 items, the item is aligned on a fullword boundary. For COMPUTATIONAL-2 items, the item is aligned on a doubleword boundary. The SYNCHRONIZED clause and slack bytes are fully discussed in the publication IBM System/360 Disk Operating System: American National Standard COBOL.

Special Considerations for DISPLAY and COMPUTATIONAL Fields

NUMERIC DISPLAY FIELDS: Zeros are not inserted into numeric DISPLAY fields by the instruction set. When numeric DISPLAY data is moved, the compiler generates instructions that insert any necessary zeros into the DISPLAY fields. When numeric DISPLAY data is compared, and one field is smaller than the other, the compiler generates instructions to move the smaller item to a work area where zeros are inserted.

COMPUTATIONAL FIELDS: COMPUTATIONAL fields can be aligned on either a halfword or fullword boundary. If an operation involves COMPUTATIONAL fields of different lengths, the halfword field is automatically expanded to a fullword field. Therefore, mixed halfword and fullword fields require no additional operations.

COMPUTATIONAL-1 AND COMPUTATIONAL-2 FIELDS: If an arithmetic operation involves a mixture of short-precision and long-precision fields, the compiler generates instructions to expand the short-precision field to a long-precision field before the operation is executed.

COMPUTATIONAL-3 FIELDS: The compiler does not have to generate instructions to insert high-order zeros for ADD and SUBTRACT statements that involve COMPUTATIONAL-3 data. The zeros are inserted by the instruction set.

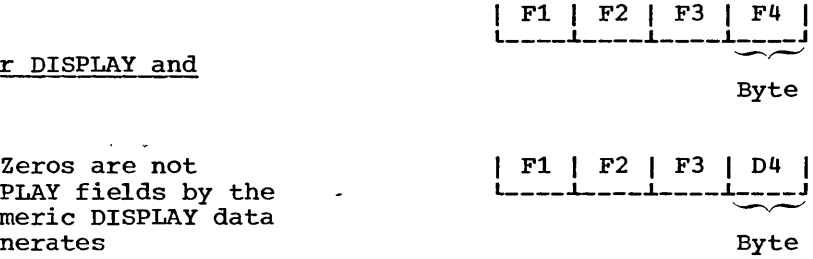
Data Formats in the Computer

The following examples illustrate how the various COBOL data formats appear in the computer in EBCDIC (Extended Binary-Coded-Decimal Interchange Code) format. More detailed information about these data formats appear in the publication IBM System/360 Principles of Operation.

Numeric DISPLAY (External Decimal): Suppose the value of an item is -1234, and its PICTURE and USAGE clauses are:

```
PICTURE 9999 DISPLAY.

        or

PICTURE S9999 DISPLAY.
```

The item appears in the computer in the following forms, respectively:

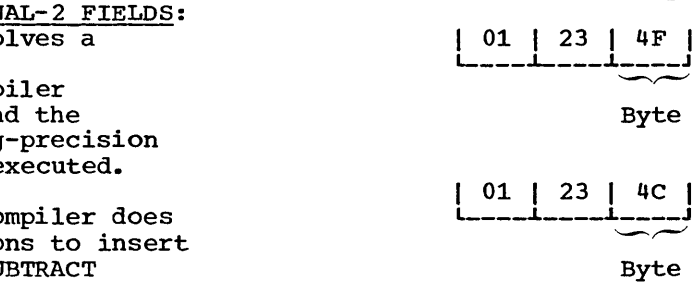


Hexadecimal F is treated arithmetically as positive; hexadecimal D represents a minus sign.

COMPUTATIONAL-3 (Internal Decimal): Suppose the value of an item is +1234, and its PICTURE and USAGE clauses are:

```
PICTURE 9999 COMPUTATIONAL-3.

        or

PICTURE S9999 COMPUTATIONAL-3.
```

The item appears internally in the following forms, respectively:



Hexadecimal F is treated arithmetically as positive; hexadecimal C represents a plus sign.

Note: Since the low-order byte of an internal decimal number always contains a sign field, an item with an odd number of digits can be stored more efficiently than an item with an even number of digits. Note that a leading zero is inserted in the above example.

COMPUTATIONAL (Binary): Suppose the value of an item is 1234, and its PICTURE and USAGE clauses are:

```
PICTURE S9999 COMPUTATIONAL.
```

The item appears internally in the following form:

```
| 0000 | 0100 | 1101 | 0010 |
        ^
      Sign
   Position
```

A 0 in the sign position indicates that the number is positive. Negative numbers are represented in two's complement form; thus, the sign position of a negative number will always contain a 1.

For example -1234 would appear as follows:

```
| 1111 | 1011 | 0010 | 1110 |
        ^
      Sign
   Position
```

Binary Item Manipulation: A binary item is allocated storage ranging from one halfword to two fullwords, depending on the number of 9's in its PICTURE. Table 24 is an illustration of how the compiler allocates this storage. Note that it is possible for a value larger than that implied by the PICTURE clause to be stored in the item. For example, PICTURE S9(4) implies a maximum value of 9,999, although it could actually hold the number 32,767.

Because most binary items are manipulated according to their allotted storage capacity, the programmer can ignore this situation. For the following reasons, however, he must be careful of his data:

1. When the ON SIZE ERROR option is used, the size test is made on the basis of the maximum value allowed by the picture of the result field. If a size error condition exists, the value of the result field is not altered and control is given to the imperative-statements specified by the error option.

2. When a binary item is displayed or exhibited, the value used is a function of the number of 9's specified in the PICTURE clause.

3. When the actual value of a positive number is significantly larger than its picture value, a value of 1 could appear in the sign position of the item, causing the item to be treated as a negative number in subsequent operations.

Table 24. Relationship of PICTURE to Storage Allocation

PICTURE	Maximum Working Value	Assigned Storage
S9 through S9(4)	32,767	One halfword
S9(5) through S9(9)	2,147,483,647	One fullword
S9(10) through S9(18)	9,223,372,036,854,775,807	Two fullwords

Case	Hexadecimal Result of Binary Calculation	Decimal Equivalent	Actual Decimal Value in Halfword of Storage	DISPLAY or EXHIBIT Value
A	0008	8	+8	8
B	000A	10	+10	0
C	C350	50000	-15536	6

Figure 52. Treatment of Varying Values in a Data Item of PICTURE S9

Figure 52 illustrates three binary manipulations. In each case, the result field is an item described as PICTURE S9 COMPUTATIONAL. One halfword of storage has been allocated, and no ON SIZE ERROR option is involved. Note that if the ON SIZE ERROR option had been specified, it would have been executed for cases B and C.

COMPUTATIONAL-1 or COMPUTATIONAL-2 (Floating-point): Suppose the value of an item is +1234 and that its USUAGE is COMPUTATIONAL-1, the item appears internally in the following form:

```
|0|100  0011|0100 1101 0010 0000 0000 0000|
 S 1       7 8                            31
```

S is the sign position of the number.

A 0 in the sign position indicates that the sign is plus.

A 1 in the sign position indicates that the sign is minus.

Bits 1 through 7 are the exponent (characteristic) of the number.

Bits 8 through 31 are the fraction (mantissa) of the number.

This form of data is referred to as floating point. The example illustrates short-precision floating-point data (COMPUTATIONAL-1). In long-precision (COMPUTATIONAL-2), the fraction length is 56 bits. (For a detailed explanation of floating-point representation, see the publication IBM System/360 Principles of Operation.)

PROCEDURE DIVISION

The Procedure Division of a program can often be made more efficient or easier to debug by using some of the techniques described below.

MODULARIZING THE PROCEDURE DIVISION

Modularization involves organizing the Procedure Division into at least three functional levels: a main-line routine, processing subroutines, and input/output subroutines. When the Procedure Division is modularized, programs are easier to maintain and document. In addition, modularization makes it simple to break down a program using the segmentation feature, resulting in a more efficient segmented program.

Main-Line Routine

The main-line routine should be short and simple, and should contain all the major logical decisions of the program. This routine controls the order in which second-level subroutines are executed. All second-level subroutines should be invoked from the main-line routine by PERFORM statements.

Processing Subroutines

Processing subroutines should be broken down into as many functional levels as necessary, depending on the complexity of the program. These must be completely closed subroutines, with one entry point and one exit point. The entry point should be the first statement of the subroutine. The exit point should be the EXIT statement. Processing subroutines can PERFORM only lower level subroutines; return to the higher level subroutine (processing subroutine) must be accomplished by a GO TO statement that references the EXIT statement.

Input/Output Subroutines

The input/output subroutines should be the lowest level subroutines, since all higher level subroutines have access to them. There should be one OPEN subroutine and one CLOSE subroutine for the program, and only one functional (READ or WRITE) subroutine for each file. Having one READ or WRITE subroutine per file has several advantages:

1. Coding can be added to count records on a file, transform blanks into zeros, check for 9's padding, etc.

2. Input and output files can be reformatted without changing the logic of the program.

3. DEBUG statements can be added during testing to create input or to DISPLAY formatted output, instead of having to create a test file.

INTERMEDIATE RESULTS

The compiler treats arithmetic statements as a succession of operations and sets up intermediate result fields to contain the results of these operations. Examples of such statements are the arithmetic statements and statements containing arithmetic expressions. See the appendix "Intermediate Results" in the publication IBM System/360 Disk Operating System: American National Standard COBOL for a description of the algorithms used by the compiler to determine the number of places reserved for intermediate result fields.

Intermediate Results and Binary Data Items

If an operation involving binary operands requires an intermediate result greater than 18 digits, the compiler converts the operands to internal decimal before performing the operation. If the result field is binary, the result will be converted from internal decimal to binary.

If an intermediate result will not be greater than nine digits, the operation is performed most efficiently on binary data fields.

Intermediate Results and COBOL Library Subroutines

If a decimal multiplication operation requires an intermediate result greater than 30 digits, a COBOL library subroutine is used to perform the multiplication. The result of this multiplication is then truncated to 30 digits.

A COBOL library subroutine is used to perform division if:

1. The divisor is equal to or greater than 15 digits.

2. The length of the divisor plus the length of the dividend is greater than 16 bytes.

3. The scaled dividend is greater than 30 digits. (A scaled dividend is a number that has been multiplied by a power of ten in order to obtain the desired number of decimal places in the quotient.)

Intermediate Results Greater Than 30 Digits

Whenever the number of digits in a decimal intermediate result is greater than 30, the field is truncated to 30 digits. A warning message will be generated during compilation, and program flow will not be interrupted at execution time. This truncation may cause a result to be incorrect.

If binary or internal decimal data is in agreement with its data description, no interrupt can occur because of an overflow condition in an intermediate result. This is due to the truncation described in the preceding paragraph.

If the possibility exists that an intermediate result field may exceed 30 digits, truncation can be avoided by the specification of floating-point operands (COMPUTATIONAL-1 or COMPUTATIONAL-2); however, accuracy may not be maintained.

Intermediate Results and Floating-point Data Items

If a floating-point operand has an intermediate result field in which exponent overflow occurs, the job will be abnormally terminated.

Intermediate Results and the ON SIZE ERROR Option

The ON SIZE ERROR option applies only to the final calculated results and not to intermediate result fields.

PROCEDURE DIVISION STATEMENTS

COMPUTE Statement

The use of the COMPUTE statement generates more efficient coding than does the use of individual arithmetic statements, since the compiler can keep track of internal work areas and does not have to store the results of intermediate calculations. It is the user's responsibility, however, to ensure that the data is defined with the level of significance required in the answer.

IF Statement

Nested and compound IF statements should be avoided as the logic is difficult to debug.

MOVE Statement

Performing a MOVE operation for an item longer than 256 bytes requires the generation of more instructions than are required for that of a MOVE operation for an item of 256 bytes or less.

When a MOVE statement with the CORRESPONDING option is executed, data items are considered as "corresponding" only if their respective data-names are the same, including all implied qualification up to, but not including, the data-names used in the MOVE statement itself.

For example:

```
01 AA                    01  XX
   05 BB                     05 BB
     10 CC                     10 CC
     10 DD                     10 DD
   05 EE                     05 YY
     10 FF                     10 FF
```

The statement MOVE CORRESPONDING AA TO XX will result in moving CC, but not FF, since FF of EE does not correspond to FF of YY.

Note: The other rules for MOVE CORRESPONDING, of course, must still be satisfied.

NOTE Statement

An asterisk (*) should be used in place of the NOTE statement since there is the possibility that when NOTE is the first sentence in a paragraph, it will inadvertently cause the whole paragraph to be treated as part of the NOTE.

PERFORM Statement

PERFORM is a useful statement if the programmer adheres to the following rules:

1. Always execute the last statement of a series of routines being operated on by a PERFORM statement. When branching out of the routine, make sure control will eventually return to the last statement of the routine, which should be an EXIT statement. Although no code is generated, the EXIT statement allows a programmer to immediately recognize the extent of a series of routines within the range of a PERFORM statement.

2. Always either PERFORM routine-name THRU routine-name-exit, or PERFORM section-name. A PERFORM paragraph-name can create problems for the programmer trying to maintain the program. For example, if one paragraph must be broken into two paragraphs, the programmer must examine every statement to determine whether this paragraph is within the range of the PERFORM statement. As a result, all statements referencing the paragraph-name must be changed to PERFORM THRU statements.

READ INTO AND WRITE FROM OPTIONS

Always use READ INTO and WRITE FROM, and process all files in the Working-Storage Section for the following reasons:

1. Debugging is much simpler. Working-Storage areas are easier to locate in a dump than are buffer areas. And, if files are blocked, it is much easier to determine which record in a block was being processed when the abnormal termination occurred.

2. Trying to access a record-area after the AT END condition has occurred (for example, AT END MOVE HIGH-VALUE TO INPUT-RECORD) can cause problems if the record area is defined only in the File Section.

Note: The programmer should be aware that additional time is used to execute the move operation involved in each READ INTO or WRITE FROM instruction.

When a READ INTO statement is used for a V-mode or U-mode file, the size of the longest record for that file is used in the MOVE statement. All other rules of the MOVE statement apply.

TRACE Statement

The programmer should remember that the RESET or READY options of the TRACE statement are initialized with each execution of a CALL statement.

TRANSFORM Statement

The TRANSFORM statement generates more efficient code than the EXAMINE REPLACING BY statement when only one character is being transformed. The TRANSFORM statement, however, uses a 256-byte table.

USING THE REPORT WRITER FEATURE

REPORT Clause in a File Description (FD) Entry

A given report-name may appear in a maximum of two file description entries. The file description entries need not have the same characteristics. If the same report-name is specified in two file description entries, the report will be written on both files. For example:

```
ENVIRONMENT DIVISION.
    SELECT FILE-1  ASSIGN SYS005-UR-1403-S.
    SELECT FILE-2  ASSIGN SYS001-UT-2400-S.
    .
    .
DATA DIVISION.
FD  FILE-1  RECORDING MODE F
            RECORD CONTAINS 121 CHARACTERS
            REPORT IS REPORT-A.
FD  FILE-2  RECORDING MODE V
            RECORD CONTAINS 101 CHARACTERS
            REPORT IS REPORT-A.
```

For each GENERATE statement, the records for REPORT-A will be written on FILE-1 and FILE-2, respectively. The records on FILE-2 will not contain columns 102 through 121 of the corresponding records on FILE-1.

Summing Techniques

Execution time in an object program can be made more efficient by keeping in mind that Report Writer source coding is treated as though the programmer had written the program in COBOL without the Report Writer feature. Therefore, a complex source statement or series of statements will generally be executed faster than simple statements that perform the same function. The following example shows two coding techniques for the Report Section of the Data Division. Method 2 uses the more complex statements.

```
RD...CONTROLS ARE YEAR MONTH WEEK DAY.
```

Method 1:

```
01 TYPE CONTROL FOOTING YEAR.
   02 SUM COST.
01 TYPE CONTROL FOOTING MONTH.
   02 SUM COST.
01 TYPE CONTROL FOOTING WEEK.
   02 SUM COST.
01 TYPE CONTROL FOOTING DAY.
   02 SUM COST.
```

Method 2:

```
01 TYPE CONTROL FOOTING YEAR.
   02 SUM A.
01 TYPE CONTROL FOOTING MONTH.
   02 A SUM B.
01 TYPE CONTROL FOOTING WEEK.
   02 B SUM C.
01 TYPE CONTROL FOOTING DAY.
   02 C SUM COST.
```

Method 2 will execute faster. One addition will be performed for each day, one more for each week, and one for each month. In Method 1, four additions will be performed for each day.

Use of SUM

Unless each identifier is the name of a SUM counter in a TYPE CONTROL FOOTING report group at an equal or lower position in the control hierarchy, the identifier must be defined in the File, Working-Storage, or Linkage Sections as well as in a TYPE DETAIL report group as a source item. A SUM counter is algebraically

incremented just before presentation of the TYPE DETAIL report group in which the item being summed appears as a source item or the item being summed appeared in a SUM clause that contained an UPON option for this DETAIL report group. This is known as SOURCE-SUM corresponding. In the following example, SUBTOTAL is incremented only when DETAIL-1 is generated.

```
FILE SECTION.
       .
       .
       .
    02 NO-PURCHASES PICTURE 99.
       .
       .
       .
REPORT SECTION.
01 DETAIL-1 TYPE DETAIL.
    02 COLUMN 30 PICTURE 99 SOURCE
       NO-PURCHASES.
       .
       .
       .
01 DETAIL-2 TYPE DETAIL.
       .
       .
       .
01 DAY TYPE CONTROL FOOTING
    LINE PLUS 2.
       .
       .
       .
    02 SUBTOTAL COLUMN 30 PICTURE 999
       SUM NO-PURCHASES.
       .
       .
       .
01 MONTH TYPE CONTROL FOOTING
    LINE PLUS 2 NEXT GROUP
    NEXT PAGE.
```

SUM Routines

A SUM routine is generated by the Report Writer for each DETAIL report group of the report. The operands included for summing are determined as follows:

1. The SUM operand(s) also appears in a SOURCE clause(s) for the DETAIL report group.

2. The UPON detail-name option was specified in the SUM clause. In this case, all the operands are included in the SUM routine for only that DETAIL report group, even if the operand appears in a SOURCE clause in other DETAIL report groups.

When a GENERATE detail-name statement is executed, the SUM routine for that DETAIL report group is executed in its logical sequence. When GENERATE report-name statement is executed and the report contains more than one DETAIL report group, the SUM routine is executed for each one. The SUM routines are executed in the sequence in which the DETAIL report groups are specified.

The following two examples show the SUM routines that are generated by the Report Writer. Example 1 illustrates how operands are selected for inclusion in the routine on the basis of simple SOURCE-SUM correlation. Example 2 illustrates how operands are selected when the UPON detail-name option is specified.

Example 1: The following statements are coded in the Report Section:

```
01 DETAIL-1 TYPE DE ...
   02 ...SOURCE A.
                .
                .
                .
01 DETAIL-2 TYPE DE ...
   02 ...SOURCE B.
   02 ...SOURCE C.
                .
                .
                .
01 DETAIL-3 TYPE DE ...
   02 ...SOURCE B.
                .
                .
                .
01 TYPE CF ...
   02 SUM-CTR-1 ...SUM A, B, C.
                .
                .
                .
01 TYPE CF ...
   02 SUM-CTR-2 ...SUM B.
```

A SUM routine is generated for each DETAIL report group, as follows:

SUM-ROUTINE FOR DETAIL-1

```
   REPORT-SAVE
      ADD A TO SUM-CTR-1.
   REPORT-RETURN
```

SUM-ROUTINE FOR DETAIL-2

```
   REPORT-SAVE
      ADD B TO SUM-CTR-1.
      ADD C TO SUM-CTR-1.
      ADD B TO SUM-CTR-2.
   REPORT-RETURN
```

SUM-ROUTINE FOR DETAIL-3

```
   REPORT-SAVE
      ADD B TO SUM-CTR-1.
      ADD B TO SUM-CTR-2.
   REPORT-RETURN
```

Example 2: This example uses the same coding as Example 1, with one exception: the UPON detail-name option is used for SUM-CTR-1, as follows:

```
01 TYPE CF ...
   02 SUM-CTR-1 ...SUM A, B, C
      UPON DETAIL-2.
```

The following SUM routines would then be generated instead of those shown in the previous example:

SUM Routine for DETAIL-1

```
   REPORT-SAVE
   REPORT-RETURN
```

SUM Routine for DETAIL-2

```
   REPORT-SAVE
      ADD A TO SUM-CTR-1.
      ADD B TO SUM-CTR-1.
      ADD C TO SUM-CTR-1.
      ADD B TO SUM-CTR-2.
   REPORT-RETURN
```

SUM Routine for DETAIL-3

```
   REPORT-SAVE
      ADD B TO SUM-CTR-2.
   REPORT-RETURN
```

Output Line Overlay

The Report Writer output line is created using an internal REDEFINES specification, indexed by integer-1. No check is made to prevent overlay on any line. For example:

```
    02  COLUMN 10  PICTURE X(23)
        VALUE "MONTHLY SUPPLIES REPORT".
    02  COLUMN 12  PICTURE X(9)
        SOURCE CURRENT-MONTH.
```

A length of 27 in column 10, followed by a specification for column 12, will cause field overlay when this line is printed.

Page Breaks

The Report Writer page break routine operates independently of the routines that are executed after any control breaks (except that a page break will occur as the result of a LINE NEXT PAGE clause). Thus, the programmer should be aware of the following facts:

1. A Control Heading is not printed after a Page Heading except for first generation. If the programmer wishes to have the equivalent of a Control Heading at the top of each page, he must include the information and data to be printed as part of the Page Heading. Since only one Page Heading may be specified for each report, he should be selective in considering his Control Heading because it will be the same for each page, and may be printed at inappropriate times (see "Control Footings and Page Format" in this chapter).

2. GROUP INDICATE items are printed after page and control breaks. Figure 53 contains a GROUP INDICATE clause and illustrates the execution output.

```
REPORT SECTION.
   .
   .
   .
01  DETAIL-LINE TYPE IS DETAIL LINE
   NUMBER IS PLUS 1.
   02  COLUMN IS 2 GROUP INDICATE
       PICTURE IS A(9) SOURCE IS
       MONTHNAME OF RECORD-AREA (MONTH).
   .
   .
   .

(Execution Output)
-----------------------------------------
JANUARY   15   A00...
               A02...

PURCHASES AND COST...
-----------------------------------------
JANUARY   21   A03...
               A03...
```

Figure 53. Sample of GROUP INDICATE Clause and Resultant Execution Output

WITH CODE Clause

When more than one report is being written on a file and the reports are to be selectively written, a unique 1-character code must be given for each report. A mnemonic-name is specified in the RD-level entry for each report and is associated with the code in the Special-Names paragraph of the Environment Division.

Note: If a report is written with the CODE option, the report should not be written directly on a printer device.

This code will be written as the first character of each record that is written on the file. When the programmer wishes to write a report from this file, he needs

only to read a record, check the first character for the desired code, and have it printed if the desired code is found. The record should be printed starting from the third character, as illustrated in Figure 54.

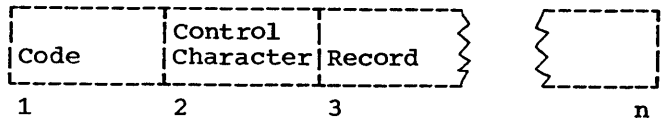


Figure 54. Format of a Report Record When the CODE Clause is Specified

The following example shows how to create and print a report with a code of A. A Report Writer program contains the following statements:

```
ENVIRONMENT DIVISION.
.
.
SPECIAL-NAMES.  A IS CODE-CHR-A
                B IS CODE-CHR-B.
.
.
DATA DIVISION.
.
.
REPORT SECTION.
RD  REP-FILE-A    CODE CODE-CHR-A ...
.
.
RD  REP-FILE-B    CODE CODE-CHR-B ...
```

A second program could then be used to print only the report with the code of A, as follows:

```
DATA DIVISION.
FD  RPT-IN-FILE
    RECORD CONTAINS 122 CHARACTERS
    LABEL RECORDS ARE STANDARD
    DATA RECORD IS RPT-RCD.
01  RPT-RCD.
    05  CODE-CHR          PICTURE X.
    05  PRINT-PART.
        10                 PICTURE X.
        10                 PICTURE X(120).
FD  PRINT-FILE
    RECORD CONTAINS 121 CHARACTERS
    LABEL RECORDS ARE STANDARD
    DATA RECORD IS PRINT-REC.
01  PRINT-REC.
    05  FILLER            PICTURE X(121).
.
.
PROCEDURE DIVISION.
.
.
LOOP.  READ RPT-IN-FILE AT END
           GO TO CONTINUE.
           IF CODE-CHR = "A"
           WRITE PRINT-REC FROM PRINT-PART
           AFTER POSITIONING CTL-CHR LINES.
           GO TO LOOP.
CONTINUE.
           .
           .
           .
```

Control Footings and Page Format

Depending on the number and size of Control Footings (as well as the page depth of the report), all of the specified Control Footings may not be printed on the same page if a control break occurs for a high-level control. When a page condition is detected before all required Control Footings are printed, the Report Writer will print the Page Footing (if specified), skip to the next page, print the Page Heading (if specified) and then continue to print Control Footings.

If the programmer wishes all of his Control Footings to be printed on the same page, he must format his page in the RD-level entry for the report (by setting the LAST DETAIL integer to a sufficiently low line number) to allow for the necessary space.

NEXT GROUP Clause

Each time a CONTROL FOOTING report group with a NEXT GROUP clause is printed, the clause is activated only if the report group is associated with the control that causes the break. This is illustrated in Figure 55.

```
RD  EXPENSE-REPORT CONTROLS ARE FINAL,
    MONTH, DAY
       .
       .
       .
01  TYPE CONTROL FOOTING DAY
    LINE PLUS 1 NEXT GROUP
    NEXT PAGE.
       .
       .
       .
01  TYPE CONTROL FOOTING MONTH
    LINE PLUS 1 NEXT GROUP
    NEXT PAGE.
       .
       .
       .

(Execution Output)

EXPENSE REPORT
       .
       .
       .
January 31.....29.30
    (Output for CF DAY)
January total.....131.40
    (Output for CF MONTH)
```

Figure 55. Activating the NEXT GROUP Clause

Note: The NEXT GROUP NEXT PAGE clause for the Control Footing DAY is not activated.

Floating First Detail

The first presentation of a body group (PH, PF, CH, CF, DE) that contains a relative line as its first line will have its relative line spacing suppressed; the first line will be printed on either the value of FIRST DETAIL or INTEGER PLUS 1 of a NEXT GROUP clause from the preceding page. For example:

1. If the following body group was the last to be printed on a page

   01 TYPE CF NEXT GROUP NEXT PAGE

   then this next body group

   01 TYPE DE LINE PLUS 5

   would be printed on value of FIRST DETAIL (in PAGE clause).

2. If the following body group was the last to be printed on a page

   01 TYPE CF NEXT GROUP LINE 12

   and after printing, line-counter = 40, then this next body group

   01 TYPE DETAIL LINE PLUS 5

   would be printed on line 12 + 1 (i.e., line 13).

Report Writer Routines

At the end of the analysis of a report description (RD) entry, the Report Writer routines are generated, based on the contents of the RD. Each routine has its own compiler-generated card number. Therefore, in the source listing, the last compiler-generated card number for an RD and that of the next source statement are not sequential.

TABLE HANDLING CONSIDERATIONS

Subscripts

If a subscript is represented by a constant and if the subscripted item is of fixed length, the location of the subscripted data item within the table or list is resolved during compilation.

If a subscript is represented by a data-name, the location is resolved at execution time. The most efficient format in this case is COMPUTATIONAL, with PICTURE size less than five integers.

The value contained in a subscript is an integer which represents an occurrence number within a table. Every time a subscripted data-name is referenced in a program, the compiler generates up to 16 instructions to calculate the correct displacement. Therefore, if a subscripted data-name is to be processed extensively, move the subscripted item to an unsubscripted work area, do all necessary processing, and then move the item back into the table. Even when subscripts are described as COMPUTATIONAL, subscripting takes time and core storage.

Index-names

Index-names are compiler-generated items, one fullword in length, assigned storage in the TGT (Task Global Table). An index-name is defined by the INDEXED BY clause. The value in an index-name represents an actual displacement from the beginning of the table that corresponds to an occurrence number in the table. Address calculation for a direct index requires a maximum of four instructions; address calculation for a relative index requires a few more. Therefore, the use of index-names in referencing tables is more efficient than the use of subscripts. The use of direct indexes is faster than the use of relative indexes.

Index-names can only be referenced in the PERFORM, SEARCH, and SET statements.

Index Data Items

Index data items are compiler-generated storage positions, one fullword in length, that are assigned storage within the COBOL program area. An index data item is defined by the USAGE IS INDEX clause. The programmer can use index data items to save values of index-names for later reference.

