60497200

CONTROL DATA

COBOL
VERSION 5
USER'S GUIDE

CDC® OPERATING SYSTEMS:
NOS 2
NOS/BE 1

REVISION RECORD

Revision	Description
A (04/30/76)	Original release.
B (06/01/77)	This revision reflects COBOL 5.1 (feature CP176) at PSR level 446.
C (04/11/80)	This revision reflects COBOL 5.3 at PSR level 508. Changes update documentation for the Basic Access Methods 1.5, Advanced Access Methods 2.1, and CYBER Database Control System 2.1. New sections include an interface to CYBER Record Manager, an interface to MCS 1.0, and interactive usage.
D (07/24/81)	This revision reflects COBOL 5.3 at PSR level 538. Changes include the addition of Multiple-Index Processor material and minor corrections.
E (03/11/86)	This revision reflects COBOL 5.3 at PSR level 647. Changes include the deletion of all references to operation under NOS 1 and miscellaneous technical and editorial changes.

REVISION LETTERS I, O, Q, AND X ARE NOT USED

©COPYRIGHT CONTROL DATA CORPORATION
1976, 1977, 1980, 1981, 1986
All Rights Reserved
Printed in the United States of America

Address comments concerning this manual to:

CONTROL DATA CORPORATION
Publications and Graphics Division
P. O. Box 3492
SUNNYVALE, CALIFORNIA 94088-3492

or use Comment Sheet in the back of this manual

LIST OF EFFECTIVE PAGES

New features, as well as changes, deletions, and additions to information in this manual are indicated by bars in the margins or by a dot near the page number if the entire page is affected. A bar by the page number indicates pagination rather than content has changed.

Page	Revision
Front Cover	-
Title Page	-
ii	E
iii/iv	E
v/vi	A
vii	E
viii	E
ix	D
x	C
xi	D
xii	D
xiii/xiv	D
xv	C
1-1	C
1-2	C
2-1 thru 2-4	C
2-5	E
2-6	C
3-1	E
3-2	C
3-3 thru 3-5	D
3-6	E
3-6.1	D
3-6.2	D
3-7	C
3-8	C
3-9	D
3-10 thru 3-12	C
3-13	E
3-14	C
3-15	D
3-16 thru 3-18	C
3-19	E
3-20	C
3-21	E
3-22	C
3-23	C
3-24 thru 3-27	D
3-28 thru 3-38	C
3-39	E
3-40 thru 3-42	C
3-43	E
3-44 thru 3-47	C
3-48	E
3-49 thru 3-51	C
3-52	E
3-53 thru 3-56	C
4-1 thru 4-4	C
4-5	E
4-6 thru 4-8	C
4-9	E
5-1 thru 5-6	C
5-7 thru 5-9	E
5-10 thru 5-16	C
6-1	C
6-2	D
6-3 thru 6-8	C
6-9	E
6-10	C
6-11	C
7-1 thru 7-6	C
8-1 thru 8-9	C
9-1	C
9-2	C
10-1	E
10-2	C
10-3	E
10-4	E
10-5	D
10-6 thru 10-9	C
11-1	C
11-2	D
11-3	D
11-4	E
11-4.1/11-4.2	D
11-5	E
11-6	C
11-7	C
11-8	E
11-9	C
11-10	E
11-11 thru 11-13	C
11-14	E
11-15	E
12-1 thru 12-3	C
13-1	E
13-2	E
13-3 thru 13-6	C
14-1 thru 14-6	E
14-6.1/14-6.2	D
14-7	E
14-8 thru 14-10	C
15-1 thru 15-3	D
15-4	E
15-5 thru 15-15	D
15-16	E
15-17	D
16-1 thru 16-3	E
16-4	C
16-5	E
16-6	E
16-7	C
16-8	E
16-9	C
16-10	C
16-11	E
16-12	E
17-1 thru 17-8	C
A-1	E
A-2	D
A-3	C
A-4	C
A-5	E
A-6	C
A-7	E
A-8	C
B-1	C
B-2	E
B-3	D
B-4	D
B-5 thru 5-9	E
C-1	C
C-2	E
C-3	C
D-1 thru D-4	C
Index-1	D
Index-2	E
Index-3	E
Index-4	D
Index-5	D
Index-6	E
Comment Sheet/Mailer	E
Back Cover	-

ACKNOWLEDGEMENT

The following acknowledgement is reproduced in its entirety at the request of the American National Standards Institute.

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

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

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

The authors and copyright holders of the copyrighted material used herein

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

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

PREFACE

This guide describes the usage of the COBOL Version 5.3 language. As described in this publication, COBOL 5 operates under control of the following operating systems:

NOS 2 for the CONTROL DATA® CYBER 180 Computer Systems; CYBER 170 Series; CYBER 70 Models 71, 72, 73, 74; and 6000 Series Computer Systems.

NOS/BE 1 for the CDC® CYBER 180 Computer Systems; CYBER 170 Series; CYBER 70 Models 71, 72, 73, 74; and 6000 Series Computer Systems.

COBOL 5 is designed to be a superset of the language specified in American National Standard X3.23-1974, COBOL. In this guide, no distinction is made between the standard language and Control Data extensions to the language.

This guide is written for a programmer familiar with the COBOL language and with the operating system under which the COBOL 5 compiler is operating. The language is presented in relation to specific features of COBOL 5. The formats of statements and clauses are illustrated by examples rather than by format specifications.

Detailed information can be found in the listed publications. The publications are listed alphabetically within groupings that indicate relative importance to readers of this manual.

The NOS System Information Manual is an online manual that includes brief descriptions of all NOS and NOS product manuals. To access this manual, log in to NOS and enter the command EXPLAIN.

The following publications are of primary interest:

Publication	Publication Number	NOS 2	NOS/BE 1
COBOL Version 5 Diagnostic Handbook	60482500	X	X
COBOL Version 5 Reference Manual	60497100	X	
COBOL Version 5 Reference Manual online	L60497100	X	
COBOL Version 5 Report Writer's User Guide	60496900	X	X
NOS Version 2 Reference Set, Volume 3, System Commands	60459680	X	
NOS/BE Version 1 Reference Manual	60493800		X

The following publications are of secondary interest:

Publication	Publication Number	NOS 2	NOS/BE 1
CYBER Record Manager Advanced Access Methods Version 2 Reference Manual	60499300	X	X
CYBER Record Manager Basic Access Methods Version 1.5 Reference Manual	60495700	X	X
DMS-170 CYBER Database Control System Version 2 Application Programming Reference Manual	60485300	X	X
DMS-170 DDL Version 3 Reference Manual Volume 2: Subschema Definition for CYBER Database Control System Use With: COBOL QUERY UPDATE	60482000		X

DMS-170 FORM Version 1 Reference Manual	60496200	X	X
Message Control System Version 1 Reference Manual	60480300	X	
NOS Full Screen Editor User's Guide	60460420	X	
Update Version 1 Reference Manual	60449900	X	X

Sites within the United States can order CDC manuals from Control Data Corporation, Literature and Distribution Services, 308 North Dale Street, St. Paul, Minnesota 55103.

Other sites can order CDC manuals by contacting the local country sales office.

This manual describes a subset of the features and parameters documented in the COBOL 5 reference manual. Control Data cannot be responsible for the proper functioning of any features or parameters not documented in the COBOL 5 reference manual.

CONTENTS

APPENDIXES

INDEX

FIGURES

TABLES

NOTATIONS

—	Underlining in examples indicates terminal user input.
A B C	Boxes in examples indicate character position in storage. An empty box means an unpredictable result.
. . .	Ellipses in examples indicate missing text.

Numerals are represented in decimal unless indicated otherwise.

The COBOL 5 language is a high-level programming language that is problem oriented rather than machine oriented. The programmer can thus concentrate on the logic of the problem. The COBOL 5 language consists of ordinary English words and arithmetic symbols that are used in an ordered manner to define data and procedures. Although the language is relatively unrestricted in its simulation of English, it is governed by rules that enable the COBOL 5 compiler to translate a COBOL source program into an object program intelligible to the computer.

COBOL 5 FEATURES

The COBOL 5 language provides a wide range of features. In addition to implementing the 1974 ANSI COBOL standard (X3.23-1974), a powerful set of Control Data extensions is provided in COBOL 5. The features described in this guide are summarized in the following paragraphs.

Six file organizations are available in COBOL 5: sequential, relative, indexed, direct, actual-key, and word-address. These organizations provide efficient processing for a wide range of applications. File access can be sequential, random, or dynamic; dynamic access allows records to be accessed both sequentially and randomly. Records in all file organizations except sequential organization have an associated primary key that is used for random access. Input/output statements are provided to read, write, rewrite, and delete records in a file and to position a file for subsequent processing. The open mode established when the file is opened determines which input/output statements can be executed for the file.

Indexed, direct, and actual-key file organizations can be installed as either initial or extended. Extended Advanced Access Methods (AAM) files are more efficient and are the default for COBOL programs. Any further discussion of AAM files in this manual implies extended AAM file organization.

Indexed, direct, and actual-key file organizations allow records to be accessed by a choice of keys. In these file organizations, alternate keys can be defined in addition to the primary key. When alternate keys are specified for a file, any key (primary or alternate) can be used to access records in the file.

A complete set of arithmetic statements provides the means to perform operations involving addition, subtraction, multiplication, and division. Each type of operation is accomplished by an individual statement: ADD, SUBTRACT, MULTIPLY, or DIVIDE. A series of different arithmetic operations can be accomplished by the COMPUTE statement; exponentiation, unary plus, and unary minus can also be specified. Results of an arithmetic operation can be rounded. Size error detection on a result is also provided.

Boolean operations allow two operands to be compared bit by bit for equality or inequality. Operations are accomplished by statements that include the reserved words BOOLEAN-AND, BOOLEAN-OR, BOOLEAN-EXOR or BOOLEAN-NOT. The COMPUTE statement can be used to assign values to boolean variables.

Conditional operations in COBOL 5 provide the means to specify an alternate path of control that is followed only when designated conditions are true. This decision making capability allows the program to specify that certain procedures or statements are executed under specified conditions. The IF, PERFORM, and SEARCH statements can specify an explicit condition that is tested each time the statement is executed; the next statement or procedure executed depends on the truth of the condition. Implicit conditions are specified through the AT END, INVALID KEY, ON SIZE ERROR, ON OVERFLOW, and AT END-OF-PAGE options that can be included in arithmetic and input/output statements. An implicit or explicit condition must be true before the alternate path of control is followed.

Delimited scope statements allow the COBOL user to write structured programs more easily. Explicit scope terminators END-IF, END-SEARCH, and END-PERFORM, are used to explicitly terminate the scope of IF, SEARCH, and PERFORM statements, respectively. A capability similar to the FORTRAN do-loop is possible in COBOL with a PERFORM VARYING statement followed by in-line imperative statements, and terminated by END-PERFORM. The inclusion of END-IF with IF or END-SEARCH with SEARCH allows the IF and SEARCH statements to be used anywhere an imperative statement can be used.

Tables of fixed or variable length can be specified in a COBOL 5 program. A table can be described with up to 48 levels of OCCURS clauses. Table elements can be referenced by subscripting or indexing. Indexing and subscripting can be mixed. The SET statement can be used to manipulate indexes. The SEARCH statement is used to search a table for a specific element.

Records in sequentially organized files are sorted or merged automatically by the SORT or MERGE statement. Input and output procedures can be defined or the input and output files can be named. One or more data items are used as keys for the sort or merge operation. The collating sequence for the sort or merge operation can be explicitly specified by the SET statement. The SORT statement causes records from one or more files to be sorted by the specified key data items. The MERGE statement is used to combine two or more identically sequenced files.

A COBOL 5 program can be segmented to reduce memory requirements during program execution. In a segmented program, the entire Procedure Division is written in sections. Each section is assigned a number that designates the segment to which the section belongs. A segment is either fixed or overlayable. Fixed segments are in memory at all times during execution; overlayable segments are made available in memory when they are needed.

Independently compiled subprograms can be accessed by a COBOL 5 program. Subprograms can be written in COBOL, COMPASS, and FORTRAN Extended. The CALL statement is used to access a COBOL subprogram; other subprograms are accessed by the ENTER statement. Data can be passed between the main program and the subprogram by specifying a parameter list in the ENTER or CALL statement; the Common-Storage Section can also be

used for passing data. Fast Dynamic Loader processing allows COBOL subprograms to be dynamically called and canceled.

Portions of a COBOL 5 program can be copied from a COBOL source library. The COPY statement can be specified anywhere in the source program. During compilation, the specified library deck is incorporated into the program and replaces the COPY statement. Modifications to the library deck can be specified in the REPLACING phrase of the COPY statement.

Debugging procedures can be specified in the COBOL 5 program to monitor files, data items, or procedures during program execution. A debugging section is included in the Declaratives portion of the Procedure Division. A USE FOR DEBUGGING statement specifies the files, data items, or procedures for which the debugging section is executed. A special register, DEBUG-ITEM, provides information related to the condition that caused the debugging section to execute. The paragraph trace feature can be used to trace the flow of the program during execution. The termination dump feature can be used to obtain a formatted map of the contents of all data items within the program.

Data base files can be processed by COBOL 5 programs through an interface with the CYBER Database Control System (CDCS). All references to CDCS in this manual refer to CDCS 2 only. The files are described by a sub-schema instead of by File Description entries in the program. The files can be locked and unlocked by the ENTER statement. Access control keys, as well as recovery points, can be specified within the program. Input/output operations are performed using standard COBOL 5 statements. In addition to reading an individual file, a read operation can retrieve records from several files joined together in a logical relationship.

CYBER Record Manager (CRM) interfaces to COBOL programs are generally transparent to the user. A file information table (FIT) exists for each file and contains descriptions of the file, such as file organization, blocking structure, and record type. The FIT is the most important element used for communication between COBOL and CRM. The COBOL compiler uses the source statements to set many FIT values. Other values are set as defaults. The USE clause can set or override certain FIT fields that might also be set or overridden by a FILE control card.

A COBOL 5 program can be written, compiled, and executed interactively through the terminal. The ACCEPT statement provides the means to input data to the executing program from the terminal. Similarly, the DISPLAY statement can be used to output data from the program to the terminal.

The Message Control System (MCS), the Network Access Method (NAM), and the COBOL Communication Facility (CCF) together allow a COBOL program to communicate with terminals. Messages are routed to and from terminals with the SEND and RECEIVE statements.

COBOL 5 JOB PROCESSING

Creating a COBOL 5 program can be considered as a three-step procedure. The first step is writing the source program. The next step is compiling the source program into executable code. The last step involves executing the program and determining that no errors exist in the logic of the program.

The COBOL 5 program is written according to the specifications of the language. The coded program is then punched on 80-column cards or entered through a terminal as card images. The resulting source deck is input to the COBOL 5 compiler.

Before the source program can be input to the COBOL 5 compiler, a set of control statements must be prepared to precede the source program. This set includes the job statement, the COBOL5 control statement, the program call control statement for execution, permanent file control statements when permanent files are involved during execution, and any other control statements required by the operating system or by a particular installation. The set of control statements must be terminated by a 7/8/9 card or its equivalent. The source program can follow the control statements.

The source program is input to the COBOL 5 compiler for translation into an object program containing executable code. The compiler checks the program and reports any errors in the diagnostics listing. The object program can be punched on cards or written on a disk file for subsequent execution of the program.

Execution of the COBOL 5 object program can be included in the compilation run or it can be a separate job. When execution immediately follows compilation, the object program usually is written on the system file LGO and the input data must follow the source program in the input file. Once the program has been completely debugged, the object program is usually punched on cards or stored on disk by making it a permanent file or part of a permanent file library in either absolute or relocatable form. Job processing then consists of executing the object program and providing the input data on the input files specified by the program.

A COBOL 5 program defines the data to be used and specifies the manner in which the data is manipulated in order to produce the desired results. The program is organized according to a predefined structure. Various components of the language are used in structuring the program.

PROGRAM ORGANIZATION

A COBOL 5 program consists of a series of entries that are organized into divisions, sections, and paragraphs. An entry contains one or more language elements and is terminated by a period. A COBOL 5 program has four divisions; each division contains a specific type of information. Within three of the divisions, the information can be further organized into sections that contain a series of related paragraphs.

IDENTIFICATION DIVISION

The Identification Division is the first division in a COBOL 5 program. It identifies the program by assigning a program name. In addition, the Identification Division can document the author's name, the date the program was written, the date it was compiled, the installation's identification, and a security entry.

Sections are not used in the Identification Division. Each type of information is presented in a paragraph that begins with a predefined name. Only the PROGRAM-ID paragraph is required.

ENVIRONMENT DIVISION

The Environment Division documents the equipment to be used to compile and execute the COBOL 5 source program and assigns each data file in the program to a specific file. It can also include other information related to input and output and the assignment of special names. The Environment Division is organized into two sections: the Configuration Section and the Input-Output Section.

The Configuration Section documents the source and object computers, which are the computers used, respectively, to compile and to execute the source program. The debugging feature can be activated for compilation and the collating sequence to be used during execution of the program can be specified. The SPECIAL-NAMES paragraph provides the means to assign user-defined names to implementor-names recognized by the compiler, to designate the name by which a specific character code set is recognized, and to specify the sub-schema for accessing data base files. In addition, this paragraph can specify an alternate character for the currency sign, the decimal point, and the quotation mark; the default position of the operational sign for signed numeric display data items can also be specified.

The Input-Output Section contains two paragraphs. The FILE-CONTROL paragraph names each file used in the source program, assigns the file to a system file-name, and specifies other file related information such as file organization, access mode, and key fields. The I-O-CONTROL paragraph specifies the points at which rerun is to be established, whether memory area is to be shared by different files, and the location of files on a multiple-file reel.

DATA DIVISION

The Data Division defines all data that is processed by the object program. Each data item referenced in the Procedure Division is described in one of the seven possible sections in the Data Division.

The File Section describes the data items within each file processed by the object program. The Common-Storage Section describes data items that are shared between the main program and an independently compiled subprogram. Data that is developed internally during operation of the program is described in the Working-Storage Section. The Secondary-Storage Section describes data to be stored in extended memory. Data to be passed to a COBOL subprogram (through the CALL statement in the main program) is described in the Linkage Section of the COBOL subprogram. When the Report Writer capability is used, the output report is described in the Report Section. The Communication Section is used when a COBOL object program communicates, through the Message Control System (MCS), with local or remote communication devices such as terminals.

Only those sections that are applicable need be specified in the source program. Each item defined in the Data Division contains (or will contain as a result of processing) data used by the program.

The Data Division does not have paragraphs. Each data item in a section is described completely in a Data Description entry. This entry describes a data item in terms of size and class.

PROCEDURE DIVISION

The Procedure Division specifies the manner in which data is manipulated. Statements in this division perform input/output operations, arithmetic processing, program control, and data movement. In addition, files can be sorted, tables can be searched, and reports can be written through the Report Writer capability. Terminal messages can be received from buffer areas called input queues and placed in areas called output queues through the COBOL Communication Facility (CCF). Refer to section 17.

Statements in the Procedure Division are combined into sentences; sentences are organized into paragraphs. One or more paragraphs can be designated as a section. If sections are to be used, the entire Procedure Division must be organized into sections. Section-names and paragraph-names in this division are user-defined.

Declaratives can be included at the beginning of the Procedure Division to specify procedures that are automatically executed at the appropriate time. Declarative procedures can be used with error checking, debugging, report writing, and key manipulation. Each procedure is contained in a section that begins with a USE statement.

LANGUAGE ELEMENTS

The COBOL 5 language is composed of various elements that are combined to form entries in the source program. The use of these elements is governed by specific rules. Source program entries consist of reserved words, user-defined words, literals, and punctuation. In the Data Division, entries also include level numbers and picture-specifications. The following paragraphs briefly describe these COBOL 5 program elements. Detailed descriptions are contained in the COBOL 5 reference manual.

RESERVED WORDS

Reserved words are English words and abbreviations that have special meanings to the COBOL 5 compiler. These words can be used only as shown in the format specifications and must be spelled correctly. Reserved words are divided into five categories:

Keywords

Words that are required in the format specifications. A keyword conveys a special meaning to the COBOL 5 compiler and is necessary to correctly compile the entry or statement.

Optional Words

Words that can be included in the format specification to improve readability. An optional word is not needed to compile the entry or statement.

Connectives

Words used to associate a name with its qualifier (OF and IN) or to logically join conditions (AND and OR).

Special Registers

Words that identify compiler-generated data related to specific COBOL 5 features. Five special registers are available: LINE-COUNTER, PAGE-COUNTER, LINAGE-COUNTER, DEBUG-ITEM, and HASHED-VALUE.

Figurative Constants

Words that represent fixed values. Six different figurative constants are available: ZERO, SPACE, HIGH-VALUE, LOW-VALUE, QUOTE, and ALL. ALL can precede any of the other figurative constants or their plural equivalents, or it can be followed by a nonnumeric literal.

USER-DEFINED WORDS

Many of the format specifications include words that are supplied by the user. Various types of names (such as data-names, paragraph-names, section-names, and file-names) are defined by the user.

A user-defined name can be up to 30 characters in length. Only the characters A through Z, 0 through 9, and the hyphen can be used; the hyphen cannot be the first or the last character of a user-defined word. Level numbers and segment numbers must be numeric. Paragraph-names and section-names can be entirely numeric. All other user-defined words must contain at least one alphabetic character. A user-defined word cannot be spelled exactly the same as a reserved word.

LITERALS

In some of the format specifications, literals are supplied by the user. A literal is a string of characters that represents a specific value. Literals are either numeric or nonnumeric.

Numeric Literals

A numeric literal contains a combination of the digits 0 through 9, the decimal point, and the plus sign or minus sign. The decimal point can be in any character position except the rightmost position. If the plus sign or the minus sign is included in the numeric literal, it must be the leftmost character. A numeric literal can contain up to 18 digits.

Floating point numeric literals can also be specified in a COBOL 5 program. These literals can be used only as follows:

In the Data Description entry of an elementary COMPUTATIONAL-2 data item.

In a Procedure Division statement that allows a noninteger numeric literal.

A floating point numeric literal consists of a mantissa, the letter E, and an exponent; a plus sign or minus sign can be included in the mantissa and in the exponent. The mantissa can contain up to 14 digits and must include a decimal point. The largest value that can be specified as the exponent is +308 or -279. If a sign character is specified, it must be the leftmost character in the mantissa or exponent.

Nonnumeric Literals

A nonnumeric literal is a string of up to 255 characters. The string of characters must be enclosed in quotation marks. Any character in the character set, including the space, can be used in a nonnumeric literal. A quotation mark can be included in the literal by specifying the quotation mark twice for each occurrence. For example, "PROGRAM ""ONE"" REPORT" would yield the literal PROGRAM "ONE" REPORT.

When the QUOTE IS APOSTROPHE clause is specified in the Environment Division or the APO parameter is included in the COBOL5 control statement, the apostrophe character is used to delimit nonnumeric literals.

PUNCTUATION

Most punctuation marks in a COBOL 5 program are optional. In some instances, punctuation is essential to program compilation and the rules must be followed exactly.

Commas and semicolons are included or omitted at the user's option and have no effect on program compilation. A period is required to terminate each of the following elements in a COBOL 5 program:

- Division header
- Section header
- Paragraph name
- Complete paragraph
- Environment or Data Division entry
- Procedure Division sentence

A period, comma, or semicolon must be followed by at least one space.

A colon must be used in reference modified items, as described in section 7.

Parentheses are used to delimit subscripts, indexes, arithmetic expressions, reference modifiers, and conditions. Parentheses must be specified in balanced pairs of left and right parentheses.

Quotation marks are used to enclose nonnumeric literals and must be specified in balanced pairs except when the literal is continued on more than one line (refer to COBOL 5 Coding Conventions in this section). The opening quotation mark cannot be followed by a space, and the closing quotation mark cannot be preceded by a space, unless the space is considered part of the nonnumeric literal.

Periods, commas, and parentheses can appear in picture-specifications and as such characters are not considered punctuation marks.

LEVEL NUMBERS

Level numbers can be used in the Data Division to designate the hierarchical structure of the data items being defined. Level numbers 01 through 49 are used to define the structure of a record or a report. Level numbers 66, 77, and 88 are special level numbers that do not designate a hierarchical position.

Record Level Numbers

The hierarchy of data items within a record is defined with level numbers 02 through 49. Level number 01 identifies the record and is used in the entry that specifies the record-name. The organization of the data items within the record is indicated by the level numbers. Elementary data items are assigned higher level numbers than the level number of the group item to which they belong. Level numbers need not be consecutive. Level number 01 can also be used to define an independent data item or the highest element in a group item that is not part of a file description.

Special Level Numbers

Three level numbers have been provided for particular types of entries; these level numbers do not define the hierarchy of data items:

Level Number 66

Used to rename a data item. Level number 66 can be used in any section of the Data Division, except the Secondary-Storage and Report Sections, to rename one or more elementary items or group items.

Level Number 77

Used in the Working-Storage, Common-Storage, and Linkage Sections to define independent data items. Level number 77 is used to define elementary items that are not a part of any record.

Level Number 88

Used to assign one or more values to a condition-name. Level number 88 can appear in any section in the Data Division except the Secondary-Storage and Report Sections.

Report Level Numbers

In the Report Section of the Data Division, level numbers are used to identify group and elementary items. A report group item is assigned level number 01 and describes one type of report line (heading, detail, or footing). Subordinate group and elementary items are assigned level numbers 02 through 49; these items further describe the characteristics of the report group item. Refer to the Report Writer user's guide.

PICTURE-SPECIFICATIONS

Picture-specifications are used in the Data Division to describe the characteristics of a data item and to specify editing requirements for a data item. A picture-specification can be associated only with an elementary data item and is specified by the PICTURE clause. A complete description of the use of the PICTURE clause is contained in the COBOL 5 reference manual. The following paragraphs summarize picture-specifications.

The type of characters used in the picture-specification determines the data category of the data item. Each data item belongs to one of six categories: alphabetic, numeric, boolean, alphanumeric, alphanumeric-edited, or numeric-edited. Table 2-1 lists each character that can be used in a picture-specification and specifies the character representation and the data categories for which it can be used.

The picture-specification can contain up to 30 characters; however, the size of the data item being described can be more than 30 characters. Consecutively repeated characters in the picture-specification can be abbreviated by specifying the character followed by a number enclosed in parentheses. For example, A(20) is equivalent to the character A repeated 20 times.

TABLE 2-1. PICTURE-SPECIFICATION CHARACTERS

Character	Representation	Data Category
A	Alphabetic character (including space)	Alphabetic, alphanumeric, or alphanumeric-edited
B	Blank (space) insertion	Alphabetic, alphanumeric-edited, or numeric-edited
9	Numeric character	Numeric, alphanumeric, alphanumeric-edited, or numeric-edited
P	Assumed decimal scaling position	Numeric or numeric-edited
S	Operational sign	Numeric
V	Assumed decimal point	Numeric or numeric-edited
X	Alphanumeric character	Alphanumeric or alphanumeric-edited
Z	Zero suppression	Numeric-edited
0	Zero insertion	Alphanumeric-edited or numeric-edited
1	Boolean character 0 or 1	Boolean
/	Slash insertion	Alphanumeric-edited or numeric-edited
,†	Comma insertion	Numeric-edited
.†	Decimal point insertion	Numeric-edited
CR	CR insertion for negative value	Numeric-edited
DB	DB insertion for negative value	Numeric-edited
+	Plus sign insertion	Numeric-edited
-	Minus sign insertion	Numeric-edited
*	Asterisk insertion	Numeric-edited
$††	Currency sign insertion	Numeric-edited

†If the DECIMAL-POINT IS COMMA clause is specified, the representations of the comma and decimal point characters are exchanged.

††The character # or the character specified in the CURRENCY SIGN clause can be used in place of the character $ in the picture-specification.

The following rules apply to the picture-specification for each data category:

Alphabetic

Only the characters A and B can be used.

Numeric

Only the characters 9, P, S, and V can be used.

Up to 18 digit positions can be described.

Each of the characters S and V can be used only once.

Boolean

Only the character 1 can be used.

Alphanumeric

Only the characters A, X, and 9 can be used.

At least one X, or at least one A and one 9, must be specified.

Alphanumeric-Edited

Only the characters A, X, 9, B, 0, and / can be used.

At least one X and one B, 0, or / must be specified, or at least one A and one 0 or / must be specified.

Numeric-Edited

Only the characters B, /, P, V, Z, 0, 9, comma (,), decimal point (.), *, +, -, CR, DB, and the currency symbol can be used.

Up to 18 digit positions can be described.

At least one character other than P, V, or 9 must be specified.

Each of the characters V, decimal point, CR, and DB can be used only once; CR and DB cannot be used in the same picture-specification.

COBOL 5 CODING CONVENTIONS

The COBOL 5 source program is written on COBOL coding forms that correspond to 80-column punched card format. The coding form is divided into five reference areas:

Reference Area	Columns
Sequence Number Area	1-6
Indicator Area	7
Area A	8-11
Area B	12-72
Program Identification Area	73-80

SOURCE PROGRAM ENTRIES

A source program entry is written according to the applicable format specification. Entries on the coding form must conform to the following rules:

- Division headers and the keywords DECLARATIVES and END DECLARATIVES must begin in Area A; the remainder of the line following the terminating period must be blank.
- Section headers must begin in Area A; only a COPY or USE sentence can follow a section header on the same line.
- Paragraph names must begin in Area A; at least one space must follow the terminating period.
- Sentences must be written in Area B; a sentence can begin on a new line or follow a preceding sentence separated by at least one space.
- Level indicators FD, SD, and RD and level numbers 01 and 77 must begin in Area A and must be followed by at least one space.
- Level numbers 02 through 49, level number 66, and level number 88 must begin in Area B and must be followed by at least one space.
- Level numbers 01 through 09 can be written as a single digit or can be preceded by a zero.

SEQUENCE NUMBERS

If the program sequence (PSQ) parameter is specified in the COBOL5 control statement, each line of the source program must have a numeric sequence number not greater than 65535, not equal to zero, and not consisting of all spaces. Diagnostics issued at compile time and at execution time then reference the applicable sequence numbers. Sequence numbers do not have to be in ascending order; however, the numbers should be in order to facilitate locating source lines referenced in diagnostics. If a sequence number contains any character other than the digits 0 through 9 and a space, a diagnostic is issued and the last valid sequence number is used.

Processing of sequence numbers depends on the PSQ parameter in the COBOL5 control statement. When the PSQ parameter is omitted, sequence numbers are optional and can include any character in the computer character set. The compiler does not perform any checking on the sequence number.

When a program has been created with line sequence numbers through a NOS interactive text editor facility (EDITOR or XEDIT), or the NOS Full Screen Editor (FSE), the PSQ parameter causes those sequence numbers to be used for diagnostic message references. Refer to section 16.

CONTINUATION LINES

A source program entry can be written on more than one line. Continuation lines must begin in Area B. When a word or a literal is continued from one line to the next, a hyphen must be entered in the Indicator Area, and the continuation is processed as follows:

- For a continued word or numeric literal, the first nonblank character in Area B of the continuation line is assumed to immediately follow the last nonblank character of the preceding line.
- For a nonnumeric literal, the first nonblank character in Area B of the continuation line must be a quotation mark; the first character following the quotation mark is assumed to immediately follow the character in column 72 of the previous line. All spaces at the end of the continued line are considered part of the nonnumeric literal.

A continuation line that does not contain a hyphen in the Indicator Area assumes that a space follows the last nonblank character in the preceding line.

COMMENT LINES

Comment lines can appear anywhere in the source program after the Identification Division header. A comment line is designated by entering an asterisk or a slash in the Indicator Area. An asterisk causes the line to be printed in the source program listing immediately following the preceding line. A slash causes page ejection before the line is printed. All characters in Area A and Area B are considered to be a comment line and are printed on the output listing.

PROGRAM TEXT REPLACEMENT

Source program text can be replaced anywhere in the COBOL 5 source program by using the REPLACE statement. Two contiguous equal signs are used to delimit pseudo-test.

```
REPLACE ==TEST== BY ==TEST-AMT==.
```

This statement replaces the characters TEST with the characters TEST-AMT, from the point at which the statement is used, until either the end of the program or until REPLACE OFF is encountered. TEST appears in the source listing but the COBOL compiler uses TEST-AMT instead. New reserved words can be replaced in this manner to avoid diagnostics.

All REPLACE statements are processed by the compiler after all COPY statements (see section 12) have been processed; the program is then checked for syntactical correctness.

Most data items used during execution of a COBOL 5 program are contained in files. The structure of a file as specified in the source program determines the type of device on which the file can reside, the organization of records within the file, and the method used to input and output records in the file.

COBOL 5 files can reside on magnetic tape or on mass storage devices; card and line printer files are mass storage files. Records are positioned in a file sequentially or according to a specified key. Depending on the file organization, the access mode can be sequential, random, or dynamic. Dynamic access allows records to be accessed sequentially as well as randomly during program execution.

Six different file organizations are available for COBOL 5 files: sequential, relative, indexed, direct, actual-key, and word-address. Records in all file organizations except sequential are stored according to a primary key value. The description of the file and the format of the statements used to manipulate the file depend on the file organization selected. File organization is established when the file is created and remains the same as long as the file exists.

Indexed, direct, and actual-key file organizations can be of two types: extended or initial. Extended file organizations are more efficient and are the COBOL default. All references to indexed, direct, and actual-key files in this section imply extended file organization unless otherwise stated.

Alternate keys can be defined for a file with indexed, direct, or actual-key organization. Alternate key processing allows records to be accessed by any one of several key fields. An index of the alternate keys is automatically created and maintained on an alternate key index file that is separate from the data file. Alternate keys can be included in the index file or omitted from it, depending on conditions specified when the keys are defined.

A file can be declared an External file. This allows the file to be shared by programs in the same run unit. External files are discussed in section 10, Subprogram Interface.

Data base files, which are accessed through the CYBER Database Control System (CDCS), can be processed by a COBOL 5 program. The files are described in a subschema instead of in the COBOL 5 program. Data base file processing is discussed in section 14, CDCS Interface.

Errors and exception conditions encountered during file processing can be handled in several different ways. Special procedures can be specified in the COBOL 5 program; other procedures are performed automatically by the system.

File processing is described in this section for each of the six file organizations. The definition of a file and the use of the applicable input/output statements are discussed separately for each file organization. The user of this section can refer to a specific file organization for information related to file definition and file manipulation. More detailed information on the interface with CYBER Record Manager is in section 15.

ALTERNATE KEY PROCESSING

File processing can be greatly enhanced by the use of alternate keys. This capability is provided for files with indexed, direct, and actual-key organizations. Alternate keys allow records in the file to be accessed by various keys. As many as 255 alternate keys can be defined for a file.

CREATING ALTERNATE KEYS

Alternate keys are defined when the file is being created. An ALTERNATE RECORD KEY clause is included in the FILE-CONTROL paragraph for each alternate key. Alternate key fields are described in a Record Description entry for the file. Alternate key fields can overlap and can differ in length. An alternate key field can begin in the same location as the primary key field or any other alternate key field; however, overlapping keys must not be the same length. The location and description of alternate key fields must remain the same for the life of the file.

A data item described with the OCCURS clause can be an alternate key field. This type of field is called a repeating group, whether or not the data item is a group item. Specifying a repeating group as an alternate key field allows a record to have more than one value for the alternate key. Each unique occurrence of the alternate key provides a value by which the record can be accessed.

When the data file is being created, index entries are automatically generated by Advanced Access Methods (AAM) for each alternate key. An alternate key value is included or excluded from the alternate key index file depending on conditions specified by the USE or OMITTED phrase in the ALTERNATE RECORD KEY clause. Specifying one of these phrases allows an alternate key value to be included in the index file or omitted from it on the basis of a code value contained in the record. Keys that have some values of little or no interest (sparse keys) can be ignored.

The index file is specified in the ASSIGN clause for the data file. The appearance of two file names in the ASSIGN clause indicates an alternate key file, as shown in the following statement:

```
SELECT CUSTOMERS ASSIGN TO CSTMRS, CSTINDX.
```

Whenever the data file CSTMRS is updated, the index file CSTINDX is automatically updated by AAM. The index file is a mass storage permanent file that must be preserved between jobs. It must be made available to a job that updates the data file or reads the data file by alternate key.

The index for an alternate key contains an entry for each alternate key value encountered as records are written on the data file. The entries for the alternate key are maintained in sorted order by AAM. Each alternate key entry contains the primary key values associated with that alternate key value.

When the alternate key is a repeating group, the primary key value for a record is associated with each occurrence of the alternate key data item. If duplicate alternate key values are not allowed, only one primary key value is associated with an alternate key value. The structure of the index file is illustrated in figure 3-1.

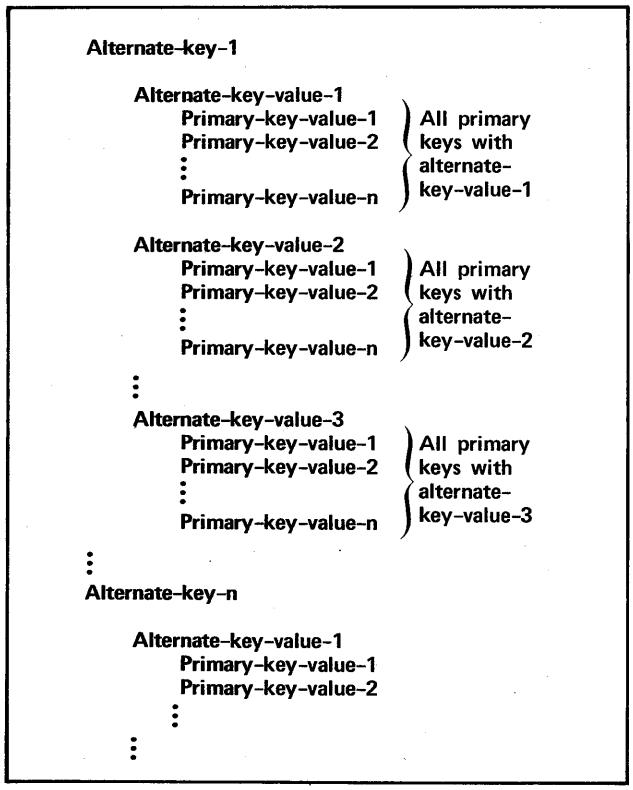


Figure 3-1. Structure of the Alternate Key Index File

The order of primary key values in an alternate key entry depends on whether or not the ASCENDING option is included in the DUPLICATES phrase of the ALTERNATE RECORD KEY clause. If ASCENDING is specified, the primary key values are maintained in ascending sequence (indexed). If ASCENDING is omitted, the primary key values are maintained in the order in which the records are written (FIFO).

ESTABLISHING THE KEY OF REFERENCE

For indexed, direct, and actual-key files, records are accessed according to the current value of the key of reference. The key of reference can be the primary key or an alternate key. An alternate key is established as the key of reference by executing either a START statement or a random READ statement. The key of reference remains the same until another START statement, a random READ statement, or an OPEN statement is executed.

The START statement is used to position the file to a record that satisfies a specific condition. The file can then be processed sequentially from that position. An alternate key is established as the key of reference by specifying either an alternate key or the leading portion of an alternate key in the START statement. The alternate key index is searched for the alternate key value that satisfies the specified condition. Sequential processing then retrieves records in order by alternate key value. This allows records to be accessed in alternate key sequence beginning with any desired alternate key value. For example, an employee file with the date hired field defined as an alternate key can be positioned such that the records subsequently accessed are those for employees hired after a certain date; the records retrieved are in sequence by alternate key value. The START statement is described in more detail in the paragraphs related to the specific file organizations.

A random READ statement can be used to establish an alternate key as the key of reference. The KEY IS phrase of the READ statement specifies the alternate key. When the statement is executed, the alternate key index is searched for a value equal to the current value of the key of reference (the alternate key data item); the record retrieved is the record with the first primary key associated with the alternate key value. Sequential READ statements can then be executed to access records with the same alternate key value or in sequence by alternate key value.

ACCESSING BY ALTERNATE KEY

Records can be read sequentially or randomly by alternate key. The key of reference is established as an alternate key before the record is accessed. When duplicate alternate key values are allowed, the specific record retrieved depends on the order of primary key values in the alternate key index. The order is determined by the ASCENDING option.

When the ASCENDING option is not specified for duplicate alternate keys, records with duplicate alternate key values are retrieved in the same chronological order they were written on the file (first in, first out). When the ASCENDING option is specified, records with duplicate alternate key values are retrieved in ascending sequence of primary key values. Access by alternate key is considerably more efficient when primary key values are in ascending sequence; therefore, the ASCENDING option should be specified in the DUPLICATES phrase unless chronological sequence is required for the application.

When the USE phrase or OMITTED phrase is specified for an alternate key, only those records with keys satisfying the condition specified in the phrase are retrieved. For example, when the following clauses are specified, key values equal to zero are not included in the index.

```
ALTERNATE RECORD KEY IS DAYS-DQ
    OMITTED WHEN DAYS-DQ IS ZERO.
```

Similarly, particular values that are rare or of no concern (sparse keys) can be excluded as illustrated in the following clauses:

```
ALTERNATE RECORD KEY IS EVENODD
    OMITTED WHEN EVENODD CONTAINS
    CHARACTER FROM "13579".
```

In this example, records are not to be indexed when the EVENODD value is 1, 3, 5, 7, or 9.

Random access by alternate key retrieves the first record (chronologically or sequentially) for the alternate key value. Additional records with the same alternate key value can then be accessed by executing sequential READ statements. When all records with the same alternate key value have been read, the next sequential READ statement retrieves the first record for the next alternate key value in the index file. The final occurrence of a particular alternate key value can be detected through the status code returned in the FILE STATUS clause by testing for a code value of 02.

SEQUENTIAL FILE ORGANIZATION

Records are read and written in sequence when the file organization is sequential. The position of each record in the file determines the order of access. Records can only be written following the last record in the file. Sequential file organization is most effective for files that are normally read from beginning to end.

Magnetic tape, punched card, and line printer files must have sequential organization. Other mass storage files can have sequential organization if desired. Keys are not used for sequential file organization; the file can only be accessed sequentially.

FILE DEFINITION

The structure of a file with sequential organization is described through the FILE-CONTROL paragraph in the Environment Division and the File Description and Record Description entries in the Data Division.

FILE-CONTROL Paragraph

For sequential file organization, the FILE-CONTROL paragraph requires two clauses: SELECT and ASSIGN. Five optional clauses can be included in this paragraph. Refer to figure 3-2.

```
    .
    .                                Required Clauses
    .
ENVIRONMENT DIVISION.
INPUT-OUTPUT SECTION.
FILE-CONTROL.
    SELECT CUSTOMER-FILE ASSIGN TO CSTMER
        ORGANIZATION IS SEQUENTIAL
        ACCESS MODE IS SEQUENTIAL
        RESERVE 2 AREAS
        FILE STATUS IS FILE-CODE
        USE "RT=Z".
    .
    .
    .
```

Figure 3-2. FILE-CONTROL Paragraph for a Sequential File

The SELECT clause specifies the file-name used by the COBOL 5 program; the same file-name is referenced in a File Description entry. The ASSIGN clause associates the program file-name with a logical file name that is used by the operating system. If the logical file name duplicates a name used in the program or a reserved word, with the exception of the words INPUT and OUTPUT, it must be enclosed in quotation marks.

The ORGANIZATION clause and the ACCESS MODE clause can be specified in the FILE-CONTROL paragraph for a sequential file. These clauses, if included, must specify SEQUENTIAL; this is the default value for both clauses.

The RESERVE clause is included to specify the number of input/output areas to be used for file buffers. If the clause is not specified, five input/output areas are allocated. In many cases, performance can be improved significantly when the RESERVE clause is used. The size of each input/output area is the maximum block size.

The FILE STATUS clause names a data item that is used to receive a status code whenever an input/output statement is executed. The value of the status code indicates whether or not the statement executed successfully. The status code is discussed further in section 15.

The USE clause supplies file information used by Basic Access Methods (BAM) to process the file. Certain FILE control statement parameters can be specified in the USE clause. These parameters supply file information that cannot be specified through the clauses and statements in the source program, or they override parameter values normally obtained from the source program. Refer to section 15 for a complete list of the parameters that can be specified.

One additional parameter can be included in the USE clause for a sequential file. A file that is not assigned to OUTPUT in the ASSIGN clause can be designated as a print file by specifying USE "PRINTF=YES". When this parameter is specified, the user program does not supply a carriage control character as the first character in the record area. Line spacing can be established with the BEFORE/AFTER ADVANCING phrase of the WRITE statement. If this phrase is not specified, all lines are single spaced.

A FILE-CONTROL paragraph for a file with sequential organization is illustrated in figure 3-2. The file-name CUSTOMER-FILE is used within the COBOL 5 program to reference the file; the logical file name recognized by the operating system is CSTMER. The ORGANIZATION and ACCESS MODE clauses are included for documentary purposes.

File Description Entry

A sequential file named in a SELECT clause must be defined by a File Description entry (FD entry) in the File Section of the Data Division. The FD entry defines the structure of the file, the manner in which data is stored, and tape labeling conventions. Six clauses in the FD entry are applicable to sequential file organization. Refer to figure 3-3.