Great care must be taken when setting values of index data items. Since an index data item is not part of any table, the compiler is unable to change any displacement value contained in an index-name when an index data item is set to the value of an index-name or another index data item. See the SET statement examples later in this chapter.

Index data items can only be referenced in SEARCH and SET statements.

OCCURS Clause

If indexing is to be used to reference a table element and the Format 2 (SEARCH ALL) statement is also used, the KEY option must be specified in the OCCURS clause. A table element is represented by the subject of an OCCURS clause, and is equivalent to one level of a table. The table element must then be ordered upon the key(s) and data-name(s) specified.

DEPENDING ON Option

If a data item described by an OCCURS clause with the DEPENDING ON data-name option is followed by nonsubordinate data items, a change in the value of data-name during the course of program execution will have the following effects:

1. The size of any group described by or containing the related OCCURS clause will reflect the new value of data-name.
2. Whenever a MOVE to a field containing an OCCURS clause with the DEPENDING ON option is executed, the MOVE is done on the basis of the current contents of the object of the DEPENDING ON option.
3. The location of any nonsubordinate items following the item described with the OCCURS clause will be affected by the new value of data-name. If the user wishes to preserve the contents of these items, the following procedure can be used: prior to the change in data-name, move all nonsubordinate items following the variable item to a work area; after the change in data-name, move all the items back.

Note: The value of data-name may change because a move is made to it or to the group in which it is contained; or the value of data-name may change because the group in which it is contained is a record area that has been changed by execution of a READ statement.

For example, assume that the Data Division of a program contains the following coding:

```
01  ANYRECORD.
    05  A PICTURE S999 COMPUTATIONAL-3.
    05  TABLEA PICTURE S999 OCCURS 100
        TIMES DEPENDING ON A.
    05  GROUPB.

            Subordinate data items.
                    End of record.
```

GROUPB items are not subordinate to TABLEA, which is described by the OCCURS clause. Assuming that WORKB is a work area with the same data structure as GROUPB, the following procedural coding could be used:

```
MOVE GROUPB TO WORKB

Calculate a new value of A

MOVE WORKB TO GROUPB
```

The preceding statements can be avoided by placing the OCCURS clause with the DEPENDING ON option at the end of the record.

Note: data-name can also change because of a change in the value of an item that redefines it. In this case, the group size and the location of nonsubordinate items as described in the two preceding paragraphs cannot be determined.

SEARCH ALL Statement

The SEARCH ALL statement is used to search an entire table for an item without having to write a loop procedure. For example, a user-defined table may be the following:

```
01  TABLE.
    05  ENTRY-IN-TABLE OCCURS 90 TIMES
        ASCENDING KEY-1,KEY-2
        DESCENDING KEY-3
        INDEXED BY INDEX-1.
        10  PART-1  PICTURE 9(2).
        10  KEY-1   PICTURE 9(5).
        10  PART-2  PICTURE 9(6).
        10  KEY-2   PICTURE 9(4).
        10  PART-3  PICTURE 9(33).
        10  KEY-3   PICTURE 9(5).
```

A search of the entire table can be initiated with the following instruction:

```
SEARCH ALL TABLE AT END GO TO NOENTRY
WHEN KEY-1 = VALUE-1 AND KEY-2 = VALUE-2
AND KEY-3 = VALUE-3 MOVE PART-1
(INDEX-1) TO OUTPUT-AREA
```

The preceding instructions will execute a search on the given array TABLE, which contains 90 elements of 55 bytes and 3 keys. The primary and secondary keys (KEY-1 and KEY-2) are in ascending order whereas the least significant key (KEY-3) is in descending order. If an entry is found in which the three keys are equal to the given values (i.e., VALUE-1, VALUE-2, VALUE-3), PART-1 of that entry will be moved to OUTPUT-AREA. If matching keys are not found in any of the entries in TABLE, the NOENTRY routine is entered.

If a match is found between a table entry and the given values, the index (INDEX-1) is set to a value corresponding to the relative position within the table of the matching entry. If no match is found, the index remains at the setting it had when execution of the SEARCH ALL statement began.

Note: It is more efficient to test keys in order of significance (i.e., KEY-1 should be specified before KEY-2 in the WHEN statement). The WHEN statement can only test for equality, and only one side of the equation may be a key.

SET Statement

The SET statement is used to assign values to index-names and to index data items.

When an index-name is set to the value of a literal, identifier, or an index-name from another table element, it is set to an actual displacement from the beginning of the table that corresponds to the occurrence number indicated by the second operand in the statement. The compiler performs the necessary calculations. If an index-name is set to another index-name for the same table, the compiler need make no conversion of the actual displacement value contained in the second operand.

However, when an index data item is set to another index data item or to an index-name, or when an index-name is set to an index data item, the compiler is unable to change any displacement value it finds, since an index data item is not part of any table. Thus, no conversion of values can take place. Remember this to avoid making programming errors.

For example, suppose that a table has been defined as:

```
01 A.
   05 B OCCURS 2 INDEXED BY I1, I5.
      10 C OCCURS 2 INDEXED BY I2, I6.
         15 D OCCURS 3 INDEXED BY I3, I4.
            20 E PIC X(20).
            20 F PIC 9(5).
```

The table appears in core storage as shown in Figure 56.

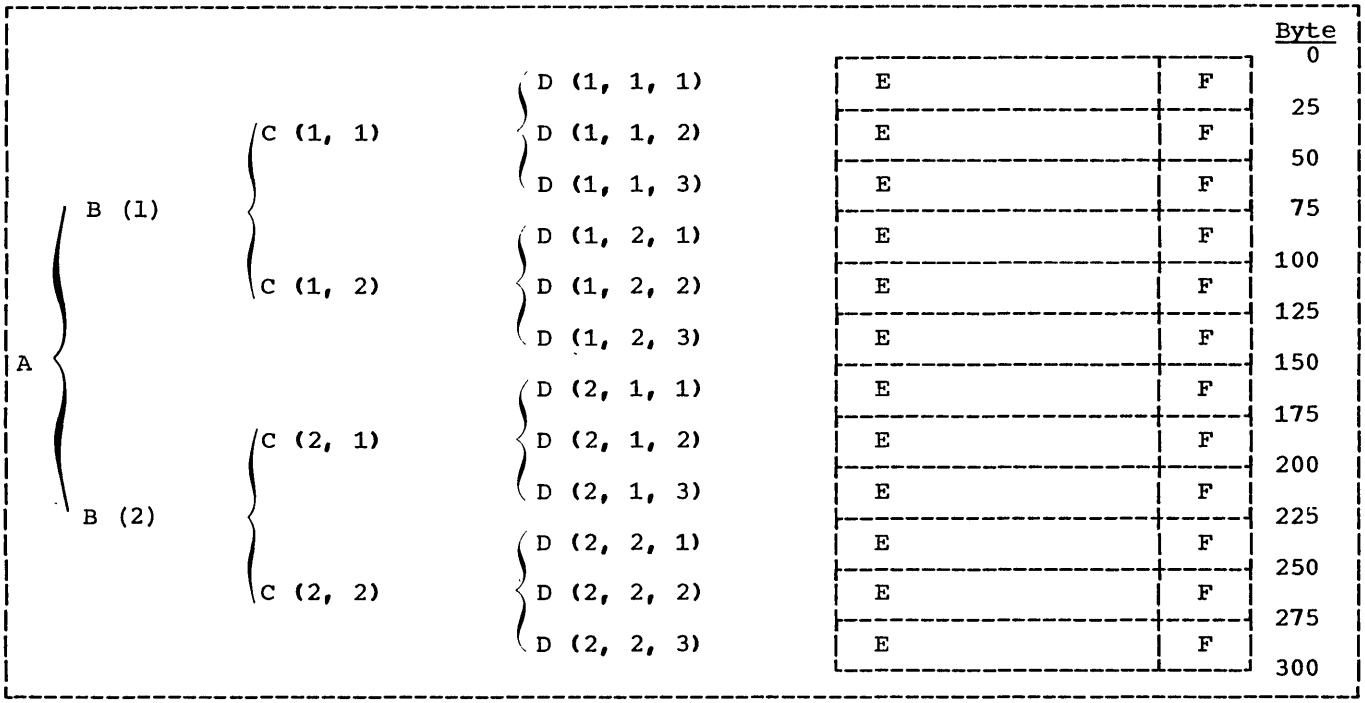


Figure 56. Table Structure in Core Storage

Suppose that a reference to D (2, 2, 3) is necessary. The following method is incorrect:

```
SET I3 TO 2.
SET INDX-DATA-ITM TO I3.
SET I3 UP BY 1.
SET I2, I1 TO INDX-DATA-ITM.
MOVE D (I1, I2, I3) TO WORKAREA.
```

The value contained in I3 after the first SET statement is 25, which represents the beginning point of the second occurrence of D. When the second SET statement is executed, the value 25 is placed in INDX-DATA-ITM, and the fourth SET statement moves the value 25 into I2 and I1. The third SET statement increases the value in I3 to 50. The calculation for the address D (I1, I2, I3) would then be as follows:

```
(address of D (1, 1, 1)) + 25 + 25 + 50
  = (address of D (1, 1, 1)) + 100
```

This is not the address of D (2, 2, 3).

The following method will find the correct address:

```
SET I3 TO 2.
SET I2, I1 TO I3.
SET I3 UP BY 1.
```

In this case, the first SET statement places the value 25 in I3. Since the compiler is able to calculate the lengths of B and C, the second SET statement places the value 75 in I2, and the value 150 in I1. The third SET statement places the value 50 in I3. The correct address calculation will be:

```
(address of D (1, 1, 1)) + 150 + 75 + 50
  = (address of D (1, 1, 1)) + 275
```

The rules for the SET statement are shown in Table 25.

Use care when setting the value of index-names associated with tables described as OCCURS DEPENDING ON. If the table entry length is changed, the value contained within the index-name will become invalid unless a new SET statement corrects it.

Table 25. Rules for the SET Statement

Receiving \ Sending	Index-name	Index data item	Identifier or Literal
Index-name	Set to value corresponding to occurrence number[1]	Move without conversion	Set to value corresponding to occurrence number
Index data item	Move without conversion	Move without conversion	Illegal
Identifier	Set to occurrence number represented by index-name	Illegal	Illegal

[1]If index-names refer to the same table element, move without conversion.

SEARCH Statement

Only one level of a table (a table element) can be referenced with one SEARCH statement. Note that SEARCH statements cannot be nested, since an imperative-statement must follow the WHEN condition, and the SEARCH statement is itself conditional.

To write a series of statements that will search the 3-dimensional table defined in the discussion of the SET statement, the programmer could write:

```
.
.
.

77   COMPARAND1 PIC X(5).
77   COMPARAND2 PIC 9(5).

01   A.
   02 B OCCURS 2 INDEXED BY I1 I5.
      03 C OCCURS 2 INDEXED BY I2 I6.
         04 D OCCURS 3 INDEXED BY I3 I4.
            05 E PIC X(5).
            05 F PIC 9(5).
.
.
.

(Initialize COMPARAND1 and COMPARAND2)

   PERFORM SEARCH-TEST1 THRU SEARCH-EXIT1
   VARYING I1 FROM 1 BY 1 UNTIL I2 IS
   GREATER THAN 2.
ENTRY-NOENTRY1.
   GO TO ERROR-RECOVERY1.
SEARCH-TEST1.
   SET I3 TO 1.
   SEARCH D WHEN E (I1, I2, I3) =
      COMPARAND1 AND F (I1, I2, I3) =
      COMPARAND2
   SET I5 TO I1
   SET I6 TO I2
   SET I2 TO 3
   SET I1 TO 3
   ALTER ENTRY-NOENTRY1 TO PROCEED
      TO ENTRY-PROCESSING1.
SEARCH-EXIT1. EXIT.
.
.
.
ERROR-RECOVERY1.
.
.
.
ENTRY-PROCESSING1.
   MOVE E (I5, I6, I3) TO OUTAREA1.
   MOVE F (I5, I6, I3) TO OUTAREA2.
.
.
.
```

The PERFORM statement varies the indexes (I1 and I2) associated with table elements B and C; the SEARCH statement varies index I3 associated with table element D.

The values of I1 and I2 that satisfy the WHEN conditions of the SEARCH statement are saved in I5 and I6. I1 and I2 are then both set to 3, so that upon return from the SEARCH statement, control will fall through the PERFORM statement to the GO TO statement.

Subsequent references to the desired occurrence of table elements E and F make use of the index-names I5 and I6 in which the correct value was saved.

Format 1 SEARCH statements perform a serial search of a table. If it is certain that the "found" condition is beyond some intermediate point in the table, the

index-names can be set at that point and only that part of the table be searched; this speeds up execution. If the table is large and must be searched from the first occurrence to the last, Format 2 (SEARCH ALL) is more efficient than Format 1, since it uses a binary search technique; however, the table must then be ordered.

In Format 1, the VARYING option allows the programmer to:

- Vary an index-name other than the first index-name stated for this table element. Thus, with two SEARCH statements, each using a different index-name, more than one value can be referenced in the same table element for comparisons, etc.
- Vary an index-name from another table element. In this case, the first index-name specified for this table is used for the SEARCH, and the index-name specified in the VARYING option is incremented at the same time. Thus, the programmer can search two table elements at once.

In Format 1, the WHEN condition can be any relation condition and there can be more than one. If multiple WHEN conditions are stated, the implied logical connective is OR -- that is, if any one of the WHEN conditions is satisfied, the imperative-statement following the WHEN condition is executed. If all conditions are to be satisfied before exiting from the SEARCH, the compound WHEN condition with AND as the logical connective must be written.

In Format 2, the SEARCH ALL statement, the table must be ordered on the key(s) specified in the OCCURS clause. Any key may be specified in the WHEN condition, but all preceding data-names in the KEY option must also be tested. The test must be an "equal to" (=) condition, and the KEY data-name must be either the subject or object of the condition, or the name of a conditional variable with which the tested condition-name is associated. The WHEN condition can also be a compound condition, formed from one of the simple conditions listed above, with AND as the only logical connective. The KEY data item and the item with which it is compared must be compatible, as given in the rules of the relation test.

Compilation is faster if keys are tested in the SEARCH statement in the same order as they appear in the KEY option.

Note that if KEY entries within the table do not contain valid values, then the results of the binary search will be unpredictable.

Building Tables

When reading in data to build an internal table:

1. Check to make sure the data does not exceed the space allocated for the table.
2. If the data must be in sequence, check the sequence.
3. If the data contains the subscript that determines its position in the table, check the subscript for a valid range.

When testing for the end of a table, use a named value giving the item count, rather than using a literal. Then, if the table must be expanded, only one value need be changed, instead of all references to a literal.

The following is a sample COBOL program and the output listing resulting from its compilation, linkage editing, and execution. The program creates a blocked, labeled, standard sequential file, writes it out on tape, and then reads it back in. It also does a check on the field called NO-OF-DEPENDENTS. All data records in the file are displayed. Those with a zero in the NO-OF-DEPENDENTS field are displayed with the special character Z. The records of the file are not altered from the time of creation, despite the fact that the NO-OF-DEPENDENTS field is changed for display purposes. The individual records of the file are created using the subscripting technique. TRACE is used as a debugging aid during program execution.

The output formats illustrated in the listing are described in the chapter "Interpreting Output."

```
// JOB SAMPLE                                                    00.47.14
// OPTION NODECK,LINK,LIST,LISTX,SYM,ERRS
  PHASE TEST,*
// EXEC FCOBOL
  CBL QUOTE


00001   000010 IDENTIFICATION DIVISION.
00002   000020 PROGRAM-ID. TESTRUN.
00003   000030    AUTHOR. PROGRAMMER NAME.
00004   000040    INSTALLATION. NEW YORK PROGRAMMING CENTER.
00005   000050    DATE-WRITTEN. SEPTEMBER 10, 1968.
00006   000060 DATE-COMPILED. 06/20/69
00007   000070    REMARKS. THIS PROGRAM HAS BEEN WRITTEN AS A SAMPLE PROGRAM FOR
00008   000080      COBOL USERS. IT CREATES AN OUTPUT FILE AND READS IT BACK AS
00009   000090      INPUT.
00010   000100
00011   000110 ENVIRONMENT DIVISION.
00012   000120 CONFIGURATION SECTION.
00013   000130 SOURCE-COMPUTER. IBM-360-H50.
00014   000140 OBJECT-COMPUTER. IBM-360-H50.
00015   000150 INPUT-OUTPUT SECTION.
00016   000160 FILE-CONTROL.
00017   000170      SELECT FILE-1 ASSIGN TO SYS008-UT-2400-S.
00018   000180      SELECT FILE-2 ASSIGN TO SYS008-UT-2400-S.
00019   000190
00020   000200 DATA DIVISION.
00021   000210 FILE SECTION.
00022   000220 FD  FILE-1
00023   000230      LABEL RECORDS ARE STANDARD
00024   000240      BLOCK CONTAINS 5 RECORDS
00025   000250      RECORDING MODE IS F
00026   000255      RECORD CONTAINS 20 CHARACTERS
00027   000260      DATA RECORD IS RECORD-1.
00028   000270 01  RECORD-1.
00029   000280      05 FIELD-A PIC X(20).
00030   000290 FD  FILE-2
00031   000300      LABEL RECORDS ARE STANDARD
00032   000310     BLOCK CONTAINS 5 RECORDS
00033   000320     RECORD CONTAINS 20 CHARACTERS
00034   000330     RECORDING MODE IS F
00035   000340     DATA RECORD IS RECORD-2.
00036   000350 01  RECORD-2.
00037   000360     05 FIELD-A PIC X(20).
```

```
00038    000370 WORKING-STORAGE SECTION.
00039    000380 01   FILLER.
00040    000390      02 COUNT PIC S99 COMP SYNC.
00041    000400      02 ALPHABET PIC X(26) VALUE IS "ABCDEFGHIJKLMNOPQRSTUVWXYZ".
00042    000410      02 ALPHA REDEFINES ALPHABET PIC X OCCURS 26 TIMES.
00043    000420      02 NUMBR PIC S99 COMP SYNC.
00044    000430      02 DEPENDENTS PIC X(26) VALUE "01234012340123401234012340".
00045    000440      02 DEPEND REDEFINES DEPENDENTS PIC X OCCURS 26 TIMES.
00046    000450 01   WORK-RECORD.
00047    000460      05 NAME-FIELD PIC X.
00048    000470      05 FILLER PIC X.
00049    000480      05 RECORD-NO PIC 9999.
00050    000490      05 FILLER PIC X VALUE IS SPACE.
00051    000500      05 LOCATION PIC AAA VALUE IS "NYC".
00052    000510      05 FILLER PIC X VALUE IS SPACE.
00053    000520      05 NO-OF-DEPENDENTS PIC XX.
00054    000530      05 FILLER PIC X(7) VALUE IS SPACES.
00055    000540
00056    000550 PROCEDURE DIVISION.
00057    000560 BEGIN. READY TRACE.
00058    000570      NOTE THAT THE FOLLOWING OPENS THE OUTPUT FILE TO BE CREATED
00059    000580      AND INITIALIZES COUNTERS.
00060    000590 STEP-1. OPEN OUTPUT FILE-1. MOVE ZERO TO COUNT, NUMBR.
00061    000600      NOTE THAT THE FOLLOWING CREATES INTERNALLY THE RECORDS TO BE
00062    000610      CONTAINED IN THE FILE, WRITES THEM ON TAPE, AND DISPLAYS
00063    000620      THEM ON THE CONSOLE.
00064    000630 STEP-2. ADD 1 TO COUNT, NUMBR. MOVE ALPHA (COUNT) TO
00065    000640      NAME-FIELD.
00066    000650      MOVE DEPEND (COUNT) TO NO-OF-DEPENDENTS.
00067    000660      MOVE NUMBR TO RECORD-NO.
00068    000670 STEP-3. DISPLAY WORK-RECORD UPON CONSOLE. WRITE RECORD-1 FROM
00069    000680      WORK-RECORD.
00070    000690 STEP-4. PERFORM STEP-2 THRU STEP-3 UNTIL COUNT IS EQUAL TO 26.
00071    000700      NOTE THAT THE FOLLOWING CLOSES THE OUTPUT FILE AND REOPENS
00072    000710      IT AS INPUT.
00073    000720 STEP-5. CLOSE FILE-1. OPEN INPUT FILE-2.
00074    000730      NOTE THAT THE FOLLOWING READS BACK THE FILE AND SINGLES
00075    000740      OUT EMPLOYEES WITH NO DEPENDENTS.
00076    000750 STEP-6. READ FILE-2 RECORD INTO WORK-RECORD AT END GO TO STEP-8.
00077    000760 STEP-7. IF NO-OF-DEPENDENTS IS EQUAL TO "0" MOVE "Z" TO
00078    000770      NO-OF-DEPENDENTS. EXHIBIT NAMED WORK-RECORD. GO TO STEP-6.
00079    000780 STEP-8. CLOSE FILE-2.
00080    000790      STOP RUN.
```

INTRNL NAME	LVL	SOURCE NAME	BASE	DISPL	INTRNL NAME	DEFINITION	USAGE	R	O	Q	M
DNM=1-148	FD	FILE-1	DTF=01		DNM=1-148		DTFMT				F
DNM=1-178	01	RECORD-1	BL=1	000	DNM=1-178	DS 0CL20	GROUP				
DNM=1-199	02	FIELD-A	BL=1	000	DNM=1-199	DS 20C	DISP				
DNM=1-216	FD	FILE-2	DTF=02		DNM=1-216		DTFMT				F
DNM=1-246	01	RECORD-2	BL=2	000	DNM=1-246	DS 0CL20	GROUP				
DNM=1-267	02	FIELD-A	BL=2	000	DNM=1-267	DS 20C	DISP				
DNM=1-287	01	FILLER	BL=3	000	DNM=1-287	DS 0CL56	GROUP				
DNM=1-306	02	COUNT	BL=3	000	DNM=1-306	DS 1H	COMP				
DNM=1-321	02	ALPHABET	BL=3	002	DNM=1-321	DS 26C	DISP				
DNM=1-339	02	ALPHA	BL=3	002	DNM=1-339	DS 1C	DISP	R	O		
DNM=1-357	02	NUMBR	BL=3	01C	DNM=1-357	DS 1H	COMP				
DNM=1-372	02	DEPENDENTS	BL=3	01E	DNM=1-372	DS 26C	DISP				
DNM=1-392	02	DEPEND	BL=3	01E	DNM=1-392	DS 1C	DISP	R	O		
DNM=1-408	01	WORK-RECORD	BL=3	038	DNM=1-408	DS 0CL20	GROUP				
DNM=1-432	02	NAME-FIELD	BL=3	038	DNM=1-432	DS 1C	DISP				
DNM=1-452	02	FILLER	BL=3	039	DNM=1-452	DS 1C	DISP				
DNM=1-471	02	RECORD-NO	BL=3	03A	DNM=1-471	DS 4C	DISP-NM				
DNM=1-490	02	FILLER	BL=3	03E	DNM=1-490	DS 1C	DISP				
DNM=2-000	02	LOCATION	BL=3	03F	DNM=2-000	DS 3C	DISP				
DNM=2-018	02	FILLER	BL=3	042	DNM=2-018	DS 1C	DISP				
DNM=2-037	02	NO-OF-DEPENDENTS	BL=3	043	DNM=2-037	DS 2C	DISP				
DNM=2-063	02	FILLER	BL=3	045	DNM=2-063	DS 7C	DISP				

MEMORY MAP

TGT	003F0
SAVE AREA	003F0
SWITCH	00438
TALLY	0043C
SORT SAVE	00440
ENTRY-SAVE	00444
SORT CORE SIZE	00448
NSTD-REELS	0044C
SORT RET	0044E
WORKING CELLS	00450
SORT FILE SIZE	00580
SORT MODE SIZE	00584
PGT-VN TBL	00588
TGT-VN TBL	0058C
SORTAB ADDRESS	00590
LENGTH OF VN TBL	00594
LNGTH OF SORTAB	00596
PGM ID	00598
A(INIT1)	005A0
UPSI SWITCHES	005A4
OVERFLOW CELLS	005AC
BL CELLS	005AC
DTFADR CELLS	005B8
TEMP STORAGE	005C0
TEMP STORAGE-2	005C8
TEMP STORAGE-3	005C8
TEMP STORAGE-4	005C8
BLL CELLS	005C8
VLC CELLS	005CC
SBL CELLS	005CC
INDEX CELLS	005CC
SUBADR CELLS	005CC
CNCTL CELLS	005D4
PFMCTL CELLS	005D4
PFMSAV CELLS	005D4
VN CELLS	005D8
SAVE AREA =2	005DC
XSASW CELLS	005DC
XSA CELLS	005DC
PARAM CELLS	005DC
RPTSAV AREA	005E0
CHECKPT CTR	005E0
IOPTR CELLS	005E0

LITERAL POOL (HEX)

```
00628 (LIT+C)     00000001  001A5B5B  C2D6D7C5  D5405B5B  C2C3D3C6  E2C55B5B
00640 (LIT+24)    C2C6C3D4  E4D3FCE9  C0000000
```