The BLOCK CONTAINS clause is used to determine the block type and block size for a sequential file that is on a tape with S or L format. Table 3-1 shows the various block types and block sizes that result from the BLOCK CONTAINS clause for a tape with S or L format. For other sequential files, the block type is always block type C; the block size is determined as follows:

- On a mass storage device, block size is 640 characters.
- On a tape with SI or I format, block size is one physical record unit (PRU). PRU device sizes are:

  Binary SI tapes - 5120 characters

  I tapes - 5120 characters (supported on NOS only)

  Coded SI tapes - 1280 characters (supported on NOS/BE only)

```
                .
                .
                .
DATA DIVISION.
FILE SECTION.
FD  CUSTOMER-FILE
        LABEL RECORDS ARE STANDARD
            VALUE OF FILE-ID IS "CF123"
        BLOCK CONTAINS 10 RECORDS
        RECORD VARYING FROM 60 TO 100 CHARACTERS
            DEPENDING ON REC-LENGTH
        CODE-SET IS UNI
        RECORDING MODE IS DECIMAL
        DATA RECORD IS CUSTOMER-REC.
                .
                .
                .
```

Figure 3-3. File Description Entry for a Sequential File

TABLE 3-1. BLOCK TYPE AND SIZE FOR S AND L TAPE FILES

BLOCK CONTAINS Clause	Block Type	Block Size†
Omitted	K	Number of characters in one record; number varies as actual record lengths vary
BLOCK CONTAINS integer RECORDS	K	Number of characters in the specified number of records; number varies as actual record lengths vary
BLOCK CONTAINS integer TO integer RECORDS	E	Number of characters within the range first integer times minimum record size and second integer times maximum record size; maximum number of records within the specified range, not exceeding maximum block size
BLOCK CONTAINS integer CHARACTERS	E	Specified number of characters
BLOCK CONTAINS integer TO integer CHARACTERS	E	Number of characters within the specified range; maximum number of records without exceeding maximum block size

†Block types K and E always have an even number of characters; if necessary, the system adds a padding character.

The CODE-SET clause is applicable primarily to tape and card files. The clause indicates that the data in the file is to be read or written according to the external alphabet named in the ALPHABET clause of the Environment Division. This allows information from another manufacturer's system to be processed in correspondence with the internal display code of the CDC system. The conversion parameter on the LABEL statement (CV on NOS or N on NOS/BE) overrides any external alphabet named in the ALPHABET clause.

The LABEL RECORDS clause must be specified in every FD entry. It indicates whether or not labels exist on the file. Labels can be specified only for magnetic tape files. When labels exist, values can be specified for certain fields in the label record. For an input file, values specified in the FD entry are checked against the values in the label fields. For an output file, values specified in the FD entry are placed in the label fields.

The RECORD clause specifies the number of characters in a record. If all records in the file are not the same length, this clause indicates the least number of characters and the most number of characters a record can contain. The information supplied in the RECORD clause is used to determine the record type and record size for input/output processing by BAM. If the clause is omitted, record type and size are determined by the Record Description entry. Table 3-2 lists the record type for each format of the RECORD clause. Refer to table 4-2 in section 4 of the COBOL 5 reference manual for record size information.

The RECORDING MODE clause is applicable only to tape files. It specifies whether the tape file is recorded in binary or decimal code. Conversion between internal and external code sets occurs when the recording mode is decimal.

TABLE 3-2. RECORD TYPES FOR SQ, IS, DA AND AK FILES

RECORD CLAUSE (FD entry)	RECORD DESCRIPTION ENTRY			
	01 Entries of Same Length	01 Entries of Different Length	Entry with OCCURS/DEPENDING ON data-name in record	Entry with OCCURS/DEPENDING ON data-name not in record
Clause omitted	F	W	T	W
RECORD CONTAINS integer CHARACTERS	F†	F	F	F
RECORD CONTAINS integer-1 TO integer-2 CHARACTERS	W	W	T	W
RECORD CONTAINS integer-1 TO integer-2 CHARACTERS DEPENDING ON data-name in record	D	D	D	D
RECORD CONTAINS integer-1 TO integer-2 CHARACTERS DEPENDING ON data-name outside record	W	W	W	W

†Record type is Z if file name is INPUT, OUTPUT, or PUNCH.

Note: For each RECORD CONTAINS format, an equivalent RECORD VARYING format exists, giving the same respective record type.

RECORDING MODE IS DECIMAL causes the CM field of the FIT to be set to YES. RECORDING MODE IS BINARY causes the CM field of the FIT to be set to NO (no conversion occurs). For ASCII and EBCDIC conversions the tape driver makes the conversion only when the CM field in the FIT contains YES. (YES is the COBOL default). For 7-track UNIVAC conversions, the COBOL library makes the conversion.

If the RECORDING MODE clause is omitted, conversion is assumed (except for block-I-type-W records).

The terms CM=YES, RECORDING MODE IS DECIMAL, and coded tapes with even parity (7 track) are equivalent. Likewise, the terms CM=NO, RECORDING MODE IS BINARY, and binary tapes with odd parity (7 or 9 track) are equivalent.

Figure 3-3.1 illustrates a COBOL program that converts an IBM EBCDIC tape to CDC display code, on the NOS operating system. In this program, the ALPHABET clause and the CODE-SET clause are documentary only. The CV=EB parameter on the LABEL statment overrides the COBOL statements and causes the conversion.

The LINAGE clause can be specified for a file that is to be printed. It indicates the number of lines on a logical page and can optionally define the top margin and footing area within the page. When the LINAGE clause is included in the FD entry, the ADVANCING and AT END-OF-PAGE phrases of the WRITE statement can be used to position print lines within the boundaries of the logical page. The value of the special register LINAGE-COUNTER indicates the current line number. The LINAGE clause cannot be specified for a report file generated through the Report Writer feature.

A File Description entry for a tape file is illustrated in figure 3-3. The file contains a standard label. If it is an input file, the operating system issues a diagnostic message and terminates the job if the FILE-ID field of the LABELS RECORDS clause does not agree with the file-id set by the file identifier (FI) parameter of the NOS LABEL control statement. The usage of the FILE-ID field is not required.

The block type for the tape file is K and each block contains 10 records; the actual length of the block can be from 600 to 1000 characters, depending on the actual size of each record. The record type is D and each record contains from 60 to 100 characters; the value of the data item REC-LENGTH, which is contained within the record, specifies the actual size of the individual record. The recording mode for the tape file is decimal. The DATA RECORD clause is included for documentary purposes only; it indicates that all records in the file are formatted according to the Record Description entry for the record named CUSTOMER-REC.

Record Description Entry

The File Description entry must include one Record Description entry for each record format applicable to the sequential file. A Record Description entry describes the physical structure of a record and provides data-names that are used to access specific data items within the record. Records in sequential files can be either fixed length or variable length.

When the RECORD clause is not included in the FD entry, record type and record size are determined from the Record Description entries for the file. Variable-length records can be described with an OCCURS clause that includes the DEPENDING ON option.

Control Statements (NOS)

```
JOB Statement
USER Statement
LABEL,EB1,R,D=PE,F=S,PO=RM,CV=EB,VSN=XXXXX
COBOL5.
LGO.
```

Source Program

```
    IDENTIFICATION DIVISION.
    PROGRAM-ID.                          TAPTST.
    ENVIRONMENT DIVISION.
    CONFIGURATION SECTION.
    SPECIAL-NAMES.
        ALPHABET IBM-CODE IS EBCDIC.
    INPUT-OUTPUT SECTION.
    FILE-CONTROL.
        SELECT REFORMAT-FILE ASSIGN TO "EB1";
            USE "RT=S,BT=C,EO=AD,EFC=3".
        SELECT CUSTOMER-FILE ASSIGN TO "CSTMER";
            USE "RT=Z,BT=C,EO=AD,EFC=3".
    DATA DIVISION.
    FILE SECTION.
    FD  REFORMAT-FILE
        LABEL RECORDS ARE STANDARD
        BLOCK CONTAINS 107 CHARACTERS;
        CODE-SET IS IBM-CODE;
        RECORDING MODE IS DECIMAL;
        DATA RECORD IS CUSTOMER-REC1.
    01  CUSTOMER-REC1.
        02  REC-1                       PIC X(80).
    FD  CUSTOMER-FILE
        LABEL RECORDS ARE OMITTED
        DATA RECORD IS CUSTOMER-REC2.
    01  CUSTOMER-REC2.
        02  REC-2                       PIC X(80).
    PROCEDURE DIVISION.
    OPENING.
        OPEN INPUT REFORMAT-FILE .
        OPEN OUTPUT CUSTOMER-FILE.
        PERFORM 26 TIMES
            READ REFORMAT-FILE
            WRITE CUSTOMER-REC2 FROM CUSTOMER-REC1
        END-PERFORM
        PERFORM WRAPUP.
    WRAPUP.
        CLOSE REFORMAT-FILE, CUSTOMER-FILE.
        STOP RUN.
```

Figure 3-3.1. IBM EBCDIC Tape Conversion (Sheet 1 of 2)

Converted File

```
S358244038ADAMS          BARBARA    220070900141400
S570327591BURCHELL       DONALD     220070670152200
S463445549CLEVELAND      WILLIAM    220070200170500
S207243050DAVIES         DAVID      220070510219000
S571649674ELLIS          ALAN      F220070680081500
S562460661FERRERA        ROBERT     220070060137100
S148169725GRAME          CARL       220070800105000
S566208909HARVEY         LAURENCE E220070450383500
S132246243IMMITT         SALVATOREJ220070690204300
S572548172JENSEN         HOWARD    M220070070091250
S576246405KANE           DAVID     H220070190146900
S087222701LEAVITT        MURRAY     220070640175700
S359306744MILTON         JOHN       220070220070200
S551482678NEWTON         PAULINE    220070410068550
S564388352O'DONNELL      DANIEL    J220070630150800
S550429831PETERSON       DENNIS     220070820778700
S091403215QUEENSBURY     TATIANA    220070400244950
S545016985ROKITIANSKY    N          220070410280350
S384186384SOWUL          JEROME     220070630225600
S571202817TREJO          PAUL       220070090202250
S568283442UTTERBACH      WILLIAM    220070270076300
S560461439VAN            FOSSEN,   L220070751365950
S555244713WILLEY         GEORGE     220070720185800
S548546977XANDTHRUS      ROGER      220070260135500
S293305616YOLLES         ROBERT     220070870044500
S568462813ZOFFMAN        NORMA     B220070620036850
```

Figure 3-3.1. IBM EBCDIC Tape Conversion (Sheet 2 of 2)

A Record Description entry for a mass storage file with sequential organization is illustrated in figure 3-4. The record type, determined from the Record Description entry, is record type T. All records in the file contain at least 70 characters (60 characters in the fixed portion and 10 characters in one item in the trailing portion). The maximum record size is 210 characters (60 characters in the fixed portion and 10 characters in each of 15 items in the trailing portion).

FILE MANIPULATION

Sequential files are processed through the use of five Procedure Division statements. Records can be added at the end of an existing sequential file; if the file resides on a mass storage device, records can also be rewritten. Individual data items in a record are processed by various statements that are discussed in other sections of this guide.

Opening Sequential Files

Before records in a sequential file can be input or output, the file must be opened by the execution of an OPEN statement. Any sequential file can be opened for input or for output. A mass storage sequential file can also be opened for input and output.

An input file is opened with the OPEN INPUT statement. The file is available for read-only processing. If the file contains a label, label checking is performed. The file is positioned at the first record unless the file resides on the file INPUT or unless the REVERSED or NO REWIND phrase is specified in the OPEN statement. Records in the file can then be read in sequence until the end of the file is reached.

```
    .
    .
    .
ENVIRONMENT DIVISION.
    .
    .
    .
    SELECT INVOICES ASSIGN TO INVFLE.
    .
    .
    .
DATA DIVISION.
FILE SECTION.
FD  INVOICES
        LABEL RECORDS ARE OMITTED
        DATA RECORD IS INVOICE-REC.
01  INVOICE-REC.
    03  CUST-NAME          PICTURE X(18).
    03  CUST-ADDRESS.
        05  STREET         PICTURE X(18).
        05  CITY           PICTURE X(15).
        05  STATE          PICTURE AA.
        05  ZIP            PICTURE 9(5).
    03  NUM-ITEMS          PICTURE 99.
    03  ITEMS-ORDERED OCCURS 1 TO 15 TIMES
            DEPENDING ON NUM-ITEMS.
        05  ITEM-NO        PICTURE XXX.
        05  QUANTITY       PICTURE 99.
        05  COST           PICTURE 999V99.
    .
    .
    .
```

Figure 3-4. Record Description Entry for a Sequential File

OPEN INPUT CUSTOMER-FILE.

Execution of this statement opens the file CUSTOMER-FILE for input and positions the file at the first record. Records can be read from the file but cannot be written on it.

For a tape file that is contained on a single reel, the REVERSED phrase can be included in the OPEN INPUT statement. This phrase causes the file to be positioned at its end; records are then read in sequence from the end to the beginning. The REVERSED phrase can only be used when the tape file records are record type F, W, R, or Z. (Record type R or Z is specified through a FILE control statement parameter or in the USE clause.) For record type F, record length must be a multiple of 10 characters. Reading a tape file from the end to the beginning results in inefficient processing.

When the NO REWIND phrase is included in the OPEN INPUT statement, the file is opened at its present position. If this phrase is not specified, the file is rewound during execution of the OPEN statement.

The OPEN OUTPUT statement is specified when a new sequential file is being created. When this statement is executed, the file is available for write-only processing. If the LABEL RECORDS ARE STANDARD clause is specified for a tape file, the beginning label is written on the file; the file is then positioned immediately after the label. Records are written in sequence on the file. When the last record has been written, an ending label (if applicable) is written on the file.

OPEN OUTPUT INVENTORY-FILE.

Execution of this statement causes the file INVENTORY-FILE to be opened for output. If a label has been specified for the file, the label is written. Records are then written on the file in sequence.

The NO REWIND phrase can be included in the OPEN OUTPUT statement to open the file at its present position. This phrase is normally used to open an output file that has already been closed during program execution. When records are subsequently read from the file, an end-of-file condition is encountered at the point where the CLOSE statement was executed.

When records are to be added at the end of an existing sequential file, the OPEN EXTEND statement is specified. If the file contains labels, the labels are checked. The file is positioned immediately after the last record in the file. Records are then written following the last record. When the file is subsequently opened for input, no distinction exists between the records originally written and the extended records. When a file is created through a COPY control statement, the OPEN EXTEND statement causes it to be positioned after an end-of-file condition.

OPEN EXTEND INVENTORY-FILE.

The existing file INVENTORY-FILE is opened for output and the file is positioned immediately following the last record in the file. Records are then written in sequence on the file.

When more than one tape file is contained on a single reel or on a set of reels, only one of the files can be open at any given time. The files can be opened in any order when they are opened for input. When a file is opened for output (OPEN OUTPUT or OPEN EXTEND), the position number of the file being written must be higher than that of any existing files in the set. Once a file is opened for output, any subsequent WRITE statements for the file destroy all files positioned after the one being written.

When a file to be opened is contained in a multifile set, that file must be specified in the MULTIPLE FILE TAPE clause of the I-O-CONTROL paragraph. Refer to the COBOL 5 reference manual for the format and usage of this clause.

A mass storage sequential file is opened for input and output by the OPEN I-O statement. When this statement is executed, the file is available for reading or updating records. Records can be read or updated (through the REWRITE statement) in sequence. The OPEN I-O statement should only be used when the file is being updated.

```
OPEN I-O INVOICE-FILE.
```

The file INVOICE-FILE is a mass storage file and is opened for both input and output. Records can be read from or rewritten on the file. No additional records can be written on this file.

Writing Sequential Files

The WRITE statement is used to write a record on a sequential file that has been opened with the OPEN OUTPUT or OPEN EXTEND statement. Three optional phrases can be included in a WRITE statement for a sequential file.

If the output file is not a print file, only the FROM phrase is applicable. This phrase causes the data in the specified area to be moved into the output record area and the record to be written on the output file.

```
WRITE OUT-REC FROM TEMP-REC.
```

This statement causes the data in the storage area named TEMP-REC to be moved to the output record area OUT-REC. The output record is then written on its associated file.

The ADVANCING phrase specifies print line positioning before or after the output record is printed. A number of lines to be skipped before or after printing can be specified by either an integer or a data item that contains an integer. If the integer is 1, the print line is single spaced.

```
WRITE PRINTLINE
    BEFORE ADVANCING 3 LINES.
```

The data in the output record PRINTLINE is written on the output file. The next record written on the output file contains a carriage control character that causes the line printer to advance, or skip ahead, three lines before the record is printed.

The keyword PAGE can be specified in the ADVANCING phrase to position the output to the top of the next page, either for the present line or for the next line to be printed. If the FD entry for the file contains the LINAGE clause, the output is positioned to the top of the next logical page; otherwise the output is positioned at the top of the physical page (if a physical page concept exists).

```
WRITE PRINTLINE
    AFTER ADVANCING PAGE.
```

Execution of this statement causes the record to be written as the first line of the next page.

A mnemonic-name can also be specified in the ADVANCING phrase to insert a carriage control character as the first character of the output record. Mnemonic-name is defined in the SPECIAL-NAMES paragraph as one of the carriage control characters recognized by the operating system. If the file is defined with the LINAGE clause, a mnemonic-name cannot be specified for line positioning.

```
WRITE PRINTLINE
    BEFORE ADVANCING TRIPLE.
```

The mnemonic-name is defined in the SPECIAL-NAMES paragraph as "-" IS TRIPLE. The PRINTLINE record is written on the output file and a hyphen is inserted as the first character of the next record to be written.

When a WRITE statement without the ADVANCING phrase is executed following a WRITE statement with the ADVANCING phrase (with no intervening OPEN statement), the output record is written after advancing one line if the first character of the line is a blank. If the first character of the output record is a nonblank character, the record is written according to the rules for AFTER ADVANCING.

The END-OF-PAGE phrase is only applicable to a print file that includes the LINAGE clause in the FD entry for the file. This phrase causes execution of an imperative statement when the end of the page is reached. The END-OF-PAGE phrase is described in more detail in section 5, Conditional Operations.

Reading Sequential Files

Once a sequential file has been opened for input (OPEN INPUT or OPEN I-O), individual records in the file are made available to the COBOL program by the READ statement. Records are read in the sequence in which they were written. The AT END phrase is included in the READ statement to specify the action to be taken after the last record has been read.

```
READ INVENTORY-FILE RECORD
    AT END GO TO END-IT.
```

When this statement is executed, a record is read from the file INVENTORY-FILE. Control is transferred to the paragraph named END-IT when the end of the file is reached.

The INTO phrase of the READ statement causes the record to be read from the file and stored in a specified area. The record is available in both the input record area and the specified storage area. When the file is defined by more than one Record Description entry, the INTO phrase cannot be used if any entry is a level 01 elementary item that is described as a numeric or numeric-edited data item.

```
READ INVOICE-FILE RECORD INTO TEMP-REC
    AT END GO TO CLOSING.
```

Each time this statement is executed, a record from the file INVOICE-FILE is read and is stored in both the input record area and the storage area designated TEMP-REC. When the end of the file is reached, control is transferred to the paragraph named CLOSING.

Updating Sequential Files

Existing sequential mass storage files can be updated by using the REWRITE statement to replace an existing record in the file. Only files with record type F (fixed length) or W (control word) can be rewritten. The file must be open for input and output (OPEN I-O). Tape files cannot be updated by the REWRITE statement. Rewriting records does not result in efficient processing and should be avoided when possible.

The record replaced is the last record read before the REWRITE statement is executed. The new record must contain the same number of characters as the record being replaced. After the record has been read, individual data items can be changed by program statements. The updated record is then written in place of the original record.

```
REWRITE CUSTOMER-REC.
```

The current data in the record area for the file is written in place of the last CUSTOMER-REC record read from the file.

The FROM phrase is included in the REWRITE statement when the new record is created in a storage area that is not the record area for the file. The storage area must be the same size as the record area.

```
REWRITE CUSTOMER-REC FROM TEMP-REC.
```

The data in the storage area TEMP-REC is moved to the record area for CUSTOMER-REC. The record is then written in place of the last record read from the file.

Closing Sequential Files

A sequential file that has been opened for processing is closed by the CLOSE statement to terminate processing of the file. When a CLOSE statement is executed for a file, no input/output statement can reference that file until it has been opened again.

The simplest form of the CLOSE statement specifies only the file-name. The file is closed and labels are processed as appropriate.

```
CLOSE INVENTORY-FILE.
```

The file INVENTORY-FILE is rewound and closed. If the file has labels, the labels are checked for an input file or written for an output file. No subsequent input/output statement can access INVENTORY-FILE unless the file is reopened.

When the REEL (or UNIT) phrase is included in the CLOSE statement, a checkpoint takes place if the RERUN EVERY END OF REEL clause is specified in the I-O-CONTROL paragraph. For a mass storage file, no further action takes place; for a tape file, processing stops on the current reel and resumes on the next reel.

```
CLOSE INVENTORY-FILE REEL
    WITH NO REWIND.
```

A checkpoint takes place if established for the end of a reel; processing resumes with the next reel. The WITH NO REWIND phrase inhibits the rewinding that normally takes place during processing of the CLOSE statement.

The WITH LOCK phrase is included in the CLOSE statement to prevent the file from being reopened during execution of the current job step. The file is returned to the system. If an attempt is made to reopen the file, the program aborts.

When the file being closed is to be reopened immediately, the C.FILE routine should be entered to override the default COBOL setting of the CF field in the file information table (FIT). If the value in the CF field is changed from DET (the default) to R prior to the close, buffer space and BAM capsules that would otherwise be returned to the system are retained by the program. User setting of the CF field is overridden if the REEL phrase or WITH LOCK phrase is included in the CLOSE statement.

```
ENTER "C.FILE" USING INVENTORY-FILE,"CF=R".
    CLOSE INVENTORY-FILE.
```

The CF field of the FIT for the file INVENTORY-FILE is altered to contain the R option, and the file is closed.

RELATIVE FILE ORGANIZATION

A relative file is a mass storage file in which a record key specifies the physical position of a record within the file. Record position is relative to the first record in the file. The key value for the first record is 1, for the second record is 2, and so forth. All records in the file are fixed length; if the file has multiple record descriptions of different lengths, the length of the largest record is the length used for all records. The access mode for a relative file can be sequential, random, or dynamic.

Relative file organization can be effectively used for files needing rapid access. Key values should be contiguous beginning with key value 1. Space exists on the file for unused key values. If the first record written on the file has a key value of 100, the file is created with 99 empty entries preceding the record with key value 100.

FILE DEFINITION

The FILE-CONTROL paragraph in the Environment Division, as well as the File Description and Record Description entries in the Data Division, describe the structure of a file with relative organization.

FILE-CONTROL Paragraph

The FILE-CONTROL paragraph for a file with relative organization must include three clauses: SELECT, ASSIGN, and ORGANIZATION. Five optional clauses can be included in this paragraph as needed. Refer to figure 3-5.

```
      .                                    Required Clauses
      .                                          |
      .                                          |
ENVIRONMENT DIVISION.                            |
INPUT-OUTPUT SECTION.                            |
FILE-CONTROL.                                    |
    SELECT BOX-FILE ASSIGN TO BOXFLE <-----------|
        ORGANIZATION IS RELATIVE <---------------|
        ACCESS MODE IS RANDOM
        RELATIVE KEY IS REL-KEY.
      .
      .
      .
```

Figure 3-5. FILE-CONTROL Paragraph for a Relative File

The file-name used by the program is specified in the SELECT clause; the logical file name recognized by the operating system is specified in the ASSIGN clause. If the logical file name is the same as any other name used in the program or as a reserved word, with the exception of INPUT and OUPUT, it must be enclosed in quotation marks. The ORGANIZATION clause must specify RELATIVE for a relative file.

The ACCESS MODE clause establishes the manner in which records can be processed during program execution. If SEQUENTIAL is specified or if the clause is omitted, records can only be processed sequentially; for a read operation, empty record entries in the file are bypassed. Records are processed randomly according to key value when RANDOM is specified. If DYNAMIC is specified, records can be processed both randomly by key value and sequentially by position during program execution.

The key value used to access records randomly is contained in the data item specified in the RELATIVE KEY clause. This clause must be included when the access mode is random or dynamic; it is required for sequential access only if the START statement is used to position the file for subsequent processing. The relative key data item cannot be a data item contained in the record. When the RELATIVE KEY clause is specified, it must immediately follow the ACCESS MODE clause. If the file is an External file, the relative key data item must be defined in the Common-Storage Section.

The number of input/output buffer areas can be increased by the RESERVE clause. Each input/output area is 64 words. When the file is processed sequentially, additional buffer areas can improve program performance because more records can be stored in memory at one time and the number of accesses is reduced. For random processing, however, the RESERVE clause should not be specified. If the clause is omitted, two buffer areas are reserved.

The FILE STATUS clause is specified to make a status code available to the program whenever an input/output statement is executed for the file. The status code is a value that designates successful or unsuccessful execution of the statement. The value further identifies the type of error that prevented the statement from executing. Refer to section 15 for a description of the status code.

File information used by BAM can be specified in the USE clause. Certain FILE control statement parameters can be specified to override file information obtained from the source program or to provide information that cannot be obtained from the program. The parameter list is enclosed in quotation marks and is in the same format used for the FILE control statement. Refer to section 15 for a complete list of parameters that can be specified.

One additional USE statement parameter can be specified for a relative file. All records begin on a physical record unit (PRU) boundary when PRUF=YES is specified in the USE clause. Records are then read and written in multiples of PRU size; PRU size is 640 characters. This results in very efficient processing when the record size is a multiple of PRU size minus 10 characters; COBOL adds 10 characters at the beginning of each record. If USE "PRUF=YES" is specified, the RESERVE clause has no effect on buffers; the record area is used as the buffer.

A FILE-CONTROL paragraph for a file with relative organization is illustrated in figure 3-5. The COBOL 5 program uses the file-name BOX-FILE while the operating system recognizes the file as BOXFLE. Random access is specified for the file; therefore, Procedure Division statements must access the file randomly by key value. The data item REL-KEY contains the key value used for random access.

File Description Entry

The structure of a relative file is defined in a File Description entry (FD entry) in the File Section of the Data Division. The FD entry specifies the program file-name from the SELECT clause. Two clauses in the FD entry are applicable to files with relative organization. Refer to figure 3-6.

```
      .
      .
      .
DATA DIVISION.
FILE SECTION.
FD  BOX-FILE
          LABEL RECORDS ARE OMITTED
          RECORD CONTAINS 60 CHARACTERS
          DATA RECORD IS BOX-REC.
      .
      .
      .
```

Figure 3-6. File Description Entry for a Relative File

The LABEL RECORDS clause is required in every FD entry. A relative file cannot have labels; this clause must specify OMITTED.

All records in a relative file are fixed-length records (record type F). The RECORD clause can be specified to indicate the number of characters in each record. If a range of characters is specified, the record size is the maximum number of characters; otherwise, the record size is the specified number of characters. If the RECORD clause is omitted, record size is determined by the Record Description entry.

Figure 3-6 illustrates a File Description entry for a relative file. Each record in the file contains 60 characters. The DATA RECORD clause documents the name of the record description for the file.

Record Description Entry

A Record Description entry is included in the File Description entry for each record format applicable to the relative file. All records in a relative file must be fixed-length records. The Record Description entry identifies each data item by a data-name and describes the physical structure of a record.

Because all records are fixed length, the record type is always record type F for relative files. The record size, if not specified in the RECORD clause, is determined by the number of character positions described in the Record Description entry. If the File Description entry includes more than one Record Description entry, the size of the longest record described is the size of each record in the file. For variable-length records, the record size of each record is the maximum record length.

The Record Description entry shown in figure 3-7 is applicable to the FILE-CONTROL paragraph (figure 3-5) and the File Description entry (figure 3-6) for a relative file. The format of the record BOX-REC describes 60 character positions, which is the number of characters specified in the RECORD clause. Each record in the file has a fixed length of 60 characters. The record key, which is used to access the file randomly, is described in the Working-Storage Section as a three-digit integer.

```
      .
      .
      .
DATA DIVISION.
FILE SECTION.
      .
      .
      .
01   BOX-REC.
     03   CUST-NAME          PICTURE X(20).
     03   STREET             PICTURE X(18).
     03   CITY               PICTURE X(15).
     03   STATE              PICTURE AA.
     03   ZIP-CODE           PICTURE 9(5).
      .
      .
      .
WORKING-STORAGE SECTION.
01   REL-KEY                 PICTURE 999.
      .
      .
      .
```

Figure 3-7. Record Description Entry for a Relative File

FILE MANIPULATION

Relative file input/output processing is specified through seven Procedure Division statements. Once the relative file has been created, records can be read, replaced, deleted, inserted, and added to the file. Various statements, which are discussed in other sections of this guide, are available to process data items within the records.

Opening Relative Files

A relative file is opened for input, for output, or for input and output. The specific format of the OPEN statement determines the open mode for the file.

A file is opened for input with the OPEN INPUT statement. Records within the file are processed sequentially or randomly depending on the access mode. The OPEN INPUT statement makes the file available for read-only processing. The file is positioned at the first record stored in the file.

```
OPEN INPUT BOX-FILE.
```

This statement specifies that the file BOX-FILE is to be opened for input. When the statement is executed, the file is positioned at the first record. Records can be read from BOX-FILE, but no record can be written on BOX-FILE.

When a relative file is being created, the OPEN OUTPUT statement is specified for the file. Execution of this statement makes the file available for write-only processing. Records are written on the file sequentially or randomly by key value.

```
OPEN OUTPUT REL-FILE.
```

When this statement is executed, the file REL-FILE is opened for output and is positioned for the first record. Records can then be written on the file in sequence or randomly by key value.

The OPEN I-O statement is used to open an existing relative file for input and output. Records in the file can be read or updated. If the access mode is random or dynamic, records can also be added to the file.

```
OPEN I-O REL-FILE.
```

Execution of this statement causes the file REL-FILE to be opened for input and output. Records can subsequently be read, deleted, and rewritten. If random or dynamic access mode is specified for REL-FILE, records can also be written on the file.

Writing Relative Files

Records are written on a relative file that has been opened for output (OPEN OUTPUT for file creation or OPEN I-O for file updating). The access mode established for the file determines whether the records can be written sequentially or randomly.

When the access mode is sequential, records are automatically written in sequence. The system generates the key values beginning with key value 1 for the first record written on the file. As each record is written, the key value is incremented to indicate the next record position in the file. When a record is written, the relative key data item, if specified, contains the key value for the record just written. Creating a relative file by writing records sequentially ensures that all record positions in the file are filled.

```
WRITE BOX-REC.
```

Each time this statement is executed, a BOX-REC record is written on its associated file. The record positions in the file are used in sequence; no empty record positions exist when the file is closed.

If the access mode for the file is random or dynamic, records are written on the file according to key values that are supplied by the program. When the WRITE statement is executed, the current value of the relative key data item specifies the record position for the record being written. An invalid key condition exists if the key value specifies a record position that already contains a record.

```
WRITE REL-REC
    INVALID KEY GO TO BAD-KEY.
```

For a relative file with random or dynamic access, this statement causes a record (REL-REC) to be written on its associated file in the record position corresponding to the value of the relative key data item. If the key value is not valid, control is transferred to the paragraph named BAD-KEY.

The FROM phrase can be included in the WRITE statement for either a sequential or random write operation. The data in the specified storage area is moved to the output record area before the record is written.

```
WRITE BOX-REC FROM TEMP-REC.
```

When this statement is executed, the data in the storage area named TEMP-REC is moved to the output record area for BOX-REC. The record is then written on its associated file.

Positioning Relative Files

A relative file can be positioned to a specific record in the file for subsequent sequential processing. The file must be open for input (OPEN INPUT or OPEN I-O) and the access mode must be either sequential or dynamic.

The START statement positions the file according to the current value of the relative key data item. Records are then retrieved sequentially beginning with the record at which the file is positioned.

The KEY phrase, if included in the START statement, must specify the relative key data item. Depending on the relational operator selected, the file is positioned at the record position equal to, greater than, or not less than the current value of the relative key data item. The relational operator NOT LESS THAN is equivalent to the equal to or greater than condition. If the KEY phrase is not specified, the file is positioned at the record position equal to the value of the relative key data item.

```
START BOX-FILE
    KEY IS EQUAL TO REL-KEY.
```

Execution of this statement causes the file BOX-FILE to be positioned at the record position indicated by the value of the relative key data item REL-KEY. Positioning of the file is the same whether or not the KEY phrase is specified.

The INVALID KEY phrase is included in the START statement to indicate the action to be taken when the specified condition cannot be satisfied by any record in the file.

```
START REL-FILE
    KEY IS GREATER THAN REC-NO
    INVALID KEY GO TO CANT-FIND.
```

When this statement is executed, the file is positioned at the first record position following the record position indicated by the current value of the relative key data item REC-NO. If the value of REC-NO indicates the last record in the file, the condition cannot be satisfied and control is transferred to the paragraph named CANT-FIND.

Reading Relative Files

When a relative file has been opened for input (OPEN INPUT or OPEN I-O), the READ statement makes a record in the file available to the COBOL 5 program for subsequent processing. Depending on the access mode established for the file, records are read sequentially by position in the file or randomly by relative key value. The format of the READ statement differs for reading sequentially and randomly.

Accessing Sequentially

When the access mode for a relative file is established as sequential or dynamic, records can be read sequentially. The first time the READ statement is executed, the record retrieved is either the first record in the file or the record at which the file has been positioned by the START statement. If the access mode is dynamic and a random READ statement has been executed, the next record in sequence is retrieved. Subsequent executions of the READ statement cause the records to be read in the order they appear in the file. Only records that have been written are retrieved; empty record positions are bypassed. After a successful read operation, the relative key data item, if specified, contains the key value for the record just read. The AT END phrase designates the action to be taken when the last record in the file has been read.

```
READ BOX-FILE RECORD
    AT END GO TO CLOSE-FILE.
```

This statement causes records to be read sequentially from the file BOX-FILE. When the end of the file has been reached, control is transferred to the paragraph named CLOSE-FILE.

The INTO phrase can be included in the READ statement to store the record in a specified storage area. The record is then available in both the storage area and the input record area. When the file is defined by more than one Record Description entry, the INTO phrase cannot be used if any entry is a level 01 elementary item that is described as a numeric or numeric-edited data item.

```
READ REL-FILE RECORD INTO REC-AREA
    AT END GO TO FINISHED.
```

When this statement is executed, the next record in sequence in the file REL-FILE is read and stored in the input record area and in the storage area named REC-AREA. Control is transferred to the paragraph named FINISHED when the end of the file has been reached.

If the access mode is dynamic, the keyword NEXT must be included in the READ statement to access the records sequentially. For sequential access, the keyword NEXT only provides documentation.

```
READ BOX-FILE NEXT RECORD
    AT END GO TO CLOSING.
```

Execution of this statement causes records from the file BOX-FILE to be read in sequence. If dynamic access is established for BOX-FILE, the keyword NEXT is required; otherwise, NEXT is optional. When the end of the file is reached, control is transferred to the paragraph named CLOSING.

Accessing Randomly

Relative file records are read randomly by relative key when the access mode is established as random. Records can also be read randomly when the access mode is dynamic. The value of the data item defined as the relative key indicates the record number of the record to be read. If the key value indicates an empty record position or is greater than the record number of the last record on the file, an invalid key condition exists. The INVALID KEY phrase is included in the READ statement to designate the action to be taken when the key value is not valid.

```
READ BILL-FILE RECORD
    INVALID KEY GO TO END-IT.
```

This statement reads a record from the file BILL-FILE according to the value of the relative key data item. If the key value is not valid, control is transferred to the paragraph named END-IT.

The INTO phrase can also be included in a random access READ statement to store the record in a specified storage area. This phrase is executed in the same manner as for a sequential read operation.

Updating Relative Files

Existing relative files are updated through the DELETE and REWRITE statements. The WRITE statement is used to insert records in empty record positions and to add records to the end of a file. The file must be open for input and output (OPEN I-O).

The DELETE statement is used to remove a record from the file. Once a record is deleted, the record position is considered to be an empty record position. Depending on the access mode for the file, the record deleted is either the last record read or the record in the record position indicated by the current value of the relative key data item.

When the access mode is sequential, the last input/output statement executed before the DELETE statement must be a valid sequential READ statement. The record retrieved by the READ statement is then the record that is deleted.

```
READ REL-FILE RECORD.
.
.
.
DELETE REL-FILE RECORD.
```

Execution of this statement causes the record retrieved by the READ statement to be deleted from the file REL-FILE. The record can no longer be accessed from the file.

When the access mode for the file is either random or dynamic, the record identified by the value of the relative key data item is the record that is deleted. The INVALID KEY phrase is included in the DELETE statement to specify the action to be taken when the record to be deleted does not exist on the file.

```
DELETE BOX-FILE RECORD
    INVALID KEY GO TO NO-RECORD.
```

The record in the record position identified by the relative key data item is deleted from the file BOX-FILE. If the designated record position does not contain a record, control is transferred to the paragraph named NO-RECORD.

The REWRITE statement is used to replace an existing record in the file. The current data in the record area replaces the data stored on the file. The FROM phrase is included in the REWRITE statement when the updated record is stored in an area other than the record area. The data in the specified area is moved to the record area before the record is rewritten on the file.

If the access mode is sequential, the input/output statement preceding the REWRITE statement must be a sequential READ statement. The record replaced is then the last record read.

```
READ REL-FILE RECORD INTO TEMP-REC.
.
.
.
REWRITE REL-REC FROM TEMP-REC.
```

When the REWRITE statement is executed, the data in the storage area TEMP-REC is moved to the record area for REL-REC. The record read by the preceding READ statement is then replaced by the data in the record area.

If the access mode is random or dynamic, the record to be replaced is identified by the value of the relative key data item. An invalid key condition occurs when the key value does not identify an existing record.

```
REWRITE BILL-REC
    INVALID KEY GO TO BAD-KEY.
```

Execution of this statement rewrites the BILL-REC record stored in the record position that is indicated by the value of the relative key data item. If the key value does not specify an existing record, control is transferred to the paragraph named BAD-KEY.

Closing Relative Files

Processing of a relative file is terminated by closing the file with the CLOSE statement. Once the CLOSE statement is executed, input/output statements cannot access the file until it is opened again. When a relative file is closed, a partition boundary exists at the end of the file. The boundary is overwritten when records are added to the end of the file.

When the WITH LOCK phrase is included in the CLOSE statement, the file is closed and returned to the system. It cannot be reopened during execution of the current control statement; an attempt to reopen the file causes the program to abort.

When a file is to be reopened immediately after being closed, the C.FILE routine should be entered to change the FIT setting of the CF field from DET to R. If this action is taken before the CLOSE statement is executed, buffer space and BAM capsules that are normally returned to the system are preserved for the program. If the WITH LOCK phrase is specified in the CLOSE statement, the CF field setting cannot be overridden by the user.

```
ENTER "C.FILE" USING REL-FILE, "CF=R".
    CLOSE REL-FILE.
```

Execution of these statements causes the CF field setting for the file REL-FILE to be changed to the R option. When the file is closed, buffer space and system capsules used by the file are not released.

INDEXED FILE ORGANIZATION

When the file organization is indexed, records are stored in sequence according to the primary key values. Records can be accessed sequentially and randomly. Indexed files can reside only on mass storage devices.

Indexed file organization is used most effectively for very large mass storage files that need to be accessed both randomly and sequentially. Each record is identified by a primary key. The value of the primary key is unique for each record in the file. Alternate keys can also be specified and used to access records in the file.

FILE DEFINITION

The structure of an indexed file is described through the FILE-CONTROL paragraph in the Environment Division and the File Description and Record Description entries in the Data Division.

FILE-CONTROL Paragraph

For indexed file organization, four clauses are required in the FILE-CONTROL paragraph: SELECT, ASSIGN, ORGANIZATION, and RECORD KEY. Four additional clauses can be included in this paragraph as needed. Refer to figure 3-8.

The SELECT clause specifies the file-name used by the program; the ASSIGN clause specifies the logical file name recognized by the operating system. If alternate keys are specified for the file, the ASSIGN clause must also include the logical file name of the alternate key index file. If either logical file name is identical to any other name used in the program or to a reserved word, it must be enclosed in quotation marks. The ORGANIZATION clause must specify INDEXED for an indexed file.

The RECORD KEY clause designates the data item that is the primary key for indexed file records. The primary key must be a data item embedded in each record or defined in the Working-Storage Section. The data item can be an elementary or group item. It can be described as an alphanumeric or unsigned numeric data item. Each primary key value in the file must be unique. As records are written on the file, they are stored in ascending sequence by primary key value.

Alternate keys are specified for indexed files by ALTERNATE RECORD KEY clauses. The clause is included once for each alternate key desired. If the primary key is embedded in the record, it must begin in a unique location within the record. Alternate keys can begin in unique or identical locations; however, if they begin in the same location, they must not be the same length. The location and description of the key data items must remain the same for the life of the file. Duplicate alternate key values can exist in the file only if the DUPLICATES phrase is included in the ALTERNATE RECORD KEY clause; otherwise, each value for the alternate key must be unique.

The ASCENDING option of the DUPLICATES phrase determines the order in which records with duplicate alternate key values are retrieved for sequential access by alternate key. If ASCENDING is not specified, the records with duplicate key values are retrieved in the order they were written. The records are retrieved in ascending primary key order when ASCENDING is specified.

Conditions for including an alternate key index entry in the alternate key index file are specified in the USE phrase and the OMITTED phrase of the ALTERNATE RECORD KEY clause. If neither phrase is included in the clause, index entries for all alternate keys in the file are stored in the index file. THE USE WHEN phrase identifies a one-character data item contained within each record and a literal of one to 36 characters in length. The alternate key index entry is stored in the index file when the character contained in the data item is the same as one of the characters in the literal. Both the data item and the literal must be alphanumeric. A single data-name can be used in more than one ALTERNATE RECORD KEY clause. Each character in the literal must be unique.

The OMITTED phrase of the ALTERNATE RECORD KEY clause can include either the KEY option or, as in the USE phrase, a data-name and literal. When KEY IS SPACES is specified, the alternate key index entry is omitted from the index file if the key value is all spaces and is described with USAGE IS DISPLAY. If KEY IS ZEROS is specified, the index entry is not stored in the index file when the key value is all zeros and has a usage of COMPUTATIONAL-1 or COMPUTATIONAL-2.

When the option of specifying a data-name and a literal is chosen in the OMITTED WHEN phrase, the alternate key index entry is excluded from the index file if the data item contains a character included in the literal. The literal and data item are set up in the same manner as described in the USE WHEN phrase. Refer to the example in the discussion on alternate key processing at the beginning of this section.

An alternate key in an indexed file record has more than one value when the alternate key is described with the OCCURS clause. When a record is written on the file, the value in each unique occurrence of the alternate key is indexed on the alternate key index file. The record can then be retrieved by the value in any occurrence of the alternate key.

The manner in which records are accessed during program execution is determined by the ACCESS MODE clause. If this clause is omitted or if SEQUENTIAL is specified, records can only be processed sequentially. All records are accessed randomly by key value when RANDOM is specified. Dynamic access mode allows records to be processed both sequentially and randomly during program execution. When an indexed file is being created, the sequential access mode should be used.

The FILE STATUS clause specifies a data item to receive a status code each time an input/output statement is executed. The status code value indicates whether or not the statement executed successfully. The status code is described in section 15.

The USE clause can supply file information used by AAM to process the indexed file. Certain FILE control statement parameters can be specified to supply file information that cannot be obtained from other clauses and statements in the source program, or to override file information normally obtained from the source program. The parameter list specified in the USE clause is enclosed in quotation marks. Refer to section 15 for a complete list of the parameters that can be specified in this clause.

The type of indexed file to be used is determined by the ORG parameter of the USE clause. If extended AAM files have been installed and either the parameter is omitted or ORG=NEW is specified, the file is treated as an extended indexed sequential file. ORG=OLD is required for files in the initial indexed sequential format.

The FILE-CONTROL paragraph illustrated in figure 3-8 describes a file with indexed organization. The file-name used within the COBOL 5 program is EMP-FILE. The ASSIGN clause specifies two logical file names; EMPFLE identifies the data file and INDFLE identifies the index file for the alternate keys. Dynamic access is specified for the file; therefore, the file can be processed both randomly and sequentially. The primary key for each record is the data item EMP-ID. Two alternate keys, HIRE-DATE and JOB-ID, are also specified; duplicate alternate key values are allowed. If records are retrieved by either alternate key, records with duplicate key values are returned in ascending sequence by primary key values.

File Description Entry

The File Description entry (FD entry) for an indexed file defines the physical structure of the file. The same program file-name specified in the SELECT clause is specified in the FD entry. Three specific clauses in the FD entry are applicable to indexed files. Refer to figure 3-9.

```
    .
    .                                  Required Clauses
    .
ENVIRONMENT DIVISION.
INPUT-OUTPUT SECTION.
FILE-CONTROL.
    SELECT EMP-FILE
        ASSIGN TO EMPFLE, INDFLE
        ORGANIZATION IS INDEXED
        ACCESS MODE IS DYNAMIC
        RECORD KEY IS EMP-ID
        ALTERNATE RECORD KEY IS HIRE-DATE
            WITH DUPLICATES ASCENDING
        ALTERNATE RECORD KEY IS JOB-ID
            WITH DUPLICATES ASCENDING
        FILE STATUS IS CODE-RETURN
        USE "RT=Z".
    .
    .
    .
```

Figure 3-8. FILE-CONTROL Paragraph for an Indexed File

```
      .
      .
      .
  DATA DIVISION.
  FILE SECTION.
  FD  EMP-FILE
          LABEL RECORD IS OMITTED
          BLOCK CONTAINS 20 RECORDS
          RECORD CONTAINS 90 CHARACTERS
          DATA RECORD IS EMPLOYEE.
      .
      .
      .
```

Figure 3-9. File Description Entry for an Indexed File

The LABEL RECORDS clause, which is required in every FD entry, specifies whether or not labels exist on the file. Indexed files cannot have labels and the clause must specify OMITTED.