DISPLAY LITERALS (BCD)

```
0064C (LTL+36)   'WORK-RECORD'
```

PGT	005E8
OVERFLOW CELLS	005E8
VIRTUAL CELLS	005E8
PROCEDURE NAME CELLS	005F4
GENERATED NAME CELLS	00608
SUBDTF ADDRESS CELLS	00620
VNI CELLS	00620
LITERALS	00628
DISPLAY LITERALS	0064C

```
REGISTER ASSIGNMENT

  REG 6    BL =3
  REG 7    BL =1
  REG 8    BL =2

57                000658                          START   EQU   *
                  000658  58 F0 C 004                     L     15,004(0,12)          V(ILBDDSP0)
                  00065C  05 1F                           BALR  1,15
                  00065E  C00140                          DC    X'0C0140'
                  000661  04F5F7404040                    DC    X'04F5F7404040'
57                000668  96 40 D 048                     CI    048(13),X'40'         SWT+0
60                00066C  58 F0 C 004                     L     15,C04(0,12)          V(ILBDDSP0)
                  000670  05 1F                           BALR  1,15
                  C00672  CC014C                          DC    X'000140'
                  000675  04F6F0404040                    DC    X'04F6F0404040'
60                00067C  41 10 C 046                     LA    1,046(0,12)           LIT+6
                  000680  58 C0 D 1C8                     L     0,1C8(0,13)           DTF=1
                  000684  18 40                           LR    4,0
                  000686  C5 FC                           BALR  15,0
                  000688  50 00 F 008                     ST    0,008(0,15)
                  00068C  45 00 F 00C                     BAL   0,00C(0,15)
                  00069C  CCCCC00C                        DC    X'00000000'
                  000694  0A 02                           SVC   2
                  000696  41 00 D 1C8                     LA    0,1C8(0,13)           DTF=1
                  00069A  58 F0 C 008                     L     15,008(0,12)          V(ILBDIML0)
                  00069E  C5 EF                           BALR  14,15
                  0CC6AC  58 10 D 1C8                     L     1,1C8(0,13)           DTF=1
                  0006A4  96 10 1 020                     CI    020(1),X'10'
                  0006A8  50 20 D 1BC                     ST    2,1BC(0,13)           BL =1
                  0CC6AC  58 7C D 1BC                     L     7,1BC(0,13)           BL =1
60                0006B0  D2 01 6 000 C 040               MVC   000(2,6),040(12)      DNM=1-306       LIT+0
                  0006B6  D2 01 6 01C C 040               MVC   01C(2,6),040(12)      DNM=1-357       LIT+0
64                0006BC                          PN=01   EQU   *
                  0006BC  58 FC C 004                     L     15,004(0,12)          V(ILBDDSP0)
                  0006CC  C5 1F                           BALR  1,15
                  0006C2  000140                          DC    X'0C0140'
                  0006C5  C4F6F4404040                    DC    X'04F6F4404040'
64                0006CC  48 30 C 042                     LH    3,042(0,12)           LIT+2
                  0006D0  4A 3C 6 000                     AH    3,000(0,6)            DNM=1-306
                  0006D4  4E 3C D 1D0                     CVD   3,1D0(0,13)           TS=01
                  0C06D8  D7 05 D 1D0 D 1D0               XC    1D0(6,13),1D0(13)     TS=01           TS=01
                  0006DE  94 0F D 1D6                     NI    1D6(13),X'0F'         TS=01+6
                  0C06E2  4F 30 D 1D0                     CVB   3,1D0(0,13)           TS=01
                  0006E6  40 30 6 000                     STH   3,000(0,6)            DNM=1-306
                  0006EA  48 30 C 042                     LH    3,042(0,12)           LIT+2
                  0006EE  4A 3C 6 01C                     AH    3,01C(0,6)            DNM=1-357
                  0006F2  4E 30 D 1D0                     CVD   3,1D0(0,13)           TS=01
                  0C06F6  D7 05 D 1D0 D 1D0               XC    1D0(6,13),1D0(13)     TS=01           TS=01
```

```
                0006FC  94 0F D 1D6              NI    1D6(13),X'0F'       TS=01+6
                0C0700  4F 3C D 1D0              CVB   3,1D0(0,13)         TS=01
                000704  40 3C 6 01C              STH   3,01C(0,6)          DNM=1-357
64              CC0708  41 40 6 002              LA    4,002(0,6)          DNM=1-339
                00070C  48 2C 6 000              LH    2,000(0,6)          DNM=1-306
                0CC710  4C 20 C 042              MH    2,042(0,12)         LIT+2
                0CC714  1A 42                    AR    4,2
                000716  5B 40 C 040              S     4,040(0,12)         LIT+0
                0C071A  50 4C D 1DC              ST    4,1DC(0,13)         SBS=1
                00071E  58 EC D 1DC              L     14,1DC(0,13)        SBS=1
                000722  D2 00 6 038 E 000        MVC   038(1,6),000(14)    DNM=1-432      DNM=1-339
66              000728  41 4C 6 01E              LA    4,01E(0,6)          DNM=1-392
                00072C  48 20 6 000              LH    2,000(0,6)          DNM=1-306
                000730  4C 20 C 042              MH    2,042(0,12)         LIT+2
                000734  1A 42                    AR    4,2
                000736  5B 4C C 040              S     4,040(0,12)         LIT+0
                00073A  50 4C D 1E0              ST    4,1E0(0,13)         SBS=2
                0C073E  58 EC D 1E0              L     14,1E0(0,13)        SBS=2
                000742  D2 00 6 043 E 000        MVC   043(1,6),000(14)    DNM=2-37       DNM=1-392
                CC0748  92 4C 6 044              MVI   044(6),X'40'        DNM=2-37+1
67              00074C  48 3C 6 01C              LH    3,C1C(0,6)          DNM=1-357
                0C0750  4E 30 D 1D0              CVD   3,1D0(0,13)         TS=01
                CCC754  F3 31 6 03A D 1D6        UNPK  03A(4,6),1D6(2,13)  DNM=1-471      TS=07
                00075A  96 F0 6 03D              OI    03D(6),X'F0'        DNM=1-471+3
68              00075E  58 FC C 004              L     15,004(0,12)        V(ILBDDSP0)
                000762  05 1F                    BALR  1,15
                000764  000140                  DC    X'000140'
                000767  C4F6F84C4040             DC    X'04F6F8404040'
68              00076E  58 FC C 004              L     15,C04(0,12)        V(ILBDDSP0)
                000772  05 1F                    BALR  1,15
                000774  0002                    DC    X'0002'
                000776  00                      DC    X'00'
                0CC777  000014                   DC    X'000014'
                CC077A  CDCCC1C4                 DC    X'0D0001C4'         BL =3
                00077E  0C38                     DC    X'0038'
                CC078C  FFFF                     DC    X'FFFF'
68              000782  D2 13 7 000 6 038        MVC   000(20,7),038(6)    DNM=1-178      DNM=1-408
                000788  58 10 D 1C8              L     1,1C8(0,13)         DTF=1
                00078C  18 41                    LR    4,1
                00078E  58 F0 1 010              L     15,C1C(0,1)
                000792  45 E0 F 00C              BAL   14,C0C(0,15)
                C00796  50 20 D 1BC              ST    2,1BC(0,13)         BL =1
                00079A  58 70 D 1BC              L     7,1BC(0,13)         BL =1
                00C79E                   GN=01   EQU   *
                0CC79E  58 10 D 1E8              L     1,1E8(0,13)         VN=01
                0007A2  C7 F1                    BCR   15,1
70              CCC7A4                   PN=02   EQU   *
                0007A4  58 FC C 004              L     15,C04(0,12)        V(ILBDDSP0)
                CC07A8  05 1F                    BALR  1,15
                CCC7AA  000140                   DC    X'000140'
                0007AC  04F7F04C4040             DC    X'04F7F0404040'
70              0007B4  58 CC D 1E8              L     0,1E8(0,13)         VN=01
                0C07B8  50 CC D 1E4              ST    0,1E4(0,13)         PSV=1
                0007BC  58 0C C 024              L     0,024(0,12)         GN=02
                0CC7CC  50 CC D 1E8              ST    0,1E8(0,13)         VN=01
                0007C4                   GN=02   EQU   *
                0C07C4  48 3C 6 000              LH    3,C00(0,6)          DNM=1-306
                0007C8  49 3C C 044              CH    3,044(0,12)         LIT+4
                0007CC  58 FC C 028              L     15,C28(0,12)        GN=03
                0007DC  C7 8F                    BCR   8,15
```

```
                0007D2  58 1C C 00C                    L       1,00C(0,12)             PN=01
                C0C7D6  07 F1                          BCR     15,1
                0007D8                         GN=03   EQU     *
                0007D8  58 0C D 1E4                    L       0,1E4(0,13)             PSV=1
                0CC7DC  50 CC D 1E8                    ST      0,1E8(0,13)             VN=01
     73         0007E0  58 FC C 004                    L       15,C04(0,12)            V(ILBDDSPO)
                0007E4  05 1F                          BALR    1,15
                0007E6  C0014C                         DC      X'000140'
                0007E9  04F7F3404040                   DC      X'04F7F3404040'
     73         0C07FC  58 1C D 1C8                    L       1,1C8(0,13)             DTF=1
                0007F4  94 EF 1 020                    NI      020(1),X'EF'
                0007F8  18 C1                          LR      0,1
                0007FA  18 4C                          LR      4,0
                0007FC  41 1C C 04E                    LA      1,04E(0,12)             LIT+14
                CC08CC  C7 CC                          BCR     0,0
                CC08C2  C5 FC                          BALR    15,0
                000804  50 CC F 008                    ST      0,008(0,15)
                CCC8C8  45 CC F 00C                    BAL     0,00C(0,15)
                00080C  CCCCC000                       DC      X'00000000'
                0C0810  0A 02                          SVC     2
                0CC812  58 CC D 1C8                    L       0,1C8(0,13)             DTF=1
                000816  41 10 C 05o                    LA      1,056(0,12)             LIT+22
                00081A  0A 02                          SVC     2
     73         00081C  41 1C C 046                    LA      1,046(0,12)             LIT+6
                000820  58 CC D 1CC                    L       0,1CC(0,13)             DTF=2
                0CC824  18 4C                          LR      4,0
                000826  05 FC                          BALR    15,0
                000828  5C CC F 008                    ST      0,008(0,15)
                00082C  45 CC F 00C                    BAL     0,00C(0,15)
                00083C  CCCCC000                       DC      X'000C0000'
                0CC834  CA C2                          SVC     2
                0CC836  41 CC D 1CC                    LA      0,1CC(0,13)             DTF=2
                00083A  58 FC C 008                    L       15,C0E(0,12)            V(ILBDIMLO)
                CC083E  05 EF                          BALR    14,15
                00084C  58 10 D 1CC                    L       1,1CC(0,13)             DTF=2
                000844  96 10 1 020                    OI      020(1),X'10'
     76         0CC848                         PN=03   EQU     *
                000848  58 FC C 004                    L       15,C04(0,12)            V(ILBDDSPO)
                0CC84C  05 1F                          BALR    1,15
                00C84E  0C0140                         DC      X'0C0140'
                C0C851  04F7F64C4040                   DC      X'04F7F64C4040'
     76         0CC85E  58 1C D 1CC                    L       1,1CC(0,13)             DTF=2
                00085C  58 FC C 02C                    L       15,02C(0,12)            GN=04
                0C0860  91 20 1 010                    TM      010(1),X'20'
                000864  07 1F                          BCR     1,15
                000866  18 41                          LR      4,1
                000868  41 FC C 02C                    LA      15,02C(0,12)            GN=04
                00086C  D2 C2 1 025 F 001              MVC     025(3,1),001(15)
                000872  58 F0 1 010                    L       15,C1C(0,1)
                CCC876  45 EC F 008                    BAL     14,008(0,15)
                00087A  50 2C D 1C0                    ST      2,1C0(0,13)             BL =2
                00C87E  58 8C D 1C0                    L       8,1C0(0,13)             BL =2
                CCC882  D2 13 6 038 8 000              MVC     038(20,6),000(8)        DNM=1-408       DNM=1-?46
                000888  58 F0 C 018                    L       15,C18(0,12)            PN=04
                0C088C  07 FF                          BCR     15,15
     76         00088E                         GN=04   EQU     *
                00088E  58 1C C 01C                    L       1,01C(0,12)             PN=05
                C0C892  C7 F1                          BCR     15,1
     77         0CC894                         PN=04   EQU     *
                000894  58 FC C 004                    L       15,C04(0,12)            V(ILBDDSPO)
```

```
                 0CC898  05 1F                          BALR  1,15
                 0CC89A  CC014C                        DC    X'000140'
                 0CC89D  C4F7F74C4040                  DC    X'04F7F7404040'
77               0008A4  58 1C C 034                   L     1,034(0,12)          GN=06
                 CCC8A8  58 2C C 030                   L     2,030(0,12)          GN=05
                 0008AC  D5 00 C 05E 6 043             CLC   05E(1,12),043(6)     LIT+30          DNM=2-37
                 0008B2  07 72                         BCR   7,2
                 0C08B4  95 4C 6 044                   CLI   044(6),X'40'         DNM=2-37+1
                 0008B8  07 72                         BCR   7,2
77               CCC8BA                        GN=06   EQU   *
                 0008BA  D2 CC 6 043 C 05F             MVC   043(1,6),05F(12)     DNM=2-37        LIT+31
                 0C08C0  92 40 6 044                   MVI   044(6),X'40'         DNM=2-37+1
78               CCC8C4                        GN=05   EQU   *
                 0008C4  58 1C C 060                   L     1,060(0,12)          LIT+32
                 0CC8C8  50 10 D 1EC                   ST    1,1EC(0,13)          PRM=1
                 0C08CC  41 20 D 1EC                   LA    2,1EC(0,13)          PRM=1
                 0008D0  58 FC C 004                   L     15,004(0,12)         V(ILBDDSP0)
                 0008D4  05 1F                         BALR  1,15
                 0CC8D6  8001                          DC    X'8001'
                 0C08D8  10                            DC    X'10'
                 CCC8D9  CC000B                        DC    X'00000B'
                 0008DC  CCC00064                      DC    X'CC0C0064'          LIT+36
                 0CC8E0  0000                          DC    X'0000'
                 0CC8E2  00                            DC    X'00'
                 0008E3  000014                        DC    X'000C14'
                 0CC8E6  0DC0C1C4                      DC    X'0D0001C4'          BL =3
                 0008EA  CC38                          DC    X'0C38'
                 0008EC  FFFF                          DC    X'FFFF'
78               0008EE  58 1C C 014                   L     1,014(0,12)          PN=03
                 0008F2  07 F1                         BCR   15,1
79               0C08F4                        PN=05   EQU   *
                 0008F4  58 FC C 004                   L     15,004(0,12)         V(ILBDDSP0)
                 0008F8  05 1F                         BALR  1,15
                 0CC8FA  0C0140                        DC    X'000140'
                 0008FC  C4F7F94C4040                  DC    X'04F7F9404040'
79               0C0904  58 10 D 1CC                   L     1,1CC(0,13)          DTF=2
                 CC0908  94 EF 1 020                   NI    020(1),X'EF'
                 00090C  18 01                         LR    0,1
                 00090E  18 4C                         LR    4,0
                 0C0910  41 1C C 04E                   LA    1,04E(0,12)          LIT+14
                 000914  07 CC                         BCR   0,0
                 CCC916  C5 FC                         BALR  15,0
                 000918  5C CC F 008                   ST    0,008(0,15)
                 00091C  45 0C F 00C                   BAL   0,00C(0,15)
                 CCC92C  CCOCC000                      DC    X'00000000'
                 000924  0A 02                         SVC   2
                 000926  58 CC D 1CC                   L     0,1CC(0,13)          DTF=2
                 00092A  41 1C C 056                   LA    1,056(0,12)          LIT+22
                 00092E  0A 02                         SVC   2
80               0CC930  0A CE                         SVC   14
                 0CC932  CA CE                         SVC   14
                 0C0934  50 DC 5 008           INIT2   ST    13,C08(0,5)
                 0C0938  50 5C D 004                   ST    5,004(0,13)
                 0C093C  58 2C C 000                   L     2,000(0,12)          VIR=1
                 0C0940  95 00 2 000                   CLI   000(2),X'00'
                 0CC944  C7 79                         BCR   7,9
                 000946  92 FF 2 000                   MVI   000(2),X'FF'
                 00094A  96 10 D 048                   CI    048(13),X'10'        SWT+0
                 CCC94E  50 EC D 054           INIT3   ST    14,054(0,13)
                 000952  C5 FC                         BALR  15,C
```

```
000954  91 20 D 048                TM    048(13),X'20'        SWT+0
000958  47 E0 F 016                BC    14,016(0,15)
00095C  58 C0 B 048                L     0,048(0,11)
000960  98 2D B 050                LM    2,13,050(11)
000964  58 E0 D 054                L     14,054(0,13)
000968  07 FE                      BCR   15,14
00096A  96 20 D 048                CI    048(13),X'20'        SWT+0
00096E  41 60 0 004                LA    6,004(0,0)
000972  41 10 C 00C                LA    1,00C(0,12)          PN=01
000976  41 70 C 040                LA    7,040(0,12)          LIT+0
00097A  06 70                      BCTR  7,0
00097C  05 50                      BALR  5,0
00097E  58 40 1 000                L     4,000(0,1)
000982  1E 4B                      ALR   4,11
000984  50 40 1 000                ST    4,000(0,1)
000988  87 16 5 000                BXLE  1,6,000(5)
00098C  41 80 D 1BC                LA    8,1BC(0,13)          OVF=1
000990  41 70 D 1CF                LA    7,1CF(0,13)          TS=01-1
000994  05 10                      BALR  1,0
000996  58 C0 8 000                L     0,000(0,8)
00099A  1E CB                      ALR   0,11
00099C  50 C0 8 000                ST    0,000(0,8)
0009A0  87 86 1 000                BXLE  8,6,000(1)
0009A4  D2 C3 D 1E8 C 038          MVC   1E8(4,13),038(12)    VN=01-C      VNI=1
0009AA  58 60 D 1C4                L     6,1C4(0,13)          BL =3
0009AE  58 70 D 1BC                L     7,1BC(0,13)          BL =1
0009B2  58 80 D 1C0                L     8,1C0(0,13)          BL =2
0009B6  58 E0 D 054                L     14,054(0,13)
0009BA  07 FE                      BCR   15,14
0009C0  05 F0              INIT1   BALR  15,0
0009C2  07 00                      BCR   0,0
0009C4  90 CE F 00A                STM   0,14,00A(15)
0009C8  47 F0 F 082                BC    15,082(0,15)
0009CC                             DS    30F
000A84  58 C0 F 0C6                L     12,0C6(0,15)
000A88  58 E0 C 000                L     14,000(0,12)         VIR=1
000A8C  58 D0 F 0CA                L     13,0CA(0,15)
000A90  95 00 E 000                CLI   000(14),X'00'
000A94  47 70 F 0A2                BC    7,0A2(0,15)
000A98  96 10 D 048                CI    048(13),X'10'        SWT+0
000A9C  92 FF E 000                MVI   000(14),X'FF'
000AA0  47 F0 F 0AC                BC    15,0AC(0,15)
000AA4  98 CE F 03A                LM    12,14,03A(15)
000AA8  90 E0 D 00C                STM   14,12,00C(13)
000AAC  18 5D                      LR    5,13
000AAE  98 9F F 0BA                LM    9,15,0BA(15)
000AB2  91 10 D 048                TM    048(13),X'10'        SWT+0
000AB6  07 19                      BCR   1,9
000AB8  07 FF                      BCR   15,15
000ABA  07 00                      BCR   0,0
000ABC  00000A94                   ADCON L4(INIT3)
000AC0  00000000                   ADCON L4(INIT1)
000AC4  00000000                   ADCON L4(INIT1)
000AC8  00000E8                    ADCON L4(PGT)
000ACC  00000F0                    ADCON L4(TGT)
000AD0  00000658                   ADCON L4(START)
000AD4  00000934                   ADCON L4(INIT2)
000AD8  C3D6C2C6F0F0F0F0           DC    X'C3D6C2C6F0F0F0F0'
000AE0  E3C5E2E3D9E4D540           DC    X'E3C5E2E3D9E4D540'
```

CROSS-REFERENCE DICTIONARY

DATA NAMES	DEFN	REFERENCE					
FILE-1	00017	00060	00060	00068	00073		
RECORD-1	00028	00068					
FILE-2	00018	00073	00073	00076	00076	00079	
RECORD-2	00036	00076					
COUNT	00040	00060	00064	00064	00064	00066	00070
ALPHA	00042	00064	00064				
NUMBR	00043	00060	00064	00064	00067		
DEPEND	00045	00066	00066				
WORK-RECORD	00046	00068	00068	00076	00078		
NAME-FIELD	00047	00064					
RECORD-NO	00049	00067	00067				
NO-OF-DEPENDENTS	00053	00066	00066	00077	00077	00077	00077

PROCEDURE NAMES	DEFN	REFERENCE
STEP-2	00064	00070
STEP-6	00076	00078
STEP-8	00079	00076

CARD	ERROR MESSAGE	
64	ILA5011I-W	HIGH ORDER TRUNCATION MIGHT OCCUR.
64	ILA5011I-W	HIGH ORDER TRUNCATION MIGHT OCCUR.

```
 ENTRY
// EXEC LNKEDT
```

JOB SAMPLE 06/20/69 DISK LINKAGE EDITOR DIAGNOSTIC OF INPUT

```
ACTION TAKEN    MAP
LIST   PHASE TEST,*
LIST   AUTOLINK   IJFFBZZN
LIST   AUTOLINK   ILBDDSPC
LIST   INCLUDE IJJCPD1
LIST   AUTOLINK   ILBDIMLC
LIST   AUTOLINK   ILBDMNSC
LIST   AUTOLINK   ILBDSAEO
LIST   ENTRY
```

06/20/69	PHASE	XFR-AD	LOCORE	HICORE	DSK-AD	ESD TYPE	LABEL	LOADED	REL-FR
	TEST	C03000	003000	0048CB	50 07 2	CSECT	TESTRUN	003000	003000
						CSECT	IJFFBZZN	0039C0	0039C0
						* ENTRY	IJFFZZZN	0039C0	
						* ENTRY	IJFFBZZZ	0039C0	
						* ENTRY	IJFFZZZZ	0039C0	
						CSECT	ILBDSAE0	0047D8	0047D8
						ENTRY	ILBDSAE1	0047F8	
						CSECT	ILBDMNS0	0047D0	0047D0
						CSECT	ILBDDSP0	003F90	003F90
						* ENTRY	ILBDDSP1	0044E0	
						* ENTRY	ILBDDSP2	004578	
						* ENTRY	ILBDDSP3	004730	
						CSECT	ILBDIML0	004768	004768
						CSECT	IJJCPD1	003DC8	003DC8
						ENTRY	IJJCPD1N	003DC8	
						* ENTRY	IJJCPD3	003DC8	

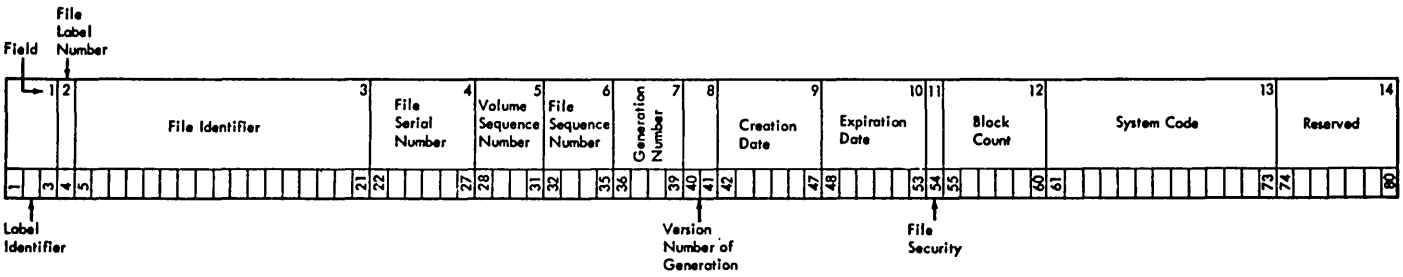
```
// ASSGN SYS008,X'282'
// TLBL SYS008,'TAPEFILE',69/365,,0001,0001
// EXEC

60
64
68
70
64
68
64
68
64
68
64
68
64
68
64
68
64
68
64
68
64
68
64
68
64
68
64
68
64
68
64
68
64
68
64
68
64
68
64
68
64
68
64
68
64
68
64
68
64
68
64
68
73
76
77
WORK-RECORD = AC0001 NYC Z
76
77
WORK-RECORD = BC0002 NYC 1
```

```
76
77
WORK-RECORD = CC0003 NYC 2
76
77
WORK-RECORD = DC0004 NYC 3
76
77
WORK-RECORD = EC0005 NYC 4
76
77
WORK-RECORD = FC0006 NYC Z
76
77
WORK-RECORD = GC0007 NYC 1
76
77
WORK-RECORD = HC0008 NYC 2
76
77
WORK-RECORD = IC0009 NYC 3
76
77
WORK-RECORD = JC0010 NYC 4
76
77
WORK-RECORD = KC0011 NYC Z
76
77
WORK-RECORD = LC0012 NYC 1
76
77
WORK-RECORD = MC0013 NYC 2
76
77
WORK-RECORD = NC0014 NYC 3
76
77
WORK-RECORD = OC0015 NYC 4
76
77
WORK-RECORD = PC0016 NYC Z
76
77
WORK-RECORD = QC0017 NYC 1
76
77
WORK-RECORD = RC0018 NYC 2
76
77
WORK-RECORD = SC0019 NYC 3
76
77
WORK-RECORD = TC0020 NYC 4
76
77
WORK-RECORD = UC0021 NYC Z
76
77
WORK-RECORD = VC0022 NYC 1
76
77
```

```
WORK-RECORC = WC0023 NYC 2
76
77
WORK-RECORC = XC0024 NYC 3
76
77
WORK-RECORD = YC0025 NYC 4
76
77
WORK-RECORD = ZC0026 NYC Z
76
79


EOJ SAMPLE
```

~~~~~~

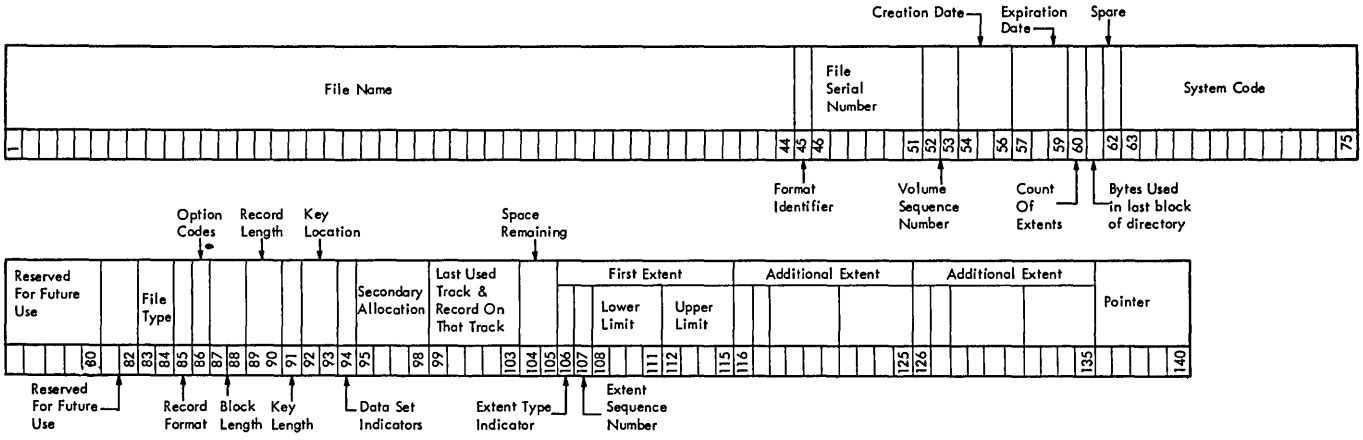
```
                    0I10A GIVE IPL CONTROL COMMANDS
set date=06/20/69,clock=00/40/00
BG 0I20I DOS IPL COMPLETE
BG 1I00A  READY FOR COMMUNICATIONS.
BG
BG // JOB SAMPLE
   00.40.13
BG
   4110A NO VOL1 LBL FOUND     TLBL=     SYS008    SYS008=282
BG 111111
BG AC0001 NYC 0
BG BC0002 NYC 1
BG CC0003 NYC 2
BG DC0004 NYC 3
BG EC0005 NYC 4
BG FC0006 NYC 0
BG GC0007 NYC 1
BG HC0008 NYC 2
BG IC0009 NYC 3
BG JC0010 NYC 4
BG KC0011 NYC 0
BG LC0012 NYC 1
BG MC0013 NYC 2
BG NC0014 NYC 3
BG OC0015 NYC 4
BG PC0016 NYC 0
BG QC0017 NYC 1
BG RC0018 NYC 2
BG SC0019 NYC 3
BG TC0020 NYC 4
BG UC0021 NYC 0
BG VC0022 NYC 1
BG WC0023 NYC 2
BG XC0024 NYC 3
BG YC0025 NYC 4
BG ZC0026 NYC 0
BG 000101 69171
   4133D ERROR IN HDR LBL               SYS008    SYS008=282
BG ignore
BG EOJ SAMPLE
   00.45.07,DURATION 00.04.54
```
~~~~~~



The standard tape file label format and contents are as follows:

Field	Name and Length	Description
1.	LABEL IDENTIFIER 3 bytes, EBCDIC	Identifies the type of label. HDR = Header (beginning of a data file) EOF = End-of-file (end of a set of data) EOV = End-of-volume (end of the physical reel)
2.	FILE LABEL NUMBER 1 byte, EBCDIC	Always a 1.
3.	FILE IDENTIFIER 17 bytes, EBCDIC	Uniquely identifies the entire file, may contain only printable characters. Some other systems will not accept embedded blanks in the file identifier.
4.	FILE SERIAL NUMBER 6 bytes, EBCDIC	Uniquely identifies a file/volume relationship. This field is identical to the volume serial number in the volume label of the first or only volume of a multivolume file or a multifile set. This field will normally be numeric (000001 to 999999), but may contain any six alphanumeric characters.
5.	VOLUME SEQUENCE NUMBER 4 bytes	Indicates the order of a volume in a given file or multifile set. The first must be numbered 0001, and subsequent numbers must be in proper numeric sequence.
6.	FILE SEQUENCE 4 bytes	Assigns numeric sequence to a file within a multi-file set. The first must be numbered 0001.
7.	GENERATION TIME 4 bytes	Uniquely identifies the various editions of the file. May be from 0001 to 9999 in proper numeric sequence.
8.	VERSION NUMBER OF GENERATION 2 bytes	Indicates the version of a generation of a file.

Field	Name and Length	Description
9.	CREATION DATE 6 bytes	Indicates the year and the day of the year that the file was created. Position: 1 — Code: blank — Meaning: none Position: 2-3 — Code: 00-99 — Meaning: year Position: 4-6 — Code: 001-366 — Meaning: day of year (e.g., January 31, 1971 would be entered as 71031).
10.	EXPIRATION DATE 6 bytes	Indicates the year and the day of the year when the file may become a scratch tape. The format of this field is identical to field 9. On a multifile reel processed sequentially, all files are considered to expire on the same day.
11.	FILE SECURITY 1 byte	Indicates security status of the file. 0 = No security protection. 1 = Security protection. Additional identification of the file is required before it can be processed.
12.	BLOCK COUNT 6 bytes	Indicates the number of data blocks written in the file from the last header label to the first trailer label, exclusive of tapemarks. Count does not include checkpoint records. This field is used in trailer labels.
13.	SYSTEM CODE 13 bytes	Uniquely identifies the operating system.
14.	RESERVED 7 bytes	Reserved. Should be recorded as blanks.



Format 1: This format is common to all data files on disk.

Field	Name and Length	Description
1.	FILE NAME 44 bytes, alphanumeric EBCDIC	This field serves as the key portion of the file label. It can consist of three sections: 1. File ID is an alphanumeric field assigned by the user and identifies the file. It can be 1 through 35 bytes in length if generation and version numbers are used, or 1 through 44 bytes in length if they are not used. 2. Generation Number. If used, this field is separated from File ID by a period. It has the format Gnnnn, where G identifies the field as the generation number and nnnn (in decimal) identifies the generation of the file. 3. Version Number of Generation. If used, this section immediately follows the generation number and has the format Vnn, where V identifies the field as the version of generation number and nn (in decimal) identifies the version of generation of the file.

Note: IBM System/360 Disk Operating System compares the entire field against the filename given in the DLAB and DLBL cards. The generation and version numbers are treated differently by the IBM System/360 Operating System.

Fields 2 through 33 constitute the DATA portion of the file label.

Field	Name and Length	Description
2.	FORMAT IDENTIFIER 1 byte, EBCDIC numeric	1 = format 1
3.	FILE SERIAL NUMBER 6 bytes, alphanumeric EBCDIC	Uniquely identifies a file/volume relationship. It is identical to the volume serial number of the first or only volume of a multivolume file.
4.	VOLUME SEQUENCE NUMBER 2 bytes, binary	Indicates the order of a volume relative to the first volume on which the data file resides.
5.	CREATION DATE 3 bytes, discontinuous binary	Indicates the year and the day of the year the file was created. It is of the form YDD, where Y signifies the year (0-99) and DD the day of the year (1-366).
6.	EXPIRATION DATE 3 bytes, discontinuous binary	Indicates the year and the day of the year the file may be deleted. The form of this field is identical to that of field 5.
7a.	EXTENT COUNT 1 byte, binary	Contains a count of the number of extents for this file on this volume. If user labels are used, the count includes the user label track as a separate extent. This field is maintained by the Disk Operating System.
7b.	BYTES USED IN LAST BLOCK OF DIRECTORY 1 byte, binary	Used by IBM System/360 Operating System only for partitioned (library structure) data sets. Not used by the Disk Operating System.
7c.	SPARE 1 byte	Reserved for future use.
8.	SYSTEM CODE 13 bytes	Uniquely identifies the operating system.
9.	RESERVED 7 bytes	Reserved for future use.
10.	FILE TYPE 2 bytes	The contents of this field uniquely identify the type of data file.

Hex Code	Meaning
4000	Sequential organization
2000	Direct organization
8000	Indexed organization
0200	Library organization
0000	Organization not defined in the file label

Field	Name and Length	Description
11.	RECORD FORMAT 1 byte	The contents of this field indicate the type of records contained in the file.

Bit Position	Content	Meaning
0 and 1	01	Variable-length records
	10	Fixed-length records
	11	Undefined format
2	0	No track overflow
	1	File is organized using track overflow (IBM System/360 Operating System only)
3	0	Unblocked records
	1	Blocked records
4	0	No truncated records
	1	Truncated records in file
5 and 6	01	Control character ASA code
	10	Control character machine code
	00	Control character not stated
7	0	Records are written without keys
	1	Records are written with keys

12.	OPTION CODES 1 byte	Bits within this field are used to indicate various options used in building the file.

Bit Position	Meaning
0	If on, indicates data file was created using write validity check.
1-7	Unused.

13.	BLOCK LENGTH 2 bytes, binary	Indicates the block length for fixed-length records, or maximum block size for variable-length blocks.
14.	RECORD LENGTH 2 bytes, binary	Indicates the record length for fixed-length records, or the maximum record length for variable-length records.
15.	KEY LENGTH 1 byte, binary	Indicates the length of the key portion of the data records in the file.
16.	KEY LOCATION 2 bytes, binary	Indicates the high-order position of the data record.

Field	Name and Length	Description
17.	DATA SET INDICATORS 1 byte	Bits within this field are used to indicate the following:

Bit Position	Meaning
0	If on, indicates that this is the last volume on which this file normally resides. This bit is used by the Disk Operating System DTFSR routine only. None of the other bits in this byte are used by the Disk Operating System.
1	If on, indicates that the data set described by this file must remain in the same absolute location on the direct-access device.
2	If on, indicates that block length must always be a multiple of eight bytes.
3	If on, indicates that this data file is security protected; a password must be provided in order to access it.
4-7	Space. Reserved for future use.

18.	SECONDARY ALLOCATION 4 bytes, binary	Indicates the amount of storage to be requested for this data file at end-of-extent. This field is used by the IBM System/360 Operating System only. It is not used by the Disk Operating System routines.
19.	LAST USED TRACK AND RECORD ON THAT TRACK 5 bytes, discontinuous binary	Indicates the last occupied track in a consecutive file organization data file. This field has the format CCHHR. It is all binary zeros if the last track in a consecutive data file is not on this volume, or if it is not consecutive organization.
20.	AMOUNT OF SPACE REMAINING ON LAST TRACK USED 2 bytes, binary	A count of the number of bytes of available space remaining on the last track used by this data file on this volume.
21.	EXTENT TYPE INDICATOR 1 byte	Indicates the type of extent with which the following fields are associated:

Hex Code	Meaning
00	Next three fields do not indicate any extent.
01	Prime area (indexed) or consecutive area, etc., (i.e., the extent containing the user's data records).
02	Overflow area of an indexed file.
04	Cylinder index or master index area of an indexed file.
40	User label track area.
80	Shared cylinder indicator.

Field	Name and Length	Description
22.	EXTENT SEQUENCE NUMBER 1 byte, binary	Indicates the extent sequence in a multi-extent file.
23.	LOWER LIMIT 4 bytes, discontinuous binary	The cylinder and the track address specifying the starting point (lower limit) of this extent component. This field has the format CCHH.
24.	UPPER LIMIT 4 bytes	The cylinder and the track address specifying the end point (upper limit) of this extent component. This field has the format CCHH.
25-28.	ADDITIONAL EXTENT 10 bytes	These fields have the same format as the fields 21 through 24, above.
29-32.	ADDITIONAL EXTENT 10 bytes	These fields have the same format as fields 21 through 24, above.
33.	POINTER TO NEXT FILE LABEL WITHIN THIS LABEL SET 5 bytes, discontinuous binary	The disk address (format CCHHR) of a continuation label is needed to further describe the file. If field 9 indicates indexed organization, this field will point to a Format 2 file label within this label set. Otherwise, it points to a Format 3 file label, and then only if the file contains more than three extent segments. If no additional file label is pointed to, this field contains all binary zeros.

APPENDIX D: TRACK FORMATS FOR THE 2311, 2314, AND 2321 DIRECT-ACCESS STORAGE DEVICES

The track format for the 2311, 2314, and 2321 direct-access storage devices is illustrated in Figure 57. The names of the fields are given in the following discussion.

Index Marker: All tracks start with an index marker. It is a signal to the hardware that indicates beginning of the track.

Home Address: The home address, preceded by a gap, follows the index marker. The home address uniquely identifies each track by specifying the cylinder and head number.

Track Descriptor Record (Record 0): Record 0 consists of two parts: a count portion and a data portion. The count portion is the same as it is for any other record (see the following description of count for record 1. The 8-byte data portion is used to record information used by LIOCS. The information in the data portion depends on the data organization (direct or indexed) that is being used.

For direct organization, this portion in the form of CCHHR contains the address of the last record on the track and the number of bytes remaining on the track. This information is used to determine whether there is space for another record on the track. For indexed organization, the data portion contains the address of the last record in the cylinder overflow area and the number of tracks remaining in the cylinder overflow area. Record 0 is then used as the cylinder overflow control record.

Address Marker: All records after record 0 will be preceded by a 2-byte address marker. The address marker is a signal to the hardware that a record is starting.

Data Records: Data records (see R1 in Figure 49) can consist of a count and data portion for sequential organization, or a count, key, and data portion for direct and indexed organizations.

1. Count Portion. The count portion contains the identification of each record, the key length, and the data length.

   a. Identification. Each record is identified with its cylinder number, head number, or record number. The cylinder and head numbers will be the same as those of the home address. The record number will indicate a particular record on the track. That is, the first record after record 0 will be record 1, followed by record 2, etc. This 5-byte binary field in the form of CCHHR is often referred to as the record ID.

   b. Key Length. The key length is specified in an 8-bit byte; its length can range from 0 to 255. This field will contain a zero if there is no key.

   c. Data Length. The data length is specified in the 16 bits of the next two bytes.

   Note: It is the count portion that identifies the presence or absence of a key, in addition to indicating the data length. In this way, each record is unique and self formatting.

2. Key Portion. The key portion of the record is normally used to store the control field of the data record such as a man number. Direct and indexed files must have a key portion.

3. Data Portion. The data portion of the record contains the data record.

Note that all records, including the data record, terminate with a 2-byte cyclic check. The hardware uses this cyclic check to ensure that is correctly reread what it had written. The cyclic check is cumulative and is appended to each record when it is written. Upon reading the record, the cyclic check is again accumulated and then compared with the appended cyclic check. If they do not agree, a data check is initiated.

The first byte of the count portion of each record and the home address is reserved for a flag byte. If a track becomes defective, a utility program may be used to transfer the data to an alternate track. (Cylinders 200 through 202 are reserved for alternate tracks on the 2321. Strips 6 through 9 of subcell 19 of each cell are reserved for alternate tracks on the 2321.) In this case, a flag bit within the byte is set on to indicate that this is a defective track and the address of an alternate track will be placed in the record ID of record 0. Subsequent references to this defective track will result in the Supervisor accessing record 0 for the address of the alternate track.

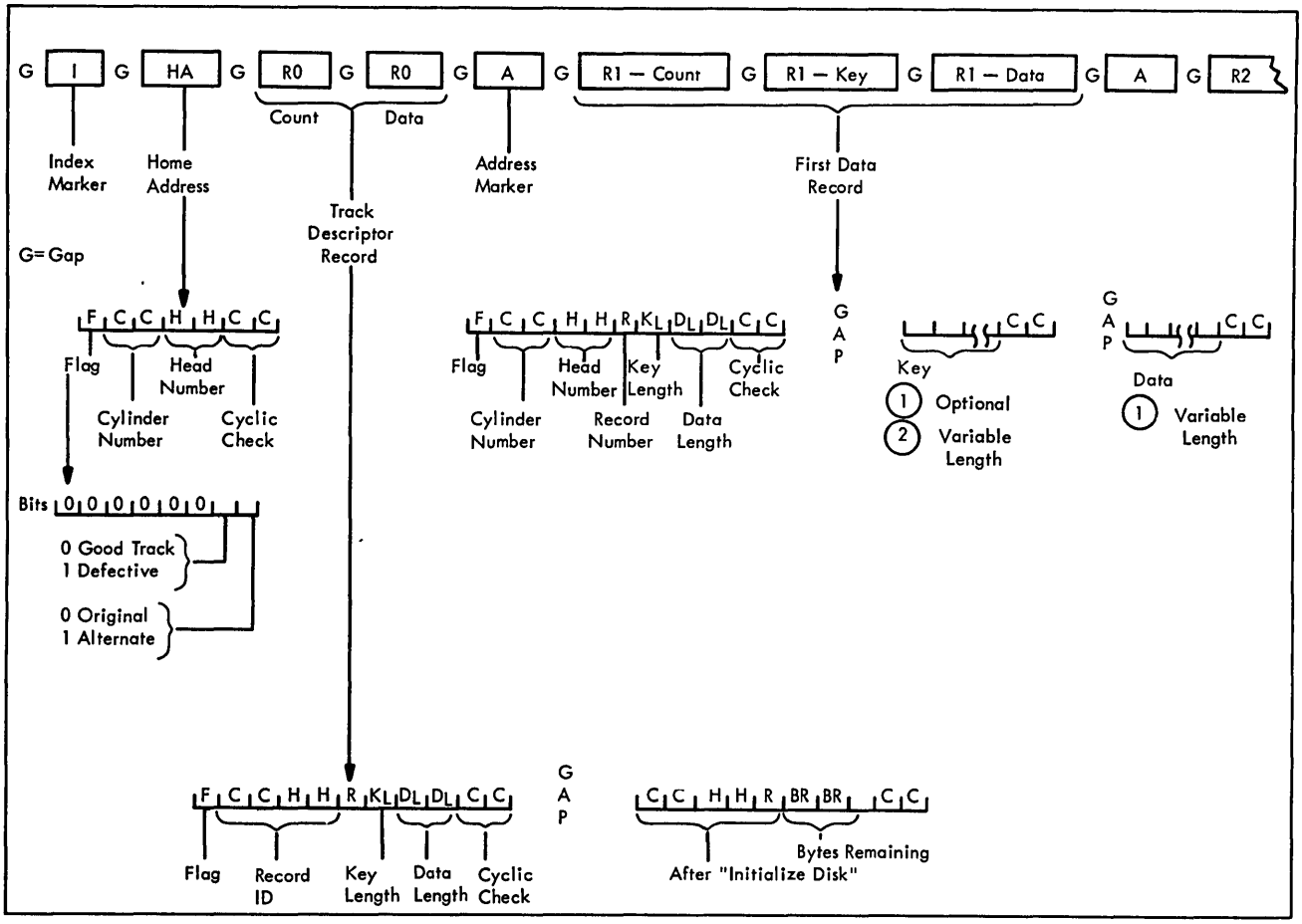


Figure 57. Track Format

COBOL library subroutines perform operations requiring extensive coding. For this reason it would be inefficient to place the coding in the object module each time it is needed. Most COBOL library subroutines are stored in the relocatable library. When required, they are combined at link-edit time with the object module produced by the compiler.

There are several major categories of COBOL library subroutines:

- Input/output verb routines
- Conversion routines
- Arithmetic verb routines
- Sort Feature interface routines
- Checkpoint (RERUN) routines
- Segmentation Feature routines
- Other verb routines

INPUT/OUTPUT SUBROUTINES

The input/output subroutines are used for the COBOL verbs DISPLAY (TRACE and EXHIBIT), ACCEPT, STOP (literal), READ, WRITE, and REWRITE, printer spacing, printer overflow, input/output errors, disk formatting and extent handling, and tape and sequential disk labels.

Printer Spacing

The ILBDSPA0 subroutine is used to control printer spacing when the WRITE statement with the BEFORE/AFTER ADVANCING or POSITIONING option is specified in the source program.

Tape and Sequential Disk Labels

The ILBDUSL0 and ILBDNSL0 subroutines are used when user or nonstandard labels, respectively, are to be processed (LABEL RECORDS ARE data-name).

CLOSE WITH LOCK Subroutine

The ILBDCLK0 subroutine is given control to issue an object-time message when an OPEN statement is used to open a file previously closed WITH LOCK.

WRITE Statement Subroutines

The ILBDVBL0 subroutine is used to write variable-length blocked records.

The ILBDDIO0 subroutine is used for writing files with direct organization (DTFDA).

The ILBDISM0 subroutine is used for writing files with indexed organization.

READ Statement Subroutines

The ILBDDSR0 subroutine is used to read sequentially the records of a directly organized file.

The ILBDDIO0 subroutine is used to read randomly the records of a directly organized file.

The ILBDISM0 subroutine is used to read an indexed file.

REWRITE Statement Subroutines

The ILBDDIO0 subroutine is used to update records on a directly organized file.

The ILBDISM0 subroutine is used to update an indexed file.

DISPLAY (EXHIBIT and TRACE) Subroutines

The ILBDDSP0 subroutine formats one or more operands into printed lines, performing conversions as needed.

The ILBDOSY0 and ILBDASY0 subroutines open SYSLST and/or SYSPCH and/or SYSIPT if there are DISPLAY or ACCEPT statements in a label declarative.

ACCEPT and STOP (literal) Statement Subroutines

The ILBDACP0 subroutine is used to handle ACCEPT statements for both SYSIPT and the console, as well as the STOP (literal) statement. The ILBDACP0 subroutine does not format or convert operands. For operands greater than 80 characters in length, any remainder in excess of the nearest multiple of 80 is ignored when accepting data from SYSIPT.

CLOSE Subroutine

The ILBDCRD0 subroutine is given control when a CLOSE UNIT statement is issued for a sequential input file with direct organization.

Multiple File Tape Subroutine

The ILBDMFT0 subroutine is given control when a reel contains more than one file and there are no standard labels.

Input/Output Error Subroutines

The ILBDSAE0 subroutine is used for processing input/output errors that occur on tape and sequential disk.

The ILBDDAE0 subroutine is used for processing input/output errors that occur on directly organized files.

The ILBDISE0 subroutine is called whenever an input/output error occurs during the processing an indexed file.

The ILBDABX0 subroutine is used to issue a STXIT macro instruction causing control to be passed to it if there is an error on a unit-record device.

Disk Extent Subroutines

The ILBDFMT0 subroutine writes record 0 (R0) on each track of each extent of a directly organized file opened as output, and writes an end-of-file (EOF) record as the last record in the file. This subroutine is called after the file has been opened.

The ILBDXTN0 subroutine stores for subsequent use the extent information for directly organized files.

Auxiliary Subroutines

Certain input/output subroutines use auxiliary subroutines as follows:

Auxiliary Routine	Used By
ILBDMOV0	ILBDSPA0, ILBDNSL0, ILBDVBL0
ILBDIDA0	ILBDFMT0, ILBDDSR0

CONVERSION SUBROUTINES

Eight numeric data formats are permitted in COBOL: five external (for input and output) and three internal (for internal processing).

The five external formats are:

- External or zoned decimal
- External floating-point
- Sterling display
- Numeric edited
- Sterling report

The three internal formats are:

- Internal or packed decimal
- Binary
- Internal floating-point

The conversions from internal decimal to external decimal, from external decimal to internal decimal, and from internal decimal to numeric edited are performed in-line. The other conversions are performed by the COBOL library subroutines shown in Table 26.

Table 26. Functions of COBOL Library Conversion Subroutines

Subroutine Name and Entry Points	Conversion	
	From	To
ILBDEFL2	External floating-point	Internal decimal
ILBDEFL1	External floating-point	Binary
ILBDEFL0	External floating-point	Internal floating-point
ILBDBID0[1]	Binary	Internal decimal
ILBDBID1[1]		
ILBDBID2[1]		
ILBDBIE0[1]	Binary	External decimal
ILBDBIE1[1]		
ILBDBIE2[1]		
ILBDBII0[2]	Binary	Internal floating-point
ILBDBII1[2]		
ILBDTEF0[2]	Binary	External floating-point
ILBDTEF1[2]		
ILBDTEF2	Internal decimal	External floating-point
IFBDTEF3	Internal floating-point	External floating-point
ILBDIDB0	Internal decimal	Binary
ILBDIDB1	External decimal	Binary
ILBDDCI1	Internal decimal	Internal floating-point
ILBDDCI0	External decimal	Internal floating-point
ILBDIFD0	Internal floating-point	Internal decimal
ILBDIFD1	Internal floating-point	External decimal
ILBDIFB1	Internal floating-point	Binary integer and a power of 10 exponent
ILBDIFB2[3]		
ILBDIFB0[3]	Internal floating-point	Binary
ILBDIDR0	Internal decimal	Sterling report
ILBDIDT0	Internal decimal	Sterling non-report
ILBDSTI0	Sterling non-report	Internal decimal

[1]The entry points used depend on whether the double-precision number is in registers 0 and 1, 2 and 3, or 4 and 5, respectively.
[2]The entry points are for single-precision binary and double-precision binary, respectively.
[3]This entry point is used for calls from other COBOL library subroutines.

ARITHMETIC VERB SUBROUTINES

Most arithmetic operations are performed in-line. However, involved calculations with very large numbers, such as decimal multiplication of two 30-digit numbers, are performed by COBOL library arithmetic subroutines. These subroutine names and their functions are shown in Table 27.

SORT FEATURE INTERFACE ROUTINE

Communication between the Sort/Merge program and the COBOL program is maintained by ILBDSRT0.

CHECKPOINT (RERUN) SUBROUTINE

The ILBDCKP0 subroutine issues the checkpoint macro instruction, which will write checkpoint records on a user-specified tape or disk checkpoint device. There are two calling sequences to this subroutine. The first, ILBDCKP1, is activated during initialization when the addresses of all files in the program are entered in a table. The second, ILBDCKP2, is required to take checkpoints during a sorting operation.

If RERUN is requested during a sorting operation, ILBDSRT0 must gather a list of physical IOCS files in use by the Sort program every time Sort exits at E11, E21, and E31. ILBDSRT0 then calls the checkpoint subroutine which will take a checkpoint of all active files.

SEGMENTATION FEATURE SUBROUTINE

The Segmentation Feature requires an object time subroutine, ILBDSEM0. The ILBDSEM0 subroutine performs the following functions when segments are needed:

1. Loads and initializes independent segments not in core.
2. Loads overlayable segments not in core.
3. Initializes independent segments if the segment is in core.
4. Branches to desired entry points if the segment is in the root segment.

OTHER VERB ROUTINES

There are also COBOL library subroutines for comparisons, the verbs MOVE and TRANSFORM, and other features of the COBOL language.

Compare Subroutines

The ILBDVC00 subroutine compares two operands, one or both of which is variable in length. Each may exceed 256 bytes.

The ILBDIVL0 subroutine is used in comparisons involving the figurative constant ALL 'literal', where literal is greater than one character.

Table 27. Functions of COBOL Library Arithmetic Subroutines

Subroutine Name	Function
ILBDXMU0	Internal decimal multiplication (30 digits * 30 digits = 60 digits)
ILBDXDI0	Internal decimal division (60 digits/30 digits = 30 digits)
ILBDXPR0	Decimal fixed-point exponentiation
ILBDFPW0	Floating-point exponentiation
ILBDGPW0[1]	Floating-point exponentiation

[1]The ILBDGPW0 entry point is used if the exponent has a PICTURE clause specifying an integer. The ILBDFPW0 entry point is used in all other cases.

MOVE Subroutines

The ILBDVMO0 subroutine is used when one or both operands is variable in length. Each may exceed 256 bytes. The subroutine has two entry points, depending on the type of MOVE: ILBDVMO0 (left-justified) and ILBDVMO1 (right-justified).

The ILBDMVL0 subroutine is used to move the figurative constant ALL 'literal', where literal is greater than one character.

The ILBDANE0 subroutine is used to perform a right-or left-justified alphanumeric edited move.

TRANSFORM Subroutine

The ILBDVTR0 subroutine transforms variable-length items.

Class Test Subroutine

The ILBDCLS0 subroutine is used to perform class tests for variable-length items and those fixed-length items longer than 256 bytes.

Note: The following tables are placed in the library for use by the in-line coding generated by the compiler and the subroutines called for by both the class test and TRANSFORM:

ILBDATB0 -- Alphabetic class test
ILBDETB0 -- External decimal class test
ILBDITB0 -- Internal decimal class test

SEARCH Subroutine

The ILBDSCH0 subroutine processes each search argument key according to type.

Main Program or Subprogram Subroutine

The ILBDMSN0 subroutine is a 1-byte switch tested in the code generated for EXIT PROGRAM, GOBACK, INIT1, and INIT2.

This appendix describes diagnostic messages generated by the compiler and by compiler-generated object code.

COMPILER DIAGNOSTIC MESSAGES

Using one of the messages as an example, COBOL compiler messages are in the following format:

```
105  ILA1002I-W  ***** SECTION HEADER
                 MISSING.  ASSUMED PRESENT.
```

The code 105 is the compiler-generated card number of the statement where the error has occurred. ILA identifies this as a Disk Operating System American National Standard COBOL compiler message; 1002 is the identifying number of the message. The symbol I indicates that this is a message to the programmer for his action. W is a level of severity in the error code with an explanation as follows:

- W Warning -- Indicates that an error was made in the source program. However, it is not serious enough to hinder the execution of the program.
- C Conditional -- Indicates that an error was made but the compiler usually makes a corrective assumption. The statement containing the error is retained. Execution can be attempted for the debugging value.
- E Error -- Indicates that a serious error was made. Usually the compiler makes no corrective assumption. The statement containing the error is dropped. Execution of the program should not be attempted.
- D Disaster -- Indicates that a serious error was made. Compilation is not completed. Results are unpredictable.

The message text usually describes the error and describes the action taken by the compiler as a result of the error. Most of the messages are self-explanatory, except in two situations:

1. When no compiler action is given. These messages are numbered in the 3000 series. They appear in combination with other messages that do have the compiler action described.
2. When messages describe errors that require an explanation too long to include in a message. These explanations appear in text under the messages.

Words in a message that must vary according to the program being compiled are denoted by five asterisks (*****) in the messages printed below. Three asterisks (***) appearing in messages on a listing indicate that the compiler has encountered unrecognizable information.

ILA0001I-D NO MORE TABLE SPACE AVAILABLE. COMPILATION ABANDONED.

Explanation: Because of the size or complexity of the source program, all of the space available for internal tables was exhausted.

User Response: Allocate more core storage for the compiler or make the program smaller or less complex.

ILA0002I-D BASIS LIBRARY NOT FOUND. COMPILATION ABANDONED.

Explanation: The source statement book specified in a BASIS card at the beginning of compilation was not found.

User Response: Correct the BASIS card or make the source code available in the library.

ILA0003I-D A TABLE HAS EXCEEDED MAXIMUM SIZE. COMPILATION ABANDONED.

Explanation: An attempt has been made to add information to a table that has been made static. The problem should never occur.

User Response: Refer the problem to a Customer Engineer.

ILA0004I- LINK OPTION RESET - D OR E LEVEL ERROR FOUND.

Explanation: The LINK option (set by a // OPTION LINK job control statement) was reset if it had been set previously. This prevents the execution of a partially compiled program or a program with serious errors

in it. If a // EXEC LNKEDT card is read later, the job control diagnostic - 1S13D STATEMENT OUT OF SEQUENCE - is logged. The operator usually cancels the job at this point.

ILA0005I-D LOGIC OR MACHINE ERROR IN TAMER. COMPILATION ABANDONED.

Explanation: A program logic error was detected in the FCOBOL table management routines.

User Response: The problem should be referred to a Customer Engineer.

Note: Messages numbered ILA0001I, ILA0003I, and ILA0005I may be printed at any time during compilation and may be followed by a dump. Message ILA0002I is printed at the beginning of compilation. Message ILA0004I follows the last message issued.

The following messages are grouped in the compiler output listing.

ILA1001I-C NUMERIC LITERAL NOT RECOGNIZED AS LEVEL NUMBER BECAUSE '*****' ILLEGAL AS USED. SKIPPING TO NEXT LEVEL, SECTION OR DIVISION.

ILA1002I-W ***** SECTION HEADER MISSING. ASSUMED PRESENT.

ILA1003I-W ***** PARAGRAPH NAME MISSING. ASSUMED PRESENT.

ILA1004I-E INVALID WORD *****. SKIPPING TO NEXT RECOGNIZABLE WORD.

ILA1005I-E INVALID ORDER IN ENVIRONMENT DIVISION. SKIPPING TO NEXT DIVISION.

ILA1006I-E DECLARATIVES SECTION WITHOUT USE SENTENCE. SECTION CAN NEVER BE EXECUTED.

ILA1007I-W ***** NOT PRECEDED BY A SPACE. ASSUMED SPACE.

ILA1008I-W RIGHT PAREN SHOULD NOT BE PRECEDED BY SPACE.

ILA1009I-E COPY MUST BE PRECEDED BY PROCEDURE-NAME. IGNORED.

ILA1010I-W LEFT PAREN SHOULD NOT BE FOLLOWED BY SPACE.

ILA1011I-C RECORDING MODE SPECIFICATION IS INVALID. ASSUMED VARIABLE.

ILA1012I-E FILE-NAME NOT UNIQUE. USING FIRST DEFINITION.

ILA1013I-E CHARACTER LENGTH IN SPECIAL-NAMES MUST BE ONE.

ILA1014I-W 'FILE' NOT PRESENT IN MULTIPLE FILE CLAUSE. ASSUMED PRESENT.

ILA1015I-E ***** INVALID AS EXTERNAL-NAME. IGNORED.

ILA1016I-E MORE THAN ONE ***** CLAUSE. SKIPPING TO NEXT CLAUSE.

ILA1017I-E ***** INVALID IN ***** CLAUSE. SKIPPING TO NEXT CLAUSE.

ILA1018I-E COPY CLAUSE INVALID IN A COPY LIBRARY. IGNORED.

ILA1019I-E NO LIBRARY NAME. COPY CLAUSE IGNORED.

ILA1020I-E ***** MUST BE PROCEDURE-NAME FOLLOWING DEBUG.*****.

ILA1021I-E ***** DOES NOT BELONG ON A DEBUG CARD. SKIPPING TO NEXT CARD.

ILA1022I-W PERIOD DOES NOT BELONG ON DEBUG CARD. DELETED.

ILA1023I-E INVALID FILE-NAME. USE IGNORED.

ILA1024I-E UNDEFINED FILE-NAME. USE IGNORED.

ILA1025I-C REDEFINES CLAUSE NOT FIRST CLAUSE FOLLOWING DATA-NAME. ASSUMED FIRST.

ILA1026I-W FOUND ***** EXPECTING ENVIRONMENT. ALL ENV. DIV. STATEMENTS IGNORED.

ILA1027I-E DUPLICATE FD. IGNORED.

ILA1028I-E ***** SENTENCE IMPROPERLY WRITTEN. SENTENCE IGNORED.

ILA1029I-E ***** IN ***** SENTENCE NOT DEFINED AS FILE-NAME. SENTENCE IGNORED.

ILA1030I-E ***** IN *****, SENTENCE IS INVALID. WORD IGNORED.