Records in an indexed file are stored in data blocks. The size of a data block is calculated by the system; however, the calculation is affected by the BLOCK CONTAINS clause. The data block size is calculated by rounding upward to a multiple of physical record unit (PRU) size less 50 characters; PRU size is 640 characters. The value used for rounding upward is as follows:

- If the clause is omitted, the value is the maximum record size. (This can be very inefficient; it is usually best to specify the clause.)
- If a number of records is specified, the value is the maximum record size multiplied by the specified number of records.
- If a number of characters is specified, the value is the specified number.

The RECORD clause is used by AAM to determine the record type and record size for input/output processing. If this clause is omitted, the Record Description entry is used. For indexed file organization, the format of the RECORD clause determines record type and record size the same as described for sequential file organization. Refer to table 3-2 for the effect of the RECORD clause.

A File Description entry for an indexed file is shown in figure 3-9. A data block in the file EMP-FILE contains 20 records of 90 characters each for a total of 1800 characters; the actual block size is 1870 characters (three PRUs less 50 characters). The record type is F (fixed length). The DATA RECORD clause documents the name of the record format for the file.

The best maximum block length can be calculated with the FLBLOK utility. Refer to the AAM reference manual for details.

Record Description Entry

A Record Description entry is included in the FD entry for each record format applicable to the indexed file. This entry provides data-names used to access individual data items and describes the physical structure of the record. Indexed file records can be fixed or variable length.

The record type and record size used by AAM are determined by the Record Description entry if the RECORD clause is not specified in the FD entry. The number of Record Description entries and the length of each entry determine record type and size in the same manner as described for sequential file organization.

The Record Description entry illustrated in figure 3-10 defines a record format for the indexed file described in figures 3-8 and 3-9. The primary key EMP-ID and the alternate keys HIRE-DATE and JOB-ID are contained within each EMPLOYEE record. Records can be accessed by any of the three keys. The Record Description entry describes a fixed-length record containing 90 character positions; this corresponds to the record type and record size specified by the RECORD clause in the FD entry.

```
      .
      .
      .
DATA DIVISION.
FILE SECTION.
      .
      .
      .
01  EMPLOYEE.
      03  EMP-ID                PICTURE 999.
      03  EMP-NAME              PICTURE X(20).
      03  EMP-ADDRESS.
          05  STREET            PICTURE X(20).
          05  CITY              PICTURE X(20).
          05  STATE             PICTURE AA.
          05  ZIP-CODE          PICTURE 9(5).
      03  JOB-ID                PICTURE X(5).
      03  DEPT                  PICTURE 999.
      03  DIV                   PICTURE 999.
      03  HIRE-DATE             PICTURE 9(6).
      03  LOCATION              PICTURE 999.
      .
      .
      .
```

Figure 3-10. Record Description Entry for an Indexed File

FILE MANIPULATION

Input/output processing of indexed files is accomplished through the use of seven Procedure Division statements. An existing indexed file can have records read, replaced, deleted, and inserted. Individual data items within a record are manipulated by various statements discussed in other sections of this guide.

Opening Indexed Files

Before any record in an indexed file can be accessed, the file is opened for input, for output, or for input and output. The open mode established by the OPEN statement determines the input/output statements that can be executed. The OPEN statement establishes the primary key as the key of reference; for an existing file, the current value of the key of reference is the primary key value for the first record in the file.

An input file is opened with the OPEN INPUT statement. The file is positioned at the first record; because records are stored in ascending sequence by primary key value, the record with the lowest primary key value is the first record in the file. Records are then read from the file sequentially or randomly depending on the access mode. The collating sequence for an indexed file with alternate keys is determined the first time the file is opened for output; the collating sequence in use within the program when the file is opened becomes the collating sequence for the file and remains the same as long as the file exists.

```
OPEN INPUT EMP-FILE.
```

When this statement is executed, the file EMP-FILE is opened for input processing. Records can be read from but not written on the file.

An indexed file is created by opening the file for output. Records are then written on the file randomly or sequentially depending on the access mode.

```
OPEN OUTPUT INV-FILE.
```

Execution of this statement causes the file INV-FILE to be opened in the output mode for file creation. Records can only be written on the file.

An indexed file is opened for input and output processing by the OPEN I-O statement. The file is positioned at the first record; the primary key of the first record becomes the current key of reference.

```
OPEN I-O EMP-FILE.
```

This statement causes the file EMP-FILE to be opened for input and output. Records in the file can be read, deleted, inserted, and updated.

Writing Indexed Files

When an indexed file is opened for output (OPEN OUTPUT for file creation or OPEN I-O for file updating), records are written on the file sequentially or randomly depending on the access mode for the file. If the access mode is sequential, the file must be open for output only. Records are then written on the file sequentially; the records must be in ascending sequence by primary key value. An invalid key condition exists if the primary key value of the record being written is not greater than the primary key value of the previous record. If duplicate alternate keys are not allowed, duplication of an alternate key value creates an invalid key condition.

```
WRITE EMPLOYEE
    INVALID KEY GO TO BAD-RECORD.
```

This statement causes an EMPLOYEE record to be written on the indexed file. In sequential access mode, the records must be in ascending sequence by primary key value. An invalid key condition causes control to be transferred to the paragraph named BAD-RECORD; the record is not written on the file.

Records are written randomly when the access mode is random or dynamic. The records are placed in the file according to the primary key values. When the file is closed, the records are in order by ascending sequence of primary key values. An invalid key condition exists if the primary key value is not unique or if an alternate key value is duplicated when the NO DUPLICATES phrase is used in the ALTERNATE RECORD KEY clause. The format of the WRITE statement is the same for sequential and random writing. Creating an indexed file randomly can result in very inefficient processing; it is always best to create the file sequentially.

When alternate keys are defined for the indexed file without the USE or OMITTED phrase, an entry is made in the alternate key index file for each alternate key in the record being written. When the USE or OMITTED phrase is included in the definition, entries are made for keys according to conditions specified in the phrase. For a repeating group alternate key, an entry is made for each occurrence of the alternate key.

The FROM phrase is included in the WRITE statement when the data for the record to be written is stored in an area other than the output record area. The data in the specified area is moved to the record area before the record is written.

```
WRITE STOCK-REC FROM NEW-REC
    INVALID KEY GO TO DUP-KEY.
```

The data in the storage area named NEW-REC is moved to the STOCK-REC record area; the record is then written on the file. If the primary key value for the record to be written already exists on the file, control is transferred to the paragraph named DUP-KEY.

Positioning Indexed Files

Sequential processing of an indexed file can begin at a position other than the first record in the file. The access mode must be sequential or dynamic and the file must be open for input (OPEN INPUT or OPEN I-O).

The START statement positions the file at the record that satisfies a specified condition and establishes the key of reference for subsequent sequential READ statements. The primary key, an alternate key, or the leading portion of either key can be specified in the relational condition. The designated primary or alternate key becomes the key of reference when the START statement is executed. The value of the key in the record at which the file is positioned becomes the current value of the key of reference.

The KEY phrase specifies the data item and the relational condition to be tested. If the phrase is omitted, the primary key is the key of reference and the file is positioned at the record with the primary key value equal to the current value of the primary key data item. When the KEY phrase is specified, the file is positioned at the first record with a key value that is equal to, greater than, or not less than the current value of the designated key data item.

```
START EMP-FILE
    KEY IS GREATER THAN EMP-ID.
```

When this statement is executed, the file EMP-FILE is positioned at the first record with a primary key value that is greater than the current value of the EMP-ID data item. Sequential processing of the file then begins at that position.

A repeating group alternate key can be used to position the file; however, the data-name of the alternate key cannot be subscripted or indexed in the START statement. The current value in the first occurrence of the alternate key data item is the value that is used to position the file. In the record at which the file is positioned, the value satisfying the condition can be in any occurrence of the alternate key.

The data item specified in the KEY phrase can be the first subordinate item of the primary key. It can be the first subordinate item of an alternate key if the alternate key begins in a unique location. If two alternate keys begin in the same character position, the KEY phrase cannot specify an item subordinate to either key. If a subordinate item is specified, it must begin in the first character position of the key field and must be described as an alphanumeric data item. For example:

```
03  HIRE-DATE.
    05  YEAR       PICTURE XX.
    05  MONTH      PICTURE 99.
    05  DAE        PICTURE 99.
```

These three entries describe an alternate key. The KEY phrase of the START statement can specify either HIRE-DATE or YEAR; the data items MONTH and DAE cannot be specified.

```
START EMP-FILE
    KEY IS NOT LESS THAN YEAR.
```

In this statement, the KEY phrase specifies the leading portion of the alternate key HIRE-DATE. Only the first two characters of the alternate key values in the index file are checked against the two characters of the current value of the YEAR data item. The file is positioned at the first record with a YEAR value that is equal to or greater than the current value of YEAR.

The INVALID KEY phrase specifies the action to be taken when no record in the file satisfies the condition of the START statement.

```
START STOCK-FILE
    INVALID KEY GO TO BAD-ID.
```

The primary key values in the file STOCK-FILE are checked for a value equal to the current value of the primary key data item. If no record in the file satisfies this condition, control is transferred to the paragraph named BAD-ID.

Reading Indexed Files

Once an indexed file has been opened for input (OPEN INPUT or OPEN I-O), records are read from the file by the READ statement. Depending on the access mode established for the file and the format of the READ statement, records are read sequentially or randomly.

Accessing Sequentially

Records in an indexed file can be accessed sequentially when the access mode is established as sequential or dynamic. When the READ statement is executed, the order in which records are retrieved depends on the key of reference. The key of reference is determined as follows:

- When the file is opened, the primary key is the key of reference.
- If the file is positioned by the START statement, the primary or alternate key used to position the file becomes the key of reference.
- In dynamic access mode, a random read executed before the sequential read establishes the key used for the random read as the key of reference.

When the primary key is the key of reference, records are retrieved in the order they are stored in the file. In an indexed file, records are stored in ascending sequence by primary key value. When the primary key is the key of reference and it is defined in the Working-Storage Section (rather than embedded in the record), the primary key value is stored in the data item named in the RECORD KEY clause. If an alternate key is the key of reference, records are retrieved in the order of the key values in the alternate key index file.

If the access mode is sequential, the first record read is either the first record in the file or the record at which the file has been positioned by the START statement. Records are then read in sequence according to the key of reference until the end of the file is reached. The FILE STATUS clause can be used to determine the final occurrence of a particular value for the key of reference by testing for a status code of 02.

```
READ EMP-FILE RECORD
    AT END GO TO FINISHED.
```

This statement causes the records in the file EMP-FILE to be read sequentially; the access mode for EMP-FILE is sequential. Records are read in stored order if the primary key is the key of reference or in index file order if an alternate key is the key of reference. When the end of the file is reached, control is transferred to the paragraph named FINISHED.

If the access mode is dynamic, the first record retrieved by a sequential READ statement is one of the following:

- The first record in the file.
- The record at which the file has been positioned by the START statement.
- The next record in sequence according to the key of reference used in the preceding random READ statement.

Subsequent records are retrieved sequentially by stored position (if the primary key is the key of reference) or by the order of alternate key values in the index file (if an alternate key is the key of reference). The keyword NEXT must be included in a sequential READ statement when the access mode is dynamic. A change in values for the key of reference can be detected through the FILE STATUS clause.

```
READ STOCK-FILE NEXT RECORD
    AT END GO TO CLOSING.
```

The access mode for the file STOCK-FILE is dynamic. This statement causes records in STOCK-FILE to be retrieved sequentially according to the key of reference. When the end of the file is reached, control is transferred to the paragraph named CLOSING.

The INTO phrase is included in the READ statement to store the record in a specified area as well as in the input record area. The record is moved into the specified storage area when the READ statement is executed. When the file is defined by more than one Record Description entry, the INTO phrase cannot be used if any entry is a level 01 elementary item that is described as a numeric or numeric-edited data item.

```
READ INV-FILE NEXT RECORD INTO TEMP-REC.
```

When this statement is executed, the next record in sequence is read from the file INV-FILE. The record is stored in the input record area and in the storage area named TEMP-REC.

Accessing Randomly

Records in an indexed file can be read randomly when the access mode is random or dynamic. The primary key or an alternate key can be the key of reference for reading a record randomly.

The key of reference for a random read is established by the KEY IS phrase of the READ statement. If this phrase is not specified, the primary key is the key of reference. The key of reference designated by the KEY IS phrase can be either the primary key or an alternate key. When the READ statement is executed, the current value of the key of reference is compared with the key values of records in the file. The first record that contains a key of equal value is retrieved from the file. If no record contains a key of equal value, an invalid key condition exists.

```
READ EMP-FILE RECORD
    KEY IS JOB-ID
    INVALID KEY GO TO NO-JOB-ID.
```

In the file EMP-FILE, the data item JOB-ID is an alternate key and duplicate keys are allowed. Execution of this READ statement causes the index file to be searched for the first alternate key value that is equal to the current value of the JOB-ID data item. If no record in the file has an equal value, control is transferred to the paragraph named NO-JOB-ID. When this statement executes successfully, a sequential read can be executed to retrieve the next record in alternate key sequence.

When a repeating group alternate key is specified in the KEY IS phrase, the data-name of the alternate key cannot be subscripted or indexed. The current value in the first occurrence of the alternate key data item is the value that is used to retrieve a record from the file. When the READ statement is executed, the index file is searched for the matching alternate key value and the first record with that value is read from the file. The matching value can be in any occurrence of the alternate key in the record that is retrieved. A sequential read can then be performed to retrieve the next record in sequence in the alternate key index. Because a record has multiple values for a repeating group alternate key, the same record can be retrieved more than once.

A record read randomly can be stored in a specified area by including the INTO phrase in the READ statement. The record is then available in both the input record area and the specified storage area.

```
READ EMP-FILE RECORD INTO NEW-REC
    INVALID KEY GO TO BAD-KEY.
```

This statement reads a record randomly; because the KEY IS phrase is not specified, the primary key is the key of reference. After a successful read, the record is available in the storage area named NEW-REC as well as in the input record area. If the file does not contain a record with a primary key equal to the current value of the key of reference, control is transferred to the paragraph named BAD-KEY.

Updating Indexed Files

The DELETE and REWRITE statements are used to update existing records in indexed files. The WRITE statement is used to write additional records on the file; it cannot be used to replace an existing record. The file must be open for input and output (OPEN I-O); any access mode is allowed.

The DELETE statement removes a record from the indexed file. Once the DELETE statement is executed, the record can no longer be accessed. The record deleted is either the last record read or the record with the primary key equal to the current value of the primary key data item.

If the access mode is sequential, the input/output statement preceding the DELETE statement must be a valid sequential READ statement. The last record read is then the record that is deleted.

```
READ EMP-FILE RECORD.
.
.
.
DELETE EMP-FILE RECORD.
```

The record read from the file EMP-FILE is removed from the file when the DELETE statement is executed. The record can no longer be accessed.

Records are deleted by primary key when the access mode is random or dynamic. The current value of the primary key data item identifies the record to be deleted. The INVALID KEY phrase is included to specify the action to be taken if the file does not contain the record to be deleted.

```
DELETE EMP-FILE RECORD
    INVALID KEY GO TO NO-RECORD.
```

This statement deletes the EMP-FILE record whose primary key is equal to the current value of the primary key data item. If no record in the file contains the designated primary key value, control is transferred to the paragraph named NO-RECORD.

The REWRITE statement is used to update data in an existing record in the file. The primary key value identifies the record to be rewritten. The data in the record area replaces the data stored in the file. If the FROM phrase is included in the REWRITE statement, the data in the specified storage area is moved to the record area before the record is rewritten.

If the access mode is sequential, the input/output statement preceding the REWRITE statement must be a sequential READ statement. The record replaced is then the last record read. The primary key value cannot be changed between execution of the READ statement and execution of the REWRITE statement.

```
READ INV-FILE RECORD INTO UPD-REC.
.
.
.
REWRITE INV-REC FROM UPD-REC.
```

A record is read from the file INV-FILE and stored in the area named UPD-REC. Statements executed before the REWRITE statement can update fields other than the primary key field in UPD-REC. When the REWRITE statement is executed, the data in UPD-REC is moved to the record area (INV-REC) and the updated record replaces the record read by the previous READ statement.

If the access mode for the file is random or dynamic, the record to be replaced is identified by the current value of the primary key data item, which must correspond to the primary key of an existing record in the file. If the primary key does not identify an existing record, an invalid key condition exists. When the file contains alternate keys, the replacing record can specify new values for the existing key items; however, the new key values cannot duplicate any current alternate key values in the file unless the DUPLICATES phrase is included in the key definitions.

```
REWRITE EMPLOYEE
    INVALID KEY GO TO BAD-KEY.
```

Execution of this statement causes the data in the EMPLOYEE record area to replace the record that contains the primary key value currently stored in the primary key data item. If the file has no record with the current primary key value, control is transferred to the paragraph named BAD-KEY.

Closing Indexed Files

When a CLOSE statement is executed for an indexed file, processing of the file is terminated. AAM updates the internal tables that are part of the file. If the file is subsequently reopened, it is positioned at the first record in the file.

When the WITH LOCK phrase is included in the CLOSE statement, the file is returned to the system and cannot be reopened during the execution of the current control statement. If an attempt is made to reopen the file within the same program, the program is aborted.

If the file is to be reopened immediately after the close and the WITH LOCK phrase is not specified in the CLOSE statement, the routine C.FILE should be entered. Through this routine, the CF field of the FIT can be reset from the default setting DET to the setting R, thus preventing the release to the system of buffer space and AAM capsules needed for the file.

```
ENTER "C.FILE" USING EMP-FILE, "CF=R".
    CLOSE EMP-FILE.
```

Execution of these statements causes the file EMP-FILE to be closed and buffer space and system capsules associated with the file to be retained by the program.

DIRECT FILE ORGANIZATION

When file organization is direct, records are stored randomly in blocks on a mass storage device. Each record contains a primary key field; the value in this field is hashed to a number that indicates a home block in the file. The hashing technique uses a formula to convert the primary key value to a number that distributes records evenly across the home blocks. A hashing routine is provided by the system; however, a user hashing routine can be supplied in the Declaratives portion of the Procedure Division.

The primary key must be specified for a direct file. In addition, alternate keys can be specified and used to access the file. Alternate keys are not hashed; the values are indexed on a separate file that must be maintained as a permanent file. The access mode for a direct file can be sequential, random, or dynamic.

Direct file organization is used most effectively for large mass storage files requiring rapid random access. A direct file can be read sequentially; however, the order of the records has no relationship to the primary key values or to the order in which the records were written.

FILE DEFINITION

The structure of a file with direct organization is specified through the FILE-CONTROL paragraph in the Environment Division and the File Description and Record Description entries in the Data Division.

FILE-CONTROL Paragraph

In the FILE-CONTROL paragraph for a file with direct organization, five clauses are required: SELECT, ASSIGN, ORGANIZATION, BLOCK COUNT, and RECORD KEY. Four optional clauses can also be included in this paragraph. Refer to figure 3-11.

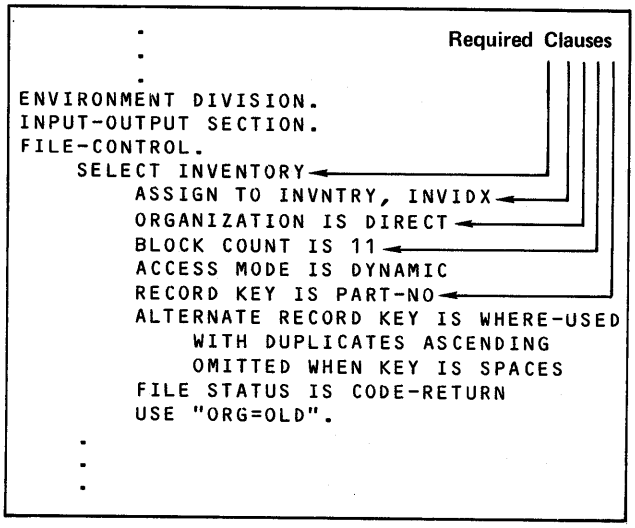


Figure 3-11. FILE-CONTROL Paragraph for a Direct File

The SELECT clause specifies the file-name used in the COBOL program. The ASSIGN clause specifies the logical file name recognized by the operating system. When alternate keys are specified for the file, the ASSIGN clause must also include the logical file name of the alternate key index file. If either logical file name is the same as any other name in the program or as a reserved word, it must be enclosed in quotation marks. The ORGANIZATION clause must specify DIRECT for a direct file.

The BLOCK COUNT clause is used only when the direct file is opened in the output mode; it is ignored if it is specified at any other time. This clause designates the number of home blocks for the file being created. The specific block in which a record is written is determined by the hashing routine. For the system hashing routine, more efficient processing results when the number of blocks is a prime number.

The primary key for direct file records is specified by the RECORD KEY clause. The data item designated as the primary key must be a fixed-length data item embedded in each record or within the Working-Storage Section. The primary key data item can be an alphanumeric elementary or group item or it can be an unsigned numeric elementary item. The primary key value for each record must be a unique value; duplicate primary keys are not allowed in a direct file. When a record is written on the file, the hashing routine uses the primary key value to determine the home block for the record.

The ALTERNATE RECORD KEY clause specifies an alternate key for the direct file. Multiple alternate keys are specified by repeating the clause for each desired key. An alternate key must be a data item contained in a direct file record. When the primary key is contained in the record, it must begin in a unique character position if they are different lengths; alternate keys of the same length must begin in unique positions. Duplicate alternate key values are not allowed unless the DUPLICATES phrase is included in the ALTERNATE RECORD KEY clause.

The ASCENDING option of the DUPLICATES phrase determines the order of retrieval when records with duplicate alternate key values are read sequentially by alternate key. If the option is omitted, records with duplicate alternate keys are retrieved in the order they were written. Including the ASCENDING option causes the records with duplicate alternate key values to be retrieved in ascending sequence by primary key value.

The condition specified in the USE phrase or the OMITTED phrase determines whether or not an entry is made in the alternate key index file for each alternate key in the data file. When neither phrase is included, an entry is made for every key in the file. The USE WHEN phrase specifies a data-name and an alphanumeric literal. The item referenced by the data-name must be defined within the record as a one-character alphanumeric item. The same data-name can be used in more than one ALTERNATE RECORD KEY clause for a file. The literal must contain from one to 36 unique characters. When the character in the data item of an alternate key duplicates a character in the literal, an entry is made for the key in the alternate key index. If the character in the item is not present in the literal, no entry is made.

The OMITTED phrase of the ALTERNATE RECORD KEY clause can specify either a data-name and literal, as in the USE phrase, or the KEY option. When the data-name and literal are included in the phrase, no entry is made in the index file for an alternate key if the character in the data item duplicates any character in the literal. The data item and literal are set up in the same manner as described in the USE phrase.

The KEY option of the OMITTED phrase can specify either SPACES or ZEROS. SPACES indicates that no entry is to be made in the index file for any key with the usage of DISPLAY and a value of spaces. When ZEROS is specified, no entry is made for a key described as COMPUTATIONAL-1 or COMPUTATIONAL-2 with a value of zero.

An alternate key described with the OCCURS clause has more than one value for the alternate key. When the record is written, the value in each unique occurrence of the alternate key is indexed on the alternate key index file. Reading the file by alternate key can then retrieve the record for the value in any occurrence of the alternate key.

The ACCESS MODE clause determines the manner in which records in the direct file can be accessed during program execution. If SEQUENTIAL is specified or if the clause is omitted, records can only be accessed sequentially. Records are accessed randomly by primary or alternate key value when RANDOM is specified. If DYNAMIC is specified, both random and sequential processing can be intermixed in the program.

The FILE STATUS clause is used to specify a data item to receive a status code whenever an input/output statement is executed for the file. The status code is a value that designates whether or not the statement executed successfully. Refer to section 15 for a description of the status code.

The USE clause supplies file information used by AAM to process the direct file. Certain FILE control statement parameters can be specified in this clause. These parameters can supply file information that cannot be specified through other clauses and statements in the source program, or they can override parameter values normally obtained from the source program. The parameter list is enclosed in quotation marks. Refer to section 15 for a complete list of parameters that can be specified.

The type of direct file to be used is determined by the ORG parameter of the USE clause. If extended AAM files have been installed and either the parameter is omitted or ORG=NEW is specified, the file is treated as an extended direct file. ORG=OLD is required for files in the initial direct format.

A FILE-CONTROL paragraph for a file with direct organization is illustrated in figure 3-11. The direct file is referenced in the COBOL 5 program by the file-name INVENTORY; the logical file name used by the operating system is INVNTRY. The index file for the alternate keys is identified by the logical file name INVIDX. The number of home blocks for the file is 11. Dynamic access allows the file to be processed sequentially and randomly during program execution. The hashed value of the primary key (PART-NO) determines the home block for the record. The alternate key (WHERE-USED) can be duplicated within the file; duplicate alternate key values are retrieved in ascending sequence of the primary key values.

File Description Entry

The physical structure of a direct file is defined by a File Description entry (FD entry) in the File Section of the Data Division. The program file-name specified in the SELECT clause is also specified in the FD entry. Three clauses in the FD entry are applicable to files with direct organization. Refer to figure 3-12.

```
      .
      .
      .
DATA DIVISION.
FILE SECTION.
FD   INVENTORY
          LABEL RECORDS ARE OMITTED
          BLOCK CONTAINS 20 RECORDS
          RECORD CONTAINS 55 TO 80
          DATA RECORD IS INV-REC.
```

Figure 3-12. File Description Entry for a Direct File

The LABEL RECORDS clause is required in every FD entry and specifies whether or not the file contains labels. Direct files cannot contain labels and the clause must specify OMITTED.

Direct file records are stored in home blocks. The size of a home block is calculated by the system using the BLOCK CONTAINS clause. The home block size is calculated by rounding upward to a multiple of physical record unit (PRU) size less 50 characters; PRU size is 640 characters. The value used for rounding upward is as follows:

- If the clause is omitted, the value is the average record size multiplied by two. (This can result in inefficient processing; it is best to specify the clause.)
- If a number of records is specified, the value is the maximum record size multiplied by the specified number of records.
- If a number of characters is specified, the value is the specified number.

The RECORD clause is used by AAM in determining the record type and record size for input/output processing of direct files. If this clause is not specified, the Record Description entry is used. The record type and record size for a direct file are determined according to the specific format of the RECORD clause in the same manner as described for sequential file organization. Refer to table 3-2 for the effect of the RECORD clause.

A File Description entry for a direct file is illustrated in figure 3-12. Each home block can contain 20 records with lengths ranging from 55 to 80 characters. The home block size is rounded upward to 1870 characters (three PRUs minus 50 characters). The DATA RECORD clause is documentary only and provides the record name (INV-REC) for the file.

Record Description Entry

Each record format applicable to the direct file is described by a Record Description entry. This entry specifies the physical structure of the record and provides the data-names used to access data items within the record. Direct file records can be fixed or variable length.

When the RECORD clause is not specified in the FD entry, the Record Description entry is used to determine the record type and record size for input/output processing. The number of Record Description entries for the file and the length of each entry determine record type and size in the same manner as described for sequential file organization.

The Record Description entry illustrated in figure 3-13 defines the record format for the direct file described in figures 3-11 and 3-12. Both the primary key (PART-NO) and the alternate key (WHERE-USED) are defined within the record. Either key can be used to access an INV-REC record. The entry describes a record with a 50-character fixed-length portion and a 5-character trailer portion that occurs from one to six times. The NUM-USED data item indicates the number of occurrences of the trailer portion in a specific record.

```
      .
      .
      .
DATA DIVISION.
FILE SECTION.
      .
      .
      .
01    INV-REC.
      03   PART-NO              PICTURE 9(5).
      03   DESCRIPTION          PICTURE X(15).
      03   QTY-ON-HAND          PICTURE 9(4).
      03   QTY-ON-ORDER         PICTURE 9(4).
      03   QTY-RESERVED         PICTURE 9(5).
      03   ORDER-DATE           PICTURE 9(6).
      03   REORDER-POINT        PICTURE 9(4).
      03   REORDER-QTY          PICTURE 9(4).
      03   QTY-PER-UNIT         PICTURE 99.
      03   NUM-USED             PICTURE 9.
      03   WHERE-USED           PICTURE X(5).
                OCCURS 1 TO 6 TIMES
                DEPENDING ON NUM-USED.
      .
      .
      .
```

Figure 3-13. Record Description Entry for a Direct File

FILE MANIPULATION

Input/output processing of direct files is specified through seven Procedure Division statements. Records in an existing direct file can be read, replaced, deleted, and inserted. Various statements, which are discussed in other sections of this guide, are provided to manipulate individual data items in the records.

Opening Direct Files

A direct file is opened for input, for output, or for input and output before file processing can begin. The input/output statements that can be executed depend on the open mode established by the OPEN statement. The primary key is established as the key of reference when the OPEN statement is executed; for existing files, the current value of the key of reference is the primary key value for the first record in the file.

The OPEN INPUT statement opens a direct file for input processing only. Records can be read from but not written on the file. Execution of the OPEN INPUT statement positions the file at the first record in the first home block. Records are then read sequentially by position in the file or randomly by key value depending on the access mode.

```
OPEN INPUT INVENTORY.
```

When this statement is executed, the file INVENTORY is opened for input. The file is positioned at the first record.

If a direct file is being created, the file is opened for output by the OPEN OUTPUT statement. Records can be written on but not read from the file. Records are written according to the hashed value of the primary key.

```
OPEN OUTPUT CUSTOMERS.
```

This statement causes the file CUSTOMERS to be opened for output. Records are subsequently written on the file at locations indicated by the primary key hashed values.

The OPEN I-O statement opens the file for input and output processing. Records can be read, inserted, deleted, or updated. The file is positioned at the first record currently existing in the file.

```
OPEN I-O PERS-FILE.
```

Execution of this statement opens the file PERS-FILE for input and output processing. Records can then be read, inserted, deleted, and rewritten.

Writing Direct Files

Records are written on a direct file at locations determined by the hashed values of the primary keys. The file must be open for output (OPEN OUTPUT for file creation or OPEN I-O for file updating) and can have any access mode. When a direct file is being created, processing is more efficient if records are written in the sequence of hashed key values. An AAM utility, CREATE, is available to create direct files efficiently. Refer to the AAM reference manual for a description of the CREATE utility.

The value of the primary key in the record being written is hashed to determine the home block in which the record is written. The hashing routine, which can be the system routine or a user-supplied routine, converts the primary key value to a home block number within the limits specified in the BLOCK COUNT clause in the FILE-CONTROL paragraph. The record is then written in the home block indicated by the hashed value.

If the home block designated by the hashed value is full, an overflow block is created and the record is written in the overflow block. Retrieving a record in an overflow block increases access time because an additional mass storage access is required to read the record.

When the direct file is closed, the physical order of the records is completely random. Primary key values and the order in which the records are written have no effect on the stored order of the records.

When alternate keys are defined for the direct file without the USE or OMITTED phrase, the primary key is entered in the alternate key index file for each alternate key in the record being written. When the USE or OMITTED phrase is included in the key definition, an entry is made in the alternate key index file according to the condition specified in the phrase. For a repeating group alternate key, an entry is made in the index file for each unique occurrence of the alternate key in a record.

An invalid key condition occurs if the primary key value is duplicated or if the hashed value is greater than the number of home blocks allocated for the file. If duplicate alternate keys are not allowed, duplication of an alternate key value also creates an invalid key condition.

```
WRITE INV-REC
    INVALID KEY GO TO BAD-RECORD.
```

When this statement is executed, the record INV-REC is written on its associated file. The hashed value of the primary key designates the home block in which the record is written. If an invalid key condition exists, control is transferred to the paragraph named BAD-RECORD.

The FROM phrase is included in the WRITE statement when the data for the record to be written is stored in an area other than the output record area. The data in the specified area is moved to the output record area and then the record is written on the file.

```
WRITE PERS-REC FROM NEW-REC
    INVALID KEY GO TO NO-GOOD.
```

This statement causes the data in the storage area named NEW-REC to be moved to the record area for PERS-REC. The record is then written in the home block indicated by the hashed value of the primary key. If an invalid key condition is encountered, control is transferred to the paragraph named NO-GOOD.

Positioning Direct Files

Direct files can be positioned to a record within the file for subsequent sequential processing. The file must be open for input (OPEN INPUT or OPEN I-O) and the access mode must be either sequential or dynamic.

The START statement establishes the key of reference for subsequent sequential READ statements and positions the file at the first record that satisfies a specified condition. The KEY phrase indicates the data item and the condition to be used for positioning the file. The data item must be an alternate key or the leading portion of an alternate key; it cannot be the primary key. When the START statement is executed, the index file is searched for a value that is greater than, equal to, or not less than the current value of the designed data item. The file is positioned at the first record that satisfies the specified condition.

```
START PERS-FILE
    KEY IS NOT LESS THAN HIRE-DATE.
```

Execution of this statement causes the index for the alternate key HIRE-DATE to be searched for a key value that is equal to or greater than the current value of the HIRE-DATE data item. The index file is positioned at the first alternate key value that satisfies the condition. Records can then be retrieved from the direct file in the sequential order of the alternate keys in the index file.

When a repeating group alternate key is specified in the KEY phrase, the data-name is not subscripted or indexed. The current value in the first occurrence of the alternate key data item is used to position the file. The value that satisfies the specified condition can be in any occurrence of the alternate key in the record at which the file is positioned.

The data item specified in the KEY phrase can be the leading portion of an alternate key if the alternate key begins in a unique character position. If two alternate keys begin in the same position, the KEY phrase cannot specify an item subordinate to either key. When the leading portion of an alternate key is used in the KEY phrase, it must begin in the first character position of the alternate key and must be described as an alphanumeric data item.

```
03  DIV-CODE.
    05  LOCATION  PICTURE XXX.
    05  FUNCTION  PICTURE 9(5).
```

The alternate key DIV-CODE is described with two subordinate items. Either DIV-CODE or LOCATION can be specified in the KEY phrase.

The INVALID KEY phrase is included in the START statement to designate the action to be taken when the specified condition is not satisfied by any record in the file.

```
START EMP-FILE
    KEY IS GREATER THAN LOCATION
    INVALID KEY GO TO BAD-KEY.
```

When this statement is executed, the alternate key values in the index file are tested for a value greater than the current value of LOCATION, which is the first three character positions of the alternate key data item DIV-CODE. If the condition cannot be satisfied, control is transferred to the paragraph named BAD-KEY.

Reading Direct Files

Records are retrieved from a direct file by the READ statement. The file must be open for input (OPEN INPUT or OPEN I-O). The access mode and the format of the READ statement determine whether records are read sequentially or randomly.

Accessing Sequentially

When the access mode for a direct file is sequential or dynamic, records can be read sequentially. The sequence in which records are read depends on the key of reference at the time the READ statement is executed. The key of reference is determined as follows:

- When the file is opened, the primary key is the key of reference.
- If the file has been positioned by the START statement, the alternate key used to position the file becomes the key of reference.
- In dynamic access mode, a random read executed before the sequential read establishes the key used for the random read as the key of reference.

If the key of reference is the primary key, the records are read in the order they are stored in the file. This order bears no relationship to the primary key values or to the order in which the records were written. If the primary key is the key of reference and it is defined in the Working-Storage Section (rather than embedded in the record), the primary key value is stored in the data item designated by the RECORD KEY clause. When an alternate key is the key of reference, the records are read in the order of the key values in the alternate key index file.

If the access mode is sequential and the key of reference is the primary key, records are read in stored sequence from the beginning of the file or from the record that satisfied the condition in the START statement. An alternate key as the key of reference causes reading to begin with the record at which the file has been positioned by the START statement; the sequential read then proceeds in the order of the alternate key values in the index file. The FILE STATUS clause, discussed in section 15, can be used to detect a change of values for the key of reference by testing for a status code value of 02.

```
READ INVENTORY RECORD
    AT END GO TO END-IT.
```

This statement reads records sequentially. The file INVENTORY is a direct file with sequential access mode. If the primary key is the key of reference, the records are read in the order they are stored in the home blocks of the file. If an alternate key is the key of reference, the records are read in sequence by the alternate key values in the index file. Control is transferred to the paragraph named END-IT when the end of the file has been reached.

If the access mode is dynamic, the first record retrieved by a sequential READ statement is one of the following:

- The first record in the first home block of the file.
- The record at which the file has been positioned by the START statement.
- The next record in sequence according to the key of reference used in the preceding random READ statement.

Subsequent records are retrieved sequentially by stored position (if the primary key is the key of reference) or by the order of alternate key values in the index file (if an alternate key is the key of reference). The keyword NEXT must be included in a sequential READ statement when the access mode is dynamic. A change in the value of the key of reference can be determined through the FILE STATUS clause, discussed in section 15.

```
READ PERS-FILE NEXT RECORD
    AT END GO TO FINISHED.
```

Before this statement is executed, the access mode for the file PERS-FILE has been established as dynamic and a record has been randomly read by alternate key. Execution of this statement then reads the next record in sequence according to the alternate key index file. When the last record has been read, control is transferred to the paragraph named FINISHED.

The INTO phrase is included in a sequential READ statement when the input record is to be stored in a specified area in addition to the input record area. When the READ statement is executed, the record is moved into the specified storage area and is then available in the storage area and in the input record area. When the file is defined by more than one Record Description entry, the INTO phrase cannot be used if any entry is a level 01 elementary item that is described as a numeric or numeric-edited data item.

```
READ INVENTORY NEXT RECORD
    INTO TEMP-REC.
```

Execution of this statement causes the next record in sequence according to the key of reference to be read from the input file INVENTORY. The record is available in both the input record area and the storage area named TEMP-REC.

Accessing Randomly

The access mode for a direct file must be random or dynamic to access records randomly. Either the primary key value or an alternate key value is used to read a record randomly.

The KEY IS phrase of a random READ statement establishes the key of reference. The data-name specified in the phrase identifies the primary key or an alternate key. If the KEY IS phrase is omitted, the primary key is the key of reference. When the READ statement is executed, the current value of the key of reference determines the record to be read. An invalid key condition exists if no record in the file contains a key of equal value.

When the key of reference is an alternate key with duplicate values, the record with the first primary key indexed for the alternate key value is the record that is retrieved. The order in which the primary keys are indexed depends on whether or not the ASCENDING option is specified in the ALTERNATE RECORD KEY clause. If ASCENDING is specified, the primary keys are indexed in ascending sequence; otherwise, the primary keys are indexed in the order the records were written on the file.

```
READ PERS-FILE RECORD
    KEY IS HIRE-DATE
    INVALID KEY GO TO NONE.
```

When this statement is executed, a record is read from the file PERS-FILE. The alternate key HIRE-DATE, which can have duplicate values, is the key of reference for the random read. The index file is searched for the alternate key value equal to the current value of the HIRE-DATE data item. The record with the first primary key indexed for that value is then read from the file. If an alternate key of equal value does not exist on the index file, control is transferred to the paragraph named NONE.

A repeating group alternate key can be specified in the KEY IS phrase; however, the data-name of the alternate key cannot be subscripted or indexed. The index file is searched for a value equal to the current value in the first occurrence of the alternate key data item. The record with the first primary key indexed for the alternate key value is read from the file. The value can be in any occurrence of the alternate key in the record read from the file. When records are read by repeating group alternate key values, the same record can be retrieved for the value in each occurrence of the alternate key.

```
READ INVENTORY RECORD
    KEY IS WHERE-USED
    INVALID KEY GO TO NOT-FOUND.
```

The key of reference for this statement is the data item WHERE-USED, which is a repeating group alternate key for the file INVENTORY. When this statement is executed, the alternate key index for WHERE-USED is searched for a value equal to the current value in the first occurrence of the WHERE-USED data item. If no key value in the alternate key index equals the current value in WHERE-USED, control is transferred to the paragraph named NOT-FOUND.

The INTO phrase in a random READ statement specifies an additional storage area for the input record. The record retrieved is stored in both the input record area and the specified storage area.

```
READ INVENTORY RECORD INTO NEW-REC
    INVALID KEY GO TO NO-USE.
```

This statement reads a record randomly from the file INVENTORY and stores it in the storage area named NEW-REC as well as in the input record area. The KEY IS phrase is omitted and the primary key is the key of reference by default. If the read operation is not successful, control is transferred to the paragraph named NO-USE.

Updating Direct Files

Records in an existing direct file are updated by the DELETE and REWRITE statements. New records are added to the file by the WRITE statement; existing records cannot be replaced through execution of the WRITE statement. The file can have any access mode; it must be opened for input and output (OPEN I-O).

A record is removed from the direct file by the DELETE statement. Depending on the access mode, the record deleted is either the last record read or the record with the primary key value equal to the current value of the key of reference.

If the access mode is sequential, a sequential READ statement must be the last input/output statement executed before the DELETE statement. The record accessed by the READ statement is then the record that is deleted.

```
READ INVENTORY RECORD.
.
.
.
DELETE INVENTORY RECORD.
```

When the DELETE statement is executed, the record retrieved by the READ statement is removed from the file INVENTORY. The record can no longer be accessed.

For random or dynamic access mode, records are deleted by primary key value. The current value of the primary key data item identifies the record to be deleted. An invalid key condition exists if the record to be deleted cannot be found on the file.

```
DELETE PERS-FILE RECORD
    INVALID KEY GO TO NOT-FOUND.
```

Execution of this statement deletes the record in the file PERS-FILE with the same primary key value as the current value of the primary key data item. If the record does not exist on the file, control is transferred to the paragraph named NOT-FOUND.

Existing records in a direct file can be updated with new data by the REWRITE statement. The record to be rewritten is identified by the primary key. The record stored on the file is replaced by the data in the record area. The FROM option is included in the REWRITE statement when the data to be rewritten is stored in an area other than the record area. The data in the specified storage area is moved to the record area and the record is then rewritten.

For sequential access mode, the last record read is the record that is replaced. Between execution of the sequential READ statement and the REWRITE statement, no other input/output statement can be executed and the primary key value cannot be changed.

```
READ INVENTORY RECORD INTO UPDATING.
.
.
.
REWRITE INV-REC FROM UPDATING.
```

When the READ statement is executed, the next record in sequence is read from the file INVENTORY and stored in the area named UPDATING. Data manipulation statements are then executed to update fields in UPDATING other than the primary key field. The REWRITE statement causes the updated record stored in UPDATING to be moved to the record area for INV-REC and the record to be rewritten in place of the record retrieved by the READ statement.

For random or dynamic access mode, the current value of the primary key data item identifies the record to be replaced. The current primary key value must correspond to the primary key value of an existing record in the file. An invalid key condition exists when the current primary key value does not identify an existing record. Alternate key values in the replacing record can differ from those in the record being replaced, but the new values cannot duplicate any existing values in the file unless the DUPLICATES option is included in the key definition.

```
REWRITE PERS-REC
    INVALID KEY GO TO BAD-KEY.
```

When this statement is executed, the record containing the primary key value currently stored in the primary key data item is replaced by the data in the PERS-REC record area. If no record in the file contains the primary key value, control is transferred to the paragraph named BAD-KEY.

Closing Direct Files

Processing of a direct file is terminated by the CLOSE statement. AAM updates the internal tables that are part of the file. Input/output statements cannot access the file until it has been opened again.

The WITH LOCK phrase of the CLOSE statement prevents a file from being opened again during the execution of the current control statement. When the CLOSE statement with the WITH LOCK phrase is executed, the file is closed and returned to the system. An attempt to reopen the file results in a program abort.

The utility routine C.FILE should be entered when a file is to be reopened immediately after it is closed. By overriding the default option DET of the FIT with the option R, this routine can prevent the release to the system of buffer space and AAM capsules associated with the file. If the WITH LOCK phrase is specified in the CLOSE statement, the C.FILE routine cannot override the default setting for the CF field.

```
ENTER "C.FILE" USING INVENTORY, "CF=R".
    CLOSE INVENTORY
```

When these statements are executed, the file INVENTORY is closed and the buffer space and system capsules used for the file are retained by the program.

ACTUAL-KEY FILE ORGANIZATION

In an actual-key file, records are stored according to the record number specified by the primary key value. Records are accessed sequentially or randomly depending on the ACCESS MODE clause. A file with actual-key organization can reside only on a mass storage device.

Actual-key file organization is used for rapid access by alternate key. The primary key is system-oriented rather than data-oriented; the value of the key identifies the actual location of the record in the file. The system generates the primary key values when the records are written on the file. Multiple alternate keys can be defined for an actual-key file. Alternate key values are stored on a separate index file.

FILE DEFINITION

The FILE-CONTROL paragraph in the Environment Division and the File Description and Record Description entries in the Data Division describe the structure of a file with actual-key organization.