ILA1031I-C USE SENTENCE NOT PRECEDED BY SECTION-NAME. SECTION-NAME ASSUMED.

ILA1032I-E ***** INCORRECTLY USED IN USE SENTENCE. SENTENCE IGNORED.

ILA1033I-W ***** FILE-NAME ALREADY ASSIGNED THIS SAME CLAUSE OPTION. USING FIRST ONE.

ILA1034I-E ***** CLAUSE ILLEGAL IN ***** LEVEL. SKIPPING TO NEXT VALID CLAUSE.

ILA1035I-E INTEGER NOT PRESENT IN MULTIPLE FILE CLAUSE.

ILA1036I-C QUALIFIED NAME INVALID AFTER LEVEL NUMBER. USING LOWEST NAME.

ILA1037I-E ***** INVALID IN DATA DESCRIPTION. SKIPPING TO NEXT CLAUSE.

ILA1038I-E ***** INVALID AFTER LEVEL NUMBER. SKIPPING TO NEXT LEVEL.

ILA1039I-W DATA-NAME IN ***** CLAUSE NEED NOT BE QUALIFIED. USING LOWEST NAME.

ILA1040I-E IMPROPER LEVEL NUMBER FOR FILE SECTION.

ILA1041I-E ***** INVALID AS USED IN ***** SECTION. SKIPPING TO NEXT LEVEL, SECTION OR DIVISION.

ILA1042I-E ASSIGN CLAUSE MISSING IN SELECT. CONTINUING.

ILA1043I-W END OF SENTENCE SHOULD PRECEDE *****. ASSUMED PRESENT.

ILA1044I-E INVALID OR MISSING USING AND/OR GIVING CLAUSE IN SORT STATEMENT. PROGRAM CANNOT BE EXECUTED.

ILA1045I-E INVALID ORDER IN ***** SECTION.

ILA1046I-E MEMBER NOT FOUND IN LIBRARY. IGNORING COPY.

ILA1047I-E SYNTAX OF COMMENT IS INCORRECT. SKIPPING TO NEXT CLAUSE.

ILA1048I-E REEL (UNIT) NOT IN ASSIGN CLAUSE. ASSUMED PRESENT.

ILA1049I-E ***** FILE-NAME ALREADY ASSIGNED THIS MULTIPLE FILE CLAUSE OPTION. USING FIRST ONE.

ILA1050I-E ***** FILE ALREADY ASSIGNED THIS APPLY OPTION. FILE-NAME IGNORED.

ILA1051I-E NO DATA-NAME IN USE SENTENCE. SENTENCE IGNORED.

ILA1052I-E ***** ILLEGALLY USED IN USE SENTENCE. END SENTENCE, RESCANNING AT NEXT RECOGNIZABLE WORD.

ILA1053I-E ***** CLAUSE INVALID. CLAUSE IGNORED.

ILA1054I-E OPERAND FOR INITIATE NOT FOUND OR ILLEGAL. OPERAND DROPPED.

ILA1055I-E VALID FILE-NAME NOT PRESENT. DESCRIPTION IGNORED.

ILA1056I-E FILE-NAME NOT DEFINED IN A SELECT. DESCRIPTION IGNORED.

ILA1057I-E FIRST WORD IN REPORT SECTION NOT RD. IGNORED.

ILA1058I-E NO REPORTS CLAUSE IN FILE SECTION. REPORT SECTION IGNORED.

ILA1059I-E NO REPORT CLAUSE FOR RD. RD IGNORED.

ILA1060I-E INVALID WORD IN REPORT WRITER STATEMENT. IGNORED.

ILA1061I-E DUPLICATE CLAUSE. DROPPED.

ILA1062I-E ***** IN COPY REPLACING STATEMENT INVALID AS BCD NAME.

ILA1063I-E DUPLICATE ENTRY IN PAGE CLAUSE. DROPPED.

ILA1064I-E NO TYPE CLAUSE SPECIFIED. SKIPPING TO NEXT 01.

ILA1065I-E INTEGER MISSING IN PAGE CLAUSE. ENTRY IGNORED.

ILA1066I-E INVALID WORD IN PAGE CLAUSE. SKIPPING TO NEXT RECOGNIZABLE WORD.

ILA1067I-E INVALID HEADER. SKIPPING TO NEXT RECOGNIZABLE WORD.

ILA1068I-E OPERAND FOR GENERATE NOT FOUND. CLAUSE DROPPED.

ILA1069-E INVALID TYPE CLAUSE. SKIPPING TO NEXT 01.

ILA1070I-C FLT-PT LIT MANTISSA EXCEEDS 16 DIGITS. TRUNCATED TO 16.

ILA1071I-C FLT-PT LIT EXPONENT EXCEEDS 2 DIGITS. TRUNCATED TO 2. RESCANNING.

ILA1072I-C FLT-PT LIT EXPONENT FOLLOWED BY NON-BLANK. RESCANNING AT NON-BLANK.

ILA1073I-C FLT-PT LIT E FOLLOWED BY INVALID CHARACTER. RESCANNING AT E.

ILA1074I-C FLT-PT LIT SIGN FOLLOWED BY INVALID CHARACTER. RESCANNING AT E.

ILA1075I-C FLT-PT LIT EXCEEDS LIMIT. ASSUME MAX OR MIN PER SIGN OF EXPONENT.

ILA1076I-C ALPHANUMERIC LIT EXCEEDS 120 CHARACTERS. TRUNCATED TO 120.

ILA1077I-C ALPHANUMERIC LIT CONTINUES IN A-MARGIN. ASSUME B-MARGIN.

ILA1078I-W ALPHANUMERIC LIT CONTINUED WITHOUT HYPHEN OR QUOTE. ASSUMED.

ILA1079I-W ALPHANUMERIC LIT HAS ZERO LENGTH. ASSUME ONE SPACE.

ILA1080I-W PERIOD PRECEDED BY SPACE. ASSUME END OF SENTENCE.

ILA1081I-W PERIOD NOT FOLLOWED BY SPACE. ASSUME END OF SENTENCE.

ILA1082I-C NUMERIC LIT EXCEEDS 18 DIGITS. TRUNCATED TO 18.

ILA1083I-C ILLEGAL CHARACTER. SCAN RESUMED AT NEXT VALID CHARACTER.

ILA1084I-W COMMA SHOULD NOT BE PRECEDED BY SPACE.

ILA1085I-C WORD OR PICTURE EXCEEDS 30 CHARACTERS. TRUNCATED TO 30 CHARACTERS.

ILA1086I-W ***** SHOULD BEGIN IN A-MARGIN.

ILA1087I-W '*****' SHOULD NOT BEGIN IN A-MARGIN.

ILA1088I-E MISSING FIRST INSERT OR DELETE CARD. PASS CARDS UNTIL FOUND. *****.

ILA1089I-E INSERT OR DELETE NUMBER OUT OF SEQUENCE. SKIPPING TO NEXT INSERT OR DELETE NUMBER. *****.

ILA1090I-E DELETE THRU NUMBER OUT OF SEQUENCE. PASS CARDS UNTIL NEXT INSERT OR DELETE. *****.

ILA1091I-C ***** IN A-MARGIN NOT VALID AS PROC-NM. ASSUME B-MARGIN.

ILA1092I-E DECLARATIVES DO NOT FOLLOW PROCEDURE DIVISION. IGNORED.

ILA1093I-E NO DECLARATIVES SECTION. END DECLARATIVES IGNORED.

ILA1094I-E INTEGER IN NEXT GROUP CLAUSE DOES NOT CONFORM TO PAGE CLAUSE SPECIFICATIONS. CONTINUING.

ILA1095I-W WORD 'SECTION' OR 'DIVISION' MISSING. ASSUMED PRESENT.

ILA1096I-E DATANAME IN UPON CLAUSE NOT SPECIFIED AS A DATANAME FOR A TYPE DETAIL REPORT GROUP IN THIS REPORT. UPON OPTION IGNORED.

ILA1097I-E PROGRAM-ID MISSING OR MISPLACED. IF PROGRAM-ID DOES NOT IMMEDIATELY FOLLOW IDENTIFICATION DIVISION, IT WILL BE IGNORED.

ILA1098I-C ALPHA LITERAL NOT CONTINUED WITH HYPHEN AND QUOTE. END LITERAL ON LAST CARD.

ILA1099I-E ***** IS INVALID AS USED.

ILA1100I-W ***** SEQUENCE ERRORS IN SOURCE PROGRAM.

ILA1101I-E NEXT PAGE NOT IN FIRST LINE CLAUSE. IGNORED.

ILA1102I-W INCOMPLETE ELEMENTARY ITEM. ASSUME VALUE SPACES.

ILA1103I-E GROUP TYPE ALLOWED ONCE FOR RD. IGNORED.

ILA1104I-E CONTROL NAME NOT SPECIFIED IN RD. SKIPPING TO NEXT 01.

ILA1105I-W ELEMENTARY ITEM EXPECTED. ASSUMED.

ILA1106I-E OPERAND FOR TERMINATE NOT FOUND OR ILLEGAL. OPERAND DROPPED.

ILA1107I-C 'NEXT GROUP' CLAUSE IS ILLEGAL FOR THIS REPORT GROUP. IGNORED.

ILA1108I-E ***** IS NOT A POSITIVE INTEGRAL NUMBER. ASSUMED ONE.

ILA1109I-E DUPLICATE USE OF CONTROL NAME. SKIPPING TO NEXT 01.

ILA1110I-W INVALID USE OF SUM CLAUSE. CLAUSE IGNORED.

ILA1111I-W ELEMENTARY LEVEL WITHOUT COLUMN OR SUM CLAUSE.

ILA1112I-E '*****' ALREADY SPECIFIED IN TWO FILE DESCRIPTION ENTRIES. IGNORED.

ILA1113I-E EXPECTING 6-DIGIT SEQUENCE NUMBER. SKIPPING TO NEXT INSERT OR DELETE NUMBER. *****.

ILA1114I-C EXTRANEOUS COMMA OR HYPHEN ON DELETE CARD. IGNORED.

ILA1115I-E NO BLANK, COMMA, OR HYPHEN FOLLOWING SEQUENCE NUMBER. ASSUME BLANK. *****.

ILA1116I-E EXPECTING 6-DIGIT SEQUENCE NUMBER AFTER HYPHEN. IGNORING DELETE FROM THRU NUMBER. *****.

ILA1117I-E DELETE NUMBER GREATER THAN LAST SEQUENCE NUMBER. STOP INSERT AND DELETE. *****.

ILA1118I-E INSERT NUMBER GREATER THAN LAST SEQUENCE NUMBER. STOP INSERT AND DELETE. *****.

ILA1119I-E INTEGER IN 'LINE' CLAUSE DOES NOT CONFORM TO PAGE CLAUSE SPECIFICATIONS. CONTINUING.

ILA1120I-W COMMA NOT FOLLOWED BY SPACE. ASSUMED.

ILA1121I-W PERIOD OR COMMA INVALID AS USED IN PICTURE CLAUSE.

ILA1122I-E EXTERNAL-NAME IN RERUN CLAUSE MUST NOT BE THE SAME AS SYSTEM-NAME USED IN ASSIGN CLAUSE. SENTENCE IGNORED.

ILA1123I-E NUMBER IS ZERO OR NEGATIVE. SENTENCE IGNORED.

ILA1124I-E NUMBER TOO LARGE FOR RERUN. SENTENCE IGNORED.

ILA1125I-C ***** FILE-NAME USED IN PREVIOUS RERUN. USING FIRST ONE.

ILA1126I-E ***** FILE-NAME SPECIFIED IN BOTH RERUN AND USING OR GIVING OPTION. RERUN IGNORED.

ILA1127I-C ***** INVALID IN ***** SENTENCE. REST OF SENTENCE IGNORED.

ILA1129I-C ID DIV. HEADER MISSING OR MISPLACED. ASSUMED PRESENT.

ILA1130I-E ***** DIV. HEADER MISSING. WORDS IN ***** STATEMENTS ARE INVALID.

ILA1131I-W INVALID PRIORITY NUMBER. ZERO ASSUMED.

ILA1132I-E INVALID SYSTEM-NAME. SKIPPING TO NEXT CLAUSE.

ILA1133I-W MORE THAN 1 USE ON STANDARD ERROR SPECIFIED FOR SAME FILE ON OPEN OPTION. USE IGNORED.

ILA1134I-E USE SPECIFIED FOR FILE WITH LABEL RECORDS OMITTED OR STANDARD. SENTENCE IGNORED.

ILA1135I-W INTEGER-1 OUTSIDE OF ALLOWABLE LIMITS. 1 ASSUMED.

ILA1136I-E DATANAME ALREADY SPECIFIED FOR A TYPE DETAIL REPORT GROUP. SKIPPING TO NEXT 01, RD, OR SECTION.

ILA1137I-W MINIMUM NUMBER OF OCCURRENCES IN OCCURS CLAUSE NOT LESS THAN MAXIMUM NUMBER. CONTINUING.

ILA1141I-C FILE ORGANIZATION FIELD INVALID IN SYSTEM-NAME. SEQUENTIAL ASSUMED.

ILA1142I-E USE FOR STANDARD ERROR OR LABEL PROCESSING SPECIFIED FOR FILE AND OPEN OPTION. USE FOR OPEN OPTION IGNORED.

ILA1143I-E USE STATEMENTS IMPLY STANDARD AND NON-STANDARD LABELS. USE IGNORED.

ILA1144I-W WRITE AFTER POSITIONING AND WRITE BEFORE ADVANCING. ILLEGALLY USED FOR 1 FILE.

ILA1145I-E ***** DUPLICATELY DEFINED IN SPECIAL NAMES PARAGRAPH. SENTENCE IGNORED.

ILA1147I-E SD FILE ILLEGALLY SPECIFIED IN SAME AREA CLAUSE. CLAUSE FOR SD IGNORED.

ILA1148I-C INVALID SEGMENT LIMIT. FIFTY ASSUMED.

ILA1149I-E FILES IN SAME AREA CLAUSE DO NOT ALL APPEAR IN THE SAME SORT/RECORD AREA CLAUSE. '*****' NOT GIVEN SAME AREA NUMBER.

ILA1151I-E ILLEGAL CHARACTER USE IN CURRENCY SIGN CLAUSE. CLAUSE IGNORED.

IAL1152I-E ON AND/OR OFF STATUS MUST BE SPECIFIED ON UPSI CLAUSE. SPECIAL NAME IGNORED.

ILA1154I-E 2 DIFFERENT LABEL PROCEDURES FOR EOF AND EOV WITH 'BEFORE'

OPTION. BOTH LABEL PROCEDURES IGNORED.

ILA1155I-E DEVICE CLASS INVALID IN SYSTEM-NAME. SKIPPING TO NEXT FIELD.

ILA1156I-C DEVICE NUMBER INVALID IN SYSTEM-NAME. '*****' ASSUMED.

ILA1158I-E '*****'IN ENTRY STATEMENT IS SAME AS PROGRAM-ID. '*****' IGNORED FOR ENTRY VERB.

ILA1156I-W PAGE LIMIT INTEGER-1 NOT SPECIFIED. ASSUME HIGH-VALUE.

ILA1160I-E CONTINUATION OF WORD FOUND IN A-MARGIN. IGNORED.

ILA1161I-W RESERVED WORD MISSING. ASSUMED PRESENT.

ILA1162I-E INTEGER IN LINE CLAUSE IS LESS THAN PREVIOUS VALUE. IGNORED.

ILA1163I-E ABSOLUTE LINE NUMBER IS PRECEDED BY A RELATIVE LINE NUMBER. IGNORED.

ILA1164I-E NO PAGE CLAUSE SPECIFIED. ALL LINE CLAUSES MUST BE 'LINE PLUS INTEGER'. IGNORED.

ILA1165I-E 'HEADING' EQUALS 'FIRST DETAIL' IN PAGE CLAUSE. PAGE HEADING IS ILLEGAL. CONTINUING.

ILA1166I-E 'FOOTING' EQUALS 'PAGE LIMIT' IN PAGE CLAUSE. PAGE FOOTING IS ILLEGAL. CONTINUING.

ILA1167I-W 'LINE NEXT PAGE' CLAUSE IS ILLEGAL FOR THIS REPORT GROUP. IGNORED.

ILA1168I-W DUPLICATE REPORT NAME. SKIPPING TO NEW RD.

ILA1169I-E AN OPERAND IN THIS SUM CLAUSE DOES NOT APPEAR AS A SOURCE ITEM IN DETAIL *****. OPERAND IGNORED.

ILA1170I-E DETAIL REPORT GROUP SPECIFIED WITH NO DATANAME. CONTINUING.

ILA1171I-E INTEGERS IN PAGE CLAUSE ARE NOT IN ASCENDING ORDER. CONTINUING.

ILA1172I-E WORD INVALID AS REPORT NAME. RD IGNORED.

ILA1173I-E GROUP INDICATE IS ILLEGAL FOR THIS REPORT GROUP. IGNORED.

ILA1174I-E NO LINE CLAUSE SPECIFIED IN PRECEDING REPORT GROUP. NO OUTPUT GENERATED.

ILA1175I-E DATANAME FOR THIS REPORT GROUP IS NOT UNIQUE. SKIPPING TO NEW 01, RD, SECTION.

ILA1176I-E SYS NUMBER NOT EQUAL TO 001 FOR SORT FILE. ASSUMED PRESENT.

ILA1178I-E RESET CLAUSE SPECIFIED, AND IS EITHER ILLEGAL FOR THIS REPORT GROUP, OR ELEMENTARY ITEM DOES NOT CONTAIN A SUM CLAUSE. CLAUSE IGNORED.

ILA1179I-E COLUMN NUMBER ILLEGAL. ASSUME COLUMN 1.

ILA2001I-C BLOCK SIZE SMALLER THAN RECORD SIZE. BLOCK CONTAINS IGNORED.

ILA2002I-W ORGANIZATION INCORRECT. USING STANDARD SEQUENTIAL.

ILA2003I-W RANDOM ACCESS ILLEGAL FOR THIS FILE. USING SEQUENTIAL.

ILA2004I-E RECORDING MODE ILLEGAL FOR ACCESS METHOD. RECORDING MODE IGNORED.

ILA2005I-W A CARD-FILE OPENED INPUT MUST HAVE FIXED RECORD FORMAT. FIXED ASSUMED.

ILA2006I-C SPANNED RECORDS INVALID FOR THIS DEVICE. USING VARIABLE.

ILA2007I-C RECORD CONTAINS CLAUSE CONFLICTS WITH RECORD DESCRIPTION. CLAUSE IGNORED.

ILA2008I-C APPLY MASTER/CYLINDER INDEX VALID ONLY FOR INDEXED FILES. CLAUSE IGNORED.

ILA2009I-C SYNCHRONIZED ITEM NOT ON PROPER BOUNDARY. NO ALIGNMENT PERFORMED BECAUSE STARTING ADDRESS OF THE REDEFINES ITEM WOULD HAVE TO BE CHANGED.

ILA2010I-E OBJECT OF REDEFINES CLAUSE IS OCCURS DEPENDING ON SUBJECT. REDEFINES CLAUSE IGNORED.

ILA2011I-E AN INDEX DATA ITEM MAY NOT BE A CONDITIONAL VARIABLE. 88(S) DISCARDED.

ILA2012I-E INDEX NAMES AND/OR KEYS IGNORED FOR TABLE WITH ILLEGAL SUBJECT.

ILA2013I-C BLOCK CONTAINS CLAUSE IMPROPERLY WRITTEN. CLAUSE IGNORED.

ILA2014I-C BLOCK CONTAINS CHARACTERS MUST BE USED FOR SPANNED RECORDS. USING VARIABLE.

ILA2015I-W CONFLICTING SPECIFICATIONS FOR RECORD FORMAT. ***** ASSUMED.

ILA2016I-E DATA-RECORD SIZE IS VARIABLE. 'RECORDING MODE F' IGNORED.

ILA2017I-E IF THE SUBJECT OF AN INDEXED BY CLAUSE IS AN ELEMENTARY ITEM ONLY THAT ITEM MAY BE SPECIFIED IN THE KEY CLAUSE. REST OF KEYS DISCARDED.

ILA2018I-E OBJECT OF RENAMES CLAUSE WAS NOT FOUND OR NON-UNIQUE IN LOGICAL RECORD.

ILA2019I-C BLOCK CONTAINS CLAUSE INVALID WHEN RECORD FORMAT IS UNDEF. CLAUSE IGNORED.

ILA2020I-C TRACK-AREA CLAUSE ILLEGAL FOR THIS ACCESS METHOD. CLAUSE IGNORED.

ILA2021I-C PICTURE DUPLICATION FACTOR TRUNCATED TO 5 SIGNIFICANT DIGITS.

ILA2022I-E THE OBJECT OF THE RENAMES OR RENAMES THRU CLAUSE CANNOT BE AN 01, 66, 77, OR 88. STATEMENT DISCARDED.

ILA2023I-E ***** KEY MISSING. FILE IGNORED.

ILA2024I-E ***** KEY IS ILLEGAL FOR THIS ACCESS METHOD. CLAUSE IGNORED.

ILA2027I-C APPLY CORE INDEX ILLEGAL FOR THIS ACCESS METHOD. CLAUSE IGNORED.

ILA2028I-W RECORD CONTAINS CLAUSE IMPROPERLY WRITTEN. CLAUSE IGNORED.

ILA2029I-C FIRST NON 77, 88 ITEM IN SECTION IS NOT AN 01. THIS ITEM WAS CHANGED TO 01.

ILA2030I-C 77 ITEM PRECEDED BY AN 01-49 ITEM OR 77 IN FILE SECTION. 77 CHANGED TO 01.

ILA2031I-C 88 ITEM MUST MUST BE PRECEDED BY 01-49 OR 77 ITEM. 88 CHANGED TO 01.

ILA2032I-E 88 ITEM CONTAINED A CLAUSE OTHER THAN VALUE CLAUSE. CLAUSE DELETED.

ILA2033I-C ITEM'S USAGE INCOMPATIBLE WITH USAGE OF GROUP IT BELONGS TO. USAGE CHANGED TO GROUP'S USAGE.

ILA2034I-E GROUP ITEM HAS PICTURE CLAUSE. CLAUSE DELETED.

ILA2035I-E GROUP ITEM HAS BLANK WHEN ZERO CLAUSE. CLAUSE DELETED.

ILA2036I-E GROUP ITEM HAS JUSTIFIED CLAUSE. CLAUSE DELETED.

ILA2037I-E BLANK WHEN ZERO CLAUSE USED INCORRECTLY. CLAUSE IGNORED.

ILA2038I-E ACTUAL KEY MUST BE GREATER THAN 4 AND LESS THAN 259 BYTES IN LENGTH. USING 5.

ILA2039I-C PICTURE CONFIGURATION ILLEGAL. PICTURE CHANGED TO 9 UNLESS USAGE IS 'DISPLAY-ST', THEN L(6)BDZ9BDZ9.

ILA2040I-E JUSTIFIED CLAUSE SPEC'D FOR NON-ALPHABETIC OR NON-ALPHANUMERIC ITEM. CLAUSE DELETED.

ILA2041I-E CONDITION NAME UNDER GROUP HAS VALUE CLAUSE THAT IS NUMERIC. 88 DISCARDED.

ILA2042I-E THIS ITEM CAUSES OVER 3 LEVELS OF SUBSCRIPTING. OCCURS CLAUSE DROPPED FOR THIS ITEM.

ILA2043I-E 01 OR 77 LEVEL HAS AN OCCURS CLAUSE. CLAUSE DELETED.

ILA2044I-E DUPLICATE SD. IGNORED.

ILA2045I-E REPORT CONTROL NAME UNDEFINED.

ILA2046I-E REPORT CONTROL NAME NOT FIXED LENGTH.

ILA2047I-E MORE THAN 12 INDEX NAMES SPECIFIED FOR TABLE. FIRST 12 ACCEPTED.

ILA2049I-C NO VALID OPEN FOR FILE. FILE IGNORED.

ILA2050I-C BLOCK SIZE TOO LARGE. USING MAXIMUM FOR DEVICE. RECORD TRUNCATED.

ILA2051I-C APPLY EXTENDED SEARCH VALID ONLY FOR DIRECT FILES. CLAUSE IGNORED.

ILA2052I-E MORE THAN 12 KEYS SPECIFIED FOR TABLE. FIRST 12 ACCEPTED.

ILA2055I-C STERLING NON-REPORT PICTURE - SIGN IN POUND FIELD MUST BE ON

HI OR LO ORDER DIGIT. PICTURE REPLACED BY 9D8D7.

ILA2056I-C STERLING NON-REPORT PICTURE - 9 IN ILLEGAL POSTION. PICTURE REPLACED BY 9D8D7.

ILA2057I-C STERLING NON-REPORT PICTURE - SIGN IN SHILLING FIELD ILLEGAL. PICTURE REPLACED BY 9D8D7.

ILA2058I-C STERLING NON-REPORT PICTURE - 8 IN ILLEGAL POSITION. PICTURE REPLACED BY 9D8D7.

ILA2059I-C STERLING NON-REPORT PICTURE - SIGN IN PENCE FIELD ILLEGAL. PICTURE REPLACED BY 9D8D7.

ILA2060I-C STERLING NON-REPORT PICTURE - 6 OR 7 IN ILLEGAL POSITION. PICTURE REPLACED BY 9D8D7.

ILA2061I-C STERLING NON-REPORT PICTURE. USAGE NOT DISPLAY-ST. PICTURE REPLACED BY 9(1).

ILA2062I-C STERLING NON-REPORT PICTURE - V IN ILLEGAL POSITION. PICTURE REPLACED BY 9D8D7.

ILA2063I-C STERLING NON-REPORT PICTURE - S IN ILLEGAL POSITION. PICTURE REPLACED BY 9D8D7.

ILA2064I-C STERLING NON-REPORT PICTURE - DIGIT LENGTH GT 2. PICTURE REPLACED BY 9D8D7.

ILA2065I-C STERLING NON-REPORT PICTURE - SHILLING FIELD GT 18. PICTURE REPLACED BY 9D8D7.

ILA2066I-C STERLING NON-REPORT PICTURE - PENCE FIELD GT 2. PICTURE REPLACED BY 9D8D7.

ILA2067I-C STERLING NON-REPORT PICTURE - NO POUND SEPARATOR. PICTURE REPLACED BY 9D8D7.

ILA2068I-C ONLY THE RENAMES CLAUSE MAY BE SPECIFIED FOR A LEVEL 66 ENTRY. CLAUSE IGNORED.

ILA2069I-C NUMERIC PICTURE - SIGN IN ILLEGAL POSITION. PICTURE REPLACED BY 9(1).

ILA2070I-C NUMERIC PICTURE - P IN ILLEGAL POSITION. PICTURE REPLACED BY 9(1).

ILA2071I-C NUMERIC PICTURE - V IN ILLEGAL POSITION. PICTURE REPLACED BY 9(1).

ILA2072I-C NUMERIC PICTURE - NO 9 IN PICTURE. PICTURE REPLACED BY 9(1).

ILA2073I-C NUMERIC PICTURE - P ENCLOSED BY 9'S. PICTURE REPLACED BY 9(1).

ILA2074I-E COMPILER ERROR - MINOR CODE FOR RENAMES ENTRY IS ILLEGAL.

ILA2075I-C NUMERIC PICTURE - DIGIT LENGTH GT 18. PICTURE REPLACED BY 9(1).

ILA2076I-C NUMERIC PICTURE - DIGIT LENGTH + SIGN SCALE GT 18. PICTURE REPLACED BY 9(1).

ILA2077I-C EXTERNAL FLOATING-POINT PICTURE - USAGE NOT DISPLAY. PICTURE CHANGED TO 9.

ILA2078I-W EXTERNAL FLOATING-POINT PICTURE - MORE THAN 1 SIGN. CHANGED TO 1.

ILA2079I-C EXTERNAL FLOATING-POINT PICTURE - SIGN IN ILLEGAL POSITION. PICTURE CHANGED TO +9.E+99.

ILA2080I-C EXTERNAL FLOATING-POINT PICTURE - SIGN MISSING. ASSUME MINUS SIGN.

ILA2081I-C EXTERNAL FLOATING-POINT PICTURE - REQUIRED CHARACTER BEFORE EXPONENT MISSING. PICTURE CHANGED TO +9.E+99.

ILA2082I-W EXTERNAL FLOATING-POINT PICTURE - NO DECIMAL POINT IN MANTISSA. ASSUME IMPLIED V.

ILA2083I-C EXTERNAL FLOATING-POINT PICTURE - MANTISSA LENGTH GT 16. PICTURE CHANGED TO +9.E+99.

ILA2084I-C EXTERNAL FLOATING-POINT PICTURE - TOTAL LENGTH GT 21. PICTURE CHANGED TO +9.E+99.

ILA2085I-C EXTERNAL FLOATING-POINT PICTURE - EXPONENT LENGTH NOT 2 DIGITS. ASSUME 2 DIGITS.