FILE-CONTROL Paragraph

The FILE-CONTROL paragraph for a file with actual-key file organization requires four clauses: SELECT, ASSIGN, ORGANIZATION, and RECORD KEY. Four optional clauses can also be included in this paragraph. Refer to figure 3-14.

```
    .
    .                                  Required Clauses
    .
ENVIRONMENT DIVISION.
INPUT-OUTPUT SECTION.
FILE-CONTROL.
    SELECT CUSTOMERS
        ASSIGN TO CSTMRS, CSTINDX
        ORGANIZATION IS ACTUAL-KEY
        ACCESS MODE IS RANDOM
        RECORD KEY IS ACT-KEY
        ALTERNATE RECORD KEY IS CUST-ID
        ALTERNATE RECORD KEY IS CUST-TYPE
            WITH DUPLICATES ASCENDING
        FILE STATUS IS CODE-RETURN
        USE "ORG=OLD".
    .
    .
    .
```

Figure 3-14. FILE-CONTROL Paragraph for an Actual-Key File

The file-name that is used in the COBOL 5 program is specified in the SELECT clause. The ASSIGN clause specifies the logical file name recognized by the operating system; if alternate keys are defined for the file, the ASSIGN clause also designates the logical file name for the alternate key index file. If either logical file name duplicates any other name used in the program or a reserved word, it must be enclosed in quotation marks. For an actual-key file, the ORGANIZATION clause must specify ACTUAL-KEY.

The RECORD KEY clause specifies the primary key, which must be an elementary COMPUTATIONAL-1 or COMPUTATIONAL-4 data item consisting of one to eight digits. The primary key must be a data item embedded in each record in the file or within the Working-Storage Section. Each primary key value is unique; duplicate primary keys cannot exist in the file because the primary key indicates the storage location for the record.

Alternate keys are specified by ALTERNATE RECORD KEY clauses. The clause is included once for each alternate key field. An alternate key must be a data item within the actual-key record. The first character position of each alternate key must be different from the primary key when the primary key is contained in the record. An alternate key must begin in a different character position from another alternate key when the keys are the same length; if the keys are not the same length, they can begin in the same position. Duplicate values for an alternate key are allowed only when the DUPLICATES option is included in the ALTERNATE RECORD KEY clause for the alternate key.

The order in which records with duplicate alternate keys are retrieved during sequential reading by alternate key depends on whether or not the ASCENDING option is included in the DUPLICATES phrase. If the option is included, records with duplicate alternate keys are retrieved in sequential order of primary keys; otherwise, the records are retrieved in the order they were written.

An alternate key value is included in or excluded from the alternate key index file depending on whether the USE phrase or the OMITTED phrase is specified in the ALTERNATE RECORD KEY clause. If neither phrase is included in the clause, an entry is made in the index file for each alternate key value in the data file. The USE phrase, which includes a one-character alphanumeric data item and an alphanumeric literal, specifies a condition for including an index entry in the index file. When the character contained in the data item is the same as one of the characters in the literal, an entry is made in the index file. When the specified condition is not satisfied, no entry is made. The literal in the phrase must consist of 1 to 36 unique characters. The data item must be contained within the record; the data-name used to reference the data item can be used in more than one ALTERNATE RECORD KEY clause for an actual-key file.

The access mode for an actual-key file can be sequential, random, or dynamic. Access mode determines the input/output statements that can be executed for the file. If the ACCESS MODE clause is omitted or if it specifies SEQUENTIAL, records can only be accessed sequentially. Records in an actual-key file can be accessed randomly if the access mode is random or dynamic. If dynamic access is specified, records can be accessed both randomly and sequentially during program execution.

The FILE STATUS clause specifies a data item to receive a status code each time an input/output statement is executed. The status code value indicates whether or not the statement executed successfully. The status code is discussed further in section 15.

The USE clause supplies file information used by AAM to process the actual-key file. Certain FILE control statement parameters can be specified to override parameter values obtained from other clauses and statements in the source program and to supply parameter values that cannot otherwise be specified in the source program. The parameter list must be enclosed in quotation marks. Refer to section 15 for a complete list of the parameters that can be specified.

The type of actual-key file to be used is determined by the ORG parameter of the USE clause. If extended AAM files have been installed and either the parameter is omitted or ORG=NEW is specified, the file is treated as an extended actual-key file. ORG=OLD is required for files in the initial actual-key format.

The OMITTED phrase of the ALTERNATE RECORD KEY clause can specify either a data-name and a literal (as in the USE phrase) or the KEY option. When a data-name and literal are specified, the alternate key index entry is not stored in the index file if the character in the data item named by the data-name is identical to any character in the literal. The literal and data item are set up in the same manner as described in the USE phrase. The option KEY IS SPACES indicates that an alternate key index entry is not stored in the index file when the alternate key item contains all spaces and has a usage of DISPLAY. When KEY IS ZEROS is specified, the alternate key index entry is omitted from the index file if the key item contains all zeros and has a usage of COMPUTATIONAL-1 or COMPUTATIONAL-2. An alternate key in an actual-key file record has more than one value when the key is described with the OCCURS clause. When a record is written, the value in each occurrence of the alternate key is indexed on the alternate key index file. The record can then be retrieved by the value in any occurrence of the alternate key.

The FILE-CONTROL paragraph shown in figure 3-14 describes a file with actual-key organization. The program name for the file is CUSTOMERS and the logical file name used by the operating system is CSTMRS. The logical file name for the alternate key index file is CSTINDX. The access mode for the actual-key file is established as random; records can only be processed randomly by key value. The data item ACT-KEY is the primary key; its value identifies the block number and record slot in which the record is stored. Two alternate keys are specified; the data item CUST-ID must be unique for each record while the data item CUST-TYPE can be duplicated within the file. Records with duplicate CUST-TYPE alternate key values are retrieved in sequential order of primary key values.

File Description Entry

The File Description entry (FD entry) in the File Section of the Data Division describes the physical structure of the actual-key file. The program file-name, as specified in the SELECT clause, is specified in the FD entry. Three specific clauses in the FD entry are applicable to actual-key files. Refer to figure 3-15.

```
      .
      .
      .
DATA DIVISION.
FILE SECTION.
FD  CUSTOMERS
          LABEL RECORDS ARE OMITTED
          BLOCK CONTAINS 10 RECORDS
          RECORD CONTAINS 165 CHARACTERS
          DATA RECORD IS CUST-REC.
      .
      .
      .
```

Figure 3-15. File Description Entry for an Actual-Key File

The LABEL RECORDS clause is required in every FD entry and specifies whether or not the file contains labels. Actual-key files cannot contain labels and OMITTED must be specified in this clause.

The BLOCK CONTAINS clause provides information that the system uses in determining the physical size of a block in the file. The block size is calculated by rounding upward to a multiple of physical record unit (PRU) size less 50 characters; PRU size is 640 characters. The value used for rounding upward is as follows:

If the clause is omitted, the value is determined by adding 90 to the result of the maximum record size multiplied by 8.

If a number of records is specified, the value is determined by adding together the following results:

1. The specified number of records plus one, multiplied by 10.
2. The maximum record size multiplied by the specified number of records.

If a number of characters is specified, the value is the number of characters.

When primary keys are user-generated, it is usually simpler to generate key values if the number of records in a block is a power of 2.

The RECORD clause determines the record type and record size used by AAM for input/output processing of actual-key files. If the clause is omitted, the Record Description entry is used for record size and type. The specific format of the RECORD clause determines record type and record size in the same manner as described for sequential file organization. Refer to table 3-2 for the effect of the RECORD clause.

Figure 3-15 illustrates a File Description entry for an actual-key file named CUSTOMERS. Each block in the file contains 10 records and a record consists of 165 characters. The DATA RECORD clause is included to document the record-name for the file.

Record Description Entry

The File Description entry for an actual-key file includes one Record Description entry for each record format applicable to the file. The Record Description entry provides data-names for individual data items within the record and describes the physical structure of the record.

If the FD entry does not include the RECORD clause, the record type and record size used for input/output processing are determined from the Record Description entry. The record type and record size are determined in the same manner as described for sequential file organization.

The record format defined by the Record Description entry illustrated in figure 3-16 is applicable to the actual-key file described in figures 3-14 and 3-15. The primary key ACT-KEY is described in the Record Description entry as an eight-digit number with COMP-1 usage. The value of ACT-KEY identifies the sequential record number in which the record is stored. The two alternate keys, CUST-ID and CUST-TYPE, are also described in the record format. A CUST-REC record can be accessed by any of the three defined keys. The record type and record size specified by the Record Description entry is the same as determined by the RECORD clause for the file. The records are fixed length (record type F) with 165 characters per record.

```
      .
      .
      .
DATA DIVISION.
FILE SECTION.
      .
      .
      .
01   CUST-REC.
     03   ACT-KEY              PICTURE 9(8)
               USAGE IS COMP-1.
     03   CUST-ID              PICTURE X(6).
     03   CUST-NAME            PICTURE X(15).
     03   CUST-TYPE            PICTURE XX.
     03   MONTHLY-ORDERS OCCURS 12 TIMES.
          05   NO-ORDERS       PICTURE 99.
          05   MONTH-AMT       PICTURE 9(5)V99.
     03   YTD-ORDERS.
          05   TOTAL-ORDERS    PICTURE 999.
          05   TOTAL-AMT       PICTURE 9(7)V99.
     03   CURRENT-BAL          PICTURE 9(6)V99.
     03   LAST-ACTIVITY        PICTURE 9(6).
      .
      .
      .
```

Figure 3-16. Record Description Entry for an Actual-Key File

FILE MANIPULATION

Seven Procedure Division statements are provided for input/output processing of actual-key files. Once the file has been created, records can be read, replaced, deleted, and inserted. Individual data items within actual-key records are manipulated through various statements that are described in other sections of the guide.

Opening Actual-Key Files

Before records in an actual-key file can be accessed, the file must be opened by the OPEN statement. The file is opened for input, for output, or for input and output processing. The open mode established for the file determines the input/output statements that can be executed. The primary key becomes the key of reference when the OPEN statement is executed; for an existing file, the current value of the key of reference is the primary key value for the first record in the file.

An actual-key file is opened for input with the OPEN INPUT statement. The file is then available for read-only processing. When the OPEN INPUT statement is executed, the file is positioned at the first record slot of the first block in the file. Depending on the access mode established for the file, records are then read sequentially or randomly.

```
OPEN INPUT CUSTOMERS.
```

This statement causes the actual-key file CUSTOMERS to be opened for input. Records can be read from but not written on the file.

The OPEN OUTPUT statement is used when an actual-key file is being created. Records can only be written on the file; they cannot be read or updated. Records are subsequently written in the location specified by the value of the primary key.

```
OPEN OUTPUT MASTER-FILE.
```

Execution of this statement opens the file MASTER-FILE for write-only processing. Records cannot be read from the file.

An existing actual-key file is opened for input and output processing by the OPEN I-O statement. The open mode established by this statement allows records in the file to be read, deleted, inserted, and updated. The file is positioned at the first record currently existing in the file.

```
OPEN I-O STOCK-FILE.
```

When this statement is executed, the file STOCK-FILE is opened for input and output processing. Records can be read, inserted, deleted, and rewritten.

Writing New Actual-Key Files

Records are written on a new actual-key file when the file has been opened for output (OPEN OUTPUT). The primary key specifies the record number. AAM converts the record number to the storage location of the record.

```
WRITE MASTER-REC.
```

Execution of this statement causes a MASTER-REC record to be written in the location specified by the system-generated primary key value. Records are written serially beginning with the first record position.

The primary key values can be generated by AAM. If AAM is to generate primary key values, the primary key data item must be set to zero before each record is written on the file. The system-generated key value is returned to the program in the primary key data item when the record is written on the file. These values must be preserved by the program if the file is to be accessed by primary key.

An invalid key condition exists if the key value indicates a location that already contains a record or if it indicates a block number that is more than one greater than the highest existing block number. If duplicate alternate keys are not allowed, a duplicate alternate key value also causes an invalid key condition.

```
WRITE CUST-REC
    INVALID KEY GO TO KEY-ERROR.
```

When this statement is executed, a CUST-REC record is written in the location indicated by the primary key value supplied by the program. If an invalid key condition is encountered, control is transferred to the paragraph named KEY-ERROR.

If alternate keys are defined for the actual-key file without the USE or OMITTED phrase, the primary key is indexed in the alternate key index file for each alternate key in the record being written. When the USE or OMITTED phrase is specified in the key definition, the primary key is entered in the index file on the basis of the condition specified in the phrase. A repeating group alternate key causes the primary key to be indexed for each occurrence of the alternate key.

The FROM phrase is included in the WRITE statement when the data for the output record is stored in an area other than the output record area. The data is moved from the storage area to the record area and the record is then written on the file.

```
WRITE STOCK-REC FROM TEMP-REC
    INVALID KEY GO TO BAD-WRITE.
```

Before the STOCK-REC record is written on the file, the data in the storage area TEMP-REC is moved to the output record area. If an invalid key condition occurs, control is transferred to the paragraph named BAD-WRITE.

Positioning Actual-Key Files With Alternative Keys

Retrieval of records in an actual-key file with alternate keys can begin with a record other than the first record in the file. The file is positioned to a record that meets a specified condition. The access mode for the file must be sequential or dynamic and the file must be open for input (OPEN INPUT or OPEN I-O).

The START statement positions the file for subsequent sequential READ statements and establishes the key of reference. The file is positioned at a record that satisfies a specified condition. The KEY phrase establishes the key of reference and specifies the condition used in positioning the file. The key of reference specified in the phrase must be an alternate key or the leading portion of an alternate key; it cannot be the primary key. When the START statement is executed, the index file is searched for a value that is greater than, equal to, or not less than the current value of the designated data item. The file is positioned at the first record that satisfies the specified condition.

```
START MASTER-FILE
    KEY IS GREATER THAN MASTER-NO.
```

When this statement is executed, the index for the alternate key MASTER-NO is searched for a value greater than the current value of the MASTER-NO data item. The index file is positioned at the first alternate key value that satisfies the condition. Retrieval of the records in the actual-key file (MASTER-FILE) proceeds in the order of the alternate key values in the index file.

A repeating group alternate key can be specified in the KEY phrase; however, the data-name of the key cannot be subscripted or indexed. The value that is used to position the file is the current value in the first occurrence of the alternate key data item. In the record at which the file is positioned, the value satisfying the specified condition can be in any occurrence of the alternate key.

The leading portion of an alternate key can be specified in the KEY phrase if the alternate key does not begin in the same character position as another alternate key; when two alternate keys begin in the same position, the KEY phrase cannot specify an item subordinate to them.

If the leading portion of an alternate key is specified in the KEY phrase, it must begin in the first character position of the alternate key and must be described as an alphanumeric data item. For example:

```
03  MASTER-NO.
    05  TYPE-CODE      PICTURE XX.
    05  MST-NUMBER     PICTURE 9(4).
```

The alternate key MASTER-NO is described with two subordinate data items. The KEY phrase can specify either MASTER-NO or TYPE-CODE, but not MST-NUMBER.

An invalid key condition occurs when the condition specified in the KEY phrase cannot be satisfied by any record in the file. The INVALID KEY phrase designates the action to be taken when this condition occurs.

```
START MASTER-FILE
    KEY IS NOT LESS THAN TYPE-CODE
    INVALID KEY GO TO NO-TYPE.
```

Execution of this statement causes the alternate key index file to be searched for a value with the first two characters of MASTER-NO equal to or greater than the current value of TYPE-CODE. If the value cannot be found, control is transferred to the paragraph named NO-TYPE.

Reading Actual-Key Files

When an actual-key file is opened for input (OPEN INPUT or OPEN I-O), records are retrieved from the file by the READ statement. Records are read randomly or sequentially depending on the access mode established for the file and the format of the READ statement. The key of reference for reading an actual-key file can be the primary key or an alternate key.

Accessing Sequentially

Sequential read operations can be executed when the access mode for the actual-key file is sequential or dynamic. At the time the READ statement is executed, the key of reference determines the order in which records are retrieved. The key of reference is determined as follows:

- When the file is opened, the primary key is the key of reference.
- If the file has been positioned by the START statement, the alternate key used to position the file becomes the key of reference.
- In dynamic access mode, a random read executed before the sequential read establishes the key used for the random read as the key of reference.

When the primary key is the key of reference, records are read in the order they are stored in the file; if the primary key is defined in the Working-Storage Section (rather than within the record), the key value is stored in the data item designated by the RECORD KEY clause. If an alternate key is the key of reference, records are read in the order of the key values in the alternate key index file.

For sequential access mode, records read by primary key are retrieved in stored sequence beginning with the first record in the first block of the file or with the record that satisfied the condition in the START statement. When an alternate key is the key of reference, the first record read is the record at which the file has been positioned by the START statement; sequential reading then proceeds in the order of the alternate key values in the index file. The final occurrence of a particular alternate key value can be detected through the FILE STATUS clause (described in section 15) by testing for a status code of 02.

```
READ MASTER-FILE RECORD
    AT END GO TO CLOSING.
```

Records in the actual-key file MASTER-FILE are read sequentially. The access mode for the file is established as sequential. If the primary key is the key of reference, records are read in stored order. When an alternate key is the key of reference, records are read in sequence by the alternate key values. When the last record has been read, control is transferred to the paragraph named CLOSING.

For dynamic access mode, the first record retrieved by a sequential READ statement is one of the following:

The first record in the first block of the file.

The record at which the file has been positioned by the START statement.

The next record in sequence according to the key of reference used in the preceding random READ statement.

Subsequent records are retrieved sequentially by stored position (if the primary key is the key of reference) or by the order of alternate key values in the index file (if an alternate key is the key of reference). The keyword NEXT must be included in a sequential READ statement when the access mode is dynamic. The FILE STATUS clause, described in section 15, can be used to determine the final occurrence of an alternate key value during sequential retrieval by alternate key.

```
READ CUSTOMERS NEXT RECORD
    AT END GO TO FINISHED.
```

This statement reads a record sequentially from the file CUSTOMERS. If the key of reference is the primary key, the next record in stored sequence is read. If an alternate key is the key of reference, the next record in alternate key sequence is read. When the end of the file has been reached, control is transferred to the paragraph named FINISHED.

A sequential READ statement includes the INTO phrase when the input record is to be stored in a specified area. The record is then available in both the input record area and the specified storage area. When the file is defined by more than one Record Description entry, the INTO phrase cannot be used if any entry is a level 01 elementary item that is described as a numeric or numeric-edited data item.

```
READ STOCK-FILE NEXT RECORD
    INTO UPDATING.
```

When this statement is executed, the next record in sequence according to the key of reference is read from the file STOCK-FILE. The record is stored in the input record area and in the storage area named UPDATING.

Accessing Randomly

Records in an actual-key file can be accessed randomly by key value when the access mode for the file is established as random or dynamic. The key of reference for the random read can be the primary key or an alternate key.

The KEY IS phrase specifies the key of reference for a random READ statement. If the phrase is omitted, the primary key is the key of reference. Either the primary key or an alternate key can be designated in the KEY IS phrase as the key of reference. At the time the READ statement is executed, the record retrieved is the record with the key value equal to the current value of the key of reference.

When the key of reference is an alternate key with duplicate values, the first primary key indexed for the alternate key value determines the record to be retrieved. The order in which the primary keys are indexed depends on whether or not the ASCENDING option is specified in the ALTERNATE RECORD KEY clause. If ASCENDING is specified, the primary keys are indexed in ascending sequence; if it is not specified, the primary keys are indexed in the order the records were written on the file.

```
READ CUSTOMERS RECORD
    KEY IS CUST-TYPE
    INVALID KEY GO TO KEY-ERROR.
```

The KEY IS phrase establishes the alternate key CUST-TYPE as the key of reference. CUST-TYPE can have duplicate values and the primary keys are indexed in ascending sequence. Execution of this statement causes the alternate key index file to be searched for a CUST-TYPE value that is equal to the current value of the data item CUST-TYPE. The record with the first primary key indexed for the CUST-TYPE value is read from the file. If a key of equal value cannot be found, control is transferred to the paragraph named KEY-ERROR.

When a repeating group alternate key is specified in the KEY IS phrase, the data-name of the alternate key cannot be subscripted or indexed. The current value in the first occurrence of the alternate key data item is used to search the index file for an equal value. The record with the first primary key indexed for the alternate key value is read from the file. The value can be in any occurrence of the alternate key in the record that is read from the file. When records are read by repeating group alternate key values, the same record can be retrieved for the value in each occurrence of the alternate key.

The INTO phrase is included in the random READ statement to specify an additional storage area for the input record. When the record is read, it is stored in the input record area and in the specified storage area.

```
READ MASTER-FILE RECORD INTO NEW-REC
    KEY IS MAST-NO
    INVALID KEY GO TO BAD-KEY.
```

When this statement is executed, a record is read from the actual-key file MASTER-FILE; the record is then available in the input record area and in the storage area named NEW-REC. The alternate key MAST-NO is established as the key of reference. If an invalid key condition exists, control is transferred to the paragraph named BAD-KEY.

Updating Actual-Key Files

Records in an existing actual-key file are updated by the DELETE and REWRITE statements. The WRITE statement is used to add new records to the file; it cannot be used to replace an existing record in the file. Any access mode is allowed and the file must be open for input and output (OPEN I-O).

Records are removed from the actual-key file by the DELETE statement. After the DELETE statement is executed, the record can no longer be accessed. The record deleted is either the last record read or the record indicated by the current value of the primary key.

If the access mode is sequential, the last input/output statement preceding the DELETE statement must be a valid sequential READ statement. The record deleted is then the last record read.

```
READ MASTER-FILE RECORD.
.
.
.
DELETE MASTER-FILE RECORD.
```

A record is read in sequence from the file MASTER-FILE. When the DELETE statement is executed, the record read from MASTER-FILE is removed from the file and can no longer be accessed.

Records are deleted by primary key value when the access mode is random or dynamic. The record in the location specified by the current value of the primary key data item is deleted from the file. If the specified location does not contain a record, an invalid key condition exists.

```
DELETE CUST-FILE RECORD
    INVALID KEY GO TO NO-RECORD.
```

This statement deletes the CUST-FILE record in the location indicated by the current value of the primary key data item. If a record does not exist in the specified location, control is transferred to the paragraph named NO-RECORD.

Data in an existing record in the actual-key file can be updated through the REWRITE statement. The record to be rewritten is identified by the primary key value. The FROM phrase is included in the REWRITE statement when the updated record is stored in a location other than the record area. The data in the storage area is moved to the record area before the record is rewritten on the file.

If the access mode is sequential, a sequential READ statement must precede the REWRITE statement. The record rewritten is then the previously read record. The primary key value must not be changed before the REWRITE statement is executed. An invalid key condition exists if the key value is changed.

```
READ STOCK-FILE RECORD INTO UPDATING.
.
.
.
REWRITE STOCK-REC FROM UPDATING.
```

The record read from the file STOCK-FILE is stored in the area named UPDATING. Data in any field except the primary key field can be updated before the REWRITE statement is executed. The updated record is moved from UPDATING to the record area for STOCK-REC and then replaces the record retrieved by the previous READ statement.

If the access mode is random or dynamic, the current value of the primary key data item specifies the record to be rewritten. An invalid key condition occurs if the record position indicated by the primary key value does not contain a record. When a record containing alternate key data items is rewritten, new values can be specified for the key items; however, the new values cannot duplicate any existing alternate key values in the file unless the DUPLICATES phrase is included in the key definition.

```
REWRITE MASTER-REC
    INVALID KEY GO TO KEY-ERR.
```

When this statement is executed, the data in the MASTER-REC record area replaces the record stored in the location identified by the current value of the primary key data item. If a record does not exist at that location, control is transferred to the paragraph named KEY-ERR.

Closing Actual-Key Files

Processing of an actual-key file is terminated by execution of the CLOSE statement. AAM updates the internal tables that are part of the file. The file can be reopened for further processing; it is then positioned at the first record in the file.

The WITH LOCK phrase is included in the CLOSE statement to prevent the actual-key file from being reopened during the execution of the current control statement. When the file is closed, it is returned to the system. If an attempt is made to reopen the file, the program aborts.

If a file is to be reopened immediately after it has been closed, the routine C.FILE should be entered to prevent the release to the system of buffer space and AAM capsules used for the file. The routine C.FILE overrides the default setting DET of the CF field in the file information table with the setting R. The R parameter indicates that the buffer space and system capsules associated with the file are retained by the program. The ENTER statement should immediately precede the CLOSE statement. If the WITH LOCK statement is included in the CLOSE statement, the default setting for the CF field cannot be overridden.

```
ENTER "C.FILE" USING STOCK-FILE, "CF=R".
    CLOSE STOCK-FILE.
```

The file STOCK-FILE is closed and the system resources associated with the file are retained for subsequent use when the file is reopened.

WORD-ADDRESS FILE ORGANIZATION

A word-address file is a mass storage file in which the word-address key specifies the number of the first word in the record. Word numbers begin with 1 for the first word of the first record in the file and continue in sequence to the end of the file. Each record begins in a new word; a word contains 10 character positions. Boundaries do not appear between records. A word-address file is similar to a table in which any entry can be identified by an index into the table.

Word-address file organization is used most often in specialized applications that require immediate access. Records are read or written beginning with the word number indicated by the word-address key value. The number of words read or written is determined by the Record Description entry. The system performs no checking of the data being read or written to ensure that the information is valid. If the file is created sequentially, unused words do not exist in the file. Alternate keys cannot be specified for word-address files.

FILE DEFINITION

The FILE-CONTROL paragraph in the Environment Division and the File Description and Record Description entries in the Data Division define the structure of a file with word-address organization.

FILE-CONTROL Paragraph

In the FILE-CONTROL paragraph for a word-address file, four clauses must be included: SELECT, ASSIGN, ORGANIZATION, and WORD-ADDRESS KEY. Four optional clauses can be included in this paragraph. Refer to figure 3-17.

```
    .                                        Required Clauses
    .                                              |||
    .                                              |||
ENVIRONMENT DIVISION                               |||
INPUT-OUTPUT SECTION.                              |||
FILE-CONTROL.                                      |||
    SELECT PARTS-FILE ASSIGN TO PARTFLE <----------|||
        ORGANIZATION IS WORD-ADDRESS <-------------||
        ACCESS MODE IS DYNAMIC                     ||
        WORD-ADDRESS KEY IS PART-KEY <-------------|
        RESERVE 1 AREA
        FILE STATUS IS FILE-CODE
        USE "DFC=3".
    .
    .
    .
```

Figure 3-17. FILE-CONTROL Paragraph for a Word-Address File

The SELECT clause specifies the file-name used in the COBOL 5 program. The ASSIGN clause specifies the logical file name recognized by the operating system. If the logical file name is the same as any other name in the program or as a reserved word, it must be enclosed in quotation marks. The ORGANIZATION clause designates the file as a word-address file.

The WORD-ADDRESS KEY clause specifies the data item that contains the key value used to access word-address records randomly. During program execution, the key value must be a numeric integer that identifies the word number at which the record begins. The data item containing the key value must not be included in the record stored in the file. If the file is an External file, the word-address key data item must be defined in the Common-Storage Section.

The access mode for a word-address file can be sequential, random, or dynamic. If the ACCESS MODE clause is omitted or if it specifies SEQUENTIAL, records are read or written sequentially. Records are accessed randomly by key value when the clause specifies RANDOM. If DYNAMIC is specified, records can be accessed sequentially and randomly during program execution.

The RESERVE clause can specify the number of input/output buffer areas for the word-address file. The specified number of areas, plus two additional words, are reserved. If the clause is omitted, the system allocates two buffer areas. The size of each input/output buffer area is the maximum block size.

The FILE STATUS clause is used to specify a data item to receive a status code whenever an input/output statement is executed for the file. The status code is a value that indicates whether or not the statement executed successfully. Refer to section 15 for a description of the status code.

The USE clause supplies file information used by BAM to process the word-address file. Certain FILE control statement parameters can be specified in this clause. These parameters supply file information that cannot be specified through the clauses and statements in the source program, or they override parameter values normally obtained from the source program. Refer to section 15 for a complete list of parameters that can be specified.

A FILE-CONTROL paragraph for a file with word-address organization is illustrated in figure 3-17. The file-name PARTS-FILE is used in the COBOL 5 program and the logical file name PARTFLE identifies the file for the operating system. Records in the file can be accessed sequentially or randomly since the access mode is established as dynamic. The key value used to access a record is contained in the data item PART-KEY.

File Description Entry

The physical structure of the word-address file is described in the File Description entry (FD entry) of the Data Division. The program file-name specified in the SELECT clause is also specified in the FD entry. Two clauses, LABEL RECORDS and RECORDS, are applicable to word-address files. Refer to figure 3-18.

```
      .
      .
      .
DATA DIVISION.
FILE SECTION.
FD  PARTS-FILE
          LABEL RECORDS ARE OMITTED
          RECORD CONTAINS 30 CHARACTERS
          DATA RECORD IS PART-REC.
      .
      .
      .
```

Figure 3-18. File Description Entry for a Word-Address File

The LABEL RECORDS clause, which is required in every FD entry, must specify that label records are omitted. Labels cannot exist on a word-address file.

Record type and record size are used by BAM for input/output processing of word-address files. The RECORD clause, if included, is used to determine record size. For a word-address file, record type is always U (unless overridden by a FILE control statement).

The File Description entry illustrated in figure 3-18 is for a word-address file named PARTS-FILE. The file contains U type records with 30 characters per record. The DATA RECORD clause documents the record name as PART-REC.

Record Description Entry

The File Description entry for a word-address file must include a Record Description entry for each record format applicable to the file. Records can be fixed or variable in length. The Record Description entry assigns data-names to individual data items and describes the physical structure of the record.

When the RECORD clause is not included in the FD entry, the record size used by BAM is determined by the Record Description entry. If all records contain the same number of characters, the record size is the maximum number of character positions described in the Record Description entry. If Record Description entries are not all the same size and no OCCURS...DEPENDING ON phrase exists in the RECORD clause, the record size is the actual size of the named record.

The Record Description entry illustrated in figure 3-19 is applicable to the word-address file described in figures 3-17 and 3-18. The record format defines a record with 30 character positions; each stored record occupies three words. The record key, which is used to access records randomly, is described in the Working-Storage Section as a four-digit integer.

```
      .
      .
      .
DATA DIVISION.
FILE SECTION.
      .
      .
      .
01   PART-REC.
     03  PART-NAME          PICTURE X(10).
     03  USED-WITH          PICTURE X(5).
     03  QTY-ON-HAND        PICTURE 9(5).
     03  MFG-CODE           PICTURE X(10).
      .
      .
      .
WORKING-STORAGE SECTION.
01   PART-KEY               PICTURE 9(4).
      .
      .
      .
```

Figure 3-19. Record Description Entry for a Word-Address File

FILE MANIPULATION

Input/output processing of word-address files is accomplished through four Procedure Division statements. Records in an existing word-address file can be read, replaced, and inserted. Individual items within a record are manipulated through various statements that are discussed in other sections of this guide.

Opening Word-Address Files

Word-address files are opened for input, for output, or for input and output. The specific format of the OPEN statement determines the open mode for the file.

The OPEN INPUT statement opens the word-address file for input only. Records are read from the file sequentially or randomly depending on the access mode. The file is positioned at the first record (word 1 of the file).

```
OPEN INPUT PARTS-FILE.
```

This statement causes the word-address file PARTS-FILE to be opened for input. Records can then be read from but not written on the file.

When a word-address file is being created, the file is opened for output. The OPEN OUTPUT statement makes the file available for write-only processing. Records are written sequentially or randomly depending on the access mode established for the file.

```
OPEN OUTPUT WORD-FILE.
```

Execution of this statement causes the file WORD-FILE to be opened for output. Records are then written on but not read from the file.

An existing word-address file can be opened for input and output. The OPEN I-O statement allows records to be read from and written on the file. The file is positioned at the first record (word 1 of the file). Subsequent statements read and write records sequentially or randomly depending on the access mode.

```
OPEN I-O PARTS-FILE.
```

When this statement is executed, the file PARTS-FILE is opened for input and output. The program can then read or write records on the file.

Writing Word-Address Files

Records are written on a word-address file sequentially or randomly depending on the access mode established for the file. The file must be open for output (OPEN OUTPUT for file creation or OPEN I-O for file updating) when records are to be written. The word-address key data item is updated after each write operation to indicate the beginning word number for the next record. The key should be initialized to one or to a specific value to avoid a large word address.

For sequential access mode, the file must be open for output only. Records are written on the file beginning with word number 1; each record uses a designated number of words. If the records are fixed length, the number of words used is the same for all records in the file. The number of words used for variable-length records depends on the actual size of the record being written. When the record is written on the file, the system returns the beginning word number for the record to the word-address key data item.

```
WRITE WORD-RECORD.
```

Execution of this statement in sequential access mode causes the record WORD-RECORD to be written on its associated file. The record begins in the next word in sequence; the system returns the word number for the record to the word-address key data item.

For random or dynamic access mode, the record is written according to the current value of the word-address key data item. The key value specifies the word number in which to begin the record. The record is written at the designated location, whether or not a record already exists at that location. An invalid key condition occurs when the key value is not an integer.

```
WRITE PART-REC
    INVALID KEY GO TO BAD-KEY.
```

When this statement is executed, the record PART-REC is written on the file beginning with the word number indicated by the word-address key value. If an invalid key condition exists, control is transferred to the paragraph named BAD-KEY.

The FROM phrase is included in the WRITE statement when the record data is in a storage area other than the record area. The data in the specified area is moved into the record area and then the record is written on the file.

```
WRITE WORD-RECORD FROM TEMP-REC.
```

This statement specifies that the data in the storage area named TEMP-REC is to be moved into the record area before the record is written on its associated file.

Reading Word-Address Files

Records are retrieved from a word-address file by the READ statement. The file must be open for input (OPEN INPUT or OPEN I-O). Depending on the access mode established for the file and the format of the READ statement, records are read sequentially or randomly.

The number of words retrieved when a READ statement is executed depends on the record size established by the Record Description entry. If multiple record descriptions are used, and there are no DEPENDING ON phrases in either the RECORD clause or the Record Description entry, the read operation uses the number of characters in the largest record description. If variable length records are to be used, the user must define the record length before a read operation occurs. Each time the READ statement is executed, the designated word-address key is updated to point to the next available word by using the record length of the record just read.

Accessing Sequentially

Word-address records can be read sequentially when the access mode is sequential or dynamic. When the file is opened, it is positioned at the first record. The READ statement retrieves the designated number of words beginning at the current file position. The AT END phrase is included in the sequential READ statement to specify the action to be taken when the end of the file is reached.

```
READ WORD-FILE RECORD
    AT END GO TO DONE.
```

Execution of this statement reads a record from the file WORD-FILE. When the end of the file is reached, control is transferred to the paragraph named DONE.

The INTO phrase can be included in the READ statement to store the record in a specified area. The record retrieved is available in both the input record area and the specified storage area. When the file is defined by more than one Record Description entry, the INTO phrase cannot be used if any entry is a level 01 elementary item that is described as a numeric or numeric-edited data item.

```
READ PARTS-FILE RECORD
    INTO REC-AREA.
```

When this statement is executed, a record is retrieved from the file PARTS-FILE. The record is available in both the input record area and the storage area named REC-AREA.

For sequential reading in dynamic access mode, the keyword NEXT must be included in the READ statement. In sequential access mode, the keyword NEXT is documentary only.

```
READ PARTS-FILE NEXT RECORD
    AT END GO TO FINISHED.
```

This statement reads the next record in sequence when the access mode for the file PARTS-FILE is dynamic. If the last record in the file has been read, control is transferred to the paragraph named FINISHED.

Accessing Randomly

If the access mode established for the word-address file is random or dynamic, records can be read randomly by key value. The data item defined as the word-address key contains a value that indicates the first word number of the record to be read. An invalid key condition exists if the key value is not an integer within the range of words in the file or if the attempt to read a record extends beyond the end of the file.

```
READ WORD-FILE RECORD
    INVALID KEY GO TO BAD-READ.
```

When this statement is executed, a record is read from the file WORD-FILE beginning at the word number indicated by the value of the word-address key. An invalid key condition causes control to be transferred to the paragraph named BAD-READ.

A random READ statement can also include the INTO phrase to store the record in a specified area. This phrase is executed in the same manner as for reading sequentially.

Closing Word-Address Files

The CLOSE statement terminates processing of a word-address file. Once the statement is executed, input/output statements cannot access the file until it has been opened again. When a word-address file is closed, a partition boundary exists at the end of the file. The boundary is overwritten when records are added to the end of the file.

The WITH LOCK phrase of the CLOSE statement specifies that the file being closed cannot be reopened during execution of the current control statement. The file is returned to the system. An attempt to reopen the file causes the program to abort.

If the file being closed is to be reopened immediately, the C.FILE routine should be entered before the file is closed. The routine changes the file information table CF field from the default DET setting to the R setting. The R parameter causes the program to retain the buffer space and BAM capsules associated with the file that are otherwise returned to the system. The setting of the CF field cannot be overridden if the WITH LOCK phrase is included in the CLOSE statement.

```
ENTER "C.FILE USING PARTS-FILE, "CF=R".
    CLOSE PARTS-FILE.
```

When these statements are executed, the file PARTS-FILE is closed; the associated buffer space and system capsules are retained for subsequent use when the file is reopened.

ERROR HANDLING

File-related errors and exception conditions encountered during program execution are handled in various ways. Some actions are performed automatically by the system while others are specified in the COBOL 5 program. Whenever an input/output statement is executed, the system generates a status code. This code can be used by the program to determine the course of action following execution of the input/output statement.

Input/output errors are handled in the following order:

1. Standard input/output error routines are automatically executed by the system.

2. If a user-supplied error procedure is specified for the file, the procedure is executed.

For at end and invalid key conditions, the program is responsible for determining subsequent action. Processing continues in one of two ways with the following order of precedence:

1. Control is transferred to the imperative statement specified in the AT END or INVALID KEY phrase of the input/output statement.

2. The user-supplied error procedure specified for the file is executed.

If neither method is used to provide for processing of at end and invalid key conditions, the program is aborted when the condition occurs.

USER-SUPPLIED ERROR PROCEDURES

The source program can specify procedures that are executed when input/output errors or exception conditions occur during file processing. A user-supplied procedure is executed after the standard input/output error routine has been performed. A USE statement introduces the error procedure to be executed. The keywords ERROR and EXCEPTION are synonymous; the choice between words is provided for documentary purposes only. An error procedure is executed for any file organization.

Execution of a user-supplied error procedure occurs under any of the following conditions:

- A standard input/output error routine has been executed.

- An at end condition exists for an input/output statement that does not include the AT END phrase.

- An invalid key condition exists for an input/output statement that does not include the INVALID KEY phrase.

User-supplied error procedures are specified in the Declaratives portion of the Procedure Division. Each error procedure is contained in a named section. The first statement in the section is a USE statement that designates the files to which the error procedure applies. This is followed by one or more paragraphs containing the statements to be executed when an input/output error or exception condition occurs.

The error procedure is executed only for those files indicated by the USE statement. If specific file-names are included in the USE statement, the error procedure is invoked for the named files regardless of the open mode of the files. If an open mode is specified in the USE statement, the error procedure applies to all files in the specified open mode.

Figure 3-20 shows two USE statements and the order in which they appear in the Procedure Division. Statements following paragraph-name INPUT-ERROR are executed when one of the conditions that invoke a USE statement is encountered for any input file. If the condition is encountered for the file named FILE-1, the statements following paragraph-name FILE1-ERROR are executed; the error procedure for input files is not executed if FILE-1 is an input file.

```
     .
     .
     .
PROCEDURE DIVISION.
DECLARATIVES.
INPUT-PROC SECTION.
USE AFTER STANDARD ERROR PROCEDURE
    ON INPUT.
INPUT-ERROR.
     .
     .
     .
FILE1-PROC SECTION.
USE AFTER STANDARD EXCEPTION PROCEDURE
    ON FILE-1.
FILE1-ERROR.
     .
     .
     .
END DECLARATIVES.
     .
     .
     .
```

Figure 3-20. Example of the USE Statement

STATUS CODE

The system generates a status code each time an input/output statement is executed. If this code is to be used by the program, the FILE STATUS clause in the Environment Division specifies the data item to receive the status code. The data item can be tested to determine the execution status of the input/output statement. Usage of the file status code (and other CRM debugging tools) is illustrated in section 15.

SAMPLE PROGRAMS

The sample programs included in this section illustrate the six file organizations. Each program uses at least one sequential file and one file with a different file organization. Two programs are shown for each of the following file organizations: relative, indexed, direct, actual-key, and word-address. The first program creates the file and the second program accesses the existing file.

RELATIVE FILE PROGRAMS

A file with relative organization is created by the program shown in figure 3-21. This file contains the name and address of each person who has a safe-deposit box. When customers are billed, the relative file is used to print address labels.

Each input record contains a customer's box number, name, and address (see figure 3-22). The box number is moved to the relative key data item (line 50), the customer's name and address are moved to the output record area (line 51), and the record is written on the relative file (line 52). The record is stored in the record position that corresponds to the box number.

The program shown in figure 3-23 uses the relative file created by the preceding program. This program is used to update the existing file and to create address labels for billing the customers. The first input card contains either the letter A or the letter B in the CARD-CODE field. The letter A indicates that the following cards contain information to update records in the relative file (lines 72 and 73). The letter B indicates that address labels are to be printed for the box numbers on the following cards (lines 74 and 75).

The UPDATING procedure reads an input record (line 79), moves the box number to the relative key data item (line 83), and rewrites the record using the new information (lines 84 and 85). This procedure is repeated until the CARD-CODE field contains the letter B (line 81) or the end of the input file is reached (line 80).

The BILLING procedure reads an input record (line 92), moves the box number to the relative key data item (line 94), and reads the corresponding record from the relative file (line 95). The name and address from the relative file record are used to create the address label on the output file (lines 97 through 103).

The format of the input records used to access the relative file is illustrated in figure 3-24. The output labels created by the program are shown in figure 3-25.

INDEXED FILE PROGRAMS

Two programs illustrate indexed file organization. The first program creates the indexed file and the second program accesses the file by alternate key.