ILA2086I-C NUMERIC EDITED PICTURE - TWO FIXED DOLLAR SIGNS, +, - OR FIXED AND FLOATING DOLLAR SIGN. PICTURE REPLACED BY 9(1).

ILA2089I-C NUMERIC EDITED PICTURE - 9, Z OR * PRECEDES FLOATING STRING. PICTURE REPLACED BY 9(").

ILA2090I-C NUMERIC EDITED PICTURE - P IN ILLEGAL POSITION. PICTURE REPLACED BY 9(1).

ILA2091I-C NUMERIC EDITED PICTURE - TWO DIFFERENT FLOATING STRING CHARACTERS. PICTURE REPLACED BY 9(1).

ILA2092I-C NUMERIC EDITED PICTURE - Z AND * IN PICTURE. PICTURE REPLACED BY 9(1).

ILA2093I-C NUMERIC EDITED PICTURE - 9 PRECEDES * OR Z. PICTURE REPLACED BY 9(1).

ILA2094I-C NUMERIC EDITED PICTURE - FLOATING STRING PRECEDES * OR Z. PICTURE REPLACED BY 9(1).

ILA2096I-C DECIMAL POINT MAY ONLY APPEAR ONCE IN A PICTURE CHARACTER STRING. PICTURE REPLACED BY 9(1).

ILA2097I-C NUMERIC EDITED PICTURE - DECIMAL POINT OR V CONTRADICTORY TO P. PICTURE REPLACED BY 9(1).

ILA2098I-C INDEXED BY AND/OR KEY CLAUSE IS ILLEGAL FOR ITEM SUBORDINATE TO GROUP THAT HAS OCCURS BUT NO INDEXED BY CLAUSE. CLAUSE IGNORED.

ILA2099I-C NUMERIC EDITED PICTURE - CR OR DB AND SIGN BOTH USED. PICTURE REPLACED BY 9(1).

ILA2100I-C NUMERIC EDITED PICTURE - CR OR DB NOT LAST TWO CHARACTERS IN PICTURE. PICTURE REPLACED BY 9(1).

ILA2101I-C NUMERIC EDITED PICTURE - SIGN IS NOT FIRST OR LAST CHARACTER IN PICTURE. PICTURE REPLACED BY 9(1).

ILA2102I-C NUMERIC EDITED PICTURE - NUMERIC CHARACTERS AFTER DECIMAL POINT ARE NOT THE SAME. PICTURE REPLACED BY 9(1).

ILA2103I-C NUMERIC EDITED PICTURE - TOTAL LENGTH GT 127. PICTURE REPLACED BY 9(1).

ILA2104I-C NUMERIC EDITED PICTURE - NUMERIC LENGTH GT 18. PICTURE REPLACED BY 9(1).

ILA2105I-E ONLY ONE KEY MAY BE SPECIFIED IF THE SUBJECT OF TABLE IS A KEY. REST OF KEYS DISCARDED.

ILA2106I-E THE RENAMES CLAUSE MUST BE THE LAST ENTRY IN A LOGICAL RECORD. SKIPPING TO NEXT LEVEL, SECTION, OR DIVISION.

ILA2107I-W NUMERIC EDITED PICTURE - USAGE NOT DISPLAY. PICTURE CHANGED TO 9.

ILA2108I-E KEYS IGNORED FOR ITEM WITH NO INDEXED BY CLAUSE.

ILA2110I-C APPLY WRITE-ONLY VALID ONLY FOR VARIABLE BLOCKED RECORDS. CLAUSE IGNORED.

ILA2113I-W ITEM WITH USAGE OF COMPUTATIONAL-1 OR COMPUTATIONAL-2 HAS PICTURE CLAUSE. CLAUSE IGNORED.

ILA2114I-E ONLY THE SYNCHRONIZED CLAUSE IS ALLOWED FOR A USAGE IS INDEX ITEM. CLAUSE IGNORED.

ILA2115I-E LENGTH OF VARIABLE GROUP GT 32K. ACCEPTED AS WRITTEN.

ILA2116I-E FIXED LENGTH GROUP ITEM IN WORKING-STORAGE OR LINKAGE SECTION IS GT 131K. ACCEPTED AS WRITTEN.

ILA2117I-E INVALID REPORT CHARACTER. PICTURE CHANGED TO 9.

ILA2118I-C LENGTH OF REDEFINES SUBJECT GREATER THAN LENGTH OF REDEFINES OBJECT. SUBJECT LENGTH USED.

ILA2119I-E VALUE CLAUSE SPECIFIED FOR AN ITEM IN A REDEFINES GROUP. CLAUSE IGNORED.

ILA2120I-E OBJECT OF REDEFINES CLAUSE UNDEFINED OR ILLEGAL. CLAUSE IGNORED.

ILA2121I-W SUBJECT OF REDEFINES IS VARIABLE LENGTH.

ILA2122I-E REDEFINES SUBJECT LEVEL NUMBER NOT EQUAL TO REDEFINES OBJECT LEVEL NUMBER OR OBJECT NOT IMMEDIATELY PRECEDING SUBJECT. CLAUSE IGNORED.

ILA2123I-W OBJECT OF REDEFINES IS SUBSCRIPTED.

ILA2124I-C OBJECT OF REDEFINES IS VARIABLE LENGTH GROUP ITEM. REDEFINES CLAUSE IGNORED.

ILA2125I-W VALUE CLAUSE TREATED AS COMMENTS FOR ITEMS IN FILE AND LINKAGE SECTIONS.

ILA2126I-C VALUE CLAUSE LITERAL TOO LONG. TRUNCATED TO PICTURE SIZE.

ILA2127I-C NUMERIC VALUE CLAUSE SPECIFIED FOR GROUP ITEM. CLAUSE IGNORED.

ILA2128I-C VALUE CLAUSE LITERAL DOES NOT CONFORM TO PICTURE. CHANGED TO BLANKS.

ILA2129I-C VALUE CLAUSE LITERAL DOES NOT CONFORM TO PICTURE. CHANGED TO ZERO.

ILA2130I-E ITEM CANNOT HAVE VALUE CLAUSE. CLAUSE IGNORED.

ILA2132I-E RECORD KEY LENGTH GREATER THAN 255 BYTES. ACCEPTED AS WRITTEN.

ILA2133I-W LABEL RECORDS CLAUSE INVALID OR MISSING. ***** ASSUMED.

ILA2134I-C VALUE FOR SCALING CHARACTER SHOULD BE ZERO. CHANGED TO ZERO.

ILA2135I-C RECORDS IN ISAM FILE CANNOT BE VARIABLE LENGTH. ASSUMED FIXED AT MAXIMUM SIZE.

ILA2136I-E NOMINAL KEY LENGTH FOR INDEXED FILE GREATER THAN 255 BYTES. KEY IGNORED.

ILA2137I-E THE OBJECT OF THE RENAMES THRU CLAUSE IS SUBORDINATE TO THE SUBJECT. STATEMENT DISCARDED.

ILA2139I-W APPLY WRITE VERIFY VALID ONLY FOR MASS STORAGE DEVICES. CLAUSE IGNORED.

ILA2140I-E VALUE CLAUSE SPECIFIED ON BOTH GROUP AND ELEMENTARY ITEM OR ON SUBORDINATE GROUP. SECOND ITEM'S VALUE CLAUSE IGNORED.

ILA2141I-C LENGTH OF LITERAL IS MORE OR LESS THAN LENGTH OF GROUP. LENGTH OF LITERAL ASSUMED.

ILA2142I-W ALPHABETIC OR ALPHANUMERIC ITEM HAS ILLEGAL USAGE. PICTURE CHANGED TO 9.

ILA2143I-W STERLING NON-REPORT PICTURE - MORE THAN ONE V OR S. ASSUMED ONE.

ILA2144I-C NUMERIC PICTURE - MORE THAN ONE V OR S. ASSUMED ONE.

ILA2145I-E ALPHABETIC OR ALPHANUMERIC ITEM LENGTH GREATER THAN 32767. TRUNCATED TO 32767.

ILA2146I-W RECORD CONTAINS DISAGREES WITH COMPUTED MAXIMUM. USING COMPUTED MAXIMUM.

ILA2148I-W ON AN 01 (77) COPY LIBRARY-NAME CLAUSE, LIBRARY DID NOT HAVE AN 01 (77) AS FIRST CARD.

ILA2149I-E VALUE CLAUSE SPECIFIED FOR ITEM WITH OCCURS OR FOR ITEM SUBORDINATE TO AN ITEM WITH OCCURS. CLAUSE IGNORED.

ILA2150I-E VALUE CLAUSE SPECIFIED FOR ITEM IN VARIABLE LENGTH PORTION OF A WORKING-STORAGE RECORD. CLAUSE IGNORED.

ILA2151I-C ELEMENTARY ITEMS NOT INTERNAL FLOATING-POINT MUST HAVE PICTURE. PICTURE ASSUMED 9.

ILA2152I-D COMPILER ERROR - PHASE 2 INPUT UNRECOGNIZABLE. SKIPPING TO NEXT PHASE.

ILA2153I-C APPLY CYLINDER OVERFLOW VALID ONLY FOR INDEXED FILES. CLAUSE IGNORED.

ILA2154I-C THE AREA BEING REDEFINED IS NOT IMMEDIATELY PRECEDING THE ENTRY WHICH REDEFINES IT OR THE LEVEL NUMBERS OF THE SUBJECT AND OBJECT OF THE REDEFINES ARE NOT THE SAME. THE OBJECT OF THE REDEFINES IS ASSUMED TO BE THE LAST ENTRY WITH THE SAME LEVEL NUMBER AS THE SUBJECT OF THE REDEFINES.

ILA2155I-C ILLEGAL STERLING NON-REPORT PICTURE CHARACTER. PICTURE REPLACED BY 9D8D7.

ILA2156I-W PICTURE DOES NOT CONTAIN A SIGN. SIGN DROPPED FROM VALUE CLAUSE LITERAL.

ILA2157I-W RESERVE CLAUSE TREATED AS COMMENTS FOR THIS FILE ORGANIZATION.

ILA2158I-D OCCURS DEPENDING ON VARIABLE IS IN VARIABLE PORTION OF A RECORD. PROGRAM INTERRUPT WILL OCCUR.

ILA2159I-C OBJECT OF REDEFINES CLAUSE NOT DEFINED. PREVIOUS 01 ASSUMED TO BE OBJECT.

ILA2160I-E THE OBJECT OF THE RENAMES OR RENAMES THRU CLAUSE CANNOT CONTAIN AN OCCURS OR OCCURS DEPENDING ON CLAUSE NOR MAY IT BE SUBORDINATE TO AN ITEM THAT

WAS ONE OF THESE CLAUSES. STATEMENT DISCARDED.

ILA2161I-C PICTURE INVALID. ADJACENT C DELIMITERS. ASSUMED PICTURE L(6)9BDZ9BDZ9.

ILA2162I-C PICTURE INVALID. ADJACENT D DELIMITERS. ASSUMED PICTURE L(6)9BDZ9BDZ9.

ILA2163I-C PICTURE INVALID. MORE THAN 2 DELIMITERS. ASSUMED PICTURE L(6)9BDZ9BDZ9.

ILA2164I-C PICTURE INVALID. NO STERLING DELIMITERS. ASSUMED PICTURE L(6)9BDZ9BDZ9.

ILA2165I-C PICTURE INVALID. ONLY 1 STERLING DELIMITER. ASSUME PICTURE L(6)9BDZ9BDZ9.

ILA2166I-C PICTURE INVALID. ERROR IN SHILLING FIELD. ASSUMED SHILLING PICTURE Z9B.

ILA2167I-C PICTURE INVALID. NUMBER OF POUND DIGITS EXCEEDS 15. ASSUMED PICTURE L(6)9BD.

ILA2168I-C PICTURE INVALID. ERROR IN WHOLE PENCE FIELD. ASSUMED PENCE PICTURE Z9.

ILA2169I-C PICTURE INVALID. ERROR IN DECIMAL PENCE FIELD. DECIMAL FIELD TRUNCATED.

ILA2170I-C PICTURE INVALID. ERROR IN POUND FIELD. ASSUMED POUND PICTURE L(6)9B.

ILA2171I-C PICTURE INVALID. NUMBER OF POUND DIGITS PLUS NUMBER OF PENCE DECIMAL EXCEEDS 15. DECIMAL PENCE DROPPED.

ILA2172I-C PICTURE INVALID. SIZE OF REPORT FIELD EXCEEDS 127 BYTES. ASSUMED PICTURE L(6)BDZ9BDZ9.

ILA2173I-C PICTURE INVALID. CR OR DB NOT VALID WITH LEADING SIGN. DECIMAL FIELD TRUNCATED.

ILA2174I-C PICTURE INVALID. SIGN IN DECIMAL PENCE FIELD NOT VALID WITH LEADING SIGN. DECIMAL FIELD TRUNCATED.

ILA2175I-C TRACK-AREA EXCEEDS AND IS REDUCED TO 32,760 BYTES.

ILA2176I-W MULTIPLE FILE TAPE CLAUSE ONLY APPLIES TO MAGNETIC TAPE FILES. CLAUSE IGNORED.

ILA2177I-W ZERO SUPPRESSION CHARACTER WILL OVERRIDE BLANK WHEN ZERO CLAUSE. CLAUSE IGNORED.

ILA2178I-E RECORD-KEY IS NOT WITHIN FILE-RECORD.

ILA2179I-E RECORD-KEY IS NOT FIXED-LENGTH.

ILA2180I-C RECORD-KEY FOR UNBLOCKED FILE INCLUDES FIRST BYTE OF RECORD.

ILA2181I-C NOMINAL OR ACTUAL KEY IS DEFINED WITHIN THE FILE.

ILA2182I-E FILE MAY BE OPENED OUTPUT ONLY. FILE IGNORED.

ILA2183I-W NO LEVEL 01 FOR FD OR SD.

ILA2184I-E VALUE CLAUSE LITERAL DOES NOT CONFORM TO PICTURE. CLAUSE IGNORED.

ILA2185I-E DATA-NAME-3 EITHER PRECEDES DATA-NAME-2 OR IS DATA-NAME-2 IN THE RENAMES THRU CLAUSE. STATEMENT DISCARDED.

ILA2186I-C PICTURE DUPLICATION FACTOR IS ZERO. ASSUMING ONE OCCURRENCE OF PICTURE CHARACTER.

ILA2187I-E OBJECT OF RENAMES CLAUSE OR RENAMES THRU CLAUSE IS NOT IN SAME LOGICAL RECORD. STATEMENT DISCARDED.

ILA2188I-C EXTERNAL FLOATING-POINT PICTURE ILLEGAL WHEN CURRENCY SIGN IS E. PICTURE CHANGED TO 9.

ILA2190I-W PICTURE CLAUSE IS SIGNED, VALUE CLAUSE UNSIGNED. ASSUMED POSITIVE.

ILA2191I-C THE SYNCHRONIZED CLAUSE SHOULD NOT BE SPECIFIED WHEN 88'S ARE UNDER GROUP. STATEMENT ACCEPTED AS WRITTEN.

ILA2192I-E ONLY USAGE IS DISPLAY SHOULD BE SPECIFIED WHEN VALUE CLAUSE IS ASSOCIATED WITH A GROUP ITEM. ACCEPTED AS WRITTEN.

ILA2193I-C LITERAL-1 IS GREATER THAN OR = TO LITERAL-2 IN VALUE THRU CLAUSE. IGNORED.

ILA2194I-C CHARACTERS OPTION IN BLOCK CONTAINS CAUSE NOT LEGAL IN INDEXED FILE. CLAUSE IGNORED.

ILA2196I-C NO VALUE CLAUSE GIVEN FOR CONDITION NAME. VALUE ASSUMED ZERO OR SPACES DEPENDING ON PICTURE.

ILA2199I-C TRACK AREA TOO SMALL. CLAUSE IGNORED.

ILA2200I-E TAPE RECORD MUST CONTAIN AT LEAST 18 CHARACTERS. FILE IGNORED.

ILA2201I-E NOMINAL KEY OR CORE-INDEX DATANAME MUST BE DEFINED IN WORKING-STORAGE SECTION.

ILA2202I-E NOMINAL KEY OR CORE-INDEX DATA-NAME MUST BE DEFINED IN THE FIXED PORTION OF A RECORD. CONTINUING.

ILA2203I-E INVALID DEVICE TYPE FOR SD. DISK ASSUMED.

ILA2204I-E RECORD KEY AND NOMINAL KEY MUST BE THE SAME LENGTH. CONTINUING.

ILA2205I-E ORGANIZATION ILLEGAL FOR ACCESS. FD IGNORED.

ILA2206I-E REWRITE ILLEGAL FOR ORGANIZATION. FD IGNORED.

ILA2207I-C APPLY CORE-INDEX LEGAL ONLY FOR INDEXED ORGANIZATION. CLAUSE IGNORED.

ILA2208I-E ***** KEY INVALID, UNDEFINED, OR NOT UNIQUE. CLAUSE IGNORED.

ILA2209I-C CORE-INDEX DATANAME INVALID. UNDEFINED, OR NOT UNIQUE. CLAUSE IGNORED.

ILA2210I-E ACTUAL KEY MUST BE GREATER THAN 8 AND LESS THAN 263 BYTES IN LENGTH. USING 9.

ILA2211I-C CYLINDER OVERFLOW TOO LARGE. CLAUSE IGNORED.

ILA2212I-W INVALID ALPHANUMERIC EDITED CHARACTER. ACCEPTED AS WRITTEN.

ILA2213I-W USER LABEL RECORD NOT DESCRIBED UNDER FD. USER LABEL IGNORED.

ILA2214I-C STERLING NON-REPORT PICTURE - NO SHILLING SEPARATOR. PICTURE REPLACED BY 9D8D7.

ILA3001I-E ***** NOT DEFINED. ***.

Explanation: This message always appears in conjunction with another message.

ILA3002I-E ***** NOT UNIQUE. ***.

Explanation: This message always appears in conjunction with another message.

ILA3003I-E HIGHEST LEVEL QUALIFIER ***** NOT DEFINED. ***.

Explanation: This message always appears in conjunction with another message.

ILA3004I-W QUALIFYING NAME ***** NOT UNIQUE. DISCARDED.

Explanation: This message always appears in conjunction with another message.

ILA3005I-E ***** NOT A VALID QUALIFIER. ***.

Explanation: This message always appears in conjunction with another message.

ILA3006I-E ***** NOT DEFINED AS PART OF *****. ***.

Explanation: This message always appears in conjunction with another message.

ILA3007I-W ***** NOT UNIQUELY QUALIFIED BY *****. DISCARDED.

Explanation: This message always appears in conjunction with another message.

ILA3008I-E ***** NOT VALID AS IDENTIFIER-1 IN ***** CORRESPONDING STATEMENT. STATEMENT DISCARDED.

ILA3009I-E ***** NOT VALID AS IDENTIFIER-2 IN ***** CORRESPONDING STATEMENT.

ILA3010I-W SUPERFLUOUS 'TO' IGNORED IN ***** CORRESPONDING STATEMENT.

ILA3011I-W NO CORRESPONDENCE FOUND BETWEEN IDENTIFIER AND *****.

ILA3012I-D COMPILER ERROR - LAST ITEM REFERENCED BY ACCESS WAS ELEMENTARY ITEM.

ILA3013I-D DICT PTR LESS THAN QVAR ENTRY FOR ELEMENTARY ITEM.

ILA3014I-D NO MATCH FOUND IN QVAR FOR ***** ELEMENTARY ITEM.

ILA3016I-D IMPOSSIBLE *****. COMPILER ERROR.

ILA3017I-D COMPILER ERROR. ***** MINOR CODE ILLEGAL.

ILA3018I-E SPECIAL REGISTERS TIME-OF-DAY OR CURRENT-DATE MAY ONLY BE USED IN THE MOVE STATEMENT.

Explanation: This message always appears in conjunction with another message.

ILA3019I-D ILLEGAL LEVEL FOR *****.

ILA3020I-E REPORT NAME ILLEGAL AS USED. DISCARDED.

ILA3021I-C ***** NOT UNIQUE IN ITS GROUP. DISCARDED.

ILA3022I-E ***** NOT VALID AS IDENTIFIER-1 IN SEARCH STATEMENT. STATEMENT DISCARDED.

ILA3023I-W ITEMS CONTAINING THE USAGE IS INDEX, REDEFINES, RENAMES, OR OCCURS CLAUSES DO NOT QUALIFY AS CORRESPONDING IDENTIFIERS.

ILA3024I-E NO KEYS WERE SPECIFIED FOR *****. STATEMENT DISCARDED.

ILA3025I-E AN ERROR WAS DETECTED PROCESSING THE 'KEY FOR' PARAMETER.

ILA3026I-E IDENTIFIER OMITTED IN ***** CORRESPONDING STATEMENT.

ILA3027I-W DATA-NAME UNDER LABEL RECORD IS NON-UNIQUE. LAST DATA DESCRIPTION OF ***** ASSUMED.

ILA3029I-E CONDITION NAME ILLEGAL AS USED IN ***** STATEMENT. *****.

ILA4001I-C OUTCOME OF PRECEDING CONDITION LEADS TO NON-EXISTENT 'NEXT SENTENCE'. 'STOP RUN' INSERTED.

ILA4002I-E ***** STATEMENT INCOMPLETE. STATEMENT DISCARDED.

ILA4003I-E EXPECTING NEW STATEMENT. FOUND *****. DELETING TILL NEXT VERB OR PROCEDURE NAME.

ILA4004I-E ***** / ***** IS ILLEGALLY USED IN ***** STATEMENT. DISCARDED.

ILA4005I-E ***** AND ***** VIOLATE RULE ABOUT LENGTH OF TRANSFORM OPERANDS. STATEMENT DISCARDED.

ILA4006I-C ***** STATEMENT CONTAINS UNPAIRED LEFT PARENTHESES. OUTERMOST IGNORED.

ILA4007I-C ***** MISSING OR MISPLACED IN ***** STATEMENT. ASSUMED IN REQUIRED POSITION.

ILA4008I-W SUPERFLUOUS ***** FOUND IN ***** STATEMENT. IGNORED.

ILA4009I-E EXAMINE STATEMENT REQUIRES FIGURATIVE CONSTANT, SINGLE ALPHANUMERIC CHARACTER, OR 1-DIGIT UNSIGNED NUMERIC INTEGRAL LITERAL. FOUND *****. STATEMENT DISCARDED.

ILA4010I-C ***** STATEMENT CONTAINS UNPAIRED RIGHT PARENTHESES. OUTERMOST IGNORED.

ILA4011I-E ***** IS NOT AN ALLOWABLE CHARACTER FOR *****. STATEMENT DISCARDED.

ILA4012I-E COMPARISON BETWEEN TWO LITERALS IS ILLEGAL. TEST DISCARDED.

ILA4013I-C RELATIONAL MISSING IN IF STATEMENT. 'EQUAL' ASSUMED.

ILA4014I-E EXAMINE STATEMENT REQUIRES IDENTIFIER WHOSE USAGE IS DISPLAY. FOUND ***** / *****. STATEMENT DISCARDED.

ILA4015I-E 'GO TO' ILLEGAL UNLESS ALTERED. STATEMENT DISCARDED.

ILA4016I-E OPERAND OF ***** APPEARS IN WRONG SEGMENT OF PROGRAM. ACCEPTED AS WRITTEN.

ILA4017I-E ELSE UNMATCHED BY CONDITION IS DISCARDED.

ILA4018I-E SET STATEMENT HAS AN ILLEGAL OPERAND BEFORE 'TO' OR INCOMPATIBLE OPERANDS. OPERAND BEFORE 'TO' DISCARDED.

ILA4019I-E ***** / ***** MAY NOT BE USED AS ARITHMETIC OPERAND IN ***** STATEMENT. ARBITRARILY SUBSTITUTING *****.

ILA4020I-C SIGN BEFORE ***** IS DISCARDED.

ILA4021I-W MINUS SIGN FOLLOWED BY SPACE ACCEPTED AS REVERSING SIGN OF FOLLOWING LITERAL.

ILA4022I-W EXIT MUST BE SINGLE-WORD PARAGRAPH PRECEDED BY A PROCEDURE-NAME. STATEMENT DISCARDED.

ILA4023I-E STORE-FIELD WHEN USED IN COMPUTATION MUST BE TO NUMERIC DATA-NAME. FOUND ***** / *****. STATEMENT DISCARDED.

ILA4024I-E TWO OPERANDS ARE REQUIRED BEFORE 'GIVING'. STATEMENT DISCARDED.

ILA4025I-E WRITE AFTER POSITIONING AND WRITE BEFORE ADVANCING ILLEGALLY USED FOR SAME FILE. BEFORE ASSUMED.

ILA4026I-E ***** / ***** IS ILLEGALLY USED IN ***** TEST. TEST DISCARDED.

ILA4027I-C RIGHT TERM OF A CONDITION MAY NOT BE NEGATED. NEGATION IS APPLIED TO THE RELATIONAL.

ILA4028I-C TWO 'NOT'S' IN SUCCESSION ILLEGAL. ACCEPTED AS CANCELLING EACH OTHER.

ILA4029I-E ***** / ***** MAY NOT BE COMPARED WITH ***** / *****. TEST DISCARDED.

ILA4030I-E FOUND ***** AFTER CONDITION. EXPECT 'OR', 'AND', OR VERB TO IMMEDIATELY FOLLOW CONDITION. DELETING TILL ONE OF THESE IS FOUND.

ILA4031I-E PROCEDURE-NAME NOT THAT OF A SINGLE GO PARAGRAPH MAY NOT BE ALTERED. STATEMENT DISCARDED.

ILA4032I-C NO ACTION INDICATED IF PRECEDING CONDITION IS TRUE. STATEMENT ACCEPTED WITH TRUE AND FALSE OUTCOMES IDENTICAL.

ILA4033I-C PROCEDURE-NAME WHICH IS THE END-OF-RANGE OF A PERFORM STATEMENT MAY NOT BE ALTERED. STATEMENT DISCARDED.

ILA4034I-C GO DEPENDING ON MUST BE FOLLOWED BY INTEGRAL IDENTIFIER LESS THAN 4 DIGITS IN LENGTH. FOUND *****. STATEMENT DISCARDED.

ILA4035I-W NO MORE THAN 3 INDEX-NAMES OR IDENTIFIERS SHOULD BE VARIED IN PERFORM STATEMENT. ACCEPTED AS WRITTEN.

Explanation: This compiler can normally handle a program varying more than three data-names, but the practice is illegal under standard COBOL language rules and is not recommended.

ILA4036I-W PERFORM RANGE IS FROM ***** TO ***** , WHICH PRECEDES IT. ACCEPTED AS WRITTEN.

Explanation: This compiler can normally handle the perform range indicated, but the practice is not recommended.

ILA4037I-E SYNTAX REQUIRES PROCEDURE-NAME TO FOLLOW 'THRU'. FOUND *****. ***** OPTION DISREGARDED.

ILA4038I-E VARYING OPTION REQUIRES NUMERIC IDENTIFIER/INDEX-NAME. FOUND LITERAL. ARBITRARILY SUBSTITUTING *****.

ILA4039I-E *****/***** IN VARYING OR TIMES OPTION IS NOT NUMERIC. ARBITRARILY SUBSTITUTING *****.

ILA4040I-E ***** FILE ***** MAY NOT BE OPENED ***** AND IS DISCARDED.

ILA4041I-E SYNTAX REQUIRES 'INPUT', 'OUTPUT', OR 'I-O' AFTER OPEN. FOUND *****. DELETING TILL ONE OF THESE IS FOUND.

ILA4042I-E SYNTAX REQUIRES FILE-NAME IN ***** STATEMENT. FOUND *****. DELETING TILL LEGAL ELEMENT FOUND.

ILA4043I-W WRITE AFTER ADVANCING AND WRITE AFTER POSITIONING ILLEGALLY USED FOR SAME FILE. ACCEPTED AS WRITTEN.

ILA4044I-C *****/***** SHOULD NOT BE MOVED TO NUMERIC FIELD. SUBSTITUTING *****.

ILA4045I-E CODE OPTION ILLEGAL FOR ON-LINE DEVICE. OPTION DELETED.

ILA4047I-E READ OR WRITE ILLEGAL FOR LABEL RECORDS. STATEMENT DISCARDED.

ILA4048I-E USE VERB MAY NOT APPEAR EXCEPT IN DECLARATIVES SECTION. STATEMENT DISCARDED.

ILA4049I-W INAPPROPRIATE OPTIONAL COBOL WORDS PRECEDING ***** IGNORED.

ILA4050I-E SYNTAX REQUIRES *****. FOUND *****. STATEMENT DISCARDED.

ILA4052I-E *****/***** MAY NOT BE TARGET FIELD FOR *****/***** IN ***** STATEMENT AND IS DISCARDED.