The indexed file EMP-FILE is created by the program shown in figure 3-26. The alternate key index file is assigned the logical file name INDFLE (line 10). Two alternate keys, HIRE-DATE and JOB-ID, are specified (lines 14 through 17); duplicate key values are allowed for both alternate keys. When records are accessed by alternate key, records with duplicate alternate key values are retrieved in ascending sequence by the value of the primary key EMP-ID.

```
 1   IDENTIFICATION DIVISION.
 2   PROGRAM-ID.  NEW-REL.
 3   ENVIRONMENT DIVISION.
 4   CONFIGURATION SECTION.
 5   SOURCE-COMPUTER.  CYBER-170.
 6   OBJECT-COMPUTER.  CYBER-170.
 7   INPUT-OUTPUT SECTION.
 8   FILE-CONTROL.
 9       SELECT CARDFILE ASSIGN TO INPUT.
10       SELECT BOX-FILE ASSIGN TO BOXFLE
11           ORGANIZATION IS RELATIVE
12           ACCESS MODE IS RANDOM
13           RELATIVE KEY IS REL-KEY.
14  DATA DIVISION.
15  FILE SECTION.
16  FD  CARDFILE
17           LABEL RECORDS ARE OMITTED
18           DATA RECORD IS CARD-IN.
19  01  CARD-IN.
20      03  BOX-NO                PICTURE 999.
21      03  CUST-NAME             PICTURE X(20).
22      03  FILLER                PICTURE X.
23      03  STREET                PICTURE X(18).
24      03  FILLER                PICTURE XX.
25      03  CITY                  PICTURE X(15).
26      03  FILLER                PICTURE X(5).
27      03  STATE                 PICTURE AA.
28      03  FILLER                PICTURE XXX.
29      03  ZIP-CODE              PICTURE 9(5).
30      03  FILLER                PICTURE X(6).
31  FD  BOX-FILE
32           LABEL RECORDS ARE OMITTED
33           RECORD CONTAINS 60 CHARACTERS
34           DATA RECORD IS BOX-REC.
35  01  BOX-REC.
36      03  CUST-NAME             PICTURE X(20).
37      03  STREET                PICTURE X(18).
38      03  CITY                  PICTURE X(15).
39      03  STATE                 PICTURE AA.
40      03  ZIP-CODE              PICTURE 9(5).
41  WORKING-STORAGE SECTION.
42  01  REL-KEY                   PICTURE 999.
43  PROCEDURE DIVISION.
44  STARTING.
45      OPEN INPUT CARDFILE.
46      OPEN OUTPUT BOX-FILE.
47  CREATING.
48      READ CARDFILE RECORD
49          AT END GO TO CLOSING.
50      MOVE BOX-NO TO REL-KEY.
51      MOVE CORRESPONDING CARD-IN TO BOX-REC.
52      WRITE BOX-REC
53          INVALID KEY GO TO BAD-KEY.
54      GO TO CREATING.
55  BAD-KEY.
56      DISPLAY "INVALID KEY " REL-KEY.
57      GO TO CREATING.
58  CLOSING.
59      CLOSE CARDFILE, BOX-FILE.
60      STOP RUN.
```

Figure 3-21. Creating a File with Relative Organization

```
Column 1  Column 4          Column 25          Column 45          Column 65  Column 70
   ↓      ↓                  ↓                  ↓                  ↓          ↓
001JOHN J SMITH          1580 HAPPY LANE       MAYS LANDING       NJ    08330
002ROBERT K RILEY        P.O. BOX 12J          EGG HARBOR CITY    NJ    08215
003ELIZABETH JONES       9377 FIRST ST         RICHLAND           NJ    08350
004MICHAEL M MARTIN      2598 LAWRENCE AVE     MC KEE CITY        NJ    08310
005RUTH L STEPHENS       6403 KILGORE RD       RICHLAND           NJ    08350
006JEFFREY J CARTER      3354 WELLINGTON ST    MAYS LANDING       NJ    08330
007RICHARD S GREEN       P.O. BOX 57A          EGG HARBOR CITY    NJ    08215
008CHRISTOPHER A BURNS   4816 PEACHTREE RD     MC KEE CITY        NJ    08310
009JEAN L RICHARDSON     P.O. BOX 36C          EGG HARBOR CITY    NJ    08215
010GEORGE R BROWN        1269 HIDDEN LANE      MAYS LANDING       NJ    08330
011JANICE WHEELER        5528 THIRD ST         RICHLAND           NJ    08350
012ALBERT L ANDERSON     P.O. BOX 35C          EGG HARBOR CITY    NJ    08215
013MARVIN VAN DYKE       2012 CENTER ST        MC KEE CITY        NJ    08310
014PAUL J GRIFFITH       7004 WILLOW LANE      MAYS LANDING       NJ    08330
015FRANK PATTERSON       4619 COLLEGE AVE      MC KEE CITY        NJ    08310
016LORETTA D BURKE       P.O. BOX 19L          EGG HARBOR CITY    NJ    08215
017EDWARD N MILLER       6825 ASHFIELD RD      RICHLAND           NJ    08350
018CRAIG SULLIVAN        1221 ORCHARD LANE     MAYS LANDING       NJ    08330
019BARBARA FINNEGAN      4545 WEBSTER AVE      RICHLAND           NJ    08350
020STEVEN TREADWELL      3779 GILBERT AVE      MC KEE CITY        NJ    08310
```

Figure 3-22. Input Data for Creating the Relative File

```
 1   IDENTIFICATION DIVISION.
 2   PROGRAM-ID.  USE-REL.
 3   ENVIRONMENT DIVISION.
 4   CONFIGURATION SECTION.
 5   SOURCE-COMPUTER.  CYBER-170.
 6   OBJECT-COMPUTER.  CYBER-170.
 7   INPUT-OUTPUT SECTION.
 8   FILE-CONTROL.
 9       SELECT CARDFILE  ASSIGN TO INPUT.
10       SELECT BOX-FILE  ASSIGN TO BOXFLE
11           ORGANIZATION IS RELATIVE
12           ACCESS MODE IS RANDOM
13           RELATIVE KEY IS REL-KEY.
14       SELECT PRINTFILE ASSIGN TO OUTPUT.
15   DATA DIVISION.
16   FILE SECTION.
17   FD  CARDFILE
18           LABEL RECORDS ARE OMITTED
19           DATA RECORD IS CARD-REC.
20   01  CARD-REC.
21       03  BOX-NO              PICTURE 999.
22       03  CUST-NAME           PICTURE X(20).
23       03  FILLER              PICTURE X.
24       03  STREET              PICTURE X(18).
25       03  FILLER              PICTURE XX.
26       03  CITY                PICTURE X(15).
27       03  FILLER              PICTURE X(5).
28       03  STATE               PICTURE AA.
29       03  FILLER              PICTURE XXX.
30       03  ZIP-CODE            PICTURE 9(5).
31       03  FILLER              PICTURE X(5).
32       03  CARD-CODE           PICTURE X.
33   FD  BOX-FILE
34           LABEL RECORDS ARE OMITTED
35           RECORD CONTAINS 60 CHARACTERS
36           DATA RECORD IS BOX-REC.
```

Figure 3-23. Updating a File With Relative Organization (Sheet 1 of 3)

```
 37   01  BOX-REC.
 38       03  CUST-NAME            PICTURE X(20).
 39       03  STREET               PICTURE X(18).
 40       03  CITY                 PICTURE X(15).
 41       03  STATE                PICTURE AA.
 42       03  ZIP-CODE             PICTURE 9(5).
 43   FD  PRINTFILE
 44           LABEL RECORD IS OMITTED
 45           DATA RECORD IS LISTLINE.
 46   01  LISTLINE                 PICTURE X(50).
 47   WORKING-STORAGE SECTION.
 48   01  REL-KEY                  PICTURE 999.
 49   01  LINE-1.
 50       03  FILLER               PICTURE X       VALUE SPACE.
 51       03  NAME-OUT             PICTURE X(20).
 52       03  FILLER               PICTURE X(29)   VALUE SPACES.
 53   01  LINE-2.
 54       03  FILLER               PICTURE X       VALUE SPACE.
 55       03  STREET-OUT           PICTURE X(18).
 56       03  FILLER               PICTURE X(31)   VALUE SPACES.
 57   01  LINE-3.
 58       03  FILLER               PICTURE X       VALUE SPACE.
 59       03  CITY                 PICTURE X(15).
 60       03  FILLER               PICTURE XX      VALUE SPACES.
 61       03  STATE                PICTURE AA.
 62       03  FILLER               PICTURE XX      VALUE SPACES.
 63       03  ZIP-CODE             PICTURE 9(5).
 64       03  FILLER               PICTURE X(23)   VALUE SPACES.
 65   PROCEDURE DIVISION.
 66   OPENING.
 67       OPEN INPUT CARDFILE.
 68       OPEN I-O BOX-FILE.
 69       OPEN OUTPUT PRINTFILE.
 70       READ CARDFILE RECORD
 71           AT END GO TO ERROR-1.
 72       IF CARD-CODE EQUALS "A"
 73           GO TO UPDATING.
 74       IF CARD-CODE EQUALS "B"
 75           GO TO BILLING.
 76       DISPLAY "INVALID CODE " CARD-CODE.
 77       STOP RUN.
 78   UPDATING.
 79       READ CARDFILE RECORD
 80           AT END GO TO CLOSING.
 81       IF CARD-CODE EQUALS "B"
 82           GO TO BILLING.
 83       MOVE BOX-NO TO REL-KEY.
 84       MOVE CORRESPONDING CARD-REC TO BOX-REC.
 85       REWRITE BOX-REC
 86           INVALID KEY GO TO BAD-RECORD.
 87       GO TO UPDATING.
 88   BAD-RECORD.
 89       DISPLAY "NO EXISTING RECORD FOR " REL-KEY.
 90       GO TO UPDATING.
 91   BILLING.
 92       READ CARDFILE RECORD
 93           AT END GO TO CLOSING.
 94       MOVE BOX-NO TO REL-KEY.
 95       READ BOX-FILE RECORD
 96           INVALID KEY GO TO NO-RECORD.
 97       MOVE CUST-NAME OF BOX-REC TO NAME-OUT.
 98       WRITE LISTLINE FROM LINE-1
 99           AFTER ADVANCING 5 LINES.
100       MOVE STREET OF BOX-REC TO STREET-OUT.
101       WRITE LISTLINE FROM LINE-2.
102       MOVE CORRESPONDING BOX-REC TO LINE-3.
103       WRITE LISTLINE FROM LINE-3.
104       GO TO BILLING.
```

Figure 3-23. Updating a File With Relative Organization (Sheet 2 of 3)

```
105   NO-RECORD.
106       DISPLAY "BOX NUMBER " REL-KEY.
107       MOVE SPACES TO LISTLINE.
108       WRITE LISTLINE
109           AFTER ADVANCING 3 LINES.
110       GO TO BILLING.
111   ERROR-1.
112       DISPLAY "NO INPUT RECORDS".
113   CLOSING.
114       CLOSE CARDFILE, BOX-FILE, PRINTFILE.
115       STOP RUN.
```

Figure 3-23. Updating a File With Relative Organization (Sheet 3 of 3)

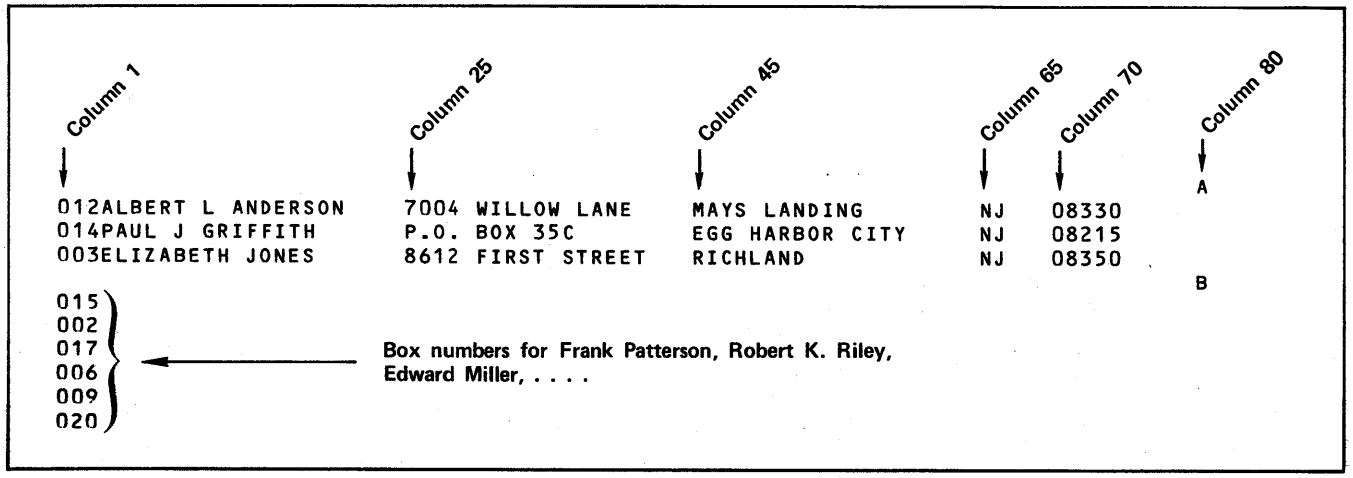


Figure 3-24. Input Data for Updating the Relative File

Two input records are used to create an indexed file record (lines 70 through 82). Since the access mode for EMP-FILE is sequential (line 12), the input records must be in ascending sequence by primary key value. An invalid key condition exists when the primary key for the record being written is not greater than the primary key for the preceding record. As each record is written on the file, the primary key is automatically entered in each alternate key index. The format of the input records is illustrated in figure 3-27.

The program shown in figure 3-28 uses the indexed file EMP-FILE to generate a listing of all employees hired since a certain date. The date to be used is accepted from the system file INPUT (line 71) and is moved to the alternate key data item HIRE-DATE (line 72). The date is represented on a punch card with the digits 700101 in columns 1 through 6. The START statement is then used to search the HIRE-DATE index for the first date that is equal to or greater than the date currently stored in HIRE-DATE (lines 73 through 75). Records are then read from the file EMP-FILE in the order of the HIRE-DATE values in the alternate key index (line 77). An output line is generated and printed for each record read from EMP-FILE (lines 79 through 83). The report shown in figure 3-29 lists the employees that were hired on or after January 1, 1970.

DIRECT FILE PROGRAMS

The two programs that illustrate direct file organization create and then access a file containing inventory records. Under certain conditions, records are updated and rewritten on the file.

The program shown in figure 3-30 creates the direct file INVENTORY. The primary key is the data item PART-NO (line 14). The hashed value of PART-NO determines the home block in which the record is stored. One alternate key (WHERE-USED) is specified for the file (line 15); duplicate alternate key values are allowed. The primary keys associated with an alternate key value are in ascending sequence because DUPLICATES ASCENDING is specified. The description of the WHERE-USED data item (lines 54 through 56) indicates that it is a repeating group alternate key. Each record contains from one to six values for the WHERE-USED data item.

Each time the WRITE-FILE paragraph is executed, an input record is read (see figure 3-31), the data is moved to the output record area, and the record is written on the file INVENTORY (lines 64 through 73). When a record is written on the file, the primary key is entered in the index file for each WHERE-USED value in the record.

```
FRANK PATTERSON
4619 COLLEGE AVE
MC KEE CITY       NJ  08310



ROBERT K RILEY
P.O. BOX 12J
EGG HARBOR CITY   NJ  08215



EDWARD N MILLER
6825 ASHFIELD RD
RICHLAND          NJ  08350



JEFFREY J CARTER
3854 WELLINGTON ST
MAYS LANDING      NJ  08330



JEAN L RICHARDSON
P.O. BOX 36C
EGG HARBOR CITY   NJ  08215



STEVEN TREADWELL
3779 GILBERT AVE
MC KEE CITY       NJ  08310
```

Figure 3-25. Output Report from Updating the Relative File

The direct file INVENTORY is accessed by alternate key in the program shown in figure 3-32. This program reads an input card containing the code of an ordered item and the quantity ordered. Records are then read from the file INVENTORY to reserve the parts needed and to determine whether parts must be ordered. The records are updated to reflect the reserved quantities and, if applicable, the ordering of parts.

The access mode for the file INVENTORY is dynamic (line 13) to allow records to be read both randomly and sequentially. The FILE STATUS clause (line 13) specifies a data item to receive a status code after execution of an input/output statement referencing the file.

The file is opened for input and output (line 86) so that records can be read and rewritten on the file. An input card is read (line 92) and the item code is moved to the first occurrence of the repeating group alternate key WHERE-USED (line 94).

The random READ statement (line 95) specifies an alternate key value from the input card as the key of reference. When this statement is executed, the index file is searched for an alternate key value that is equal to the current value in the first occurrence of WHERE-USED in the record area. The record with the first primary key indexed for the WHERE-USED value is then retrieved from the file INVENTORY. The status code returned to the KEY-CHECK data item is moved to the KEY-SAVE data item (line 97) for later reference.

After the record is updated (lines 99 through 110), a line is written on the output report and the updated record is rewritten on the file INVENTORY (lines 112 through 119). The status code saved after the read operation is then checked to determine whether the file contains another record with the same WHERE-USED value. If the status code in KEY-SAVE is 02 (lines 120 and 121), the next record in alternate key sequence is indexed for the same WHERE-USED value and a sequential read should be executed (lines 126 and 127). If KEY-SAVE is not equal to 02, all records with the same WHERE-USED value have been read and another input card should be read (line 124).

```
 1   IDENTIFICATION DIVISION.
 2   PROGRAM-ID.  NEW-IND.
 3   ENVIRONMENT DIVISION.
 4   CONFIGURATION SECTION.
 5   SOURCE-COMPUTER.  CYBER-170.
 6   OBJECT-COMPUTER.  CYBER-170.
 7   INPUT-OUTPUT SECTION.
 8   FILE-CONTROL.
 9       SELECT CARD-IN  ASSIGN TO INPUT.
10       SELECT EMP-FILE ASSIGN TO EMPFLE, INDFLE
11           ORGANIZATION IS INDEXED
12           ACCESS MODE IS SEQUENTIAL
13           RECORD KEY IS EMP-ID
14           ALTERNATE RECORD KEY IS HIRE-DATE
15               WITH DUPLICATES ASCENDING
16           ALTERNATE RECORD KEY IS JOB-ID
17               WITH DUPLICATES ASCENDING.
18   DATA DIVISION.
19   FILE SECTION.
20   FD  CARD-IN
21           LABEL RECORD IS OMITTED
22           DATA RECORDS ARE CARD-1, CARD-2.
```

Figure 3-26. Creating a File With Indexed Organization (Sheet 1 of 2)

```
01  CARD-1.
    03  EMP-ID-1              PICTURE 999.
    03  FILLER                PICTURE X.
    03  EMP-NAME              PICTURE X(20).
    03  EMP-ADDRESS.
        05  STREET            PICTURE X(20).
        05  CITY              PICTURE X(20).
        05  STATE             PICTURE AA.
        05  FILLER            PICTURE X.
        05  ZIP-CODE          PICTURE 9(5).
    03  FILLER                PICTURE X(8).
01  CARD-2.
    03  EMP-ID-2              PICTURE 999.
    03  FILLER                PICTURE X.
    03  JOB-ID-IN             PICTURE X(5).
    03  FILLER                PICTURE X(5).
    03  DEPT                  PICTURE 999.
    03  FILLER                PICTURE XX.
    03  DIV                   PICTURE 999.
    03  FILLER                PICTURE XX.
    03  HIRE-DATE-IN          PICTURE 9(6).
    03  FILLER                PICTURE X(4).
    03  LOCATION              PICTURE 999.
    03  FILLER                PICTURE X(43).
FD  EMP-FILE
        LABEL RECORD IS OMITTED
        BLOCK CONTAINS 20 RECORDS
        RECORD CONTAINS 90 CHARACTERS
        DATA RECORD IS EMPLOYEE.
01  EMPLOYEE.
    03  EMP-ID                PICTURE 999.
    03  EMP-NAME              PICTURE X(20).
    03  EMP-ADDRESS.
        05  STREET            PICTURE X(20).
        05  CITY              PICTURE X(20).
        05  STATE             PICTURE AA.
        05  ZIP-CODE          PICTURE 9(5).
    03  JOB-ID                PICTURE X(5).
    03  DEPT                  PICTURE 999.
    03  DIV                   PICTURE 999.
    03  HIRE-DATE             PICTURE 9(6).
    03  LOCATION              PICTURE 999.
PROCEDURE DIVISION.
OPEN-FILES.
    OPEN INPUT CARD-IN.
    OPEN OUTPUT EMP-FILE.
READ-CARDS.
    READ CARD-IN RECORD
        AT END GO TO CLOSE-FILES.
    MOVE EMP-ID-1 TO EMP-ID.
    MOVE CORRESPONDING CARD-1 TO EMPLOYEE.
    READ CARD-IN RECORD
        AT END GO TO INPUT-ERROR.
    IF EMP-ID-2 NOT EQUAL TO EMP-ID
        GO TO INPUT-ERROR.
    MOVE JOB-ID-IN TO JOB-ID.
    MOVE HIRE-DATE-IN TO HIRE-DATE.
    MOVE CORRESPONDING CARD-2 TO EMPLOYEE.
    WRITE EMPLOYEE
        INVALID KEY GO TO BAD-RECORD.
    GO TO READ-CARDS.
INPUT-ERROR.
    DISPLAY "CARD-2 MISSING OR IN ERROR ".
    DISPLAY EMPLOYEE.
    GO TO READ-CARDS.
BAD-RECORD.
    DISPLAY "INVALID RECORD - " EMPLOYEE.
    GO TO READ-CARDS.
CLOSE-FILES.
    CLOSE CARD-IN, EMP-FILE.
    STOP RUN.
```

Figure 3-26. Creating a File With Indexed Organization (Sheet 2 of 2)

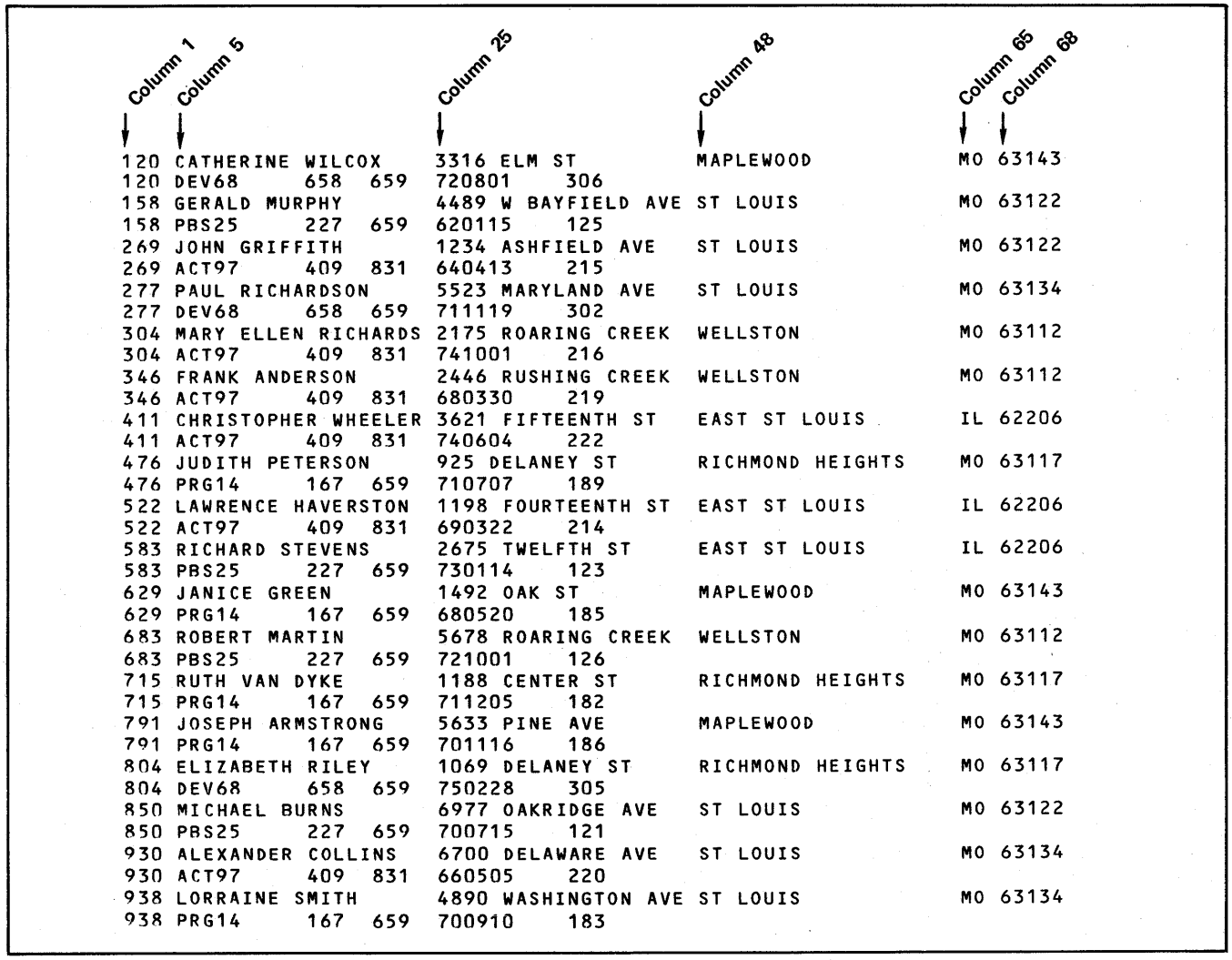


Figure 3-27. Input Data for Creating the Indexed File

The format of the input cards used to access the direct file INVENTORY is illustrated in figure 3-33. An output report generated by the program is shown in figure 3-34.

ACTUAL-KEY FILE PROGRAMS

A file with actual-key organization is created and then updated by two programs shown in this section. The actual-key file is a file of customer records. Each customer record contains order totals on a monthly basis as well as year-to-date totals.

The program shown in figure 3-35 creates the actual-key file CUSTOMERS. The primary key is the data item ACT-KEY, which is described as an eight-digit COMP-1 item (lines 13 and 34). The value of ACT-KEY specifies the location of the record in the file. Two alternate keys (CUST-ID and CUST-TYPE) are specified (lines 14 through 16). Because each customer has a unique identification number, duplicate CUST-ID values are not allowed. Duplicate values are allowed for the alternate key CUST-TYPE.

Before each record is written, the primary key ACT-KEY is set to zero (line 53). The system then generates the primary key value automatically and stores it in the ACT-KEY data item. Data from the input card (see figure 3-36) is moved into the output record area (lines 54 through 58). The remaining data items in the output record are initialized with a value of zero (lines 59 and 60). The record is then written on the file CUSTOMERS (line 61).

Records in the file CUSTOMERS are updated by the program shown in figure 3-37. Each input card contains the date and amount of an order for a customer. When an input card is read, the customer identification (CUST-ID-IN) is moved to the alternate key data item CUST-ID (lines 81 through 83). A CUSTOMERS file record is then read by the alternate key CUST-ID (line 84). The record is updated to reflect the new order (lines 86 through 88) and is rewritten on the file (line 95).

An output report is also generated during program execution. This report shows the customers for which orders were processed and the new year-to-date totals.

The format of the input cards is illustrated in figure 3-38. An output report generated by the program is shown in figure 3-39.

```
 1   IDENTIFICATION DIVISION.
 2   PROGRAM-ID.  LST-IND.
 3   ENVIRONMENT DIVISION.
 4   CONFIGURATION SECTION.
 5   SOURCE-COMPUTER.  CYBER-170.
 6   OBJECT-COMPUTER.  CYBER-170.
 7   INPUT-OUTPUT SECTION.
 8   FILE-CONTROL.
 9       SELECT EMP-FILE ASSIGN TO EMPFLE, INDFLE
10           ORGANIZATION IS INDEXED
11           ACCESS MODE IS DYNAMIC
12           RECORD KEY IS EMP-ID
13           ALTERNATE RECORD KEY IS HIRE-DATE
14               WITH DUPLICATES ASCENDING
15           ALTERNATE RECORD KEY IS JOB-ID
16               WITH DUPLICATES ASCENDING.
17       SELECT PRINTOUT ASSIGN TO OUTPUT.
18   DATA DIVISION.
19   FILE SECTION.
20   FD  EMP-FILE
21           LABEL RECORD IS OMITTED
22           BLOCK CONTAINS 20 RECORDS
23           RECORD CONTAINS 90 CHARACTERS
24           DATA RECORD IS EMPLOYEE.
25   01  EMPLOYEE.
26       03  EMP-ID                  PICTURE 999.
27       03  EMP-NAME                PICTURE X(20).
28       03  EMP-ADDRESS.
29           05  STREET              PICTURE X(20).
30           05  CITY                PICTURE X(20).
31           05  STATE               PICTURE AA.
32           05  ZIP-CODE            PICTURE 9(5).
33       03  JOB-ID                  PICTURE X(5).
34       03  DEPT                    PICTURE 999.
35       03  DIV                     PICTURE 999.
36       03  HIRE-DATE               PICTURE 9(6).
37       03  LOCATION                PICTURE 999.
38   FD  PRINTOUT
39           LABEL RECORD IS OMITTED
40           DATA RECORD IS PRINTLINE.
41   01  PRINTLINE                   PICTURE X(136).
42   WORKING-STORAGE SECTION.
43   01  DATE-CARD.
44       03  DATE-IN                 PICTURE 9(6).
45       03  FILLER                  PICTURE X(74).
46   01  HEAD-OUT.
47       03  FILLER                  PICTURE 9       VALUE 1.
48       03  FILLER                  PICTURE X(5)    VALUE " DATE".
49       03  FILLER                  PICTURE X(13)   VALUE SPACES.
50       03  FILLER                  PICTURE X(4)    VALUE "NAME".
51       03  FILLER                  PICTURE X(11)   VALUE SPACES.
52       03  FILLER                  PICTURE X(11)   VALUE "JOB-ID     ".
53       03  FILLER                  PICTURE X(11)   VALUE "EMPLOYEE-ID".
54       03  FILLER                  PICTURE X(80)   VALUE SPACES.
55   01  LINE-OUT.
56       03  FILLER                  PICTURE X       VALUE SPACES.
57       03  DATE-OUT                PICTURE 9(6).
58       03  FILLER                  PICTURE X(4)    VALUE SPACES.
59       03  EMP-NAME-OUT            PICTURE X(20).
60       03  FILLER                  PICTURE X(4)    VALUE SPACES.
61       03  ID-OUT                  PICTURE X(5).
62       03  FILLER                  PICTURE X(9)    VALUE SPACES.
63       03  EMP-ID-OUT              PICTURE 999.
64       03  FILLER                  PICTURE X(84)   VALUE SPACES.
65   PROCEDURE DIVISION.
66   OPENING.
67       OPEN INPUT EMP-FILE.
68       OPEN OUTPUT PRINTOUT.
69       PERFORM PRINT-HEAD.
```

Figure 3-28. Accessing an Indexed File by Alternate Key (Sheet 1 of 2)

```
70   SETTING-UP.
71       ACCEPT DATE-CARD.
72       MOVE DATE-IN TO HIRE-DATE.
73       START EMP-FILE
74           KEY IS NOT LESS THAN HIRE-DATE
75           INVALID KEY GO TO BAD-DATE.
76   READING.
77       READ EMP-FILE NEXT RECORD
78           AT END GO TO CLOSE-OUT.
79       MOVE HIRE-DATE TO DATE-OUT.
80       MOVE EMP-NAME TO EMP-NAME-OUT.
81       MOVE JOB-ID TO ID-OUT.
82       MOVE EMP-ID TO EMP-ID-OUT.
83       WRITE PRINTLINE FROM LINE-OUT.
84       GO TO READING.
85   PRINT-HEAD.
86       WRITE PRINTLINE FROM HEAD-OUT.
87       MOVE SPACES TO PRINTLINE.
88       WRITE PRINTLINE.
89   BAD-DATE.
90       DISPLAY "NO EMPLOYEES HIRED FROM " DATE-IN.
91   CLOSE-OUT.
92       CLOSE EMP-FILE, PRINTOUT.
93       STOP RUN.
```

Figure 3-28. Accessing an Indexed File by Alternate Key (Sheet 2 of 2)

DATE	NAME	JOB-ID	EMPLOYEE-ID
700715	MICHAEL BURNS	PBS25	850
700910	LORRAINE SMITH	PRG14	938
701116	JOSEPH ARMSTRONG	PRG14	791
710707	JUDITH PETERSON	PRG14	476
711119	PAUL RICHARDSON	DEV68	277
711205	RUTH VAN DYKE	PRG14	715
720801	CATHERINE WILCOX	DEV68	120
721001	ROBERT MARTIN	PBS25	683
730114	RICHARD STEVENS	PBS25	583
740604	CHRISTOPHER WHEELER	ACT97	411
741001	MARY ELLEN RICHARDS	ACT97	304
750228	ELIZABETH RILEY	DEV68	804

Figure 3-29. Output Report from Accessing the Indexed File

```
 1  IDENTIFICATION DIVISION.
 2  PROGRAM-ID.  NEW-DIR.
 3  ENVIRONMENT DIVISION.
 4  CONFIGURATION SECTION.
 5  SOURCE-COMPUTER.  CYBER-170.
 6  OBJECT-COMPUTER.  CYBER-170.
 7  INPUT-OUTPUT SECTION.
 8  FILE-CONTROL.
 9      SELECT CARD-FILE ASSIGN TO INPUT.
10      SELECT INVENTORY ASSIGN TO INVNTRY, INVIDX
11          ORGANIZATION IS DIRECT
12          BLOCK COUNT IS 50
13          ACCESS MODE IS DYNAMIC
14          RECORD KEY IS PART-NO
15          ALTERNATE RECORD KEY IS WHERE-USED
16              WITH DUPLICATES ASCENDING.
```

Figure 3-30. Creating a File With Direct Organization (Sheet 1 of 2)

```
17  DATA DIVISION.
18  FILE SECTION.
19  FD   CARD-FILE
20           LABEL RECORDS ARE OMITTED
21           DATA RECORD IS CARD-REC.
22  01   CARD-REC.
23       03   PART-NUM              PICTURE 9(5).
24       03   DESCRIPTION           PICTURE X(15).
25       03   QTY-ON-HAND           PICTURE 9(4).
26       03   QTY-ON-ORDER          PICTURE 9(4).
27       03   QTY-RESERVED          PICTURE 9(5).
28       03   ORDER-DATE            PICTURE 9(6).
29       03   REORDER-POINT         PICTURE 9(4).
30       03   REORDER-QTY           PICTURE 9(4).
31       03   QTY-PER-UNIT          PICTURE 99.
32       03   NO-USED               PICTURE 9.
33       03   USED-WITH             PICTURE X(5)
34                OCCURS 1 TO 6 TIMES
35                DEPENDING ON NO-USED.
39  FD   INVENTORY
40           LABEL RECORDS ARE OMITTED
41           BLOCK CONTAINS 25 RECORDS
42           RECORD IS VARYING IN SIZE FROM 55 TO 80 CHARACTERS
43           DATA RECORD IS INV-REC.
44  01   INV-REC.
45       03   PART-NO               PICTURE 9(5).
46       03   DESCRIPTION           PICTURE X(15).
47       03   QTY-ON-HAND           PICTURE 9(4).
48       03   QTY-ON-ORDER          PICTURE 9(4).
48A      03   QTY-RESERVED          PICTURE 9(5).
49       03   ORDER-DATE            PICTURE 9(6).
50       03   REORDER-POINT         PICTURE 9(4).
51       03   REORDER-QTY           PICTURE 9(4).
52       03   QTY-PER-UNIT          PICTURE 99.
53       03   NUM-USED              PICTURE 9.
54       03   WHERE-USED            PICTURE X(5)
55                OCCURS 1 TO 6 TIMES
56                DEPENDING ON NUM-USED.
57  WORKING-STORAGE SECTION.
58  01   CNTR                      PICTURE 9.
59  PROCEDURE DIVISION.
60  OPENING.
61       OPEN INPUT CARD-FILE.
62       OPEN OUTPUT INVENTORY.
63  WRITE-FILE.
64       READ CARD-FILE RECORD
65           AT END GO TO CLOSING.
66       MOVE CORRESPONDING CARD-REC TO INV-REC.
67       MOVE PART-NUM TO PART-NO.
68       MOVE NO-USED TO NUM-USED.
69       MOVE 1 TO CNTR.
70       PERFORM MOVE-ALT-KEY NO-USED TIMES.
71       WRITE INV-REC
72           INVALID KEY GO TO BAD-RECORD.
73       GO TO WRITE-FILE.
74  MOVE-ALT-KEY.
75       MOVE USED-WITH (CNTR) TO WHERE-USED (CNTR).
76       ADD 1 TO CNTR.
77  BAD-RECORD.
78       DISPLAY "RECORD NOT WRITTEN. REC IS " CARD-REC.
79       GO TO WRITE-FILE.
80  CLOSING.
81       CLOSE CARD-FILE, INVENTORY.
82       STOP RUN.
```

Figure 3-30. Creating a File With Direct Organization (Sheet 2 of 2)

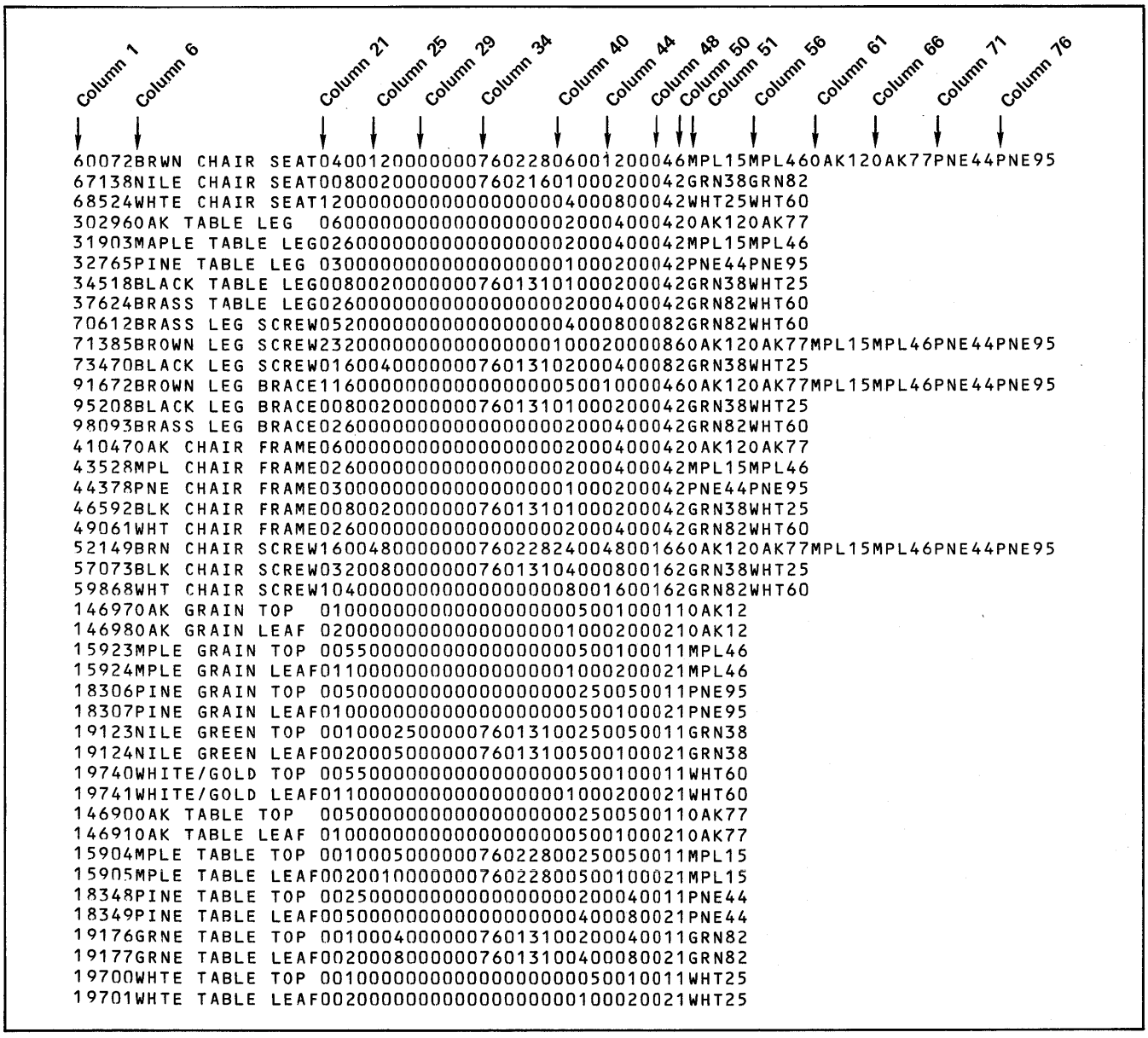


Figure 3-31. Input Data for Creating the Direct File

```
 1  IDENTIFICATION DIVISION.
 2  PROGRAM-ID.  UPD-DIR.
 3  ENVIRONMENT DIVISION.
 4  CONFIGURATION SECTION.
 5  SOURCE-COMPUTER.  CYBER-170.
 6  OBJECT-COMPUTER.  CYBER-170.
 7  INPUT-OUTPUT SECTION.
 8  FILE-CONTROL.
 9      SELECT ORDER-FILE ASSIGN TO INPUT.
10      SELECT INVENTORY ASSIGN TO INVNTRY, INVIDX
11          ORGANIZATION IS DIRECT
12          BLOCK COUNT IS 11
13          ACCESS MODE IS DYNAMIC
14          FILE STATUS IS KEY-CHECK
15          RECORD KEY IS PART-NO
16          ALTERNATE RECORD KEY IS WHERE-USED
17              WITH DUPLICATES ASCENDING.
18      SELECT PRINT-FILE ASSIGN TO OUTPUT.
19  DATA DIVISION.
20  FILE SECTION.
21  FD  ORDER-FILE
22          LABEL RECORDS ARE OMITTED
23          DATA RECORD IS ORDER-REC.
24  01  ORDER-REC.
25      03  ITEM                PICTURE XXX99.
26      03  FILLER              PICTURE X(5).
27      03  NO-ORDERED          PICTURE 999.
28      03  FILLER              PICTURE X(67).
29  FD  INVENTORY
30          LABEL RECORDS ARE OMITTED
31          BLOCK CONTAINS 20 RECORDS
32          RECORD IS VARYING IN SIZE FROM 55 TO 80 CHARACTERS
33          DATA RECORD IS INV-REC.
34  01  INV-REC.
35      03  PART-NO             PICTURE 9(5).
36      03  DESCRIPTION         PICTURE X(15).
37      03  QTY-ON-HAND         PICTURE 9(4).
38      03  QTY-ON-ORDER        PICTURE 9(4).
39      03  QTY-RESERVED        PICTURE 9(5).
40      03  ORDER-DATE          PICTURE 9(6).
41      03  REORDER-POINT       PICTURE 9(4).
42      03  REORDER-QTY         PICTURE 9(4).
43      03  QTY-PER-UNIT        PICTURE 99.
44      03  NUM-USED            PICTURE 9.
45      03  WHERE-USED          PICTURE X(5)
46              OCCURS 1 TO 6 TIMES
47              DEPENDING ON NUM-USED.
48  FD  PRINT-FILE
49          LABEL RECORDS ARE OMITTED
50          DATA RECORD IS PRINTLINE.
51  01  PRINTLINE               PICTURE X(136).
52  WORKING-STORAGE SECTION.
53  01  TEMP                    PICTURE 9(5).
54  01  QTY-NEEDED              PICTURE 9(5).
55  01  ORD-DATE                PICTURE 9(6).
56  01  KEY-CHECK               PICTURE XX.
57  01  KEY-SAVE                PICTURE 99.
58  01  HEAD.
59      03  FILLER              PICTURE 9       VALUE 1.
60      03  FILLER              PICTURE XX      VALUE SPACES.
61      03  FILLER              PICTURE X(12)   VALUE "ITEM ORDERED".
62      03  FILLER              PICTURE X(8)    VALUE SPACES.
63      03  FILLER              PICTURE X(11)   VALUE "PART NEEDED".
64      03  FILLER              PICTURE X(10)   VALUE SPACES.
65      03  FILLER              PICTURE X(11)   VALUE "DESCRIPTION".
66      03  FILLER              PICTURE X(10)   VALUE SPACES.
67      03  FILLER              PICTURE X(12)   VALUE "QTY RESERVED".
68      03  FILLER              PICTURE X(8)    VALUE SPACES.
69      03  FILLER              PICTURE X(12)   VALUE "ORDER AMOUNT".
70      03  FILLER              PICTURE X(39)   VALUE SPACES.
```

Figure 3-32. Updating a File With Direct Organization (Sheet 1 of 2)

```
 01  OUT-LINE.
     03  FILLER                  PICTURE X(6)    VALUE SPACES.
     03  ITEM-OUT                PICTURE X(5).
     03  FILLER                  PICTURE X(15)   VALUE SPACES.
     03  PART-OUT                PICTURE 9(5).
     03  FILLER                  PICTURE X(11)   VALUE SPACES.
     03  DESC-OUT                PICTURE X(15).
     03  FILLER                  PICTURE X(12)   VALUE SPACES.
     03  RESERVED-OUT            PICTURE ZZZZ.
     03  FILLER                  PICTURE X(16)   VALUE SPACES.
     03  REORD-QTY-OUT           PICTURE ZZZZ.
     03  FILLER                  PICTURE X(43)   VALUE SPACES.
 PROCEDURE DIVISION.
 OPEN-FILES.
     OPEN INPUT ORDER-FILE.
     OPEN I-O INVENTORY.
     OPEN OUTPUT PRINT-FILE.
     ACCEPT ORD-DATE FROM DATE.
     WRITE PRINTLINE FROM HEAD
         BEFORE ADVANCING 2 LINES.
 ITEM-READ.
     READ ORDER-FILE RECORD
         AT END GO TO CLOSE-FILES.
     MOVE ITEM TO WHERE-USED (1), ITEM-OUT.
     READ INVENTORY RECORD KEY IS WHERE-USED
         INVALID KEY GO TO NOT-FOUND.
     MOVE KEY-CHECK TO KEY-SAVE.
 QTY-CHECK.
     COMPUTE QTY-NEEDED = NO-ORDERED * QTY-PER-UNIT.
     COMPUTE TEMP = QTY-ON-HAND + QTY-ON-ORDER - QTY-RESERVED.
     IF QTY-NEEDED GREATER THAN TEMP
         GO TO REORDER.
     SUBTRACT QTY-NEEDED FROM TEMP.
     IF TEMP GREATER THAN REORDER-POINT
         MOVE ZERO TO REORD-QTY-OUT
         GO TO WRITE-LINE.
 REORDER.
     MOVE REORDER-QTY TO REORD-QTY-OUT.
     ADD REORDER-QTY TO QTY-ON-ORDER.
     MOVE ORD-DATE TO ORDER-DATE.
 WRITE-LINE.
     ADD QTY-NEEDED TO QTY-RESERVED.
     MOVE QTY-NEEDED TO RESERVED-OUT.
     MOVE PART-NO TO PART-OUT.
     MOVE DESCRIPTION TO DESC-OUT.
     WRITE PRINTLINE FROM OUT-LINE.
     MOVE SPACES TO ITEM-OUT.
     REWRITE INV-REC
         INVALID KEY PERFORM NO-REWRITE.
     IF KEY-SAVE IS EQUAL TO 02
         GO TO NEXT-PART.
     MOVE SPACES TO PRINTLINE.
     WRITE PRINTLINE.
     GO TO ITEM-READ.
 NEXT-PART.
     READ INVENTORY NEXT RECORD
         AT END GO TO ITEM-READ.
     MOVE KEY-CHECK TO KEY-SAVE.
     GO TO QTY-CHECK.
 NO-REWRITE.
     DISPLAY "RECORD NOT REWRITTEN FOR ABOVE PART".
 NOT-FOUND.
     DISPLAY "NO RECORDS FOR ITEM " ITEM.
     GO TO ITEM-READ.
 CLOSE-FILES.
     CLOSE ORDER-FILE, INVENTORY, PRINT-FILE.
     STOP RUN.
```

Figure 3-32. Updating a File With Direct Organization (Sheet 2 of 2)

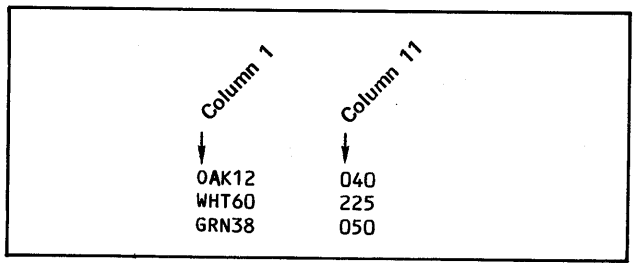


Figure 3-33. Input Data for Updating the Direct File

WORD-ADDRESS FILE PROGRAMS

Word-address file organization is illustrated in two programs. The first program creates the word-address file and the second program reads records from the file.

The word-address file PARTS-FILE is created by the program shown in figure 3-40. Input data is shown in figure 3-41. The word-address key is the Working-Storage data item PART-KEY (lines 13 and 40). When a record is written on the file, the value of PART-KEY identifies the number of the word in which the record begins. Each 30-character record is stored in three words; the part numbers, which are assigned in sequence, cannot be the key values. The key value for a record is computed by multiplying the part number by three and then subtracting two from the result (line 48).

The program shown in figure 3-42 reads records from the file PARTS-FILE in order to determine whether enough parts are on hand to satisfy the quantity needed. A record is read from the file CARD-FILE; the part number is used to calculate the word-address key value (lines 68 through 70). A record is then read from the file PARTS-FILE according to the calculated key value (line 71). The USED-FOR value in the CARD-FILE record is compared with the USED-WITH value in the PARTS-FILE record to ensure that a valid record has been read from PARTS-FILE (lines 73 and 74). A line is printed on the output report whenever the quantity on hand is less than the quantity needed (lines 79 through 83). The input records illustrated in figure 3-43 were used to create the output report shown in figure 3-44.

ITEM ORDERED	PART NEEDED	DESCRIPTION	QTY RESERVED	ORDER AMOUNT
OAK12	14697	OAK GRAIN TOP	40	
	14698	OAK GRAIN LEAF	80	
	30296	OAK TABLE LEG	160	
	41047	OAK CHAIR FRAME	160	
	52149	BRN CHAIR SCREW	640	
	60072	BRWN CHAIR SEAT	160	
	71385	BROWN LEG SCREW	320	
	91672	BROWN LEG BRACE	160	
WHT60	19740	WHITE/GOLD TOP	225	100
	19741	WHITE/GOLD LEAF	450	200
	37624	BRASS TABLE LEG	900	400
	49061	WHT CHAIR FRAME	900	400
	59868	WHT CHAIR SCREW	3600	1600
	68524	WHTE CHAIR SEAT	900	800
	70612	BRASS LEG SCREW	1800	800
	98093	BRASS LEG BRACE	900	400
GRN38	19123	NILE GREEN TOP	50	50
	19124	NILE GREEN LEAF	100	100
	34518	BLACK TABLE LEG	200	200
	46592	BLK CHAIR FRAME	200	200
	57073	BLK CHAIR SCREW	800	800
	67138	NILE CHAIR SEAT	200	200
	73470	BLACK LEG SCREW	400	400
	95208	BLACK LEG BRACE	200	200

Figure 3-34. Output Report from Updating the Direct File

```
1     IDENTIFICATION DIVISION.
2     PROGRAM-ID.  NEW-AK.
3     ENVIRONMENT DIVISION.
4     CONFIGURATION SECTION.
5     SOURCE-COMPUTER.  CYBER-170.
6     OBJECT-COMPUTER.  CYBER-170.
7     INPUT-OUTPUT SECTION.
8     FILE-CONTROL.
9         SELECT CARD-INPUT ASSIGN TO INPUT.
10        SELECT CUSTOMERS  ASSIGN TO CSTMRS, CSTINDX
11            ORGANIZATION IS ACTUAL-KEY
12            ACCESS MODE IS RANDOM
13            RECORD KEY IS ACT-KEY
14            ALTERNATE RECORD KEY IS CUST-ID
15            ALTERNATE RECORD KEY IS CUST-TYPE
16                WITH DUPLICATES ASCENDING.
17    DATA DIVISION.
18    FILE SECTION.
19    FD  CARD-INPUT
20            LABEL RECORDS ARE OMITTED
21            DATA RECORD IS CARD.
22    01  CARD.
23        03  CUST-ID-IN           PICTURE X(6).
24        03  FILLER               PICTURE XXX.
25        03  CUST-TYPE-IN         PICTURE XX.
26        03  FILLER               PICTURE XXX.
27        03  CUST-NAME-IN         PICTURE X(15).
28        03  FILLER               PICTURE X(51).
29    FD  CUSTOMERS
30            LABEL RECORDS ARE OMITTED
31            BLOCK CONTAINS 50 RECORDS
32            DATA RECORD IS CUST-REC.
33    01  CUST-REC.
34        03  ACT-KEY              PICTURE 9(8)  USAGE IS COMP-1.
35        03  CUST-ID              PICTURE X(6).
36        03  CUST-NAME            PICTURE X(15).
37        03  CUST-TYPE            PICTURE XX.
38        03  MONTHLY-ORDERS  OCCURS 12 TIMES.
39            05  NO-ORDERS        PICTURE 99.
40            05  MONTH-AMT        PICTURE 9(5)V99.
41        03  YTD-ORDERS.
42            05  TOTAL-ORDERS     PICTURE 999.
43            05  TOTAL-AMT        PICTURE 9(7)V99.
44        03  CURRENT-BAL          PICTURE 9(6)V99.
45        03  LAST-ACTIVITY        PICTURE 9(6).
46    WORKING-STORAGE SECTION.
47    01  COUNTER                  PICTURE 99.
48    PROCEDURE DIVISION.
49    BEGIN-1.
50        OPEN INPUT CARD-INPUT.
51        OPEN OUTPUT CUSTOMERS.
52    CREATING.
53        MOVE ZEROS TO ACT-KEY.
54        READ CARD-INPUT RECORD
55            AT END GO TO END-IT.
56        MOVE CUST-ID-IN TO CUST-ID.
57        MOVE CUST-TYPE-IN TO CUST-TYPE.
58        MOVE CUST-NAME-IN TO CUST-NAME.
59        INITIALIZE COUNTER, YTD-ORDERS, CURRENT-BAL, LAST-ACTIVITY.
60        PERFORM ZERO-SET 12 TIMES.
61        WRITE CUST-REC
62            INVALID KEY GO TO BAD-RECORD.
63        GO TO CREATING.
64    BAD-RECORD.
65        DISPLAY "RECORD NOT WRITTEN " CUST-ID.
66        GO TO CREATING.
67    ZERO-SET.
68        ADD 1 TO COUNTER.
69        MOVE ZEROS TO MONTHLY-ORDERS (COUNTER).
70    END-IT.
71        CLOSE CARD-INPUT, CUSTOMERS.
72        STOP RUN.
```

Figure 3-35. Creating a File With Actual-Key Organization

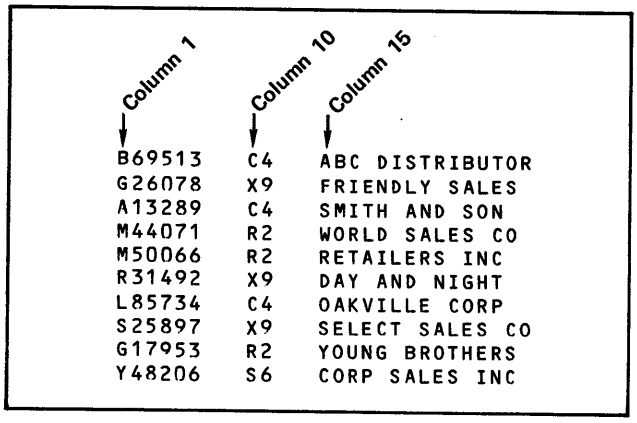


Figure 3-36. Input Data for Creating the Actual Key File

```
 1   IDENTIFICATION DIVISION.
 2   PROGRAM-ID.  UPD-AK.
 3   ENVIRONMENT DIVISION.
 4   CONFIGURATION SECTION.
 5   SOURCE-COMPUTER.  CYBER-170.
 6   OBJECT-COMPUTER.  CYBER-170.
 7   INPUT-OUTPUT SECTION.
 8   FILE-CONTROL.
 9       SELECT INVOICES ASSIGN TO INPUT.
10       SELECT CUSTOMERS ASSIGN TO CSTMRS, CSTINDX
11           ORGANIZATION IS ACTUAL-KEY
12           ACCESS MODE IS DYNAMIC
13           RECORD KEY IS ACT-KEY
14           ALTERNATE RECORD KEY IS CUST-ID
15           ALTERNATE RECORD KEY IS CUST-TYPE
16               WITH DUPLICATES ASCENDING.
17       SELECT PRINT-OUT ASSIGN TO OUTPUT.
18   DATA DIVISION.
19   FILE SECTION.
20   FD   INVOICES
21           LABEL RECORDS ARE OMITTED
22           DATA RECORD IS INV-RECORD.
23   01   INV-RECORD.
24       03   CUST-ID-IN          PICTURE X(6).
25       03   FILLER              PICTURE XXX.
26       03   ORDER-DATE          PICTURE 9(6).
27       03   FILLER              PICTURE XXXX.
28       03   ORDER-AMT           PICTURE 9(5)V99.
29       03   FILLER              PICTURE XXX.
30       03   MONTH               PICTURE 99.
31       03   FILLER              PICTURE X(49).
32   FD   CUSTOMERS
33           LABEL RECORDS ARE OMITTED
34           BLOCK CONTAINS 50 RECORDS
35           DATA RECORD IS CUST-REC.
36   01   CUST-REC.
37       03   ACT-KEY             PICTURE 9(8)    USAGE IS COMP-1.
38       03   CUST-ID             PICTURE X(6).
39       03   CUST-NAME           PICTURE X(15).
40       03   CUST-TYPE           PICTURE XX.
41       03   MONTHLY-ORDERS  OCCURS 12 TIMES.
42            05   NO-ORDERS      PICTURE 99.
43            05   MONTH-AMT      PICTURE 9(5)V99.
```

Figure 3-37. Updating a File With Actual-Key Organization (Sheet 1 of 2)

```
44          03  YTD-ORDERS.
45              05  TOTAL-ORDERS     PICTURE 999.
46              05  TOTAL-AMT        PICTURE 9(7)V99.
47          03  CURRENT-BAL          PICTURE 9(6)V99.
48          03  LAST-ACTIVITY        PICTURE 9(6).
49      FD  PRINT-OUT
50              LABEL RECORDS ARE OMITTED
51              LINAGE IS 50 LINES
52              DATA RECORD IS OUT-LINE.
53      01  OUT-LINE                 PICTURE X(136).
54      WORKING-STORAGE SECTION.
55      01  HEADER.
56          03  FILLER               PICTURE X(8)  VALUE " CUST-ID".
57          03  FILLER               PICTURE X(6)  VALUE SPACES.
58          03  FILLER               PICTURE X(13) VALUE "CUSTOMER NAME".
59          03  FILLER               PICTURE X(6)  VALUE SPACES.
60          03  FILLER               PICTURE X(10) VALUE "NO. ORDERS".
61          03  FILLER               PICTURE X(7)  VALUE SPACES.
62          03  FILLER               PICTURE X(10) VALUE "YTD AMOUNT".
63          03  FILLER               PICTURE X(76) VALUE SPACES.
64      01  LINE-1.
65          03  FILLER               PICTURE X     VALUE SPACES.
66          03  ID-OUT               PICTURE X(6).
67          03  FILLER               PICTURE X(6)  VALUE SPACES.
68          03  NAME-OUT             PICTURE X(15).
69          03  FILLER               PICTURE X(9)  VALUE SPACES.
70          03  ORDERS-OUT           PICTURE Z9.
71          03  FILLER               PICTURE X(9)  VALUE SPACES.
72          03  AMT-OUT              PICTURE $$,$$$,$99.99.
73          03  FILLER               PICTURE X(75) VALUE SPACES.
74      PROCEDURE DIVISION.
75      OPEN-FILES.
76          OPEN INPUT INVOICES.
77          OPEN I-O CUSTOMERS.
78          OPEN OUTPUT PRINT-OUT.
79          PERFORM HEAD-OUT.
80      UPDATING.
81          READ INVOICES RECORD
82              AT END GO TO CLOSE-OUT.
83          MOVE CUST-ID-IN TO CUST-ID.
84          READ CUSTOMERS RECORD KEY IS CUST-ID
85              INVALID KEY GO TO NO-RECORD.
86          MOVE ORDER-DATE TO LAST-ACTIVITY.
87          ADD 1 TO NO-ORDERS (MONTH), TOTAL-ORDERS.
88          ADD ORDER-AMT TO MONTH-AMT (MONTH), TOTAL-AMT.
89          MOVE CUST-ID TO ID-OUT.
90          MOVE CUST-NAME TO NAME-OUT.
91          MOVE TOTAL-ORDERS TO ORDERS-OUT.
92          MOVE TOTAL-AMT TO AMT-OUT.
93          WRITE OUT-LINE FROM LINE-1
94              AT END-OF-PAGE PERFORM HEAD-OUT.
95          REWRITE CUST-REC
96              INVALID KEY GO TO KEY-ERROR.
97          GO TO UPDATING.
98      HEAD-OUT.
99          WRITE OUT-LINE FROM HEADER
100             BEFORE ADVANCING 2 LINES.
101     NO-RECORD.
102         DISPLAY "NO RECORD FOR " CUST-ID.
103         GO TO UPDATING.
104     KEY-ERROR.
105         DISPLAY "RECORD NOT REWRITTEN " CUST-ID.
106         GO TO UPDATING.
107     CLOSE-OUT.
108         CLOSE INVOICES, CUSTOMERS, PRINT-OUT.
109         STOP RUN.
```

Figure 3-37. Updating a File With Actual-Key Organization (Sheet 2 of 2)

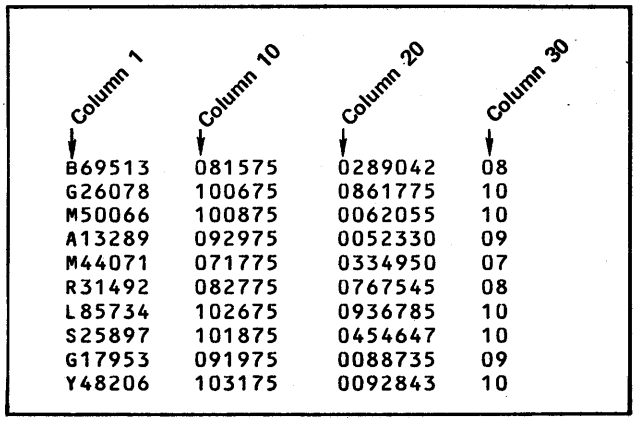


Figure 3-38. Input Data for Updating the Actual-Key File

CUST-ID	CUSTOMER NAME	NO. ORDERS	YTD AMOUNT
B69513	ABC DISTRIBUTOR	5	$12,282.92
G26078	FRIENDLY SALES	4	$28,619.83
M50066	RETAILERS INC	4	$11,203.11
A13289	SMITH AND SON	6	$9,810.29
M44071	WORLD SALES CO	4	$28,424.86
R31492	DAY AND NIGHT	4	$27,335.96
L85734	OAKVILLE CORP	5	$43,735.87
S25897	SELECT SALES CO	4	$29,467.63
G17953	YOUNG BROTHERS	5	$14,970.53
Y48206	CORP SALES INC	4	$23,122.90

Figure 3-39. Output Report from Updating the Actual-Key File

```
 1   IDENTIFICATION DIVISION.
 2   PROGRAM-ID.  NEW-WA.
 3   ENVIRONMENT DIVISION.
 4   CONFIGURATION SECTION.
 5   SOURCE-COMPUTER.  CYBER-170.
 6   OBJECT-COMPUTER.  CYBER-170.
 7   INPUT-OUTPUT SECTION.
 8   FILE-CONTROL.
 9       SELECT CARD-IN  ASSIGN TO INPUT.
10       SELECT PARTS-FILE ASSIGN TO PARTFLE
11           ORGANIZATION IS WORD-ADDRESS
12           ACCESS MODE IS DYNAMIC
13           WORD-ADDRESS KEY IS PART-KEY.
14   DATA DIVISION.
15   FILE SECTION.
16   FD  CARD-IN
17           LABEL RECORDS ARE OMITTED
18           DATA RECORD IS CARD-REC.
19   01  CARD-REC.
20       03  PART                PICTURE 9(4).
21       03  FILLER              PICTURE X(5).
22       03  PART-NAME           PICTURE X(10).
23       03  FILLER              PICTURE X(5).
24       03  USED-WITH           PICTURE X(5).
25       03  FILLER              PICTURE X(5).
26       03  QTY-ON-HAND         PICTURE 9(5).
27       03  FILLER              PICTURE X(5).
28       03  MFG-CODE            PICTURE X(10).
29       03  FILLER              PICTURE X(26).
30   FD  PARTS-FILE
31           LABEL RECORDS ARE OMITTED
32           RECORD CONTAINS 30 CHARACTERS
33           DATA RECORD IS PART-REC.
34   01  PART-REC.
35       03  PART-NAME           PICTURE X(10).
36       03  USED-WITH           PICTURE X(5).
37       03  QTY-ON-HAND         PICTURE 9(5).
38       03  MFG-CODE            PICTURE X(10).
39   WORKING-STORAGE SECTION.
40   01  PART-KEY                PICTURE 9(4).
41   PROCEDURE DIVISION.
42   OPEN-FILES.
43       OPEN INPUT CARD-IN.
44       OPEN OUTPUT PARTS-FILE.
45   CREATE-FILE.
46       READ CARD-IN RECORD
47           AT END GO TO CLOSE-FILE.
48       COMPUTE PART-KEY = PART * 3 - 2.
49       MOVE CORRESPONDING CARD-REC TO PART-REC.
50       WRITE PART-REC
51           INVALID KEY GO TO NO-GOOD.
52       GO TO CREATE-FILE.
53   NO-GOOD.
54       DISPLAY "BAD KEY " PART.
55       GO TO CREATE-FILE.
56   CLOSE-FILE.
57       CLOSE CARD-IN, PARTS-FILE.
```

Figure 3-40. Creating a File with Word-Address Organization

```
Column 1    Column 10    Column 25    Column 35    Column 45
   ↓           ↓            ↓            ↓            ↓
0001        SCREW-10     CX48J        00192        YOUNGBR010
0002        SCREW-15     GT26L        00048        ABCDIST015
0003        SCREW-23     AV50Q        00099        HGHDWRES23
0004        SCREW-58     RM13Z        00753        ABCDIST058
0005        B-BRACKT     CX48J        00298        YOUNGBR069
0006        W-BRACKT     AV50Q        00040        HGHDWRES40
0007        G-BRACKT     RM13Z        00983        ABCDIST125
0008        S-BRACKT     GT26L        00480        ABCDIST163
0009        PEDSTL15     BU57F        00316        MILLERS115
0010        PEDSTL74     KY96P        00789        JHNSNSP174
0011        PEDSTL82     HD52W        00946        MILLERS182
0012        PEDSTL36     NI38E        00130        JHNSNSP136
0013        SHADE-43     HD52W        00488        XYZSUPLY43
0014        SHADE-16     KY96P        00697        MASNDIST16
```

Figure 3-41. Input Data for Creating the Word-Address File

```
 1   IDENTIFICATION DIVISION.
 2   PROGRAM-ID.  READ-WA.
 3   ENVIRONMENT DIVISION.
 4   CONFIGURATION SECTION.
 5   SOURCE-COMPUTER.  CYBER-170.
 6   OBJECT-COMPUTER.  CYBER-170.
 7   INPUT-OUTPUT SECTION.
 8   FILE-CONTROL.
 9       SELECT CARD-FILE ASSIGN TO INPUT.
10       SELECT PARTS-FILE ASSIGN TO PARTFLE
11           ORGANIZATION IS WORD-ADDRESS
12           ACCESS MODE IS DYNAMIC
13           WORD-ADDRESS KEY IS PART-KEY.
14       SELECT LIST-FILE ASSIGN TO OUTPUT.
15   DATA DIVISION.
16   FILE SECTION.
17   FD   CARD-FILE
18           LABEL RECORDS ARE OMITTED
19           DATA RECORD IS CARD-IN.
20   01   CARD-IN.
21        03   USED-FOR            PICTURE X(5).
22        03   FILLER              PICTURE X(4).
23        03   PART-NO             PICTURE 9(4).
24        03   FILLER              PICTURE X(6).
25        03   NO-NEEDED           PICTURE 9(5).
26        03   FILLER              PICTURE X(56).
27   FD   PARTS-FILE
28           LABEL RECORDS ARE OMITTED
29           RECORD CONTAINS 30 CHARACTERS
30           DATA RECORD IS PART-REC.
31   01   PART-REC.
32        03   PART-NAME           PICTURE X(10).
33        03   USED-WITH           PICTURE X(5).
34        03   QTY-ON-HAND         PICTURE 9(5).
35        03   MFG-CODE            PICTURE X(10).
36   FD   LIST-FILE
37           LABEL RECORDS ARE OMITTED
38           DATA RECORD IS LIST-LINE.
39   01   LIST-LINE               PICTURE X(60).
40   WORKING-STORAGE SECTION.
41   01   PART-KEY                PICTURE 9(4).
42   01   HEADS.
43        03   FILLER              PICTURE 9       VALUE 1.
44        03   FILLER              PICTURE X(9)    VALUE " PART NO.".
45        03   FILLER              PICTURE X(5)    VALUE SPACES.
```

Figure 3-42. Accessing a File With Word-Address Organization (Sheet 1 of 2)

```
46          03  FILLER                 PICTURE X(11)   VALUE "QTY ON HAND".
47          03  FILLER                 PICTURE X(5)    VALUE SPACES.
48          03  FILLER                 PICTURE X(10)   VALUE "QTY NEEDED".
49          03  FILLER                 PICTURE X(5)    VALUE SPACES.
50          03  FILLER                 PICTURE X(10)   VALUE "ORDER CODE".
51          03  FILLER                 PICTURE X(4)    VALUE SPACES.
52      01  LINE-OUT.
53          03  FILLER                 PICTURE X(4)    VALUE SPACES.
54          03  PART-NUM               PICTURE 9(4).
55          03  FILLER                 PICTURE X(10)   VALUE SPACES.
56          03  ON-HAND                PICTURE ZZZZ9.
57          03  FILLER                 PICTURE X(10)   VALUE SPACES.
58          03  NEEDED                 PICTURE ZZZZ9.
59          03  FILLER                 PICTURE X(8)    VALUE SPACES.
60          03  MANUFACTURER           PICTURE X(10).
61          03  FILLER                 PICTURE X(4)    VALUE SPACES.
62      PROCEDURE DIVISION.
63      OPENING.
64          OPEN INPUT CARD-FILE, PARTS-FILE.
65          OPEN OUTPUT LIST-FILE.
66          PERFORM HEADINGS.
67      PARTS-CHECK.
68          READ CARD-FILE RECORD
69              AT END GO TO CLOSE-OUT.
70          COMPUTE PART-KEY = PART-NO * 3 - 2.
71          READ PARTS-FILE RECORD
72              INVALID KEY GO TO BAD-KEY.
73          IF USED-FOR NOT EQUAL TO USED-WITH
74              GO TO BAD-KEY.
75          IF QTY-ON-HAND LESS THAN NO-NEEDED
76              PERFORM PRINT-LINE.
77          GO TO PARTS-CHECK.
78      PRINT-LINE.
79          MOVE PART-NO TO PART-NUM.
80          MOVE QTY-ON-HAND TO ON-HAND.
81          MOVE NO-NEEDED TO NEEDED.
82          MOVE MFG-CODE TO MANUFACTURER.
83          WRITE LIST-LINE FROM LINE-OUT.
84      HEADINGS.
85          WRITE LIST-LINE FROM HEADS.
86          MOVE SPACES TO LIST-LINE.
87          WRITE LIST-LINE.
88      BAD-KEY.
89          DISPLAY "BAD NUMBER " PART-NO.
90          GO TO PARTS-CHECK.
91      CLOSE-OUT.
92          CLOSE CARD-FILE, PARTS-FILE, LIST-FILE.
93          STOP RUN.
```

Figure 3-42. Accessing a File With Word-Address Organization (Sheet 2 of 2)

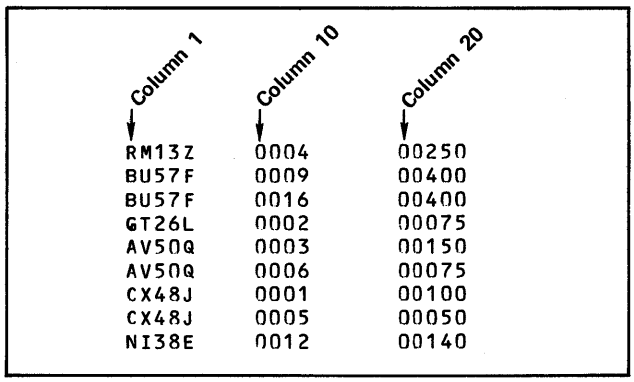


Figure 3-43. Input Data for Accessing the Word-Address File

1 PART NO.	QTY ON HAND	QTY NEEDED	ORDER CODE
0009	316	400	MILLERS115
BAD NUMBER	16		
0002	48	75	ABCDIST015
0003	99	150	HGHDWRES23
0006	40	75	HGHDWRES40
0012	130	140	JHNSNSP136

Figure 3-44. Output Report from Accessing the Word-Address File

ARITHMETIC AND BOOLEAN OPERATIONS 4

The COBOL 5 program can specify various arithmetic operations to be performed at execution time. These operations include addition, subtraction, multiplication, division, and exponentiation. Arithmetic operations can be specified through five Procedure Division statements: ADD, SUBTRACT, MULTIPLY, DIVIDE, and COMPUTE. Arithmetic expressions, which are used in COMPUTE statements, can also be specified in conditional statements. An arithmetic operation can be performed in display code, integer, or floating point mode of operation.

The COBOL 5 program can also specify boolean operations to be performed at execution time. Boolean operations can be specified through Procedure Division statements that include the boolean operators BOOLEAN-AND, BOOLEAN-OR, BOOLEAN-EXOR, or BOOLEAN-NOT. Boolean expressions can be used in COMPUTE statements and in conditional statements. The boolean character values can only be 0 or 1.

ARITHMETIC EXPRESSIONS

An arithmetic expression consists of data-names, numeric literals, and arithmetic operators, which are used to separate pairs of data-names and literals. A simple arithmetic expression contains two or more data-names and literals separated by arithmetic operators. A complex arithmetic expression contains two or more simple expressions separated by arithmetic operators. Parentheses can enclose arithmetic expressions to specify the order of evaluation or to clarify the logic of the expression.

ARITHMETIC OPERATORS

Two types of arithmetic operators can be used in arithmetic expressions: binary and unary. Five binary arithmetic operators are available:

Operator	Function
+	Addition
-	Subtraction
*	Multiplication
/	Division
**	Exponentiation

An arithmetic expression can be preceded by one of the following unary arithmetic operators:

Operator	Function
+	Multiplication by +1
-	Multiplication by -1

A unary operator can also precede an element (data-name or literal) within an arithmetic expression. The value of the data item or literal is effectively multiplied by the value +1 or -1.

An arithmetic operator must be preceded and followed by a space. With one exception, an arithmetic operator must be followed by a data-name or a literal; the one exception is that a binary operator can be followed by a unary operator. When parentheses are used, the data-name or literal can be preceded by a left parenthesis.

EVALUATION OF EXPRESSIONS

The order of evaluation for arithmetic expressions is determined by the arithmetic operators:

Operator	Order of Evaluation
Unary + and -	First
**	Second
* and /	Third
Binary + and -	Fourth

Expressions at the same level are evaluated from left to right.

Parentheses can be used to modify the normal sequence of evaluation. Expressions enclosed in parentheses are evaluated first, beginning with the innermost pair and proceeding to the outermost pair. Within a pair of parentheses, evaluation occurs according to the hierarchical order of evaluation.

SIMPLE ARITHMETIC EXPRESSIONS

A simple arithmetic expression consists of two or more data items separated by arithmetic operators. Each data item can be either a numeric literal or a data-name that identifies an elementary numeric data item described in the Data Division. The following examples are typical simple expressions:

```
SUBTOT + TAX

PRICE * DISCOUNT

SALARY / 40 * HOURS
```

COMPLEX ARITHMETIC EXPRESSIONS

A series of two or more simple expressions can be specified in a statement. The expressions are separated by arithmetic operators; parentheses can be used to enclosed each simple expression. The following examples are typical complex expressions:

```
(PRICE * QTY) + (PRE-BAL - DISCT)

(SALARY / 40) * (TOT-HRS - 40)
```

In the first example, PRICE is multiplied by QTY and DISCT is subtracted from PRE-BAL; the result of the first operation is then added to the result of the second operation.

For more complex expressions, parentheses can be nested within other pairs of parentheses. The expression within the innermost pair of parentheses is evaluated first. In the following example, C is added to D; B is then divided by the result of that operation.

```
(Z + A) * (B / (C + D))
```

ARITHMETIC STATEMENTS

Arithmetic operations are specified through arithmetic statements in the Procedure Division of a COBOL 5 program. Basic arithmetic operations can be performed using the ADD, SUBTRACT, MULTIPLY, and DIVIDE statements. More complex operations are performed through the COMPUTE statement.

Items to be used for computation in the arithmetic statements are referred to as operands. All operands must be elementary numeric items. Items in which results of arithmetic operations are stored are referred to as receiving items. A receiving item can be an operand or it can be a separate item that is not involved in the computation. Receiving items must be either elementary numeric items or elementary numeric-edited items. The storage location of a receiving item should not overlap that of any of the items involved in the computation; if the fields do overlap, unpredictable results might occur.

Numeric literals, data items described in the Data Division, and special registers can be specified as items in arithmetic statements. If the Data Division description of a data item includes the USAGE clause, the usage must be DISPLAY, COMPUTATIONAL, COMPUTATIONAL-1, COMPUTATIONAL-2, or COMPUTATIONAL-4. Display usage is assumed by default when the USAGE clause is not specified for a data item. Four special registers can be specified in arithmetic statements: LINAGE-COUNTER, HASHED-VALUE, LINE-COUNTER, and PAGE-COUNTER. The special register LINE-COUNTER cannot be used as a receiving item.

When the receiving item is a numeric-edited item, the result of the computation is edited before it is moved to the receiving item. Decimal point alignment is supplied automatically throughout computation.

ADDITION OF ITEMS

Two or more items are added together by the ADD statement. The receiving item for the result can be one of the operands or a separate item. The choice of the keyword TO or GIVING in the ADD statement determines whether the receiving item is an operand or a separate item.

The receiving item is an operand when the keyword TO is used. The operand preceding TO is added to the receiving item. When more than one operand precedes TO, the operands are added together and the result is added to the receiving item. If more than one receiving item is specified, the operand, or the result of multiple operands, is added to each receiving item.

The simplest format of the ADD statement adds one item to another item.

```
ADD ITEM-AMT TO ACCUM.

ADD 1 TO PAGE-COUNTER.
```

In the first statement, the value of the data item ITEM-AMT is added to the value of ACCUM and the result of the addition is stored as the new value of ACCUM. The second statement adds 1 to the special register PAGE-COUNTER.

A more complex addition operation occurs when multiple operands are specified.

```
ADD FED-TAX, SOC-SEC, STATE-TAX
    TO TOT-TAX, TOT-DED.
```

The three operands preceding TO are added together. The result of this addition is then added to each of the receiving items TOT-TAX and TOT-DED.

The keyword GIVING in the ADD statement indicates that the receiving item is not included in the addition operation. The operands preceding GIVING are added together and the result is stored in the receiving item. If more than one receiving item is specified, the result is stored in each receiving item.

```
ADD REG-PAY, OT-PAY GIVING GROSS-PAY.
```

When this statement is executed, the data item GROSS-PAY contains the result of adding the values of REG-PAY and OT-PAY.

The third available format of the ADD statement allows items within a group item to be added to corresponding items within another group item. The ADD statement specifies the data-name of each group item. When the statement is executed, the elementary items that have the same data-names and qualifiers are added together. The corresponding items are added and the results are stored in the receiving group item.

Figure 4-1 illustrates the use of the ADD CORRESPONDING statement. When this ADD statement is executed, only three items in RATE-TABLE are changed: the RATE data items for NEW-YORK, BOSTON, and LOS-ANGELES.

In the ADD CORRESPONDING statement, one or more receiving items are specified after the keyword TO. If more than one receiving item is specified, the corresponding items in the group item preceding TO are added to each receiving item.

Within the group items, data items described by (or subordinate to a data item described by) a REDEFINES clause or an OCCURS clause are ignored during the addition. Neither group item can contain any data item described with a RENAMES clause or a USAGE IS INDEX clause.

SUBTRACTION OF ITEMS

One or more items can be subtracted from another item through the SUBTRACT statement. The difference is stored in the receiving item, which can be an operand or a separate item.

When the receiving item is an operand, the result of the subtraction is stored as the new value of the item following the keyword FROM.

```
SUBTRACT DISCT FROM ACCUM.

SUBTRACT 5 FROM TEMP.
```

```
      .
      .
      .
DATA DIVISION.
      .
      .
      .
01  UPDATE-TABLE.
    03  EASTERN-REG.
        05  NEW-YORK.
            07  RATE        ...
        05  BOSTON.
            07    RATE       ...
    03  WESTERN-REG.
        05  LOS-ANGELES.
            07  RATE        ...
01  RATE-TABLE.
    03  EASTERN-REG.
        05  NEW-YORK.
            07  RATE        ...
        05  BOSTON.
            07  RATE        ...
        05  PHILADELPHIA.
            07  RATE        ...
      .
      .
      .
    03  WESTERN-REG.
        05  LOS-ANGELES.
            07  RATE        ...
        05  SAN-FRANCISCO.
            07  RATE        ...
      .
      .
      .
PROCEDURE DIVISION.
      .
      .
      .
      ADD CORRESPONDING UPDATE-TABLE
          TO RATE-TABLE.
```

Figure 4-1. Addition of Corresponding Items

The value of DISCT is subtracted from ACCUM; the difference is stored as the new value of ACCUM. In the second statement, the numeric literal 5 is subtracted from the value of TEMP; the difference is stored as the new value of TEMP.

More than one item can be subtracted from another item. The sum of the items preceding the keyword FROM is subtracted from the item following FROM.

```
SUBTRACT SOC-SEC, FED-TAX
    FROM GROSS-PAY.
```

The values of SOC-SEC and FED-TAX are added together and the sum is subtracted from GROSS-PAY. The difference is then stored as the new value of GROSS-PAY.

Multiple operands can also be specified after the keyword FROM. The sum of the operands preceding FROM is then subtracted from each operand following FROM and the difference is stored as the new value in each case.

```
SUBTRACT DISCT FROM TOTAL, AMT-DUE.
```

The value of DISCT is subtracted from TOTAL and from AMT-DUE. The differences are stored as the new values of TOTAL and AMT-DUE, respectively.

The difference computed by the SUBTRACT statement can be stored in a separate item by including the GIVING phrase. The subtraction is performed and the difference is stored in the item specified in the GIVING phrase.

```
SUBTRACT SOC-SEC, FED-TAX
    FROM GROSS-PAY GIVING NET-PAY.
```

The sum of SOC-SEC and FED-TAX is subtracted from GROSS-PAY. The difference is stored as the new value of NET-PAY.

Items within a group item can be subtracted from corresponding items within another group item. The data-names specified in the SUBTRACT CORRESPONDING statement identify the group items. Execution of the statement causes the elementary items in the first group item to be subtracted from the corresponding elementary items in the receiving group item. Corresponding items have the same data-names and qualifiers up to the group item level.

```
SUBTRACT CORRESPONDING UPDATE-TABLE
    FROM RATE-TABLE.
```

Using the data descriptions in figure 4-1, execution of this statement causes the three RATE data items in UPDATE-TABLE to be subtracted from the corresponding RATE data items in RATE-TABLE.

When more than one receiving item is specified in the SUBTRACT CORRESPONDING statement, the data items in the group item preceding FROM are subtracted from the corresponding data items in each receiving item.

During subtraction of corresponding items, data items described by, or subordinate to items described by, a REDEFINES clause or an OCCURS clause are ignored. Neither the sending nor the receiving group item can contain any data item described with a RENAMES clause or a USAGE IS INDEX clause.

MULTIPLICATION OF ITEMS

Multiplication of two items is accomplished through the MULTIPLY statement. The product of the multiplication process is stored in an operand or in one or more separate items.

The simplest format of the MULTIPLY statement multiplies one item by another item.

```
MULTIPLY INTEREST BY NEW-PRINC.
```

The value of INTEREST is multiplied by the value of NEW-PRINC. The resulting product is stored as the new value of NEW-PRINC.

More than one operand can be specified as a receiving item. Each of the receiving items stores the product of the operand preceding the keyword BY and the operand receiving item.

```
MULTIPLY 1.05 BY EARNINGS, OT-RATE,
    SOC-SEC, FED-TAX.
```

The numeric literal 1.05 is multiplied by each of the four receiving items. The product of each multiplication operation is stored in the respective receiving item.

The product can be stored in a receiving item that is not an operand by including the GIVING phrase in the MULTIPLY statement. One or more receiving items are specified to store the product of the two operands.

```
MULTIPLY .10 BY AMOUNT
    GIVING PERCENT, ACCUM.
```

The numeric literal .10 is multiplied by the value of AMOUNT. The product is then stored in the two receiving items PERCENT and ACCUM.

DIVISION OF ITEMS

One item can be divided by another item through the DIVIDE statement. The result of the division is stored either in an operand or in a separate item depending on the format of the statement.

A data item or a numeric literal can be divided into a data item that subsequently stores the quotient (the result of the division).

```
DIVIDE 1.5 INTO TEMP.
```

The numeric literal 1.5 (the divisor) is divided into the value of TEMP (the dividend). The quotient is then stored as the new value of TEMP.

If two or more dividends are specified, the divisor is divided into each dividend. The quotient resulting from each division operation is stored in the respective data item used as the dividend.

```
DIVIDE TEMP INTO CNTR1, CNTR2.
```

When this statement is executed, two division operations take place. The value of TEMP is divided into the value of CNTR1 and the quotient is stored as the new value of CNTR1. The value of TEMP is then divided into the value of CNTR2 and the resulting quotient is stored as the new value of CNTR2.

The GIVING phrase is used to specify one or more receiving items that are not operands. The position of the divisor and the dividend in the statement depends on the choice of the keyword INTO or BY. When INTO is specified, the first operand is the divisor and the second operand is the dividend. When BY is specified, the first operand is the dividend and the second operand is the divisor.

```
DIVIDE CNTR INTO TEMP GIVING AVERAGE.

DIVIDE TEMP BY CNTR GIVING AVERAGE.
```

In both examples, the divisor is CNTR and the dividend is TEMP. The quotient is stored as the new value of AVERAGE.

The remainder resulting from a division operation can also be stored in a data item. The remainder is calculated by subtracting the product of the quotient and the divisor from the dividend.

```
DIVIDE 12 INTO AMOUNT
    GIVING PAYMENT REMAINDER TEMP.
```

The quotient resulting from dividing 12 into the value of AMOUNT is stored in the data item PAYMENT. The remainder is calculated and stored in the data item TEMP.

The quotient that is used to calculate the remainder is an intermediate item. The picture-specification of the intermediate item is the same as the quotient and contains an operational sign; however, editing symbols are excluded from the intermediate item. Rounding, if specified for the receiving item, is not performed on the intermediate item. If the quotient is described as COMPUTATIONAL-2, the calculation for the remainder is always zero.

COMPUTING A DATA ITEM VALUE

The value of a data item is sometimes determined by performing a series of arithmetic operations. With the basic arithmetic statements already discussed, several statements could be required to obtain the desired result. A single COMPUTE statement can cause several different operations to be performed and the final result to be stored in the data item.

The arithmetic operations are specified in the COMPUTE statement as an arithmetic expression. Arithmetic operations that can be performed include addition, subtraction, multiplication, division, and exponentiation. When the COMPUTE statement is executed, the expression is evaluated and the result is stored in one or more receiving items.

The arithmetic expression consists of data-names, literals, and arithmetic operators. The structure of arithmetic expressions and the order of evaluation have been discussed previously in this section.

```
COMPUTE AMT-DUE = ACCUM - DISCT + TAX.
```

The value of DISCT is subtracted from the value of ACCUM. The difference is then added to the value of TAX and the result is stored as the new value of AMT-DUE.

The order of evaluation of the arithmetic expression can be explicitly stated by enclosing operands in parentheses. Arithmetic operations within parentheses are evaluated first.

```
COMPUTE TAX = (ACCUM - DISCT) * 0.06.
```

The expression enclosed in parentheses is evaluated; the result is then multiplied by the numeric literal 0.06. If parentheses are not used, the value of DISCT is multiplied by 0.06 and the product is then subtracted from the value of ACCUM.

In more complex expressions, parentheses can be nested. The expression within the innermost pair of parentheses is evaluated first. The result of the evaluation is then used to evaluate the expression within the next pair of parentheses. This process continues until the expression within the outermost pair of parentheses has been evaluated.

```
COMPUTE XVAL = X1 + (X2 * (X3 + X4)) -
    (X5 / (X6 + X7)).
```

In this example, four pairs of parentheses are used to explicitly specify the order of evaluation. The value of XVAL is computed as follows:

1. X3 is added to X4.
2. X2 is multiplied by the result of step 1.
3. X6 is added to X7.
4. X5 is divided by the result of step 3.

5. X1 is added to the result of step 2.

6. The result of step 4 is subtracted from the result of step 5.

Whenever possible, division should be the final arithmetic operation in order to preserve the accuracy of the result. Operands should not be described as COMPUTATIONAL-2 items.

ROUNDING A RESULT

The result computed by an arithmetic statement is stored in the receiving item according to the data description of the item. If the number of decimal places in the receiving item is less than the number of decimal places in the computed result, the excess digits are truncated. Rounding of a truncated result is performed when the ROUNDED option is specified for a receiving item.

When rounding is requested, the least significant digit of the receiving item is increased by 1 when the most significant truncated digit is 5 or greater. The following examples illustrate rounding for a receiving item that is described as PICTURE 99V99. The symbol ↑ indicates the decimal position.

Computed Result	Stored Result
25427 ↑	2543 ↑
91622 ↑	9162 ↑
63109 ↑	6311 ↑

A receiving item that is described with the character P in the rightmost positions can be rounded in the least significant stored digit position. Rounding occurs in the rightmost stored digit when the most significant truncated digit is equal to or greater than 5.

CHECKING FOR A SIZE ERROR

A size error occurs when the number of integral positions in the receiving item is less than the number of integral digits in the result. The SIZE ERROR option in an arithmetic statement provides the means to perform a specific function when a size error condition exists.

Size error checking is performed on the intermediate result and on the final result of any arithmetic operation. A size error always occurs for each of the following conditions:

- An exponentiation error.
- A floating point exponent overflow or underflow.
- Division by zero.

Exponent overflow or underflow causes program termination, unless a MODE control statement is in effect for these conditions.

The SIZE ERROR option specifies an imperative statement that is to be executed when a size error occurs. The imperative statement can transfer control to an error routine, print a message, or perform any required function.

```
ADD ITEM-AMT TO ACCUM
    ON SIZE ERROR GO TO ERR-PROC.
```

When this statement is executed, the result of the addition operation is checked for a size error. If a size error exists, control is transferred to the paragraph named ERR-PROC.

If the SIZE ERROR option is specified for an ADD CORRESPONDING or SUBTRACT CORRESPONDING statement, each individual arithmetic operation is checked for a size error condition. A size error detected for an individual operation does not cause the imperative statement to be executed immediately. All the individual additions or subtractions are performed before the imperative statement is executed.

When a size error occurs and the SIZE ERROR option is not included in the arithmetic statement, the value stored in the affected receiving item is undefined. If the SIZE ERROR option is specified, the receiving item is not changed when a size error occurs. A size error condition for one receiving item does not affect other receiving items included in the arithmetic operation.

NUMBER REPRESENTATION

All operands in arithmetic expressions and arithmetic statements must be numeric items. Arithmetic operations are performed in one of three modes: display code, integer, or floating point. The specific operation and the description of the operands determine the mode of operation.

DISPLAY CODE OPERATION

Addition and subtraction can be performed in the display code mode of operation. The operands must be described as display items with a maximum result size of 18 digits. Results obtained in the display code mode of operation are exact results.

INTEGER OPERATION

Integer mode of operation is performed on operands that are described as COMP-1 or COMP-4. This is the most efficient mode of operation and should be used whenever possible. For addition and subtraction, the size of the intermediate result, which is determined by aligning the operands on the decimal points, cannot exceed 14 digits. Multiplication in integer mode of operation requires that the sum of the operand sizes does not exceed 14 digits. Exponentiation can be performed only with zero point location.

FLOATING POINT OPERATION

Floating point mode of operation is used whenever the arithmetic operation cannot be performed in display code or integer mode. The maximum size of a floating point number is 28 digits. Floating point operations do not yield exact results.