ILA4054I-E SYNTAX REQUIRES SORT-FILE NAME. FOUND *****. STATEMENT DISCARDED.

ILA4055I-C SORT SEQUENCE NOT SPECIFIED. ASCENDING ASSUMED.

ILA4056I-E SYNTAX REQUIRES *****. FOUND *****. DISCARDED.

ILA4057I-E NUMBER OF SORT KEYS EXCEEDS MAXIMUM OR TOTAL KEY LENGTH

EXCEEDS 256 BYTES. ***** DISCARDED.

ILA4059I-E SORT-KEY MUST BE NON-SUBSCRIPTED OR NON-INDEXED FIXED-LENGTH DATA-NAME DEFINED UNDER AN SD. FOUND *****. DISCARDED.

ILA4060I-C ***** IS NOT A POSITIVE NUMERIC INTEGRAL LITERAL OF REQUIRED LENGTH. ***** OPTION DISCARDED.

ILA4061I-W NEITHER NAMED NOR CHANGED SPECIFIED. STATEMENT ACCEPTED. WILL BE TREATED AS FORMATTED DISPLAY.

ILA4062I-W 'NAMED CHANGED' ACCEPTED AS 'CHANGED NAMED'.

ILA4063I-W PREVIOUS DEBUG PACKET REFERS TO SAME PROCEDURE-NAME. CARD DELETED AND FOLLOWING STATEMENTS ATTACHED TO IMMEDIATELY PRECEDING PACKET.

ILA4064I-E ***** IS NOT A POSITIVE NUMERIC INTEGRAL LITERAL OF REQUIRED LENGTH. SUBSTITUTING *****.

ILA4065I-W NUMERIC LITERAL IN EXAMINE STATEMENT SHOULD BE UNSIGNED. SIGN IGNORED.

ILA4066I-E SYNTAX REQUIRES 01 LEVEL SD DATA-NAME IN RELEASE STATEMENT. FOUND *****. STATEMENT DISCARDED.

ILA4067I-W ALL CHARACTER SHOULD NOT BE USED AS LITERAL IN EXAMINE STATEMENT. STATEMENT ACCEPTED AS WRITTEN.

ILA4068I-D COMPILER ERROR. PHASE 4 TRYING TO GET DATA ATTRIBUTES FOR *****.

ILA4069I-C SYNTAX REQUIRES DEVICE-NAME. FOUND ***** IN ***** STATEMENT. SYSTEM UNIT ASSUMED.

ILA4070I-E ***** STATEMENT REQUIRES IDENTIFIER WHOSE USAGE IS DISPLAY. FOUND SPECIAL REGISTER. STATEMENT DISCARDED.

ILA4071I-E ***** EXCEEDS LEGAL LENGTH. DISCARDED.

ILA4072I-W EXIT FROM ***** PROCEDURE ASSUMED BEFORE *****.

ILA4073I-W ***** SHOULD NOT APPEAR IN DECLARATIVE SECTION. ACCEPTED AS WRITTEN.

Explanation: The statement will be compiled, but its use is illegal under standard COBOL rules and is not recommended.

ILA4074I-C STATEMENT CONTAINS FLOATING POINT DATA ITEMS. REMAINDER IGNORED.

ILA4075I-C 'NEXT SENTENCE' ILLEGAL AND DISCARDED. BOTH ***** AND NOT ***** WILL CAUSE EXECUTION OF NEXT VERB.

ILA4076I-E ***** REQUIRES ***** LEVELS OF SUBSCRIPTING OR INDEXING. SUBSTITUTING FIRST OCCURRENCE OF *****.

ILA4077I-E ***** MAY NOT BE USED AS A SUBSCRIPT SINCE IT REQUIRES SUBSCRIPTING ITSELF. SUBSTITUTING FIRST OCCURRENCE OF *****.

ILA4078I-E SUBSCRIPT MUST BE INTEGRAL DATA-NAME OR LITERAL. FOUND NON-INTEGER *****. SUBSTITUTING FIRST OCCURRENCE OF *****.

ILA4079I-E ***** FOUND AMONG SUBSCRIPTS. SUBSTITUTING FIRST OCCURRENCE OF *****.

ILA4080I-W DEBUG CARD MAY NOT REFER TO A PROCEDURE NAME WHICH ITSELF IS IN A DEBUG PACKET. CARD DELETED AND FOLLOWING STATEMENTS ATTACHED TO IMMEDIATELY PRECEDING PACKET.

ILA4081I-C ***** EXCEEDS ***** CHARACTERS. UP TO 114 ACCEPTED.

ILA4082I-E ***** IS NOT DEFINED AS SUBSCRIPTED OR INDEXED. SUBSCRIPTS DISCARDED.

ILA4083I-E OCCURS-DEPENDING-ON-VARIABLE MUST BE INTEGRAL NON-SUBSCRIPTED DATA-NAME. FOUND *****. ARBITRARILY SUBSTITUTING *****.

ILA4084I-C ILLOGICAL USE OF PARENTHESES ACCEPTED WITH DOUBTS AS TO MEANING.

ILA4085I-E RECORD DESCRIPTION FOR FILE ***** MISSING OR ILLEGAL. STATEMENT DISCARDED.

ILA4086I-C ***** CONDITION USED WHERE ONLY IMPERATIVE STATEMENTS ARE LEGAL MAY CAUSE ERRORS IN PROCESSING.

ILA4087I-E 'END DECLARATIVES' MISSING OR MISPLACED. PROGRAM CANNOT BE EXECUTED.

ILA4088I-D COMPILER ERROR. I-C TEXT COUNT FIELD 0. SKIPPING TO PHASE 5.

ILA4089I-W *****/***** SHOULD NOT BE TARGET FIELD FOR *****/***** IN ***** STATEMENT. STATEMENT ACCEPTED AS WRITTEN.

ILA4090I-E SORT-KEY MUST BE IN FIXED POSITION NOT MORE THAN 4092 BYTES FROM START OF RECORD. ***** DISCARDED.

ILA4091I-E SYNTAX REQUIRES OPERAND. FOUND *****. TEST DISCARDED.

ILA4092I-W EXTERNAL DECIMAL NAME USED IN TRANSFORM STATEMENT. STATEMENT ACCEPTED AS WRITTEN.

ILA4094I-W ***** IS IN A RECORD OF AN APPLY-WRITE-ONLY FILE, AND REFERRING TO IT MAY CAUSE ERRORS IF FILE IS OPENED AS OUTPUT WHEN ***** STATEMENT IS EXECUTED.

ILA4095I-E WRITE FROM IDENTIFIER REQUIRED FOR *****, TO WHICH WRITE-ONLY IS APPLIED. STATEMENT DISCARDED.

ILA4096I-W ***** STATEMENT WILL NEVER BE EXECUTED.

Explanation: The logic of the COBOL source program prevents the computer from executing the statement noted. The compiler, however, accepts the statement as written.

ILA4097I-C UNIT (REEL) OPTION ILLEGAL FOR *****. DISCARDED.

ILA4098I-E 'ALTER' STATEMENT VIOLATES RULE ABOUT REFERENCES TO A GO TO IN A DIFFERENT INDEPENDENT SEGMENT. IGNORED.

ILA4099I-E NO EXIT SPECIFIED BEFORE END OF THIS DECLARATIVE SECTION. CONTROL WILL FALL THROUGH TO NEXT SECTION.

ILA4100I-W IDENTIFIER FOLLOWING INTO(FROM) IN READ(WRITE) STATEMENT SHOULD NOT BE DEFINED UNDER SAME FD AS RECORDNAME. ACCEPTED AS WRITTEN.

ILA4101I-E SET STATEMENT REQUIRES OPERAND AFTER 'UP' OR 'DOWN' TO BE NUMERIC INTEGRAL DATANAME OR POSITIVE INTEGRAL NUMERIC LITERAL. FOUND *****. STATEMENT DISCARDED.

ILA4102I-E SET STATEMENT REQUIRES OPERAND AFTER 'TO' TO BE INDEX NAME, INDEX DATA ITEM, NUMERIC LITERAL, DATA NAME, OR POSITIVE INTEGRAL NUMERIC LITERAL. FOUND ***** . STATEMENT DISCARDED.

ILA4103I-C 'ALL' MUST BE FOLLOWED BY ALPHANUMERIC LITERAL. FOUND *****. DISCARDING 'ALL'.

ILA4104I-E (SEARCH OR) SEARCH ALL STATEMENT HAS EITHER SUBSCRIPTED OR INDEXED IDENTIFIER-1 OR ILLEGAL OPERAND. SCANNING TILL 'AT END' OR 'WHEN'. DELETING TILL ONE OF THESE IS FOUND.

ILA4105I-E DATA-NAME CANNOT BE BOTH INDEXED AND SUBSCRIPTED IN ***** STATEMENT. SUBSCRIPTS DISCARDED.

ILA4106I-E DATA-NAME MUST BE INDEXED BY INDEX NAME OR INDEX NAME PLUS OR MINUS AN INTEGRAL NUMERIC LITERAL. SUBSTITUTING FIRST OCCURRENCE OF *****.

ILA4108I-E CALLED PROGRAM MAY NOT BE SEGMENTED. ENTRY STATEMENT IGNORED.

ILA4109I-E KEY IN SEARCH-ALL FLOATING POINT OR STERLING. STATEMENT CHANGED TO SEARCH STATEMENT.

ILA4110I-E CONDITION IN SEARCH ALL STATEMENT TESTS KEY WITHOUT TESTING ALL PRECEDING KEYS. STATEMENT DISCARDED.

ILA4111I-E INVALID CONDITION OR INVALID FORMULA IN CONDITION IN SEARCH-ALL STATEMENT. STATEMENT DISCARDED.

ILA4112I-W SET UP OR DOWN SHOULD NOT INCREMENT INDEX-NAME BY INDEX DATA ITEM. ACCEPTED AS WRITTEN.

ILA4113I-C BEFORE OR AFTER ADVANCING OR AFTER POSITIONING REQUIRED FOR *****. ASSUMING *****.

ILA4114I-C INVALID ADVANCING/POSITIONING OPTION. 1 LINE ASSUMED.

ILA4115I-C 'AFTER POSITIONING' EXPECTED BUT NOT FOUND. ASSUMED PRESENT.

ILA4116I-E ILLEGAL TO ***** FILE *****. STATEMENT DELETED.

ILA4117I-C ***** CLAUSE MISSING. ***** NEXT SENTENCE USED.

ILA4118I-C NO REWIND IS AN INVALID OPTION FOR FILE *****. IGNORED.

ILA4119I-C INVALID FILE-TYPE FOR START VERB. STATEMENT DISCARDED.

ILA4120I-C REWRITE LEGAL ONLY FOR 'U' AND 'W' DIRECT FILE. ACCEPTED AS 'WRITE'.

ILA5001I-D ERROR OCCURRED WHILE TRYING TO ASSIGN A DOUBLE REGISTER. COMPILATION ABANDONED.

ILA5002I-D ERROR OCCURRED WHILE PROCESSING A SUBSCRIPTED OR INDEXED DATA-NAME. COMPILATION ABANDONED.

ILA5003I-C DIVISOR IS ZERO. RESULT WILL BE ALL 9'S.

ILA5004I-W ALPHANUMERIC SENDING FIELD TOO BIG. 18 LOW ORDER BYTES USED.

ILA5005I-D ERROR OCCURRED WHILE PROCESSING A MOVE. COMPILATION ABANDONED.

ILA5006I-D UNEXPECTED INPUT TO THE MOVE OR STORE PROCESSOR. COMPILATION ABANDONED.

ILA5007I-D UNEXPECTED INPUT TO THE ARITHMETIC CODE GENERATOR. COMPILATION ABANDONED.

ILA5008I-D UNEXPECTED INPUT TO THE FLOATING-POINT ARITHMETIC ROUTINE 'FBCVBH'. COMPILATION ABANDONED.

ILA5009I-D LOST SUBSCRIPT OR INDEX ID IN TABLE 'XSSNT'. COMPILATION ABANDONED.

ILA5010I-C HIGH ORDER TRUNCATION OF THE CONSTANT DID OCCUR.

ILA5011I-W HIGH ORDER TRUNCATION MIGHT OCCUR.

ILA5012I-D LOST INTERMEDIATE RESULT ATTRIBUTES IN 'XINTR' TABLE. COMPILATION ABANDONED.

ILA5013I-C ILLEGAL COMPARISON OF TWO NUMERIC LITERALS. STATEMENT DISCARDED.

ILA5014I-E KEY IN SEARCH ALL AT INVALID OFFSET. STATEMENT DISCARDED.

ILA5015I-E INVALID USE OF SPECIAL REGISTER. STATEMENT DISCARDED.

ILA5016I-E MORE THAN 255 SUBSCRIPT ADDRESS CELLS USED. PROGRAM CANNOT EXECUTE CORRECTLY.

ILA5017I-C INVALID ADVANCING CLAUSE OPTION FOR A DTFCD FILE. USING STACKER 1.

ILA5018I-C INTEGER IN POSITIONING CLAUSE NOT BETWEEN 0 AND 3. 1 ASSUMED.

ILA5019I-C PUNCH STACKER SELECT SPECIFIED FOR A DTFPR FILE. USING "SKIP TO CHANNEL 1".

ILA5020I-C IDENTIFIER IN EXHIBIT EXCEEDS MAXIMUM. TRUNCATED TO 120 CHARACTERS.

ILA5021I-C INTEGER IN ADVANCING OR POSITIONING CLAUSE NOT POSITIVE. POSITIVE ASSUMED.

ILA5022I-C MORE THAN 2-DIGIT INTEGER IN ADVANCING CLAUSE. USING INTEGER-1.

ILA5023I-E EOP INVALID FOR DOUBLE-BUFFERED FILE. IGNORED.

ILA5024I-E END OF PAGE OPTION REQUESTED FOR NON-DTFPR FILE. IGNORED.

ILA5025I-C ADVANCING OR POSITIONING OPTION ILLEGAL FOR NON-SEQUENTIAL FILE. IGNORED.

ILA5026I-C EXHIBIT CHANGED OPERAND GREATER THAN 256 BYTES. LENGTH OF 256 ASSUMED.

The following messages may be interspersed in phase 6 output.

ILA6003I-D ERROR FOUND PROCESSING F4 TEXT. UNKNOWN DATA A-TEXT CODE.

ILA6005I-D ERROR FOUND PROCESSING F1 TEXT. COMPILATION ABANDONED.

ILA6006I-E MAP SUPPRESS SPECIFIED AND E-LEVEL DIAGNOSTIC HAS OCCURRED. LISTX, LINK, CLIST AND DECK WILL BE IGNORED.

ILA6007I-D TABLE HAS EXCEEDED MAXIMUM SIZE. PMAP, LOAD MODULE AND DECK WILL BE INCOMPLETE.

The following messages are issued on SYSLOG and SYSLST during an FCOBOL

compilation. They are printed on SYSLST with the prefix ILA.

C100I BACKGROUND AREA IS LESS THAN 54K.

Explanation: At least 54K is required to compile using FCOBOL.

User Response: Allocate at least 54K to the background partition.

C101I DEVICE NOT ASSIGNED - SYSnnn.

Explanation: nnn is either 001, 002, 003, or 004. The specified logical unit is unassigned and must be assigned.

User Response: Make sure that the assignment is made.

C102I UNSUPPORTED DEVICE TYPE - SYSnnn.

Explanation: nnn is either 001, 002, 003, or 004. The specified file must be a disk file if SYS001, or a tape or disk file if SYS002 through SYS004.

User Response: Make the correct assignment.

C103I END OF FILE ON SYSIPT.

Explanation: End-of-file was encountered in the initialization phase - no source language was found.

User Response: Check the source module for embedded /* (slash asterisk) cards or missing source cards.

C104I WARNING. SYS001 FILE IS TAPE.

Explanation: In small, simple programs that do not require dictionary spill, it is sometimes possible to compile with the spill file (SYS001) assigned to tape. However, if any spill does occur, an input/output error may occur.

User Response: Reassign SYS001 to a disk file.

OBJECT TIME MESSAGES

The following messages are normally issued on SYSLOG.

C110A STOP literal

Explanation: The programmer has issued a STOP literal statement in the FCOBOL program.

User Response: Operator should respond with end-of-block, or with any character in order to proceed with the program.

C111A AWAITING REPLY

Explanation: This message is issued in connection with the FCOBOL ACCEPT statement.

User Response: The operator should reply as specified by the programmer.

The following messages are issued on SYSLOG and SYSLST prior to cancellation of the job. If the DUMP option is specified, a partial dump is taken from the problem program origin to the highest core location of the last phase loaded. When this occurs, the eight bytes immediately preceding the DTF are destroyed. The messages have the form:

CmmmI SYSnnn filename dtfaddress text

where mmm and text correspond as follows:

mmm	text
112	DATA CHECK
113	WRONG LENGTH RECORD
114	PRIME DATA AREA FULL
115	CYLINDER INDEX FULL
116	MASTER INDEX FULL
117	OVERFLOW AREA FULL
118	DATA CHECK IN COUNT
119	DATA CHECK IN KEY OR DATA
120	NO ROOM FOUND
121	DASD ERROR
122	DASD ERROR WHILE ATTEMPTING TO WRITE RECORD ZERO
123	FILE CANNOT BE OPENED AFTER CLOSE WITH LOCK

Explanation: Condition indicated occurred on SYSnnn, filename is 7 characters in length and is the file-name as generated in the SELECT sentence, and dtfaddress is the hexadecimal address of the file's DTF table.

User Response: Rerun the job or add a user declarative section to the Procedure Division to handle errors within the program.

124 CYLINDER AND MASTER INDEX TOO SMALL

125 NO EXTENTS

Explanation: During CLOSE UNIT processing, no extent is found for the next volume.

User Response: Rerun job with proper EXTENT (XTENT) statements.

The following message is issued on SYSLOG:

C126D IS IT EOF?

Explanation: A tapemark was just read on an unlabeled tape file described at compilation time as having more than one reel.

User Response: The operator must respond either with N if it is not the last reel, or with Y if it is the last reel.

COBOL Object Program Unnumbered Messages

xxx...

Explanation: This message is written on the console and is recognizable because it is not preceded by a message code and action indicator. It is issued by an object program originally coded in COBOL. The message text is supplied by the object program and may indicate alternative action to be taken.

System Action: The job continues.

Operator Response: Operator response, if any is needed, is determined by the message text.

This appendix contains information concerning system requirements for the DOS American National Standard COBOL compiler, execution time considerations, and the Sort Feature. Additional information used in estimating the main and auxiliary storage requirements is contained in the publication IBM System/360 Disk Operating System: Performance Estimates, Form GC24-5032.

MINIMUM MACHINE REQUIREMENTS FOR THE COMPILER

1. At least a System/360 Model 30. The compiler also operates on Models 40, 50, 65, 67 (in 65 mode), or 75. A minimum of 54K bytes of main storage is required except when using the Report Writer Feature. This necessitates a minimum partition size of 80K bytes.

2. Four utility data sets on 2400 Tape Units, 2311 Disk Drives, or 2314 Disk Storage Facility. At least one utility data set as well as the operating system must reside on a mass storage device (i.e., a 2311 or 2314). If the three remaining utility data sets reside on tape, there must be a separate tape unit for each data set. If they reside on a mass storage device, there must be enough space on that device.

   Utility data set assignments must be made as follows:

   SYS001 - disk unit
   SYS002 - disk or tape unit
   SYS003 - disk or tape unit
   SYS004 - disk or tape unit

3. A device, such as a printer keyboard, for direct operator communication.

4. A device, such as a card reader, for the job input stream.

5. A device, such as a printer or tape unit, for system output files.

6. The commercial instruction set, and floating-point arithmetic feature, if floating-point literals or calculations are used.

Note: All devices currently supported by IBM System/360 Disk Operating System COBOL are supported by IBM System/360 Disk Operating System American National Standard COBOL.

EXECUTION TIME CONSIDERATIONS

The amount of main storage must be sufficient to accomodate at least:

- The selected control program
- Support for the file processing techniques used
- Load module to be executed

SORT FEATURE CONSIDERATIONS

The Sort/Merge program must be executed under control of the Disk Operating System. The program requires the following minimum machine configuration:

1. 16K (16,384) bytes of main storage if the program is to use IBM 2400 Series Magnetic Tape Units or IBM 2311 Disk Storage Drives for intermediate storage. The Sort/Merge program uses 10,240 bytes; an additional 6K bytes are needed for the Disk Operating System and user-written routines.

2. 32K (32,768) bytes of main storage if the program is to use the IBM 2314 Direct Access Facility for intermediate storage. The Sort/Merge program uses 22,528 bytes; an additional 10K bytes are needed for the Disk Operating System and user-written routines.

   Note: Performance increases significantly if 50K is available for operation of the Sort/Merge program. At the 100K level, the performance is very high.

3. Standard instruction set.

4. One 2311 or 2314 disk unit attached to one selector channel for sort input, output, and work files. (System residence requirements may necessitate having an additional disk storage unit for sorting.)

5. One IBM 1403 and 1443 Printer, or one IBM 1052 Printer Keyboard.

6. One IBM 1442, 2501, 2520, and 2540 Card Reader, or one IBM 2400 Series Magnetic Tape Unit (7- or 9-track) assigned to SYSIPT and SYSRDR.

7. Three IBM 2400 Series Magnetic Tape Units for work files when tape units are to be used for intermediate storage.

8. One IBM 2400 Series Magnetic Tape Unit if tape input/output is to be used.

When tape units are used for intermediate storage, five input/output devices are required as the minimum for a sorting operation (one input, three work, one output). When disk units are used for intermediate storage, three extents are required (one input, one work, one output).

Three extents are required as a minimum for a disk merging operation (two input, one output). A one-way merge, which simply copies the input file, may be executed with two tape units or one disk unit.

COMMUNICATION REGION

The Communication Region is a (46-byte) ? storage area within the Supervisor used by the Supervisor and the COBOL compiler. The structure of the Communication Region is illustrated in Figure 58.

Fields in the Communication Region are addressed relative to the first byte of the region. An asterisk (*) identifies the fields available to the COBOL user.

Byte(s)	Meaning
0-7*	Calendar date supplied during the IPL procedure or by the DATE control statement. This field can be used for dating printed output of the COBOL program via the special register CURRENT-DATE. The date can be in one of two forms: mm/dd/yy or dd/mm/yy where mm is month, dd is day, and yy is year. The form is chosen by the installation at system generation time.
8,9	Address of the background program label area.
10,11	Reserved for control program use.
12-22*	User area for inter-program or intra-program communication. This field can be referenced in a COBOL program executing in the background via the special register COM-REG. All eleven bytes are initialized to binary zeros when a JOB control statement is encountered.
23*	User program switch indicators (UPSI). The condition-name associated with the status of the UPSI switches can be specified in the COBOL program via the Special-Names paragraph of the Environment Division. UPSI byte switches are set by the UPSI control statement. The condition-name associated with each may be tested in the Procedure Division of the COBOL program. UPSI byte switches are initialized to binary zeros when a JOB control statement is encountered.
24-31	Jobname for background programs located in the operand field of the JOB control statement.
32-35	Address of the uppermost byte of the background program area.
36-39	Address of the uppermost byte of the last phase loaded into the background program area.
40-43	Address of the uppermost byte used in loading any phase of the background program.
44,45	Length of the background program label area.

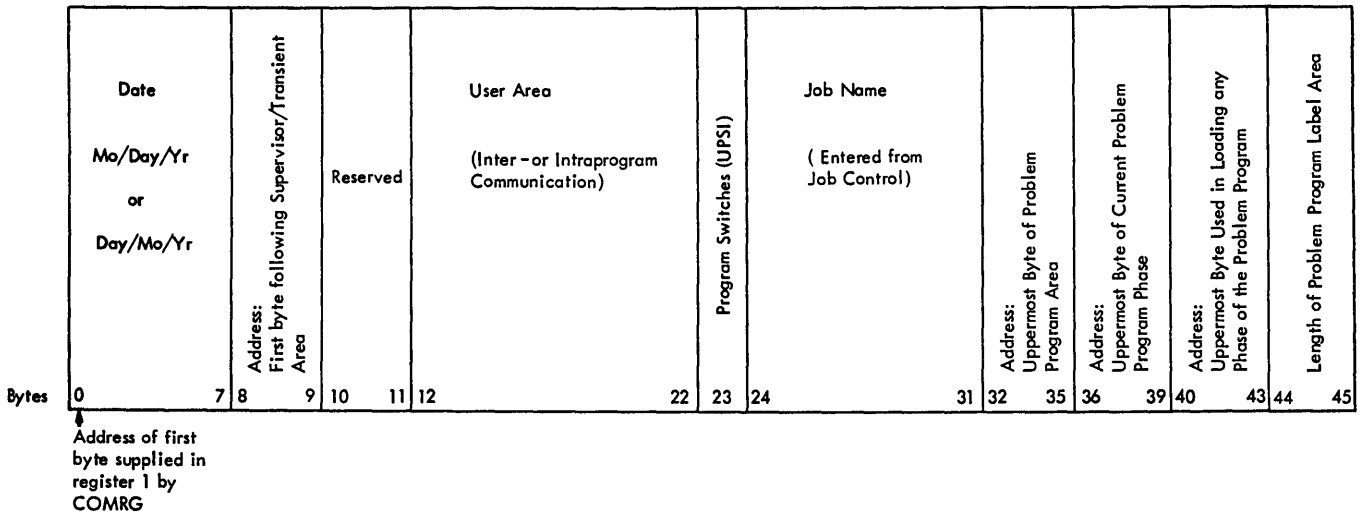


Figure 58. Communication Region in the Supervisor

This appendix illustrates the necessary job control statements and their sequence for five typical programs:

1. Creating a Direct File
2. Retrieving and Updating a Direct File
3. Creating an Indexed File
4. Retrieving and Updating an Indexed File
5. Sorting an Unlabeled Tape File

In all five programs the programmer has requested the following compiler options through the OPTION control statement:

NODECK -- No punched card output for the object program is needed.

LINK -- The object module is to be linkage edited.

LIST -- The COBOL source statements are to be printed on SYSLST.

LISTX -- A Procedure Division map is to be printed on SYSLST.

SYM -- A Data Division map is to be printed on SYSLST.

ERRS -- The diagnostic messages of the COBOL compiler are to be printed on SYSLST.

The EXEC FCOBOL statement calls for execution of the FCOBOL compiler.

By using the CBL card, the programmer indicates that in this source program the quotation mark (") is used for nonnumeric literals.

The ASSIGN clause in the COBOL source program specifies a system-name with the following fields:

SYSnnn-class-device-organization-[name]

The ASSGN control statement for a file must specify the same logical unit as the SYSnnn field of system-name. The ASSGN statement assigns the logical unit to a specific hexadecimal address. The address specified must be associated with the device whose number is given in the device field of system-name.

The DLBL control statememt for a labeled file on a mass storage device must contain the same name as system-name. This is the name by which the file is known to the control program. (The name field of system-name is optional. If name is omitted, the DLBL statement must specify the logical unit (SYSnnn) as the file-name.) The code field of the DLBL statement must correspond to the class and organization fields of system-name as follows:

DLBL "code"	ASSIGN "class"	ASSIGN "organization"
SD	DA or UT	S
DA	DA	A or U, D or W
ISC	DA	I
ISE	DA	I

The first EXTENT control statement for a file on a mass storage device must specify the same logical unit as the SYSnnn field of system-name. (Subsequent EXTENT statements for the same file, if they immediately follow the first, may omit this field.) The type of the extent must be compatible with the organization field of system-name as follows:

	EXTENT "type"	ASSIGN "organization"
1	(data area, no split cylinder)	S, A, U, I, D, W
2	(overflow area for indexed file)	I
3	(index area for indexed file)	I
4	(data area, split cylinder)	S, A, U, I, D, W

The following two examples illustrate the job control statements necessary for programs that create and update a direct file.

In the COBOL source programs, the programmer has written:

```
SELECT DA-FILE ASSIGN TO
  SYS015-DA-2311-A-MASTER...

SELECT CARD-FILE ASSIGN TO
  SYS007-UR-2540R-S...
```

In the READFILE source program, the programmer has written:

```
SELECT PRINT-FILE ASSIGN TO
  SYS008-UR-2403-S...
```

(Note the relationship between the system-names in the source programs and the control statements.)

The LBLTYP statement defines the amount of storage to be reserved to process labels for the DA file. The file has one extent.

The EXEC LNKEDT statement causes the object program to be linkage edited.

An ASSGN control statement assigns logical unit SYS007 to the hexadecimal address 00C -- a 2540R Card Reader.

In the updating program, another ASSGN statement assigns logical unit SYS008 to the hexadecimal address 00E -- a 1403 Printer.

The next series of statements identify the direct file completely.

The ASSGN statement identifies the file as residing on logical unit SYS015, which has the hexadecimal address of 192 -- a 2311 Disk Drive.