SAMPLE ARITHMETIC PROGRAM

The use of the basic arithmetic statements is illustrated in the sample program shown in figure 4-2. The input file contains one record for each item on an invoice. The output file is a report that lists each item and its computed amount as well as the computed totals for each invoice. Figure 4-3 shows the input records used to create the output report shown in figure 4-4.

```
 1   IDENTIFICATION DIVISION.
 2   PROGRAM-ID.  ARITHMETIC-EXAMPLE.
 3   ENVIRONMENT DIVISION.
 4   CONFIGURATION SECTION.
 5   SOURCE-COMPUTER.  CYBER-170.
 6   OBJECT-COMPUTER.  CYBER-170.
 7   INPUT-OUTPUT SECTION.
 8   FILE-CONTROL.
 9       SELECT CARD-IN ASSIGN TO INPUT.
10       SELECT OUTFILE ASSIGN TO OUTPUT.
11   DATA DIVISION.
12   FILE SECTION.
13   FD  CARD-IN
14           LABEL RECORD IS OMITTED
15           DATA RECORD IS IN-REC.
16   01  IN-REC.
17       03  INVOICE-NO       PICTURE 9(6).
18       03  FILLER           PICTURE X(5).
19       03  ITEM-ID          PICTURE 999.
20       03  FILLER           PICTURE X(5).
21       03  QUANTITY         PICTURE 999.
22       03  FILLER           PICTURE X(5).
23       03  COST-PER-UNIT    PICTURE 99V99.
24       03  FILLER           PICTURE X(48).
25       03  END-FLAG         PICTURE A.
26   FD  OUTFILE
27           LABEL RECORD IS OMITTED
28           DATA RECORD IS PRINTLINE.
29   01  PRINTLINE            PICTURE X(136).
30   WORKING-STORAGE SECTION.
31   01  ITEM-AMT             PICTURE 9(4)V99 VALUE ZERO.
32   01  DISCT                PICTURE 9(4)V99 VALUE ZERO.
33   01  TAX                  PICTURE 9(4)V99 VALUE ZERO.
34   01  ACCUM                PICTURE 9(5)V99 USAGE COMP-1.
35   01  TEMP                 PICTURE 9(6)V99 USAGE COMP-1.
36   01  CNTR                 PICTURE 999     USAGE COMP-1.
37   01  QUANT                PICTURE 999     USAGE COMP-1.
38   01  UNIT-COST            PICTURE 99V99   USAGE COMP-1.
39   01  AVERAGE              PICTURE $ZZ,ZZ9.99.
40   01  HEAD.
41       03  FILLER           PICTURE X     VALUE "1".
42       03  FILLER           PICTURE X(12) VALUE "   INVOICE  ".
J0       03  FILLER           PICTURE X(6)  VALUE "  ITEM".
44       03  FILLER           PICTURE X(8)  VALUE "     QTY".
45       03  FILLER           PICTURE X(8)  VALUE SPACES.
46       03  FILLER           PICTURE X(12) VALUE "COST        ".
47       03  FILLER           PICTURE X(13) VALUE "DISCOUNT     ".
48       03  FILLER           PICTURE X(12) VALUE "SALES TAX   ".
49       03  FILLER           PICTURE X(12) VALUE "  AMOUNT DUE".
50       03  FILLER           PICTURE X(52) VALUE SPACES.
51   01  OUTPUT-LINE-1.
52       03  FILLER           PICTURE X(5) VALUE SPACES.
53       03  INVOICE-NUM      PICTURE 9(6).
54       03  FILLER           PICTURE X(5) VALUE SPACES.
55       03  ITEM-IDENT       PICTURE 999.
56       03  FILLER           PICTURE X(5) VALUE SPACES.
57       03  QTY              PICTURE ZZ9.
58       03  FILLER           PICTURE X(5) VALUE SPACES.
59       03  AMOUNT           PICTURE $Z,ZZ9.99.
60       03  FILLER           PICTURE X(95) VALUE SPACES.
61   01  OUTPUT-LINE-2.
62       03  FILLER           PICTURE X(31) VALUE SPACES.
63       03  INV-TOTAL        PICTURE $ZZ,ZZ9.99.
64       03  FILLER           PICTURE X(5)  VALUE "  -  ".
65       03  DISCOUNT         PICTURE $Z,ZZ9.99.
66       03  FILLER           PICTURE X(5)  VALUE "  +  ".
67       03  SALES-TAX        PICTURE $Z,ZZ9.99.
68       03  FILLER           PICTURE X(5)  VALUE "  =  ".
69       03  AMT-DUE          PICTURE $ZZ,ZZ9.99.
70       03  FILLER           PICTURE X(52) VALUE SPACES.
```

Figure 4-2. Sample Arithmetic Program (Sheet 1 of 2)

```
 71   PROCEDURE DIVISION.
 72   OPENING.
 73       OPEN INPUT CARD-IN.
 74       OPEN OUTPUT OUTFILE.
 75       MOVE ZEROS TO ACCUM, TEMP, CNTR.
 76       WRITE PRINTLINE FROM HEAD.
 77       MOVE SPACES TO PRINTLINE, WRITE PRINTLINE.
 78   READ-CARD.
 79       READ CARD-IN AT END GO TO CLOSING.
 80       MOVE INVOICE-NO TO INVOICE-NUM.
 81       MOVE ITEM-ID TO ITEM-IDENT.
 82       MOVE QUANTITY TO QTY, QUANT.
 83       MOVE COST-PER-UNIT TO UNIT-COST.
 84       MULTIPLY UNIT-COST BY QUANT GIVING ITEM-AMT.
 85       ADD ITEM-AMT TO ACCUM.
 86       MOVE ITEM-AMT TO AMOUNT.
 87       WRITE PRINTLINE FROM OUTPUT-LINE-1.
 88       IF END-FLAG EQUALS "E"
 89           GO TO INVOICE-TOTAL.
 90       GO TO READ-CARD.
 91   INVOICE-TOTAL.
 92       MOVE ACCUM TO INV-TOTAL.
 93       MULTIPLY ACCUM BY .1 GIVING DISCT ROUNDED.
 94       MOVE DISCT TO DISCOUNT.
 95       SUBTRACT DISCT FROM ACCUM.
 96       MULTIPLY ACCUM BY .06 GIVING TAX ROUNDED.
 97       MOVE TAX TO SALES-TAX.
 98       ADD TAX TO ACCUM.
 99       MOVE ACCUM TO AMT-DUE.
100       ADD ACCUM TO TEMP.
101       ADD 1 TO CNTR.
102       WRITE PRINTLINE FROM OUTPUT-LINE-2
103           AFTER ADVANCING 2 LINES.
104       MOVE ZEROS TO ACCUM.
105       MOVE SPACES TO PRINTLINE.
106       WRITE PRINTLINE
107           AFTER ADVANCING 3 LINES.
108       GO TO READ-CARD.
109   CLOSING.
110       DIVIDE CNTR INTO TEMP GIVING AVERAGE ROUNDED.
111       DISPLAY "AVERAGE INVOICE AMOUNT IS " AVERAGE.
112       CLOSE CARD-IN, OUTFILE.
113       STOP RUN.
```

Figure 4-2. Sample Arithmetic Program (Sheet 2 of 2)

```
Column 1   Column 12   Column 20   Column 28                    Column 80
175256     465        005         1095
175256     103        020         0495
175256     916        002         2495                         E
175257     696        012         0895                         E
175258     309        100         1475
175258     682        050         2250
175258     916        010         2495
175258     277        125         0725                         E
180696     103        030         0495
180696     456        045         1650
180696     916        005         2495                         E
180697     465        023         1095
180697     599        040         3200                         E
180698     196        020         2495
180698     696        050         0895
180698     456        050         1650                         E
```

Figure 4-3. Input Data for Sample Arithmetic Program

```
INVOICE     ITEM     QTY        COST          DISCOUNT       SALES TAX      AMOUNT DUE

 175256      465       5     $    54.75
 175256      103      20     $    99.00
 175256      916       2     $    49.90
                            $   203.65   -  $   20.37   +  $   11.00   =  $   194.28

 175257      696      12     $   107.40
                            $   107.40   -  $   10.74   +  $    5.80   =  $   102.46

 175258      309     100     $1,475.00
 175258      682      50     $1,125.00
 175258      916      10     $  249.50
 175258      277     125     $  906.25
                            $ 3,755.75   -  $  375.58   +  $  202.81   =  $ 3,582.98

 180696      103      30     $  148.50
 180696      456      45     $  742.50
 180696      916       5     $  124.75
                            $ 1,015.75   -  $  101.58   +  $   54.85   =  $   969.02

 180697      465      23     $  251.85
 180697      599      40     $1,280.00
                            $ 1,531.85   -  $  153.19   +  $   82.72   =  $ 1,461.38

 180698      196      20     $  499.00
 180698      696      50     $  447.50
 180698      456      50     $  825.00
                            $ 1,771.50   -  $  177.15   +  $   95.66   =  $ 1,690.01

AVERAGE INVOICE AMOUNT IS $ 1,333.36
```

Figure 4-4. Output Report from Sample Arithmetic Program

Each input record contains the quantity of the invoice item and the cost of one item. The total cost for the specified quantity is computed (line 84) and an output line is generated for the invoice item (line 87).

The last record for an invoice is identified by the letter E in the last character position of the input record (line 88). When all records for an invoice have been processed, the totals for the invoice are computed (lines 93 through 98) and an output line is generated.

As each invoice is completed, a running total of invoice amounts is maintained (line 100). When all invoices have been processed, the average amount for all invoices is computed (line 110) and displayed on the output report.

BOOLEAN EXPRESSIONS

A boolean expression consists of boolean variables, boolean literals (consisting of the characters 0 or 1), and boolean operators that are used to separate pairs of boolean variables and literals. Boolean expressions can be used in COMPUTE statements to define a boolean variable and its value. Boolean expressions can be used in relational conditions to compare boolean variables and/or literals for equality or inequality.

A simple boolean expression can contain a single boolean variable or literal, or two boolean variables or literals separated by a boolean operator. A complex boolean expression contains two or more simple boolean expressions separated by boolean operators. Parentheses can enclose boolean expressions to specify the order of evaluation or to clarify the logic of the expression.

BOOLEAN OPERATORS

Two types of boolean operators can be used in boolean expressions: binary and unary. Three binary boolean operators are available:

Operator	Function
BOOLEAN-AND	Boolean conjunction
BOOLEAN-OR	Boolean inclusive OR
BOOLEAN-EXOR	Boolean exclusive OR

A boolean expression can also be preceded by the following unary boolean operator:

Operator	Function
BOOLEAN-NOT	Boolean negation

A boolean operator must be preceded and followed by a space. With one exception, a boolean operator must be followed by a boolean variable or literal; the one exception is that a binary operator can be followed by a unary operator. When parentheses are used, the variable or literal can be preceded by a left parentheses.

EVALUATION OF EXPRESSIONS

The order of evaluation for boolean expressions is determined by the boolean operators:

Operator	Order of evaluation
BOOLEAN-NOT	First
BOOLEAN-AND	Second
BOOLEAN-OR and BOOLEAN-EXOR	Third

Expressions at the same level are evaluated from left to right.

Parentheses can be used to modify the normal sequence of evaluation. Expressions enclosed in parentheses are evaluated first, beginning with the innermost pair and proceeding to the outermost pair. Within a pair of parentheses, evaluation occurs according to the hierarchical order of evaluation.

The following boolean expression:

```
B"001010110"
```

is a boolean literal that produces the 9-character boolean value 001010110.

The boolean expression:

```
CODE-VALUE1 BOOLEAN-AND CODE-VALUE2
```

produces the boolean conjunction (logical multiplication) of CODE-VALUE1 and CODE-VALUE2. For instance, if CODE-VALUE1 is 10110 and CODE-VALUE2 is 01101:

CODE-VALUE1	10110
CODE-VALUE2	01101
Result of conjunction	00100

The boolean expression:

```
MISC-ITEM BOOLEAN-OR(BOOLEAN-NOT
    STANDARD-ITEM)
```

produces the boolean disjunction (logical addition) of MISC-ITEM and the complement of STANDARD-ITEM. For instance, if MISC-ITEM has a value of 01101 and STANDARD-ITEM has a value of 11010:

MISC-ITEM	01101
Complement of STANDARD-ITEM	00101
Result of disjunction	01101

Boolean expressions can be used in Procedure Division statements. The COMPUTE statement can be used to define a boolean variable.

```
COMPUTE NEW = MISC-ITEM BOOLEAN-OR
    (BOOLEAN-NOT STANDARD-ITEM)
```

This statement defines the boolean variable NEW and sets its value to 01101 (using values from the previous example).

The IF statement can be used to compare two boolean variables or literals for equality.

```
IF M-CODE BOOLEAN-AND B-VAL = B"10010"
    GO TO PROC-A ELSE GO TO PROC-B.
```

In this statement, the boolean variable M-CODE is compared with the boolean variable B-VAL. If the result of the conjunction is 10010, then the branch to PROC-A is taken; otherwise, the branch to PROC-B is taken.

SAMPLE BOOLEAN PROGRAM

The use of boolean statements is illustrated in figure 4-5. VAL1 and CONST are defined in the Working-Storage Section as boolean variables. The IF statement performs a logical AND operation on VAL1 and CONST. The COMPUTE statement performs the same operation and places the result in VAL1 before displaying it.

A. **Program Listing**

```
IDENTIFICATION DIVISION.
PROGRAM-ID.          BOOL.
DATA DIVISION.
WORKING-STORAGE SECTION.
01 STRS.
    02 VAL1            PIC 1(10) VALUE B"0101011101".
    02 CONST           PIC 1(10) VALUE B"1100110101".
PROCEDURE DIVISION.
STRT.
    IF VAL1 BOOLEAN-AND CONST = B"0100010101"
                       DISPLAY "GOOD"
                ELSE
                       DISPLAY "BAD".
    COMPUTE VAL1 = CONST BOOLEAN-AND VAL1
    DISPLAY "RESULT=" VAL1
    STOP RUN.
```

B. **Program Output**

```
GOOD
RESULT=0100010101
```

Figure 4-5. Boolean Example

Conditional operations are used in a COBOL 5 program to select an alternate path of control. A conditional expression is tested for its truth value, which is either true or false. The path of control to be executed depends on the truth value of the condition. Conditional expressions can be either simple or complex and are specified in the IF, PERFORM, and SEARCH statements.

Implicit conditional operations can be designated through five different options that can be included in various Procedure Division statements. These options specify imperative statements that are executed when the implied conditions exist. The five conditional options that can be specified are AT END, AT END-OF-PAGE, INVALID KEY, ON OVERFLOW, and ON SIZE ERROR.

CONDITIONAL EXPRESSIONS

A conditional expression specifies the condition that is tested to determine the next statement to be executed. The condition can be either a simple condition or a complex condition.

In an IF or SEARCH statement, the conditional expression is followed by an imperative statement that is executed when the condition is true. In a PERFORM statement, the procedure is performed repeatedly until the specified condition is true; control is then passed to the statement following the PERFORM statement.

SIMPLE CONDITIONS

A simple condition specifies one condition that is to be tested for its truth value. Five different types of conditions can be tested: relational, class, condition-name, switch-status, and sign conditions.

Relational Conditions

A relational condition causes two numeric or boolean operands to be compared. If the specified relationship exists, the condition is true. A relational condition not involving boolean expressions specifies that the first operand is one of the following:

- Greater than or not greater than the second operand.
- Less than or not less than the second operand.
- Equal to or not equal to the second operand.
- Exceeds the second operand.

A relational condition involving boolean expressions specifies that the first operand is equal to or is not equal to the second operand.

The operands in a relational condition can be data items described in the Data Division, literals, boolean, or arithmetic expressions. When an arithmetic expression is specified, the result of evaluating the arithmetic expression is used for the comparison.

Comparing Numeric Operands

A numeric comparison is based on the algebraic values of the numeric operands. The operands can be different lengths; decimal point alignment is performed automatically. Unsigned numeric operands are considered to be positive values. Numeric comparisons can be made between operands that are described with different usages. For example, a display item can be compared with a computational item.

```
GROSS-PAY IS GREATER THAN 150.99
```

The data item GROSS-PAY is compared with the numeric literal 150.99. If the value of GROSS-PAY is greater than 150.99, the condition is true; if GROSS-PAY is equal to or less than 150.99, the condition is not true.

The keyword NOT can be included in the relational condition to test for a negative condition.

```
AGE IS NOT LESS THAN 21
```

In this example, the condition is true when the value of the data item AGE is equal to or greater than 21. If the value of AGE is less than 21, the condition is not true.

Zero is considered a unique value regardless of the sign of the operand. Positive zero values are equal to negative zero values.

Operands described as COMPUTATIONAL-2 items typically do not have exact values. Because comparisons compare exact values, a range test should be used for a COMPUTATIONAL-2 operand.

Table 5-1 lists some numeric operand values and relational operators that result in a true condition.

Comparing Nonnumeric Operands

A nonnumeric comparison is performed when one or both operands are nonnumeric items. This type of comparison is based on a specified collating sequence. The collating sequence to be used can be selected by:

- Specifying the program collating sequence in the Environment Division.
- Changing the program collating sequence with a SET statement that is executed before the relational condition is tested.

Execution of a SET statement takes precedence over a collating sequence previously specified. If a collating sequence is not explicitly specified, the default collating sequence is used for the comparison.

A numeric operand that is compared with a nonnumeric operand must be an integer data item or literal or an arithmetic expression containing only integer operands. Both operands must be display items. If the nonnumeric operand is an elementary item, the numeric operand is treated as if it were moved to an alphanumeric item of the

same size as the nonnumeric item; any sign associated with the numeric operand is ignored. A nonnumeric group item causes the numeric operand to be treated as if it were moved to a group item of the same size as the nonnumeric item.

Operands of equal length are compared character by character as specified by the relational operator. Comparison begins with the leftmost character of each operand and continues until a pair of unequal characters is encountered or the last pair of characters has been compared. If no unequal pair is detected, the operands are equal. Unequal characters are evaluated according to their relative positions in the collating sequence. The operand that contains the higher character in the collating sequence is considered the greater operand.

When the operands are not the same length, the shorter operand is considered to be extended on the right with spaces up to the length of the longer operand. The comparison then proceeds as described for operands of equal length.

Table 5-2 lists some nonnumeric comparisons that result in a true condition. The comparisons are based on the CDC 64-character collating sequence; at least one operand in each comparison is described as a nonnumeric item.

The keyword NOT can also be included in a nonnumeric comparison to test for a negative condition.

Comparing Boolean Operands

Boolean operands of equal size are compared character for character, starting from the leftmost character of each operand, until either an unequal pair is encountered or the last pair of characters has been compared. If no unequal pair is detected, the operands are equal. If the operands are of unequal length, the shorter operand is treated as though it were extended on the right with boolean character zeros to make the operands of equal length.

Table 5-3 lists some boolean comparisons that result in a true condition.

TABLE 5-1. TRUE NUMERIC RELATIONAL CONDITIONS

Value of Operand-1	Relational Operator	Value of Operand-2
45610	IS LESS THAN or IS <	50000
6500	IS GREATER THAN or IS > or EXCEEDS	9450
00395	IS EQUAL TO or IS = or EQUALS	395
21	IS NOT LESS THAN or IS NOT <	18
456	IS NOT GREATER THAN or IS NOT > or IS NOT EQUAL TO	789

TABLE 5-2. TRUE NONNUMERIC RELATIONAL CONDITIONS

Value of Operand-1	Relational Operator	Value of Operand-2
ABC	IS EQUAL TO or IS = or EQUALS	ABC
FISHER	IS GREATER THAN or IS > or EXCEEDS	FISCHER
579.00	IS LESS THAN or IS <	57900
EF8900	IS NOT LESS THAN or IS NOT <	EF7800
JOHN	IS NOT GREATER THAN or IS NOT > or IS NOT EQUAL TO	JOHNSON

TABLE 5-3. TRUE BOOLEAN RELATIONAL CONDITIONS

Value of Operand-1	Relational Operator	Value of Operand-2
0 1 1 0 0 1 1	IS EQUAL TO or IS = or EQUALS	0 1 1 0 0 1 1
0 1 1 0 1	IS EQUAL TO or IS = or EQUALS	0 1 1 0 1 0 0 0
0 1 1 0 1	IS NOT EQUAL TO or IS NOT = or IS UNEQUAL TO	0 1 1 0 1 0 1 0

Class Conditions

A class condition tests a data item to determine whether the value of the data item is either numeric or alphabetic. The keyword NOT in the expression tests for a value that is not numeric or not alphabetic.

A numeric or alphanumeric data item can be tested for a numeric value. For a true condition, the value consists of the digits 0 through 9 and optionally can contain an operational sign.

An operational sign is indicated in the data description by the character S in the picture-specification. If the SIGN IS SEPARATE clause is also included in the description, the operational sign is the plus or the minus character (+ or -). Without the SIGN IS SEPARATE clause, the sign is combined with the first or last digit of the data item.

```
PART-NO IS NUMERIC
```

The data item PART-NO must have a numeric value for the class condition to be true. If the data description of PART-NO does not specify an operational sign, the value cannot contain an operational sign; however, the operational sign can be present if the SIGN clause is specified in the data description.

When the data item is described as COMPUTATIONAL-1, the numeric test is true only if the leftmost 12 bits of the computer word containing the data item are either all ones or all zeros. For a data item described as COMPUTATIONAL-2 or COMPUTATIONAL-4, the numeric test is always true.

An alphabetic or alphanumeric data item can be tested for an alphabetic value. The data item being tested can contain only the characters A through Z and the space for a true condition.

```
EMP-ID IS ALPHABETIC
```

The data item EMP-ID is checked for an alphabetic value. If the value contains any character other than A through Z and the space, the class condition is not true.

The keyword NOT is included in the class condition expression to test for a value that is not numeric or not alphabetic. The condition is true when the value does not satisfy the specified class condition.

```
EMP-ID IS NOT ALPHABETIC
```

If the value of EMP-ID contains any character other than A through Z and the space, the condition is true.

Condition-Name Conditions

A conditional expression can specify a condition-name. This causes the value of a conditional variable to be tested. A conditional variable is a data item that is established in the Data Division; condition-names are assigned to the values that can be associated with the data item. A condition-name can be assigned one or more individual values or ranges of values.

When a condition-name is specified as a conditional expression, the value of the conditional variable is tested to determine whether or not it is equal to one of the values assigned to the condition-name. The condition is true if the value of the conditional variable is the same as a value associated with the specified condition-name.

Figure 5-1 illustrates the use of a condition-name condition in an IF statement. The conditional variable PAY-PERIOD is associated with three condition-names; each condition-name is assigned one numeric value. When the IF statement is executed, PAY-PERIOD is tested for the value 2; if the value of PAY-PERIOD is 2, control is transferred to the paragraph named BI-WEEKLY-PAY.

```
      .
      .
      .
DATA DIVISION.
      .
      .
      .
    03 PAY-PERIOD PICTURE 9.
       88 WEEKLY VALUE IS 1.
       88 BI-WEEKLY VALUE IS 2.
       88 MONTHLY VALUE IS 3.
      .
      .
      .
PROCEDURE DIVISION.
      .
      .
      .
IF BI-WEEKLY GO TO BI-WEEKLY-PAY.
      .
      .
      .
```

Figure 5-1. Using a Condition-Name Condition

Switch-Status Conditions

The on or off status of a switch can be tested through a switch-status condition. The condition-name specified as a switch-status condition is established in the Environment Division. Six external switches (SWITCH-1 through SWITCH-6) and 120 internal switches (SWITCH-7 through SWITCH-126) can be associated with condition-names; both the on status and the off status can be assigned condition-names. A condition-name should be associated with only one condition because condition-names cannot be qualified in the Procedure Division.

When a switch-status condition-name is specified as a conditional expression, the switch is tested for the on or off status associated with the condition-name. If the switch setting is the same as specified for the condition-name, the switch-status condition is true.

Figure 5-2 illustrates the use of a switch-status condition. The condition-name TAPE-OUTPUT is assigned to the on status of SWITCH-1. When the IF statement is executed, SWITCH-1 is tested to determine whether or not it has been set to on. If SWITCH-1 is on, control is transferred to the paragraph named WRITE-TAPE.

```
      .
      .
      .
ENVIRONMENT DIVISION.
      .
      .
      .
SPECIAL-NAMES.
     SWITCH-1 ON STATUS IS TAPE-OUTPUT.
      .
      .
      .
PROCEDURE DIVISION.
      .
      .
      .
     IF TAPE-OUTPUT GO TO WRITE-TAPE.
      .
      .
      .
```

Figure 5-2. Using a Switch-Status Condition

The status of an external switch can be set by the SWITCH control statement before the program is executed. This statement can also be used to set an external switch when the program is executing from a terminal and when a STOP literal statement is specified to cause a program pause. The status of an external switch or an internal switch can be set during program execution by a SET statement. Figure 5-3 illustrates switch setting during program execution. If the SET statement is executed before the switch-status condition is tested, the condition is true and control is transferred to the paragraph named WRITE-RPT.

Sign Conditions

The sign condition tests the value of a data item or an arithmetic expression to determine if it is a positive value, a negative value, or a zero value. If an arithmetic expression is specified, it must contain at least one reference to a variable item. The value zero, whether signed or not, is considered to be neither positive nor negative.

(ON-HAND - ORDERED) IS POSITIVE

```
      .
      .
      .
ENVIRONMENT DIVISION.
      .
      .
      .
SPECIAL-NAMES.
     SWITCH-4 IS PRINT-CHECK
         ON STATUS IS PRINT-OUT.
      .
      .
      .
PROCEDURE DIVISION.
      .
      .
      .
     SET PRINT-CHECK TO ON.
      .
      .
      .
     IF PRINT-OUT GO TO WRITE-RPT.
      .
      .
      .
```

Figure 5-3. Setting a Switch

The arithmetic expression is evaluated and the result is then tested for a positive value. The sign condition is true if the value is greater than zero; if the value is zero or less than zero, the condition is not true.

The keyword NOT can be included in a sign condition expression. The value is tested to determine if it is not positive, not negative, or not zero.

(ON-HAND - ORDERED) IS NOT NEGATIVE

In this example, the condition is true if the result of the arithmetic expression is not a negative value. It can be a positive value or a zero value for a true condition.

COMPLEX CONDITIONS

A complex condition tests more than one condition for a truth value. The conditions are connected by logical operators. The function of each logical operator is as follows:

AND — Both simple conditions must be true for the complex condition to be true. If either condition is not true, the complex condition is not true.

OR — At least one of the simple conditions must be true; both conditions can be true. If neither condition is true, the complex condition is not true.

NOT — The truth value is reversed (negated). If the expression is true, the complex condition is not true.

Parentheses can be used to designate the order of evaluation for complex conditional expressions. Expressions within parentheses are tested for a truth value first; the complete expression is then evaluated for a truth value. When parentheses are nested, evaluation begins with the innermost pair of parentheses and proceeds to the

outermost pair. If parentheses are not used, evaluation begins at the left; all expressions connected with AND are evaluated first and then all expressions connected with OR are evaluated.

```
AGE>20 OR MARITAL = "M" AND
    CLASS>50
```

In this example, the complex conditional expression is true under either of the following conditions:

- The data item MARITAL is equal to M and the data item CLASS is greater than 50.
- The data item AGE is greater than 20.

If at least one of these conditions is true, the complex condition is true; if neither one is true, the complex condition is false.

```
(AGE>20 OR MARITAL = "M") AND
    CLASS>50
```

This example is the same as the previous example except that parentheses have been added. The parentheses cause the complex condition to be evaluated in a different manner. The complex condition is true under either of the following conditions:

- The data item AGE is greater than 20 and the data item CLASS is greater than 50.
- The data item MARITAL is equal to M and the data item CLASS is greater than 50.

The complex condition yields a true condition when at least one of these conditions is true; the complex condition is not true when neither condition is true.

The reverse of the condition just described can be specified by including the logical operator NOT before the complex expression.

```
NOT ((AGE>20 OR MARITAL = "M") AND
    CLASS>50)
```

The inclusion of the logical operator NOT specifies that one of the following conditions must exist for the complex condition to be true:

- The data item CLASS is less than or equal to 50.
- The data item AGE is less than or equal to 20 and the data item MARITAL is not equal to M.

A complex conditional expression contains a series of elements. The following rules apply to the order in which the elements can be specified:

- A simple condition can be the first or last element in the expression.
- A simple condition can be followed by the logical operator OR or AND or by a right parenthesis; it can be preceded by a logical operator (OR, NOT, or AND) or a left parenthesis.
- The logical operator OR or AND cannot be the first or the last element in the expression. It can be preceded by a simple condition or by a right parenthesis and followed by a simple condition, NOT, or a left parenthesis.
- The logical operator NOT can be the first but not the last element in the expression. It can be preceded by OR, AND, or a left parenthesis and followed by a simple condition or a left parenthesis.
- A left parenthesis can be the first but not the last element in the expression. It can be preceded by OR, NOT, AND, or another left parenthesis and can be followed by a simple condition, NOT, or another left parenthesis.
- A right parenthesis can be the last, but not the first, element in the expression. It can be preceded by a simple condition or another right parenthesis and can be followed by OR, AND, or another right parenthesis.

Implied Elements

A complex conditional expression can contain a series of two or more relational conditions. In certain instances, some elements of the relational conditions can be omitted from the expression. These omitted elements become implied elements.

When two or more consecutive relational conditions have the same operand preceding the relational operator, the operand can be omitted in the succeeding relational conditions. The operand must be specified in the first condition.

```
AGE>20 AND<66
```

Two relational conditions are specified in this example. The operand AGE is implied in the second condition. If the value of AGE is greater than 20 and it is less than 66, the complex condition is true.

When the first operand and the relational operator are the same in consecutive relational conditions, these two elements can be omitted following the first occurrence of the elements.

```
HRS-WORKED = 40 OR TEMP
```

The operand HRS-WORKED and the relational operator = are implied in the relational condition following the logical operator OR. If the value of HRS-WORKED is equal to either the numeric literal 40 or the value of TEMP, the complex condition is true.

The keyword NOT can be a part of the relational operator or it can be a logical operator. The usage of NOT is determined as follows:

- When NOT is immediately followed by GREATER, >, LESS, < , EQUAL, or =, it is part of the relational operator.
- In all other cases NOT is a logical operator.

As a relational operator, NOT can be an implied element; however, NOT cannot be an implied element as a logical operator.

Logical operators cannot be implied elements. Parentheses can be used to apply the logical operator to more than one condition.

```
NOT (TEMP1>TEMP2 AND ACCUM)
```

This example illustrates the use of parentheses and also includes a relational condition that has two implied elements. The two relational conditions are evaluated first; the logical operator NOT is then applied to the truth value of the two conditions. If both conditions are true, the complex condition is not true; if either or both conditions are not true, the complex condition is true. This example in expanded form is:

```
NOT TEMP1>TEMP2 AND NOT TEMP1>ACCUM
```

Order of Evaluation

A complex conditional expression contains a series of simple conditions that are evaluated individually and collectively to determine a final truth value for the complex condition. The hierarchical order of evaluation for a complex condition is as follows:

1. Arithmetic expressions within simple conditions.
2. Simple conditions in the following order:

   Relational

   Class

   Condition-name

   Switch-status

   Sign

3. Conditions connected by logical operators in the following order:

   AND

   OR

   NOT

Conditions at the same level in the order of evaluation are evaluated from left to right.

Parentheses can be used to change the order of evaluation. Conditions within parentheses are evaluated first; when parentheses are nested, evaluation begins with the innermost pair of parentheses and proceeds to the outermost pair. After the conditions enclosed by parentheses have been evaluated, the final truth value is determined according to the hierarchical order of evaluation.

It is good programming practice to avoid abbreviations and to use parentheses whenever both the logical operators AND and OR are included in the conditional expression. This ensures that the expression is evaluated in the desired order.

CONDITIONAL STATEMENTS

A conditional statement specifies that the next operation to be performed depends on the truth value of a specific condition. The condition is explicitly specified in some statements as a conditional expression. In other statements, the condition is implied by the use of certain phrases.

Within the same sentence, a conditional statement can be preceded by an imperative statement. The imperative statement is executed regardless of the truth value of the conditional statement.

The inclusion of an explicit scope terminator makes a conditional statement into an imperative statement. This section includes discussion of END-IF, END-PERFORM, and END-SEARCH terminators even though the associated IF, PERFORM, and SEARCH statements are not considered conditional in this context.

EXPLICIT CONDITIONAL STATEMENTS

An explicit conditional statement depends on the evaluation of a specified conditional expression. The truth of the condition determines what statement is executed next. Three different statements in the Procedure Division can specify an explicit condition: the IF statement, the PERFORM statement, and the SEARCH statement.

IF Statement Without END-IF

An IF statement specifies a conditional expression that is evaluated to determine whether or not the next statement is executed. The statement following the conditional expression is executed if the condition is true; it is bypassed if the condition is not true.

```
IF MARITAL = "M" ADD 1 TO MARRIED.
```

The conditional expression MARITAL = "M" is evaluated for a truth value. If the data item MARITAL contains the letter M, the condition is true and 1 is added to the data item MARRIED. If the condition is not true, the ADD statement is not executed. Control is then passed to the next sentence.

The IF statement can also specify a statement to be executed when the condition is not true. The keyword ELSE precedes the statement that is executed for a false condition.

```
IF MARITAL = "M" ADD 1 TO MARRIED
    ELSE ADD 1 TO SINGLE.
```

The evaluation of the conditional expression determines which data item (MARRIED or SINGLE) is incremented. If the condition is true, 1 is added to MARRIED; if the condition is not true, 1 is added to SINGLE. Control is then passed to the next sentence.

The phrase NEXT SENTENCE can be substituted for the statement following the conditional expression or the statement following ELSE. This phrase causes control to be transferred to the next executable sentence.

```
IF PART-NO IS NUMERIC NEXT SENTENCE
    ELSE GO TO BAD-NUMBER.
```

In this example, a true condition causes control to be transferred to the next sentence. If the condition is false, control is transferred to the paragraph named BAD-NUMBER.

The statements following the conditional expression can be imperative or conditional statements; either statement can be followed by a conditional statement. IF statements are

considered to be nested when either statement following the conditional expression contains another IF statement. When IF statements are nested, each ELSE phrase is paired with the immediately preceding IF statement that is not already paired with another ELSE phrase.

```
IF INV-NO = TEMP ADD COST TO ACCUM
    IF FLAG = 1 PERFORM DISC-ITEM
    ELSE NEXT SENTENCE
ELSE GO TO NEW-INV.
```

The conditional expression in the first IF statement is evaluated. If the condition is true, the value of COST is added to the data item ACCUM and the data item FLAG is checked for the value 1. If this second condition is also true, the procedure DISC-ITEM is performed and control returns to the next executable sentence. If FLAG does not contain the value 1, control is immediately passed to the next sentence. When the first conditional expression is evaluated and the condition is false, control is immediately transferred to the paragraph named NEW-INV.

IF Statement With END-IF

The inclusion of an END-IF terminator with an IF statement makes the IF statement an imperative statement. This structure can eliminate the repetition of a condition that identifies a major category, as shown in figure 5-4.

```
      .
      .
      .
PROCEDURE DIVISION.
      .
      .
      .
      IF CATEGORY = "ANIMALS"
            IF SPECIES = "CAT"
                  IF HABITAT = "PET"
                        PERFORM CAT-PET-ROUTINE
                  END-IF
                  IF HABITAT = "ZOO"
                        PERFORM CAT-ZOO-ROUTINE
                  END-IF
               ELSE
                  PERFORM NOT-CAT-ROUTINE
            END-IF
            IF SPECIES = "FISH"
                  IF HABITAT = "PET"
                        PERFORM FISH-PET-ROUTINE
                  END-IF
                  IF HABITAT = "ZOO"
                        PERFORM FISH-ZOO-ROUTINE
                  END-IF
               ELSE
                  PERFORM NOT-ANIMAL-ROUTINE
      END-IF
      .
      .
      .
```

Figure 5-4. IF Statement with END-IF Example 1

The IF statement with the END-IF terminator is allowed in all places that allow imperative statements, such as following the AT END phrase in a READ statement. When the end of INFILE is reached in figure 5-5, control passes to ROUTINE-2, ROUTINE-3, or ERROR-ROUTINE, depending on the specific condition.

```
        READ INFILE
            AT END
                IF CONDITION = 2
                   PERFORM ROUTINE-2
                ELSE
                IF CONDITION = 3
                   PERFORM ROUTINE-3
                ELSE
                   PERFORM ERROR-ROUTINE
                END-IF
                CLOSE INFILE OUTFILE
                STOP RUN.
```

Figure 5-5. IF Statement with END-IF Example 2

PERFORM Statement Without END-PERFORM

The PERFORM statement causes a procedure or a range of procedures to be performed. It is a conditional statement when the procedure is performed repeatedly until one or more specified conditions are true.

If WITH TEST BEFORE is specified or if the WITH TEST phrase is omitted, the conditional expression is evaluated before control is transferred to the procedure. If the condition is not true, the procedure is performed and the condition is then tested again. Each time that the condition is not true, the procedure is performed. When the condition is true, control is transferred to the next statement after the PERFORM statement.

```
PERFORM ROUTINE1 WITH TEST BEFORE
    UNTIL TEMP = CNTR.
```

The conditional expression is evaluated and if it is true, the procedure is not performed and control is immediately passed to the next sentence. If the condition is not true, the procedure ROUTINE1 is performed and the condition is tested again. This process continues until the condition is true; control is then transferred to the next sentence.

If the WITH TEST AFTER phrase is specified, the conditional expression is evaluated after the last statement in the procedure is executed. The statements are executed at least once, regardless of the initial conditions.

A data item or an index can be varied while performing a procedure until a specified condition is true. The PERFORM statement specifies the initial value of the data item or index and the value by which it is incremented or decremented each time the procedure is performed.

```
PERFORM RATE-CALC THRU RC-EXIT
    VARYING QUANT FROM 50 BY 5
    UNTIL QUANT GREATER THAN TEMP.
```

Before the range of procedures is executed, the data item QUANT is set to 50 and the condition is evaluated. If the condition is true, the procedures are not performed and control is passed to the next executable statement. If the condition is not true, the procedures are executed, QUANT is incremented by 5, and the condition is evaluated again. This cycle is repeated until the value of QUANT is greater than the value of TEMP; control then passes to the next statement following the PERFORM statement.

Up to three data items or indexes can be varied during execution of the PERFORM statement. Each data item or index is varied until a specified condition is true. At the beginning of the PERFORM statement processing, each item to be varied is set to its specified initial value. If the first condition is true at this point, the procedure is not performed and control is immediately transferred to the next statement. The procedure is performed varying the last data item or index until its associated condition is true. This loop is repeated each time the preceding condition is not true. When all conditions are true, processing of the PERFORM statement is complete and control is passed to the next statement.

Execution of a PERFORM statement with more than one variable item is best described through an example. Figure 5-6 illustrates a PERFORM statement with three variable items. When this PERFORM statement is entered, the three indexes are each set to the initial value of 1. The procedure MPY-ROUTINE is executed as follows:

```
PERFORM MPY-ROUTINE
    VARYING I-INDEX FROM 1 BY 1
        UNTIL I-INDEX > CNTR
    AFTER J-INDEX FROM 1 BY 1
        UNTIL J-INDEX > CNTR
    AFTER K-INDEX FROM 1 BY 1
        UNTIL K-INDEX > CNTR.
```

Figure 5-6. Varying Indexes in a PERFORM Statement

1. MPY-ROUTINE is executed repeatedly until the value of K-INDEX is greater than the value of CNTR; K-INDEX is incremented by 1 each time the procedure is executed.

2. K-INDEX is reset to its initial value of 1; J-INDEX is incremented by 1 and tested for a value greater than CNTR.

3. If J-INDEX is not greater than CNTR, steps 1 and 2 are repeated; if it is greater than CNTR, J-INDEX is reset to its initial value of 1 and I-INDEX is incremented by 1 and tested for a value greater than CNTR.

4. If I-INDEX is not greater than CNTR, steps 1, 2, and 3 are repeated; if it is greater than CNTR, control is passed to the next statement following the PERFORM statement.

PERFORM Statement With END-PERFORM

The inclusion of an END-PERFORM terminator with a PERFORM statement makes the PERFORM statement an imperative statement and provides the COBOL user with a capability similar to a FORTRAN do-loop. This structure does not have a procedure-name following the PERFORM verb. Instead, the in-line code is performed until the END-PERFORM terminator is reached. Figure 5-7 illustrates this capability and the structure of the PERFORM statement with an END-PERFORM terminator.

A. Program Listing

```
1   IDENTIFICATION DIVISION.
2   PROGRAM-ID.  PERF.
3   DATA DIVISION.
4   WORKING-STORAGE SECTION.
5   01 STOR1 PIC 99.
6   PROCEDURE DIVISION.
7   SRTR.
8       MOVE ZEROS TO STOR1
9       PERFORM 2 TIMES
10          DISPLAY "1"
11          DISPLAY "2"
12          PERFORM VARYING STOR1
13                  FROM 1 BY 1
14                  UNTIL STOR1 = 5
15              DISPLAY "STOR1=" STOR1
16          END-PERFORM
17          DISPLAY "A"
18          PERFORM WITH TEST AFTER
19                  VARYING STOR1
20                  FROM 1 BY 1
21                  UNTIL STOR1 = 5
22              DISPLAY "STOR1=" STOR1
23          END-PERFORM
24          DISPLAY "B"
25      END-PERFORM
26      DISPLAY "Z"
27      STOP RUN.
```

B. Program Output

```
1
2
STOR1=  1
STOR1=  2
STOR1=  3
STOR1=  4
A
STOR1=  1
STOR1=  2
STOR1=  3
STOR1=  4
STOR1=  5
B
1
2
STOR1=  1
STOR1=  2
STOR1=  3
STOR1=  4
A
STOR1=  1
STOR1=  2
STOR1=  3
STOR1=  4
STO51=  5
B
Z
```

Figure 5-7. PERFORM Statement with END-PERFORM

In the example in figure 5-7, two PERFORM statements are nested within a higher level PERFORM. The PERFORM at line 9 has within it, at lines 12 and 18, two PERFORMS that are independent of each other. The PERFORM at line 18 also demonstrates the WITH TEST AFTER phrase; the data-item STOR1 is tested for a value of 5 after the DISPLAY statment has executed. The omission of the WITH TEST phrase (or inclusion of a WITH TEST BEFORE phrase) would cause STOR1 to be evaluated before the DISPLAY statement execution.

SEARCH Statement Without END-SEARCH

The SEARCH statement is used to search a table for a specific element within the table. The element is designated by specifying a condition; when the condition is true, the desired element has been located. The use of the SEARCH statement is described in detail in section 6, Table Handling.

The condition that must be satisfied to terminate a search operation depends on the type of search performed:

- A sequential search is terminated when any one of the specified conditions is true.
- A binary search is terminated when all the specified conditions are true.

The table to be searched must be described in the Data Division with an OCCURS clause that includes the INDEXED BY phrase. The index-name specified in this phrase is used to search the table. At the end of a successful search, the index-name points to the table element that satisfies the search criteria.

A sequential search begins at the current setting of the index-name and continues to the end of the table. Each table element is tested for the specified conditions. Any valid relational condition can be specified. Multiple conditions are tested in the order specified. When a table element satisfies a condition, the imperative statement associated with that condition is executed. If the end of the table is reached, control is passed to the imperative statement of the AT END phrase (if specified) or to the next executable statement.

A binary search is specified by the SEARCH ALL format of the SEARCH statement. One or more conditions can be specified; all conditions must be true for the search operation to terminate successfully. Each specified condition can be a condition-name condition or an equal condition. If no element in the table satisfies the conditions, control is passed to the imperative statement of the AT END phrase (if specified) or to the next executable statement.

The sample program at the end of this section illustrates the use of the SEARCH statement to perform a sequential search operation. Additional examples of the SEARCH statement can be found in section 6.

SEARCH Statement With END-SEARCH

The inclusion of an END-SEARCH terminator with a SEARCH statement makes the SEARCH statement an imperative statement. This structure can eliminate the repetition of searching for an item at a major level, as shown in figure 5-8.

In the example in figure 5-8, the value 13 is found in the table TBL by finding the first digit in one SEARCH and the second digit in another SEARCH. The table values above and below 13 are also displayed.

A. Program Listing

```
IDENTIFICATION DIVISION.
PROGRAM-ID.  ENDSEARCH.
DATA DIVISION.
WORKING-STORAGE SECTION.
01 TBL.
    02 LEVEL-1  OCCURS 3 INDEXED BY L1INDX.
        03 LEVEL-2  OCCURS 5 INDEXED BY L2INDX.
            13 L21  PIC 9.
            13 L22  PIC 9.
PROCEDURE DIVISION.
STRT.
    MOVE "00010203041011121314202122232 4" TO TBL
    SET L1INDX, L2INDX TO 1
    SEARCH LEVEL-1
        AT END DISPLAY "1ST DIGIT NOT FOUND"
        WHEN L21 (L1INDX, L2INDX) = 1
        DISPLAY "1ST DIGIT FOUND-" L21 (L1INDX, L2INDX)
        SEARCH LEVEL-2
            AT END DISPLAY "2ND DIGIT NOT FOUND"
            WHEN L22 (L1INDX, L2INDX) = 3
            DISPLAY "FOUND 2ND DIGIT-" L22 (L1INDX, L2INDX)
        END-SEARCH
        SET L2INDX DOWN BY 1
        DISPLAY "NUMBER BELOW IS-" LEVEL-2 (L1INDX, L2INDX)
        SET L2INDX UP BY 2
        DISPLAY "NUMBER ABOVE IS-" LEVEL-2 (L1INDX, L2INDX)
    END-SEARCH
    DISPLAY "ALL DONE"
    STOP RUN.
```

B. Program Output

```
1ST DIGIT FOUND- 1
FOUND 2ND DIGIT- 3
NUMBER BELOW IS-12
NUMBER ABOVE IS-14
ALL DONE
```

Figure 5-8. SEARCH Statement with END-SEARCH

IMPLICIT CONDITIONAL STATEMENTS

Implicit conditions are indicated in certain optional phrases that can be included in specific Procedure Division statements. When one of these phrases is included, it is followed by an imperative statement that is executed when the condition implied by the phrase is true.