The DLBL statement specifies the filename as MASTER, with an expiration date of the 365th day of 1970, and that the file has direct organization (DA).

The EXTENT statement specifies that the file residing on logical unit SYS015 has a serial number 111111, that the extent is a data area with no split cylinder and that this is the first (and only) extent for the file (type and sequence number 1,0), that the file begins on relative track 1020 (track 0 of cylinder 102), and that the file occupies 100 tracks.

(Note that in the EXTENT statement, the relative track number (1020) is not required for the input DA file of the updating program, since the system will use the file labels for this information.)

The EXEC statement begins execution of the problem program, and is followed by input data.

The /* statements indicate end-of-data, the /& statement indicates end-of-job.

Creating a Direct File

```
// JOB CREATEDA
// OPTION NODECK,LINK,LIST,LISTX,SYM,ERRS
// EXEC FCOBOL
 CBL QUOTE

   {COBOL source deck}
/*
// LBLTYP NSD(01)
// EXEC LNKEDT
// ASSGN SYS007,X'00C'
// ASSGN SYS015,X'192'
// DLBL MASTER,70/365,DA
// EXTENT SYS015,111111,1,0,1020,100
// EXEC

   {input data cards}
/*
/&
```

Retrieving and Updating a Direct File

```
// JOB READFILE
// OPTION NODECK,LINK,LIST,LISTX,SYM,ERRS
// EXEC FCOBOL
 CBL QUOTE

   {COBOL source deck}
/*
// LBLTYP NSD(01)
// EXEC LNKEDT
// ASSGN SYS007,X'00C'
// ASSGN SYS008,X'00E'
// ASSGN SYS015,X'192'
// DLBL MASTER,70/365,DA
// EXTENT SYS015,111111,1,0,1020,100

   {input data cards}
/*
/&
```

The following two examples illustrate the job control statements necessary for programs that create and update an indexed file.

In the CREATEIS source program, the programmer has written:

```
SELECT IS-FILE ASSIGN TO
  SYS015-DA-2311-I-MASTER
ACCESS IS SEQUENTIAL
RECORD KEY IS REC-ID.
```

In the RANDIS source program, the programmer has written:

```
SELECT IS-FILE ASSIGN TO
  SYS015-DA-2311-I-MASTER
ACCESS IS RANDOM
NOMINAL KEY IS KEY-ID
RECORD KEY IS REC-ID.

SELECT PRINT-FILE ASSIGN TO
  SYS008-UR-1403-S
RESERVE NO ALTERNATE AREAS.
```

In both source programs, he has written:

```
SELECT CARD-FILE ASSIGN TO
  SYS007-UR-2540R-S.

I-O-CONTROL.
    APPLY MASTER-INDEX TO 2311 ON IS-FILE.
```

(Note the relationship between the source program statements and the job control statements.)

The LBLTYP statement defines the amount of storage reserved to process lables for the indexed file. The file has three extents: a master index extent, a cylinder index extent, and a data extent.

The EXEC LNKEDT statement causes the object module to be linkage edited.

An ASSGN control statement assigns logical unit SYS007 to the hexadecimal address 00C -- a 2540R Card Reader.

In the retrieval program, another ASSGN statement assigns logical unit SYS008 to the hexadecimal address 00E -- a 1403 Printer.

The next ASSGN statement assigns logical unit SYS015 to the hexadecimal address 193 -- a 2311 Disk Drive.

The DLBL statement names the file as MASTER, and indicates the expiration date as the 365th day of 1970. In the file creation program, the file label is indexed sequential using Load Create (code ISC); in the retrieval program, the file label is indexed sequential using Load Extension, Add or Retrieve (code ISE).

The first EXTENT statement is identified as a master index (type and sequence numbers are 4,0), and the relative track is 1800 (the extent begins on cylinder 180 track 0), and the extent is 10 tracks long.

The second EXTENT statement is identified as a cylinder index (type and sequence number are 4,1), the relative track is 1810 (the extent begins on cylinder 181, track 0), and the extent is 10 tracks long.

(Note that the extents assigned to master and cylinder indexes must be contiguous, and that the master index must precede the cylinder index on the disk pack. Also note, that if a master index is not requested, the first extent is that for the cylinder index, which would be type 4, sequence number 1.)

The third EXTENT statement is identified as a data area (type 1) and is the third extent named for this file. The relative track is 0010 (the extent begins on cylinder 1, track 0), and the extent is 1750 tracks long.

End-of-data is indicated with the /* statement; end-of-job is indicated with the /& statement.

Creating an Indexed File

```
// JOB CREATEIS
// OPTION NODECK,LINK,LIST,LISTX,SYM,ERRS
// EXEC FCOBOL
 CBL QUOTE

   {COBOL source deck}
/*
// LBLTYP NSD(03)
// EXEC LNKEDT
// ASSGN SYS007,X'00C'
// ASSGN SYS015,X'193'
// DLBL MASTER,70/365,ISC
// EXTENT SYS015,111111,4,0,1800,10
// EXTENT SYS015,111111,4,1,1810,10
// EXTENT SYS015,111111,1,2,0010,1750
// EXEC

   {input data card}
/*
/&
```

Retrieving and Updating an Indexed File

```
// JOB RANDIS
// OPTION NODECK,LINK,LIST,LISTX,SYM,ERRS
// EXEC FCOBOL

   {COBOL source deck}
// LBLTYP NSD(03)
// EXEC LNKEDT
// ASSGN SYS007,X'00C'
// ASSGN SYS008,X'00E'
// ASSGN SYS015,X'193'
// DLBL MASTER,70/365,ISE
// EXTENT SYS015,111111,4,0,1800,5
// EXTENT SYS015,111111,4,1,1810,10
// EXTENT SYS015,111111,1,3,0010,1750
// EXEC

   {input data cards}
/*
/&
```

FILES USED IN A SORT OPERATION

The following example illustrates the job control statements necessary for a program that sorts an unlabeled tape file.

In the COBOL source program, the programmer has written:

```
SELECT NET-FILE-IN ASSIGN TO
  SYS007-UT-2400-S.

SELECT NET-FILE-OUT ASSIGN TO
  SYS008-UT-2400-SL.

SELECT NET-FILE ASSIGN TO 3
  SYS001-UT-2400-S.
```

NET-FILE-IN is the input file; NET-FILE-OUT is the output file; NET-FILE is the sort work file, which utilizes three tape units.

(Note the relationship between the system-names in the COBOL source program and the control statements.)

The EXEC LNKEDT statement causes the job to be linkage edited.

The first two ASSGN control statements assign the logical unit SYS007 to hexadecimal address 181, and logical unit SYS008 to hexadecimal address 182. SYS007 is the sort input file, and SYS008 is the sort output file.

The last three ASSGN statements assign logical unit SYS001 to hexadecimal address 183, logical unit SYS002 to hexadecimal address 281, and logical unit SYS003 to hexadecimal address 282. SYS001, SYS002, and SYS003 are the logical units that must be used for sort work files. The sort work files must be assigned to 9-track tape units. At this installation, 9-track tape drives are associated with hexadecimal addresses 183, 281, and 282.

Sorting an Unlabeled Tape File

```
// JOB SORTCOB
// OPTION NODECK,LINK,LIST,LISTX,SYM,ERRS
// EXEC FCOBOL
 CBL QUOTE

   {COBOL source deck}
// EXEC LNKEDT
// ASSGN SYS007,X'181'
// ASSGN SYS008,X'182'
// ASSGN SYS001,X'183'
// ASSGN SYS002,X'281'
// ASSGN SYS003,X'282'
// EXEC
/&
```

(Where more than one page reference is given, the major reference appears first.)

* 21
/* 15
/& 25

abnormal termination 54-57
ACCEPT statement 78
  subroutines 212,211
accessing a direct file 109-127
  randomly 110-111
  sequentially 110
accessing an indexed file 127-131
  randomly 130-131
  sequentially 130
accessing a sequential file 109
actual key 111-127
  actual track addressing 111-115,125-127
    sample program 115-119
  relative track addressing 111-113
    sample program 120-124
ACTUAL KEY clause 111-112
actual track addressing 111-115,125-127
  sample program 115-119
ADD statement 46
adding records to an indexed file 128-129,130-131
adding source statements to a book 46
addressing direct files
  actual track addressing 111-115,125-127
    sample program 115-119
  relative track addressing 111-113
    sample program 120-124
ALTER statement
  in a called program 81
APOST option 36
application programs 12
APPLY EXTENDED-SEARCH clause 110
APPLY WRITE-ONLY clause 157
  programming technique 165
arguments
  passed to a called assembler language program 84-85
  passed to a called COBOL program 81,82-83
arithmetic subroutines 214
assembler language routine for accomplishing overlay 88-89
assembler language subprograms 83-87
assembler sublibrary of source statement library 43
ASSGN control statement 22-23,20
ASSIGN clause 19,20
assigning storage for compiler work file buffers 36
assignment of input/output devices 19-20
asterisk (*) 21
AUTOLINK 40

Automatic Library Look-Up (AUTOLINK) 40
  and ENTRY control statement 40
  and PHASE control statement 39
auxiliary subroutines 212

background program 12
  label area 239
BASIS card 48-49,43
  used for debugging 54
batched-job mode 12
binary items 173-174
BKEND control statement 44
block descriptor field 157
block-length field
  V-mode records 154
books in the source statement library
  cataloging 43-44
  retrieving 44-45
  updating 45-47
braces 21
brackets 21
BUF option 36
building tables 186

CALL statement 81-82
  in segmented program 99
called program 81
calling an assembler language subprogram 84-87
calling and called programs 81-93
calling program 81
capacity records 109-110
CATAL option 34,41
cataloging
  a book 43-45
  a module 42-43
  a program phase 41-42,37,39
  a segmented program 101
CATALR control statement 42
CATALS control statement 43
CBL statement 36-37
checking standard labels 148,150
  DLAB control statement 29
  DLBL control statement 26
  TLBL control statement 25
  TPLAB control statement 30
  VOL control statement 28-29
checkpoint subroutine 214,65
checkpointing a COBOL program 64-66
checkpoints during a sort operation 97
  control statement requirements 65
CHKPT macro instruction 65
class test subroutine 215
CLIST option 36
CLOSE control statement 24
CLOSE UNIT subroutine 212
CLOSE WITH LOCK subroutine 211
COBOL option card (CBL card) 36-37

COBOL sublibrary of source statement library 43
comment control statement 21
comments in job control statements 21
comments on the phase map 77
Communication Region 239
  DATE control statement 24
compare subroutines 214
compilation 17,11
  options for 36-37,33-34
  work files required for 236
compile and edit job 14
compile, edit, and execute job 14
compile-only job 14
compiler diagnostic messages 217-235,74
  generation of 64
  working with 64
compiler-generated card number
  on diagnostic messages 74
  on object code listing 73
  on source statements 67
compiler machine requirements 237
compiler options
  CBL card 36-37
  OPTION control statement 33-34
compiler output 67-75
completion codes from sort program 97
computational items
  conversions involving 169-171
  internal representation of 172-174
  special considerations for 172
COMPUTE statement
  programming technique 176
  subroutines 214
COM-REG 239
condensed object listing 36
continuation of job control statements 21
  DLAB control statement 29
  TPLAB control statement 30
control fields
  S-mode records 157
  V-mode records 154
control footings and page format 180
control program 11
control sections 39
conversion subroutines 212-213
converting elementary data items 169-171
converting track addresses
  in a COBOL source program
    relative to actual 115
  in EXTENT control statement
    actual to relative 27
    relative to actual 27
copy function of Librarian 41
COPY statement 44-45
core image directory 41
core image library 41-42,37,39
correspondence of arguments and parameters
  assembler language subprograms 84
  COBOL subprograms 82-83
creating a direct file 110-111
  actual track addressing 111-119,125-126
    sample program 115-119
  relative track addressing 111-113,120-124
    sample program 120-124
  sample job decks 241-242
creating an indexed file 130
  sample job deck 243,241
creating standard mass storage file labels 149-150
  DLAB control statement 29
  DLBL control statement 26
  PARSTD option 34
  STDLABEL option 34
creating standard tape file labels 144,145-149
  PARSTD option 34
  STDLABEL option 34
  TLBL control statement 25
  TPLAB control statement 30
creating user labels 148-149,144
  USRLABEL option 34
cross-reference dictionary 74
CURRENT-DATE 239,25
cylinder index 128
cylinder overflow area 128

data, locating in a dump 57
data extents
  direct files 27,30
  indexed files 27,30
data files 17
data format conversion 169-171
data formats in the computer 172-174
data management 108,17
DATE-COMPILED 67
DATE control statement 24-25
  and Communication Region 239
debug packet 54
debugging language 51-54
DECK option 33
DEL statement 46
DELETE card 48-50
  used for debugging 54
deleting source statements
  for one run only 49-50
  from a book 46
DEPENDING ON option of OCCURS clause
  and Table Handling Feature 182-183
  and variable-length records 162-164
device assignment 19-20
  duration of effect 23
diagnostic messages
  compiler 74-75,64,217-234
  generation of 64
  linkage editor 77
  object time 79,234-235
direct files 109-127
  actual track addressing 109-119,125-126
  relative track addressing 109-113,120-124
  sample job decks 241-242
direct linkage 88
direct organization 109-127,107
disk extent subroutines 212
DISPLAY items
  conversions involving 171,170
  internal format of 172
  special considerations for 172
DISPLAY statement subroutines 211-212

division/remainder method of randomizing 113-115
used to create a direct file
actual track addressing 115-119
relative track addressing 120-124
DLAB control statement 29
DLBL control statement 26
identifying private libraries 48
DTF
creation of 108,133
locating in a dump 56
symbolic name of 29
DTF tables 133-138
dummy segment 99
DUMP option 33
dumps 54-63
errors causing 55-56
how to use 55
locating data in 57
locating DTF in 56

edit and execute job 14
editing 17-18
edit-only job 14
EJECT 67,165
ellipsis 22
END statement 47
end-of-data control statement 15
end-of-job control statement 15
ENTRY control statement 40
generated by compiler for Segmentation 100,101
entry point in a called program 81,82
ENTRY statement 82
in an overlay structure 89
error recovery
on unit-record devices 36
using an assembler language routine 141-143
using error declarative section 138-141
using INVALID KEY 138,139
ERRS option 33
EXEC control statement 15
EXEC FCOBOL statement 17,15
EXEC LNKEDT statement 17,15
execute-only job 14
execution output 77-79
execution time, machine requirements 237
EXHIBIT statement 52-53
subroutine 211
EXIT PROGRAM statement 82
extended search 110
extended source program library facility 48-50
EXTENT control statement 26-28
extents, maximum number 27,30
external-name 82
external reference 82
unresolved 40

F-mode records 153
FCOBOL 17
file integrity 110
file organization 107-108
direct 109-126,107
indexed 127-131,107
sequential 109,107

file retention
on direct-access storage devices 26
on tape devices 25
fixed-length records 153
fixed partitioned multiprogramming 12
FLAGE option 36
FLAGW option 36
foreground programs 12
format F records 153
format notation 21-22
format S records 157-162
format U records 153-154
format V records 154-157

generic terms 21
GIVING option of error declarative 139-141
global table 73
glossary 72
GOBACK statement 82

IBM-supplied processing programs 12
identification field of COBOL source statements 45
IF statement 176
ILBDCKP0 subroutine 65,214
ILBDSEM0 subroutine 101,102,214
ILBDSRT0 subroutine 96,214
in-line parameter list 85,87
INCLUDE control statement 39
independent overflow area 128
independent segment 99
index data items 182,183
index-names 182,183-186
indexed files 127-131
adding records to 128-129
sample job decks 241,243-244
indexed organization 107,127-131
improving efficiency when using 131
indexes 128
indirect addressing 112-114
Initial Program Loader (IPL) 11
input
compiler 17
Job Control Processor 20
Linkage Editor 17,37,39
for a segmented program 100,101,102
INPUT PROCEDURE option 95,96
Input/Output Control System (IOCS) 108
input/output error subroutines 212
input/output errors 138-143
INSERT card 49,50
used for debugging 54
intermediate results 175-176
interrupts, errors causing 55-56
INVALID KEY condition 138,139
direct organization 110,139
indexed organization 139
standard sequential organization 109,139
IOCS 108

job 13
types 14
job control commands 37

job control considerations
for accomplishing overlay 90
for sort program 95-96
Job Control Processor 11,21
options 33-34
JOB control statement 31,15
job control statements 15,21-37
definition 11
format notation 21
formation of 21
overlay considerations 90
sequence of 22
sort considerations 95-96
job deck 15,22
job definition 13
job definition statements 15
job step 13

label area, reserving storage for 31
label definition
DLAB control statement 29
DLBL control statement 26
TLBL control statement 25
TPLAB control statement 30
label processing 144-151
mass storage file labels 149-151
tape file labels 144-149
label processing considerations
mass storage file labels 150-151
tape file labels 148-149
label processing subroutines 211
LBLTYP control statement 31-32
level numbers 166
LIBR option 37
Librarian 41,12
line overlay (Report Writer) 179
LINK option 38,33,216
linkage 81-83
in a called program 82
in a calling program 81
correspondence of arguments and parameters 82
entry points 82
linkage conventions 83-87
argument list 85
assembler subprogram 83
generated by compiler for segmentation 100,101,102
overlay considerations 89
in-line parameter list 85
lowest level subprogram 87
register use 84
save area 84
linkage editing 17,14
with overlay 89
without overlay 83
Linkage Editor 11
linkage editor control statements 37-40
fields of 37
generated by compiler for segmentation 100,101,102
overlay considerations
placement of 38-39
linkage editor diagnostic of input 77
linkage editor input deck 18
linkage editor input for a segmented program 100,101,102
linkage editor messages 77
linkage editor output 75-77,64,17-18
linkage registers 84
linkage with the Sort Feature 96-97
LIOCS 108
LIST option 33
LISTIO control statement 32
LISTX option 33
literal pool 73
locating the Working-Storage Section in dumps 166-167
LOG option 33
logic module 108
Logical Input/Output Control System (LIOCS) 108
logical record 108
spanning physical blocks 157-162

machine considerations 231-238
main program or subprogram subroutine 215
maintenance function of Librarian 41-48
mass storage device 107
mass storage file labels 149-151
master index 120
modularizing
the Data Division 166
the Procedure Division 174-175
used by the Segmentation Feature 99
module 11
input to Linkage Editor 39
MOVE statement 176
MOVE statement subroutines 215
MTC control statement 32-33
multifile volumes
TLBL control statement 25
multiphase program 18,13,14
multiple file tape subroutine 212
multiprogramming 12
multivolume tape files with nonstandard labels 149

naming conventions used by Segmentation 100
NEXT GROUP clause 181
NODECK option 33
NODUMP option 33
NOERRS option 33
NOLIBR option 37
NOLINK option 33
NOLIST option 33
NOLISTX option 33
NOLOG option 33
NOMAP option 18,40
NOMINAL KEY clause 130,131
nonstandard tape file labels 144
multivolume file considerations 149
NOSEQ option 36
NOSYM option 33
NOTE statement 176
NOTRUNC option 37
NOXREF option 33
NSTD-REELS 149

object code listing 73-74
object module 75
produced by the compiler for Segmentation 100
object time messages 234-235

OCCURS clause
  with Table Handling Feature 182
  with S-mode records 162-164
  with U-mode records 162-164
  with V-mode records 162-164
ON statement 51
operator communication
  ACCEPT statement 78
  job control commands 37
  PAUSE control statement 34
  STOP statement 78
operator intervention between job steps 34
operator messages
  ACCEPT statement 78
  STOP statement 78
OPTION control statement 33
  duration of effect 34
OPTIONAL (SELECT clause) 23
options for compiliation
  CBL card 36-37
  OPTION control statement 33-34
organization of files 107-108
  direct 109-127,107
  indexed 127-131,107-108
  sequential 109,107
origin point of phase 39
output
  compiler 67-75,17
  complete sample program 187-199
  EXHIBIT statement 51-53
  from a segmented program 100-102
  linkage editor 75-77,64,17-18
  phase execution 77-78
  system 79
  TRACE statement 51-53
OUTPUT PROCEDURE option 96
overflow area 128-129
overlay 14
  using Segmentation Feature 99-103
  using subprogram linkage 89-93
overlay logic 89
overlay structures 87-93
  job control considerations 90
  linkage editor 89-93
  PHASE statement 39
  provided by Segmentation Feature 99-103
overlayable fixed segment 99

page breaks 179
parameter list 82,85
PARSTD option 34
PAUSE control statement 34
PERFORM statement 176
permanent segment 99
phase
  definition of 11
  origin point 39
PHASE control statement 39
  generated by compiler for Segmentation 100,101,102
  using overlay 89
phase execution 18
  output 77-78
phase map 77
Physical Input/Output Control System (PIOCS) 108
PICTURE clause 167-168
PIOCS 108
pre-DTF switch 138
prefixes 165-166
prime area 127
prime numbers 113,114
printer spacing subroutine 211
priority numbers 99
private libraries 48
private relocatable library 40
problem program area 18
Procedure Division header 82
processing
  direct files 110-111
  indexed files 130
  sequential files 109
processing programs 11
Program Global Table (PGT) 73
PROGRAM-ID paragraph
  and program linkage 81
  and Segmentation 100
program switches 35-36
  Communication Region 239
programmer logical units 19,20
programming techniques 165-186
  Data Division 165-174
  Environment Division 165
  general considerations 165
  Procedure Division 174-177
  Report Writer Feature 177-181
  Table Handling Feature 181-186

QUOTE option 36

randomizing
  for the 2311 Disk Drive 125
  for the 2321 Data Cell Drive 126
randomizing techniques 112-115
  sample programs 115-119,120-124
READ INTO statement 176-177
READ statement subroutines 211
READY TRACE statement 51,177
RECORD CONTAINS clause 166
record formats 153-164
  format F 153
  format S 157-162
  format U 153-154
  format V 154-157
RECORD KEY clause 130-131
record zero (R0) 109
recording capacities of mass storage devices 107
REDEFINES clause 167
register use for linkage 84
relocatable library 42,43
  directory 42
  INCLUDE statement 39
REP statement 46
replacing source statements in a book 46
REPORT clause 177
Report Writer Feature 177-181
Report Writer routines, generation of 181
RERUN clause
  and RSTRT control statement 65,35
  and Sort Feature 97
  subroutine 214
RESET control statement 35
RESET TRACE statement 51

restarting a checkpointed program 66,35
retrieving a book from the source statement library 44,45
  BASIS card 48,49
  COPY statement 44,45
  modifying using INSERT and DELETE cards 49,50
retrieving a direct file 111
  sample job deck 241,242
retrieving an indexed file 130
  sample job deck 241,244
retrieving a program phase 41-42
REWRITE statement subroutines 211
root phase 18
  in overlay structure 87
root phase overlay 18
root segment 99,100,101
RSTRT control statement 35,66
R0 (record zero) 109

S-mode records 157-162
sample program output 187-199
save area 84
SEARCH ALL statement 183,186
SEARCH statement 185-186
  subroutine 215
segment descriptor field 157
segment limit 99,100
Segmentation Feature 99-103
  subroutine 214
segmentation subroutine 214
segments 99-101
SELECT clause
  ASSGN control statement 23
  DLBL control statement 26
  EXTENT control statement 27
  programming technique 165
  TLBL control statement 25
  VOL control statement 29
SELECT OPTIONAL clause 23
SEQ option 36
sequence-check source statements 36
sequence of job control statements 22
sequential organization 107,109
service function of Librarian 41
SET command 25,67
SET statement 183-184
7-track tape, restriction when used as sort work files 96
sign usage 168
single-program mode 12
SKIP1 67,165
SKIP2 67,165
SKIP3 67,165
slash ampersand (/&) 15
slash asterisk (/*) 15
sort diagnostic messages 96
Sort Feature 95-97
  machine requirements 237-238
  in a segmented program 101-103
sort interface subroutine 214
sort job control requirements 95-96
sort work files 96
sorting an unlabeled tape file 241,244
  sample job deck 244
SORT-RETURN 97
source statement library 43-48
  directory 43
space allocation
  EXTENT control statement 27-28
  XTENT control statement 30-31
SPACEn option 36
spacing of source program listing 36,67,165
spanned records 157-162
  on directly organized files 159-161
  on sequentially organized files 161-162
  and Sort Feature 96
special registers
  COM-REG 239
  CURRENT-DATE 25,239
  NSTD-REELS 149
  SORT-RETURN 97
spill file 234
standard file labels
  format 1 203-207
  mass storage 149-150
  tape 144,148-149,201-202
START statement 130
statement formats 21-22
STDLABEL option 34
STOP statement 78
  subroutines 212
STXIT option 36
subordinate phases 18
subscripts 163
SUM counters 177,178
SUM routines 178-179
summing techniques 177,178
Supervisor 11
SUPMAP option 36
suppressing messages
  FLAGE option 36
  NOERRS option 33
SYM option 33
symbolic names
  of input/output devices 19-20
  of phases 39
SYNCHRONIZED clause 172
synonyms 112
SYSIN 19,20
SYSIPT 19,20
SYSLNK 19,20,38
SYSLOG 19,20
SYSLST 19,20,67
SYSOUT 19,20,23
SYSPCH 19,20
SYSRDR 19,20
  on same device as SYSIPT 15,17,19
SYSRES 19,20,48
SYSRLB 19,20,48
SYSSLB 19,20,48
system logical units 19
system message identification codes 79
system-name restriction for RERUN on a sort file 97
system output 79
system service programs 11-12
SYS000 through SYS221 19,20

table element 182
Table Handling Feature 181-186
tape file labels 144-149
Task Global Table (TGT) 73

TLBL control statement 25-26
standard tape file labels 141,142
TPLAB control statement 30
standard tape file labels 140
TRACE statement 51,177
track 107
track addressing 109-127
actual 111-115,125-127
sample program 115-119
relative 111-113
sample program 120-124
track formats for direct-access storage devices 209-210
track index 128
TRANSFORM statement 177
subroutine 215
transient area 101,102,103
TRUNC option 37
truncation of COMPUTATIONAL items 37

U-mode records 152-153
undefined records 152-153
unlabeled files 151
sorting 244
unnumbered messages 235
unresolved external references 40
unsigned items 168
UPDATE function 45-48
ADD statement 46
control statement placement 47
DEL statement 46
END statement 47
invalid operand defaults 47-48
logical unit assignment 47
REP statement 46
UPDATE statement 45-46
UPDATE statement 45-46
updating a book in the source statement library 45-48
updating a direct file 111
sample job deck 242
updating an indexed file 130-131
sample job deck 244
UPSI byte 36
UPSI control statement 35
Communication Region 239
UPSI switches 239
UPSI-0 through UPSI-7 36
USAGE clause 169-171
user labels
mass storage files 150,151
tape files 144,148
user program switch indicators 239
USING option
of CALL statement 81
of ENTRY statement 82
on Procedure Division header 82
USRLABEL option 34
utility data sets
required by compiler 237
required by sort program 238,96

V-mode records 154-157
variable-length records 154-157
VOL control statement 28-29
volume labels
mass storage 149
tape 144

WITH CODE clause 179-180
work files
required by compiler 237
required by sort program 238,96
Working-Storage Section, locating in a dump 166-167
WRITE FROM statement 176-177
WRITE statement subroutines 211

XREF dictionary 74
XREF option 33
XTENT control statement 30

7-track tape, restriction when used as sort work files 96

READER'S COMMENTS

TITLE: IBM System/360 Disk Operating System
American National Standard COBOL
Programmer's Guide

FORM: GC28-6398-1

Your comments assist us in improving the usefulness of our publications; they are an important part of the input used in preparing updates to the publications. All comments and suggestions become the property of IBM.

Please do not use this form for technical questions about the system or for requests for additional publications; this only delays the response. Instead, direct your inquiries or requests to your IBM representative or to the IBM Branch Office serving your locality.

Corrections or clarifications needed:

Page *Comment*

Please include your name and address in the space below if you wish a reply.

Thank you for your cooperation. No postage necessary if mailed in the U.S.A.

cut along this line

fold fold

FIRST CLASS
PERMIT NO. 33504
NEW YORK, N.Y.

BUSINESS REPLY MAIL

NO POSTAGE NECESSARY IF MAILED IN THE UNITED STATES

POSTAGE WILL BE PAID BY . . .

IBM CORPORATION
1271 Avenue of the Americas
New York, New York 10020

Attention: PUBLICATIONS

fold fold

IBM

International Business Machines Corporation
Data Processing Division
112 East Post Road, White Plains, N.Y. 10601
(USA Only)

World Trade Corporation
821 United Nations Plaza, New York, New York 10017

Printed in U.S.A. GC28-6398-1

Printed in U.S.A. GC28-6398-1

International Business Machines Corporation
Data Processing Division
112 East Post Road, White Plains, N.Y. 10601
[USA Only]

IBM World Trade Corporation
821 United Nations Plaza, New York, New York 10017
[International]