Five phrases imply a condition: AT END, AT END-OF-PAGE, INVALID KEY, ON OVERFLOW, and ON SIZE ERROR. Each phrase can be used with specific statements.

At End Condition

The AT END phrase can be used in conjunction with the READ, RETURN, and SEARCH statements. The condition implied by this phrase is true when the end of a file or table has been reached. When a true condition occurs, control is passed to the imperative statement associated with the AT END phrase.

Execution of a READ statement causes a record to be made available to the program. The AT END phrase can be included in a READ statement that accesses records sequentially. If it is included, the associated imperative statement is executed when the end-of-file condition is detected while attempting to read a record.

```
READ CARD-IN RECORD
    AT END GO TO CLOSING.
```

Execution of this statement causes the next sequential record in the input file CARD-IN to be read. If the end-of-file condition is true when the READ statement is executed, control is transferred to the paragraph named CLOSING.

The AT END phrase is required in the RETURN statement. This statement accesses a record in a sort file or a merge file and makes the record available for processing. The imperative statement associated with the AT END phrase is executed when the end-of-file condition is true.

```
RETURN SORT-FILE RECORD
    AT END GO TO ENDIT.
```

Each time this statement is executed, a record is retrieved from the file SORT-FILE until an end-of-file condition exists. When this condition is true, control is transferred to the paragraph named ENDIT.

The SEARCH statement can include the AT END phrase. If it is included, the imperative statement associated with the AT END phrase is executed when no element in the table satisfies the search criteria.

```
SEARCH PART-ITEM AT END GO TO NOT-FOUND
    WHEN PART-ITEM (PI-INDEX) = PART-NO
    GO TO FOUND.
```

When this statement is executed, the table PART-ITEM is searched for an element that satisfies the specified condition. If the end of the table is reached and the condition has not been satisfied, control is transferred to the paragraph named NOT-FOUND.

End-of-Page Condition

The AT END-OF-PAGE (or AT EOP) phrase can be used only with a WRITE statement that prepares an output file for printing; the FD entry for the file must include the LINAGE clause. This phrase is usually specified to control the printing of headings and footings. The AT END-OF-PAGE phrase includes an imperative statement that is executed when the end of a page is reached. The limits of the page must be defined by the LINAGE clause.

An end-of-page condition exists when execution of the WRITE statement causes printing or spacing in the footing area (if specified) or beyond the page limit. The imperative statement associated with the AT END-OF-PAGE phrase is executed before page positioning if printing occurs in the footing area or after page positioning if printing extends beyond the page limit.

```
WRITE PRINT-LINE
    AT END-OF-PAGE GO TO PRINT-HEADS.
```

When the end-of-page condition occurs, control is transferred to the paragraph named PRINT-HEADS. Assuming that the LINAGE clause for the print file does not specify a footing area, the GO TO statement is executed after the file is positioned at the next page.

```
WRITE PRINT-LINE
    AT END-OF-PAGE GO TO LAST-LINE.
```

In this example, the end-of-page condition causes control to be transferred to the paragraph named LAST-LINE. Assuming that the LINAGE clause for the print file specifies a footing area, the GO TO statement is executed before the file is positioned at the next page.

Invalid Key Condition

An invalid key condition can occur when a DELETE, READ, REWRITE, START, or WRITE statement is being executed for a file that is accessed by a key data item. The INVALID KEY phrase implies that the attempted operation is illegal. In most cases, the invalid key condition occurs when the record cannot be found in the file. For a WRITE statement, an invalid key condition occurs when the record already exists in the file.

The INVALID KEY phrase specifies an imperative statement that is executed when the condition is true; the statement containing the INVALID KEY phrase is not executed. An invalid key condition can exist under the following circumstances:

DELETE statement - any file organization except sequential or word-address; random or dynamic access mode.

The record to be deleted cannot be found.

READ statement - any file organization except sequential; random access mode or dynamic access mode when records are accessed randomly.

The record to be read cannot be found; for word-address file organization, an attempt is made to read past the last word of the file.

REWRITE statement - any file organization except sequential or word-address; random or dynamic access mode.

The record to be rewritten cannot be found.

REWRITE statement - indexed, direct, or actual-key file organization; any access mode.

The primary key has been changed between reading and rewriting in sequential access mode or the record to be rewritten would create a duplicate alternate key when duplicate alternate keys are not allowed.

START statement - any file organization except sequential or word-address; sequential or dynamic access mode.

No record in the file satisfies the specified relational condition.

WRITE statement - any file organization except sequential or word-address; any access mode.

The record to be written already exists in the file with the same primary key.

WRITE statement - indexed, direct, or actual-key file organization; any access mode.

The record to be written already exists with the same alternate key when duplicate alternate keys are not allowed.

WRITE statement - indexed file organization; sequential access.

The record to be written does not have a primary key that is greater than the primary key of the previous record written.

WRITE statement - actual-key organization; random or dynamic access mode.

The key of the record to be written is not a valid actual key.

Overflow Condition

The CALL, STRING, and UNSTRING statements include the optional ON OVERFLOW phrase. When an overflow condition occurs during execution of the statement, the imperative statement associated with the ON OVERFLOW phrase is executed.

An overflow condition exists for a STRING or UNSTRING statement when the receiving item or items cannot contain the complete sending item. If the ON OVERFLOW phrase is not specified, an overflow condition causes control to be passed to the next executable statement.

For a CALL statement, an overflow condition exists when there is insufficient room to load a dynamic subprogram. This occurs when the maximum field length would be exceeded by loading the subprogram. When the ON OVERFLOW phrase is not specified and an overflow condition occurs, the run is aborted.

Size Error Condition

The ON SIZE ERROR phrase can be included in any of the arithmetic statements: ADD, SUBTRACT, MULTIPLY, DIVIDE, and COMPUTE. A size error condition occurs when the number of integral digits in the result of an arithmetic operation exceeds the number of integral positions in the receiving item. Size error testing is performed on intermediate and final results of all arithmetic operations. When a size error exists, the imperative statement associated with the ON SIZE ERROR phrase is executed. The ON SIZE ERROR phrase is described in more detail in section 4, Arithmetic Operations.

SAMPLE CONDITIONAL PROGRAM

Various types of conditional operations are used in the sample program shown in figure 5-9. Both implicit and explicit conditions are included in the program. Figure 5-10 illustrates the format of the input records used by the program. An output report generated by the program is shown in figure 5-11.

The input file used by this program contains four different types of records. The first group of input records (CUSTOMER-REC records) is used to enter data into the table CUST-TABLE. The second group of input records is used to enter data into the table ITEM-TABLE. The third group of input records contains two types of records (NAME-REC and LINE-REC records) that are used to process an order for a customer.

Two PERFORM statements are utilized to access the input data and to enter it into the two tables. The first PERFORM statement (lines 84, 85, and 86) causes data to be stored in the table CUST-TABLE (lines 91 through 100). This statement is executed repeatedly until the data item DISC-FLAG contains the letter E (line 86). When this condition is true, the next PERFORM statement is executed. The second PERFORM statement (lines 87, 88, and 89) causes data to be stored in the table ITEM-TABLE (lines 101 through 109). This statement is executed repeatedly until the data item ITEM-ID equals zero (line 89). When this condition is true, control is passed to the procedure ORDER-PROCESSING (line 90).

A customer order consists of at least two input records. The first record for an order (NAME-REC record) is identified by the letter A in the REC-CODE field (lines 113 and 132). This record is followed by one record (LINE-REC record) for each item of the order; LINE-REC records are identified by the letter B in the REC-CODE field (line 130). When the first record is read for an order, the CUSTOMER-ID field is used to search the table CUST-TABLE for the name of the customer (lines 116 through 119). As each LINE-REC record is read, the table ITEM-TABLE is searched for the description of the item specified by the ITEM-NO field (lines 136 through 139).

As the item records are processed, a total for the order is accumulated (line 143). When all line items for an order have been read, a discount is computed if the discount conditions for the order have been satisfied (lines 150 through 153).

```
 1   IDENTIFICATION DIVISION.
 2   PROGRAM-ID.  CONDITIONAL-EXAMPLE.
 3   ENVIRONMENT DIVISION.
 4   CONFIGURATION SECTION.
 5   SOURCE-COMPUTER.  CYBER-170.
 6   OBJECT-COMPUTER.  CYBER-170.
 7   INPUT-OUTPUT SECTION.
 8   FILE-CONTROL.
 9       SELECT IN-FILE ASSIGN TO INPUT.
10       SELECT PRINT-FILE    ASSIGN TO OUTPUT.
11   DATA DIVISION.
12   FILE SECTION.
13   FD  IN-FILE
14           LABEL RECORD IS OMITTED
15           DATA RECORDS ARE CUSTOMER-REC, ITEMS-REC,
16               NAME-REC, LINE-REC.
17   01  CUSTOMER-REC.
18       03  CUST-ID              PICTURE XXX.
19       03  CUST-NAME            PICTURE X(20).
20       03  FILLER               PICTURE X(56).
21       03  DISC-FLAG            PICTURE X.
22   01  ITEMS-REC.
23       03  ITEM-ID              PICTURE 999.
24       03  ITEM-DESC            PICTURE X(20).
25       03  FILLER               PICTURE X(57).
26   01  NAME-REC.
27       03  REC-CODE             PICTURE X.
28       03  FILLER               PICTURE XXX.
29       03  INV-NO               PICTURE 9(6).
30       03  FILLER               PICTURE XXXX.
31       03  CUSTOMER-ID          PICTURE XXX.
32       03  FILLER               PICTURE X(63).
33   01  LINE-REC.
34       03  REC-CODE             PICTURE X.
35       03  FILLER               PICTURE XXX.
36       03  ITEM-NO              PICTURE 999.
37       03  FILLER               PICTURE X(7).
38       03  QUANTITY             PICTURE 999.
39       03  FILLER               PICTURE X(7).
40       03  COST                 PICTURE 9(5)V99.
41       03  FILLER               PICTURE X(49).
42   FD  PRINT-FILE
43           LABEL RECORD IS OMITTED
44           DATA RECORD IS PRINTLINE.
45   01  PRINTLINE                PICTURE X(136).
46   WORKING-STORAGE SECTION.
47   01  SAVE                     PICTURE X.
48   01  ACCUM                    PICTURE 9(7)V99 USAGE COMP-1.
49   01  DISCOUNT                 PICTURE 9(6)V99 USAGE COMP.
50   01  CUST-TABLE.
51       03  CUSTOMER OCCURS 50 TIMES
52               INDEXED BY C-INDEX.
53           05  C-IDENT          PICTURE XXX.
54           05  C-NAME           PICTURE X(20).
55           05  C-FLAG           PICTURE X.
56   01  ITEM-TABLE.
57       03  ITEM OCCURS 100 TIMES
58               INDEXED BY I-INDEX.
59           05  I-IDENT          PICTURE 999.
60           05  I-DESC           PICTURE X(20).
```

Figure 5-9. Sample Conditional Program (Sheet 1 of 3)

```
01  LINE-1.
    03  FILLER              PICTURE X(5)    VALUE SPACES.
    03  CUST-NAME           PICTURE X(20).
    03  FILLER              PICTURE X(5)    VALUE SPACES.
    03  INVOICE             PICTURE 9(6).
    03  FILLER              PICTURE X(100)  VALUE SPACES.
01  LINE-2.
    03  FILLER              PICTURE X(10)   VALUE SPACES.
    03  QTY                 PICTURE ZZ9.
    03  FILLER              PICTURE X(5)    VALUE SPACES.
    03  DESCRIPTION         PICTURE X(20).
    03  FILLER              PICTURE X(5)    VALUE SPACES.
    03  PRICE               PICTURE $$$$9.99.
    03  FILLER              PICTURE X(85)   VALUE SPACES.
01  LINE-3.
    03  FILLER              PICTURE X(42)   VALUE SPACES.
    03  AMOUNT              PICTURE $$$$$9.99.
    03  FILLER              PICTURE X(85)   VALUE SPACES.
PROCEDURE DIVISION.
SETTING-UP.
    OPEN INPUT IN-FILE.
    OPEN OUTPUT PRINT-FILE.
    MOVE SPACES TO CUST-TABLE, ITEM-TABLE.
    PERFORM TABLE-SETUP-1 THRU TSET-1A
        VARYING C-INDEX FROM 1 BY 1
        UNTIL DISC-FLAG EQUALS "E".
    PERFORM TABLE-SETUP-2 THRU TSET-2A
        VARYING I-INDEX FROM 1 BY 1
        UNTIL ITEM-ID EQUALS ZERO.
    GO TO ORDER-PROCESSING.
TABLE-SETUP-1.
    READ IN-FILE RECORD
        AT END GO TO TSET-1A.
    IF DISC-FLAG = "E"
        GO TO TSET-1A.
    MOVE CUST-ID TO C-IDENT (C-INDEX).
    MOVE CUST-NAME OF CUSTOMER-REC TO C-NAME (C-INDEX).
    MOVE DISC-FLAG TO C-FLAG (C-INDEX).
TSET-1A.
    EXIT.
TABLE-SETUP-2.
    READ IN-FILE RECORD
        AT END GO TO TSET-2A.
    IF ITEM-ID EQUALS ZERO
        GO TO TSET-2A.
    MOVE ITEM-ID TO I-IDENT (I-INDEX).
    MOVE ITEM-DESC TO I-DESC (I-INDEX).
TSET-2A.
    EXIT.
ORDER-PROCESSING.
    READ IN-FILE RECORD
        AT END GO TO ERROR-1.
    IF REC-CODE OF NAME-REC NOT EQUAL TO "A"
        GO TO ERROR-2.
HEADER-ITEM.
    SET C-INDEX TO 1.
    SEARCH CUSTOMER AT END GO TO ID-NOT-FOUND
        WHEN CUSTOMER-ID EQUALS C-IDENT (C-INDEX)
        NEXT SENTENCE.
    MOVE C-FLAG (C-INDEX) TO SAVE.
    MOVE C-NAME (C-INDEX) TO CUST-NAME OF LINE-1.
```

Figure 5-9. Sample Conditional Program (Sheet 2 of 3)

```
122   HI-1.
123       MOVE INV-NO TO INVOICE.
124       WRITE PRINTLINE FROM LINE-1
125           AFTER ADVANCING 3 LINES.
126       MOVE ZEROS TO ACCUM.
127   READ-RECORD.
128       READ IN-FILE RECORD
129           AT END GO TO CLOSING.
130       IF REC-CODE OF LINE-REC EQUALS "B"
131           GO TO LINE-ITEM.
132       IF REC-CODE OF NAME-REC EQUALS "A" PERFORM TOTALS
133           ELSE GO TO ERROR-2.
134       GO TO HEADER-ITEM.
135   LINE-ITEM.
136       SET I-INDEX TO 1.
137       SEARCH ITEM AT END GO TO NOT-FOUND
138           WHEN I-IDENT (I-INDEX) EQUALS ITEM-NO
139           MOVE I-DESC (I-INDEX) TO DESCRIPTION.
140   LI-1.
141       MOVE QUANTITY TO QTY.
142       MOVE COST TO PRICE.
143       ADD COST TO ACCUM ON SIZE ERROR PERFORM ERROR-3.
144       WRITE PRINTLINE FROM LINE-2.
145       GO TO READ-RECORD.
146   TOTALS.
147       MOVE ACCUM TO AMOUNT.
148       WRITE PRINTLINE FROM LINE-3
149           AFTER ADVANCING 2 LINES.
150       IF ACCUM GREATER THAN 100.00 AND SAVE EQUALS "D"
151           NEXT SENTENCE
152           ELSE GO TO HEADER-ITEM.
153       COMPUTE DISCOUNT ROUNDED = ACCUM * .10.
154       MOVE DISCOUNT TO AMOUNT.
155       WRITE PRINTLINE FROM LINE-3.
156       SUBTRACT DISCOUNT FROM ACCUM GIVING AMOUNT.
157       WRITE PRINTLINE FROM LINE-3
158           AFTER ADVANCING 2 LINES.
159   ERROR-1.
160       DISPLAY "NO INPUT RECORDS".
161       STOP RUN.
162   ERROR-2.
163       DISPLAY "ILLEGAL REC-CODE - " REC-CODE OF NAME-REC.
164       GO TO READ-RECORD.
165   ID-NOT-FOUND.
166       MOVE CUSTOMER-ID TO CUST-NAME OF LINE-1.
167       GO TO HI-1.
168   NOT-FOUND.
169       MOVE ITEM-NO TO DESCRIPTION.
170       GO TO LI-1.
171   ERROR-3.
172       DISPLAY "ACCUMULATOR OVERFLOW".
173   CLOSING.
174       PERFORM TOTALS.
175       CLOSE IN-FILE, PRINT-FILE.
176       STOP RUN.
```

Figure 5-9. Sample Conditional Program (Sheet 3 of 3)

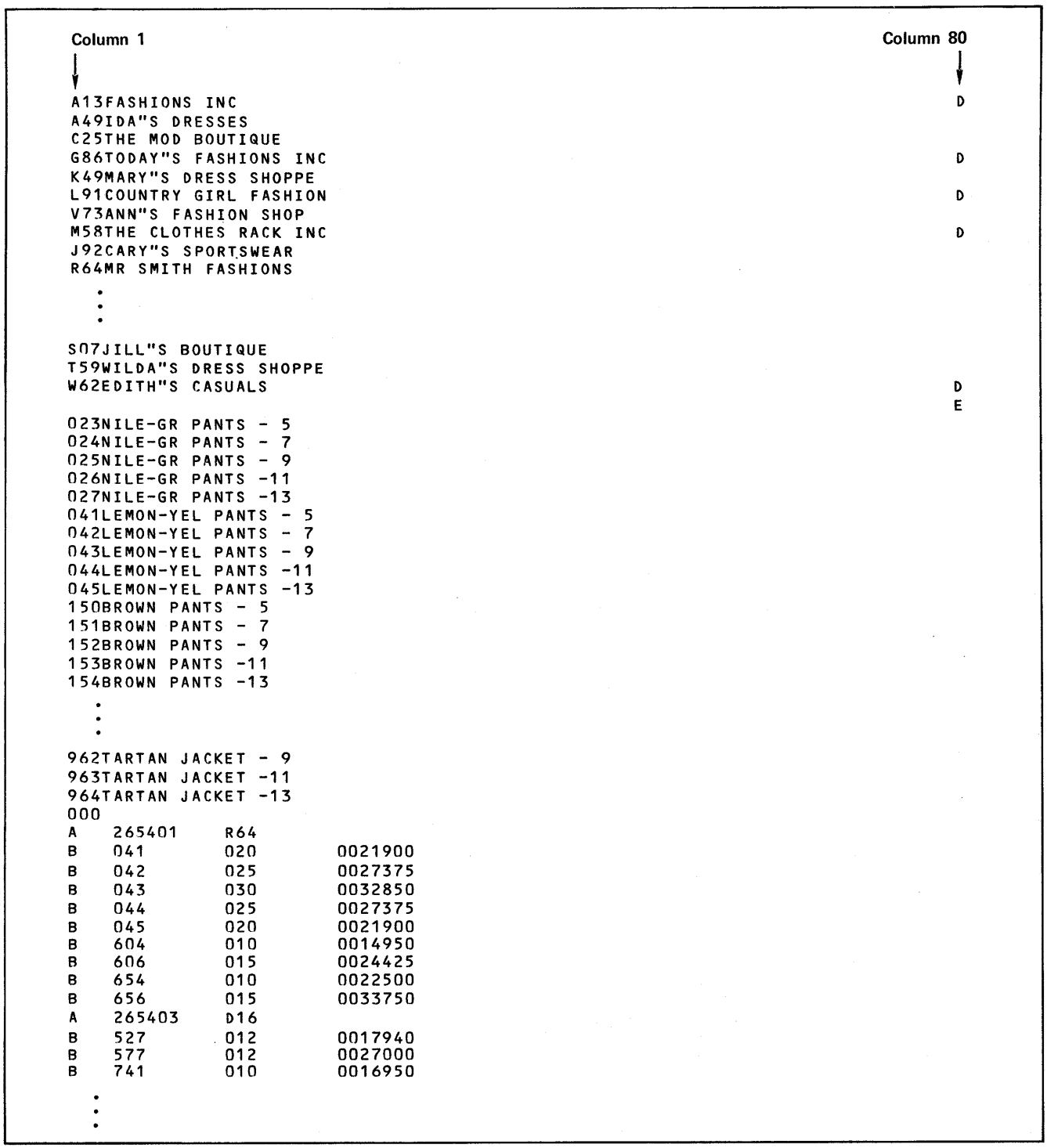


Figure 5-10. Sample Input for Conditional Program

```
MR SMITH FASHIONS          265401
      20      041                        $219.00
      25      042                        $273.75
      30      043                        $328.50
      25      044                        $273.75
      20      045                        $219.00
      10      604                        $149.50
      15      606                        $244.25
      10      654                        $225.00
      15      656                        $337.50

                                         $2270.25


HIS-N-HERS FASHIONS        265403
      12      527                        $179.40
      12      577                        $270.00
      10      741                        $169.50
      10      781                        $285.00
       6      024                         $65.70
      10      026                        $109.50
       8      044                         $87.60
      10      152                        $109.50
       6      154                         $65.70

                                         $1341.90
                                          $134.19

                                         $1207.71


GUYS AND GALS              265405
      15      195                        $207.00
      15      196                        $207.00
      15      201                        $315.00
      15      202                        $315.00
      10      152                        $109.50
      10      236                        $225.00
      10      410                        $152.50
      15      411                        $228.75
      17      412                        $259.25
      12      413                        $183.00
      10      490                        $265.00
      15      491                        $397.50
      17      492                        $450.50
      12      493                        $318.00

                                         $3633.00


THE CLOTHES RACK INC       265406
      30      363                        $457.50
      20      364                        $305.00
      30      388                        $795.00
      20      389                        $530.00
      25      816                        $387.50
      40      817                        $620.00
      35      818                        $542.50
      25      876                        $643.75
      40      877                       $1030.00
      35      878                        $901.25

                                         $6212.50
                                          $621.25

                                         $5591.25
```

Figure 5-11. Output Report from Sample Conditional Program

Data used by a COBOL 5 program is frequently organized into a table as a group of constant values. Three methods are available for describing a table and referencing specific elements within a table:

- Each table element is named, described, and assigned a value; elements are referenced by data-name.
- A table described by the above method is redefined and described with the OCCURS clause; individual elements are referenced by subscripting or indexing.
- During program execution, data can be moved into a table described with the OCCURS clause; individual elements are referenced by subscripting or indexing.

Table elements are referenced by indexing when the INDEXED BY option is included in the OCCURS clause. If the INDEXED BY option is not included, table elements must be referenced by subscripting. Indexing and subscripting can be mixed within a table. An index, which is identified by the index-name specified in the OCCURS clause, contains a value that points to a specific table element. A subscript is either an integer or a data item that contains an integer value; it locates a specific table element.

Indexing is more efficient than subscripting and should be used whenever possible. When subscripting is used, subscripts should be COMPUTATIONAL-1 items for the most efficient results. Constant indexing or subscripting should always be specified by a literal.

TABLE DEFINITION

Tables used during program execution are described in the Data Division. The values for the table elements can be assigned in the table description, or the values can be entered in the table during program execution. Table definition can involve the VALUE, REDEFINES, and OCCURS clauses as well as the PICTURE clause.

ASSIGNING INDIVIDUAL DATA-NAMES

A data-name can be assigned to each element in the table. This allows direct reference to a specific table element. Values are stored in the table by specifying the VALUE clause for each table element.

Figure 6-1 illustrates a table that contains the number of male students and the number of female students for each of the 12 grades. The 24 numbers in the table are organized by grade level with the male number preceding the female number.

During compilation, the processor associates the 24 numbers with particular areas in memory. At execution time, all of the numbers are stored in memory and are available for reference by the object program. The table elements are referenced by data-name; however, a specific male or female number must be qualified when it is referenced.

```
ADD MALE OF GRADE-2, MALE OF GRADE-3
    TO COUNTER.
```

```
     .
     .
     .
DATA DIVISION.
     .
     .
     .
01   GRADE-COUNT.
     03   GRADE-1.
          05   MALE    PICTURE 999 VALUE 215.
          05   FEMALE  PICTURE 999 VALUE 257.
     03   GRADE-2.
          05   MALE    PICTURE 999 VALUE 245.
          05   FEMALE  PICTURE 999 VALUE 289.
     03   GRADE-3.
          05   MALE    PICTURE 999 VALUE 198.
          05   FEMALE  PICTURE 999 VALUE 232.
     .
     .
     .
     03   GRADE-12.
          05   MALE    PICTURE 999 VALUE 202.
          05   FEMALE  PICTURE 999 VALUE 248.
     .
     .
     .
```

Figure 6-1. Table Definition by Data-Names

This statement causes the values 245 and 198 to be added to the data item COUNTER. The values stored in the table GRADE-COUNT are constant values that remain the same each time the program is executed.

REDEFINING A TABLE

Elements within a table can be referenced by position in the table rather than by data-name. This is accomplished by describing a table that assigns values and then redefining the table as a series of repeating data items. Procedure Division statements then reference the table data by specifying the redefined data-name along with a subscript or index-name. The REDEFINES and OCCURS clauses are required to redefine a table.

The table shown in figure 6-1 is redefined in figure 6-2. The level 01 entry in figure 6-2 must immediately follow the last level 05 entry for GRADE-12 in figure 6-1. The redefined table is given the data-name GRADE-TABLE. The table elements are organized into 12 sets named GRADE; each set contains two values named SEX-COUNT. Subsequent references to the redefined table can specify GRADE-TABLE (the entire table), GRADE (one set of two values, identified by one subscript), or SEX-COUNT (one value in one set, identified by two subscripts).

```
MOVE SEX-COUNT (12, 1) TO TEMP.
```

This statement causes the first value (number of males) in the twelfth set (GRADE-12) to be moved to a data item named TEMP. If the subscript had been (12, 2), the female number of the GRADE-12 set would have been moved.

```
      .
      .
      .
DATA DIVISION.
      .
      .
      .
01   GRADE-TABLE REDEFINES GRADE-COUNT.
     03   GRADE OCCURS 12 TIMES.
          05   SEX-COUNT PICTURE 999
                    OCCURS 2 TIMES.
      .
      .
      .
```

Figure 6-2. Table Redefinition

A data-name rather than an integer can be used as a subscript. This allows one statement to reference different table elements at different times during processing. Indexing can also be used for a redefined table. Subscripts and indexes are described in detail later in this section.

MOVING VALUES INTO A TABLE

A table that is described with the OCCURS clause cannot specify values to be stored in the table. When the entry containing the OCCURS clause is not subordinate to an entry that redefines a table with specified values, the values can be entered into the table during program execution.

When a table is described in the Data Division and values are not entered in the table, statements in the Procedure Division can supply the data and store it in the table. Figure 6-3 illustrates the description of a table for which values are supplied at execution time. The table contains 50 sets named CUSTOMER; each set contains one value for each of the three subordinate items. Reference to an element in this table must include the index-name C-INDEX unless it is a constant reference, in which case a literal can be specified. The sample program in section 5 (figure 5-5) reads an input file and stores the information in this table.

```
      .
      .
      .
DATA DIVISION.
      .
      .
      .
01   CUST-TABLE.
     03   CUSTOMER OCCURS 50 TIMES
               INDEXED BY C-INDEX.
          05   C-IDENT   PICTURE XXX.
          05   C-NAME    PICTURE X(20).
          05   C-FLAG    PICTURE X.
      .
      .
      .
```

Figure 6-3. Table Definition by the OCCURS Clause

The example in figure 6-4 illustrates how to use the PERFORM statement with an END-PERFORM terminator to store information in a table (STOCK-TABLE). Refer to section 5 for more detail on the usage of the PERFORM statement with END-PERFORM.

TABLE REFERENCE

Table elements are referenced in Procedure Division statements in one of three ways depending on the description of the table. The first type of reference is used when the table is not described with the OCCURS clause. Each element in the table can be identified by a unique data-name or a data-name that can be made unique by qualification. When a table is described with the OCCURS clause, either subscripting or indexing must be used to reference table elements.

UNIQUE REFERENCE

When a table is described without using the OCCURS clause, data-names are assigned to all table elements. A specific table element can then be referenced by its data-name. If the data-name is not unique, it must be made unique by qualification.

```
01 STOCK-TABLE.
   03 PART-ITEM
        OCCURS 10 TIMES
        INDEXED BY INDEX-B.
        05 PART PICTURE 999
             OCCURS 20 TIMES
             INDEXED BY INDX-P.
    .
    .
    .
    PERFORM VARYING INDEX-B FROM 1 BY 1
        UNTIL INDEX-B = 10
        PERFORM VARYING INDEX-P FROM 1 BY 1
            UNTIL INDEX-P = 20
          MOVE PART-LIST (INDEX-B, INDEX-P) TO PART (INDEX-B, INDEX-P)
        END-PERFORM
    END-PERFORM.
```

Figure 6-4. Using PERFORM/END-PERFORM to Fill a Table

Unique reference is used with the table shown in figure 6-1. The MALE or FEMALE element for any grade level must be qualified by the group data-name of the specific grade level.

```
FEMALE OF GRADE-1

MALE OF GRADE-9
```

The first example references the FEMALE element in the set of two values associated with the group item GRADE-1. The MALE element in the set of two values associated with GRADE-9 is referenced in the second example. Group data-names in this table can be referenced without qualification.

```
MOVE GRADE-3 TO GRADE-OUT.
```

This example references the third set of values in the table. The data-name GRADE-OUT references a group item consisting of two elementary data items to receive the MALE and FEMALE values in the GRADE-3 set.

SUBSCRIPTING

A table described by the OCCURS clause can be referenced by subscripts. A subscript is either an integer or the data-name of an elementary numeric data item containing an integer value. The integer points to a specific group or elementary item within the table.

Subscripts are enclosed in parentheses following the data-name of the table element. The lowest valid subscript number is one; the highest valid number corresponds to the maximum number of occurrences of the element as specified in the OCCURS clause. The system does not automatically check the subscript for a valid number; this can be performed by Procedure Division statements or by specifying DB=SB in the COBOL5 control statement.

The table description can contain up to 48 levels of nested OCCURS clauses. In this case, one subscript is required for the first entry and one for each subordinate entry containing the OCCURS clause. When more than one subscript is required, the subscripts are written in descending order of inclusiveness. The subscripts can, but need not, be separated by commas. A separating comma must be followed by a space. The table shown in figure 6-2 contains two levels of nesting; reference to the SEX-COUNT items requires two subscripts.

```
SEX-COUNT (4, 2)
```

The first subscript refers to the fourth occurrence of the group item GRADE and the second subscript refers to the second occurrence of SEX-COUNT within the fourth group item; that is, the subscripts point to the female population of the fourth grade.

When a data-name is used as a subscript, it must refer to a numeric elementary data item that represents an integer; the usage of the data item cannot be INDEX or COMPUTATIONAL-2. For most efficient usage, the data item should be COMPUTATIONAL-1; otherwise, it should be as small as possible. A subscript data-name can be qualified; however, it cannot be subscripted. When the table reference is executed, the current value of the subscript data item is used to calculate the table element desired.

Figure 6-5 illustrates the use of a data-name subscript. The data item COUNTER is described in the Data Division as an integer. In the Procedure Division, COUNTER is given an initial value of 1. The paragraph TABLE-LOOKUP is executed repeatedly until one of the two conditions is satisfied. Each time TABLE-LOOKUP is executed, the subscript COUNTER has been incremented by 1 and points to the next element in the table.

```
      .
      .
      .
DATA DIVISION.
      .
      .
      .
01   TEMP              PIC 9(5).
01   COUNTER           PIC 999 USAGE COMP-1.
01   PARTS-TABLE.
     03  PART-ITEM     PIC 9(5)
                OCCURS 100 TIMES.
      .
      .
      .
PROCEDURE DIVISION.
      .
      .
      .
     MOVE 1 TO COUNTER.
TABLE-LOOKUP.
     IF PART-ITEM (COUNTER) = TEMP
        GO TO PART-FOUND.
     IF COUNTER > 100 GO TO NOT-FOUND.
     ADD 1 TO COUNTER.
     GO TO TABLE-LOOKUP.
      .
      .
      .
```

Figure 6-5. Table Reference by Subscripting

INDEXING

When a table description includes the INDEXED BY option in the OCCURS clause, table elements are referenced by indexing. The INDEXED BY option specifies the index-name to be used for referencing an element within the table. The index-name cannot be described anywhere else in the program; the allocation and format of the index associated with the index-name are controlled by the compiler. The index is not data and cannot be part of a data hierarchy. Indexing is more efficient than subscripting and should be used if possible. Indexes and integers can be mixed in a table reference. Indexes and subscripts can also be mixed.

Up to 48 levels of nested OCCURS clauses can be specified in the table description. Each level is assigned an index-name in an INDEXED BY phrase. Table references include the index-names in descending order of inclusiveness. Index-names are enclosed in parentheses and can, but need not, be separated by commas. A separating comma must be followed by a space.

The value of the index at the time of execution corresponds to the occurrence number of an element in the associated table. An index must be initialized before it is used as a table reference; a SET, SEARCH, or PERFORM statement can be used to give an initial value to an index. An index value cannot be less than 1 or greater than the highest permissible occurrence number for the element.

Figure 6-6 illustrates a table description with two levels of nested OCCURS clauses. An index-name is specified for each level. The MOVE statement in the Procedure Division references a specific YEAR element in the table. The two index-names (S-INDEX and Y-INDEX) point to a location in the table corresponding to the current values of the respective indexes.

```
      .
      .
      .
DATA DIVISION.
      .
      .
      .
01   POPULATION TABLE.
     03   STATE OCCURS 50 TIMES
               INDEXED BY S-INDEX.
          05   YEAR PICTURE 9(10)
                    OCCURS 10 TIMES
                    INDEXED BY Y-INDEX.
      .
      .
      .
PROCEDURE DIVISION.
      .
      .
      .
     MOVE YEAR (S-INDEX, Y-INDEX)
          TO MALE-POP.
      .
      .
      .
```

Figure 6-6. Table Reference by Indexing

The type of indexing used in figure 6-6 is called direct indexing. The table element referenced is located by the absolute value of the index associated with the specified index-name. Relative indexing is specified when the index-name is followed by a plus sign or a minus sign and an integer. The table element referenced in this manner is located by using the value of the index and incrementing or decrementing that value by the specified integer; the actual value of the index is not changed.

The table shown in figure 6-6 contains 10 population numbers for each of 50 states. The 10 numbers represent the number of males followed by the number of females for each of five years. Assuming that execution of the MOVE statement in this figure references the first number for the first state, the male population number for the first year is moved to the data item MALE-POP. The female population number for the same year can then be referenced by relative indexing.

```
MOVE YEAR (S-INDEX, Y-INDEX + 1)
    TO FEMALE-POP.
```

The value associated with an index-name can only be used as a table reference. The value can be stored in an index data item when the index value is to be used as data. The index data item is described in the Data Division with the USAGE IS INDEX clause. A SET statement in the Procedure Division can store an index value in the index data item. An index data item can be used in SEARCH and SET statements and in relational conditions; it can also be specified in the USING phrases of the Procedure Division header and the CALL statement.

TABLE HANDLING STATEMENTS

Three COBOL 5 statements provide the means to effectively enter data in tables or to access data stored in tables. The PERFORM statement with the VARYING phrase can be used for a table that is either subscripted or indexed. The SEARCH statement is used only for a table that is indexed. An index can be set to a value by the SET statement.

Data can be entered into tables by using the PERFORM statement with the END-PERFORM terminator (see figure 6-4). A table can be searched for a specific value by using the SEARCH statement with the END-SEARCH terminator (see figure 5-8 in section 5).

PERFORM STATEMENT

An index or a data-name subscript can be automatically incremented or decremented by a specified value each time a PERFORM statement is executed. The index or subscript points to a different element in the table each time the procedure is performed.

The VARYING phrase of the PERFORM statement specifies the index-name or subscript data-name. This phrase also specifies the initial value for the index or subscript, the value by which it is incremented or decremented, and the condition that determines when execution of the PERFORM statement is complete. A negative value is specified when the index or subscript is to be decremented. The procedure is performed repeatedly until the specified condition is true.

```
PERFORM TABLE-SETUP-1 THRU TSET-1A
    VARYING C-INDEX FROM 1 BY 1
    UNTIL DISC-FLAG EQUALS "E".
```

The procedure to be performed is a series of paragraphs beginning with TABLE-SETUP-1 and ending with TSET-1A. The index associated with C-INDEX is initialized with the value 1. Each time the procedure is executed, C-INDEX is incremented by 1. When the data item DISC-FLAG contains the letter E, the procedure is not performed and control passes to the sentence following the PERFORM statement.

The PERFORM statement can specify up to 48 indexes or 48 data-name subscripts to be varied. The initial value, the increment or decrement value, and the condition to be satisfied are specified for each index or subscript. The manner in which three items are varied and the procedure (or range of procedures) is performed is discussed in section 5, Conditional Operations. The sample program in section 5 uses the PERFORM statement to enter data in a table by varying an index.

SEARCH STATEMENT

A table referenced by indexing can be searched for an element that satisfies one or more conditions. Two different types of search procedures can be specified by SEARCH statements. The format of the statement determines whether the search procedure is a sequential or binary search.

The description of the table to be searched must include the OCCURS clause with the INDEXED BY phrase. This phrase specifies the index-name that is used to search the table. After a successful search, the index-name points to a table element that satisfies the search criteria.

Sequential Search

A sequential search begins at the current setting of the index-name. If the current value of the index-name points to the first element in the table, the search starts at the beginning of the table; otherwise, the search begins at the indicated position within the table. If the current value of the index-name exceeds the upper limit defined for the table, control is passed to the imperative-statement of the AT END phrase (if specified) or to the next executable statement following the SEARCH statement.

The table element indicated by the index-name setting is tested for the specified condition. If the condition is true, the search is complete and the imperative statement associated with the condition is executed. If the condition is not true, the next element in the table is tested for the condition. This procedure continues until the condition is true or the end of the table is reached.

More than one condition can be specified for the search operation. A table element is then tested for each specified condition. When any one of the conditions is true, the search is complete and the imperative statement associated with the true condition is executed.

Figure 6-7 illustrates the SEARCH statement for a sequential search. Index-name P-INDEX is set to 1 so that the search will begin with the first element in the table. The condition PART-ITEM (P-INDEX) = PART-NO is tested for each element in the table until a true condition is encountered. When the condition is true, control is transferred to the paragraph named PART-FOUND. If the end of the table is reached before a true condition occurs, control passes to the paragraph named NOT-FOUND.

```
     .
     .
     .
DATA DIVISION.
     .
     .
     .
03   PART-NUMBERS.
         05   FILLER PICTURE 9(3) VALUE 075.
         05   FILLER PICTURE 9(3) VALUE 212.
         05   FILLER PICTURE 9(3) VALUE 153.
         .
         .
         .
         05   FILLER PICTURE 9(3) VALUE 010.
     03  PART-TABLE REDEFINES PART-NUMBERS.
         05   PART-ITEM PICTURE 9(3)
                  OCCURS 20 TIMES
                  INDEXED BY P-INDEX.
     .
     .
     .
PROCEDURE DIVISON.
     .
     .
     .
     SET P-INDEX TO 1.
     SEARCH PART-ITEM
         AT END GO TO NOT-FOUND
         WHEN PART-ITEM (P-INDEX) = PART-NO
         GO TO PART-FOUND.
     .
     .
     .
```

Figure 6-7. Table Searching, Sequential Search

Binary Search

A binary search is an efficient means of searching a large table for an element that satisfies one or more conditions. The table must be ordered in the sequence of the ASCENDING/DESCENDING KEY phrase in the OCCURS clause. If more than one condition is specified, all the conditions must be true for a successful search.

The SEARCH ALL format of the SEARCH statement designates a binary search operation. The OCCURS clause for the table must include the KEY IS phrase as well as the INDEXED BY phrase. One or more of the data-names specified in the KEY IS phrase are used to search the table. The data-name can be referenced by specifying a condition-name associated with the data-name or by specifying the data-name in a relational condition that tests for equality. The data-name must be indexed by the index-name in the INDEXED BY phrase; if more than one index-name is specified, the first index-name is used.

When a condition-name is specified, the description of the condition-name can designate only one value. The condition-name must be associated with a data-name in the KEY IS phrase of the table description.

The SEARCH ALL statement can specify a relational condition that uses the EQUALS relational operator or one of its equivalent forms. The operand preceding the relational operator must be one of the data-names in the KEY IS phrase of the table description. The operand following the relational operator can be a literal, an arithmetic expression, or a data item that is not referenced in the KEY IS phrase.

When the SEARCH ALL statement is executed, the table elements are tested for the specified condition. The search ends if the condition is true for a table element; when more than one condition is specified, all conditions must be true for a successful search. Control is then passed to the imperative statement associated with the specified conditions. If a table element that satisfies the conditions cannot be found, control is passed to the imperative statement of the AT END phrase (if specified) or to the next executable statement following the SEARCH ALL statement.

A SEARCH ALL statement with two conditions specified is illustrated in figure 6-8. The table DATA-LIST is searched for an element that satisfies both conditions; the values of ITEM1 and TEMP1 must be equal and the values of ITEM2 and TEMP2 must also be equal. ITEM1 and ITEM2 are subscripted by the index-name LIST-INDEX. When a table element that satisfies both conditions is found, control is transferred to the paragraph named LIST-ITEM. If no element in the table satisfies both conditions, control is passed to the next executable sentence after the SEARCH ALL statement.

SET STATEMENT

The SET statement can be used to initialize an index-name or to transfer the value of an index-name to another index-name, an index data item, or an integer data item. It can also be used to increment or decrement the value of an index-name.

The initial value given to an index-name can be specified as a numeric literal, a data-name, or another index-name. If a data-name is specified, it must refer to either an index data item or an elementary integer data item; the current value of the data item becomes the value of the

```
      .
      .
      .
DATA DIVISION.
      .
      .
      .
      02  DATA-LIST OCCURS 25 TIMES
                 INDEXED BY LIST-INDEX
                 DESCENDING KEY IS ITEM1   ITEM2.
          04   ITEM1   PICTURE XX.
          04   ITEMA   PICTURE 9(7).
          04   ITEM2   PICTURE 99.
      .
      .
      .
PROCEDURE DIVISION.
      .
      .
      .
      SEARCH ALL DATA-LIST
          WHEN ITEM1 (LIST-INDEX) = TEMP1
          AND ITEM2 (LIST-INDEX) = TEMP2
          GO TO LIST-ITEM.
      .
      .
      .
```

Figure 6-8. Table Searching, Binary Search

index-name. When a data-name or an index-name is specified as the initial value of an index-name, the storage areas of the sending and receiving items should not overlap; if the areas do overlap, unpredictable results might occur during execution.

SET INDX-P TO 1.

Initializes the index-name (INDX-P) with a value of 1.

SET INDX-D TO INDX-P.

Transfers the current value of one index-name (INDX-P) to another index-name (INDX-D).

SET INDX-C TO SAVE.

Initializes the index-name (INDX-C) with the current value of the data item (SAVE).

SET SAVE TO INDX-C.

Transfers the current value of the index-name (INDX-C) to the data item (SAVE).

The capability to store the value of an index-name in a data item (as shown in the last of the preceding examples) provides the means to retain the value for later reference. This feature is particularly advantageous for input/output because an index-name cannot be defined as part of a file but an index data item (or integer data item) can be defined as part of a file.

The value of an index-name can be incremented or decremented by the SET statement. A numeric literal or a data-name that references an elementary integer data item can be specified for the increment or decrement value. Incrementing is indicated by the keywords UP BY; decrementing is indicated by the keywords DOWN BY. The value of the index-name after execution of the SET statement must correspond to a valid occurrence number for the table. When the value of an index-name is incremented or decremented by a value contained in a data item, the storage areas of the items should not overlap; if the areas do overlap, unpredictable results might occur during execution.

Figure 6-9 illustrates the use of the SET statement in searching a table that has nested OCCURS clauses. The table is searched for the first population number greater than 1,000,000. The index-name values for the found item are saved for future reference. Before the search operation begins, the two index-names are initialized with a value of 1. The value of Y-IDX is incremented automatically during the search operation; the value of S-IDX does not change. If the ten YEAR elements for the current setting of S-IDX do not satisfy the condition, control is transferred to the paragraph named TRY-AGAIN. The value of S-IDX is incremented by 1, Y-IDX is set to a value of 1, and the next ten YEAR elements are searched. If S-IDX reaches a value greater than 50, the entire table has been searched unsuccessfully and control is passed to the paragraph named CANT-FIND.

```
      .
      .
      .
DATA DIVISION.
      .
      .
      .
01   SAVA   USAGE IS INDEX.
01   SAVB   USAGE IS INDEX.
      .
      .
      .
01   POPULATION-TABLE.
     03   STATE    OCCURS 50 TIMES
                   INDEXED BY S-IDX.
          05   YEAR   PICTURE 9(10)
                   OCCURS 10 TIMES
                   INDEXED BY Y-IDX.
      .
      .
      .
PROCEDURE DIVISION.
      .
      .
      .
     SET S-IDX, Y-IDX TO 1.
SRCH.
     SEARCH YEAR
          AT END GO TO TRY-AGAIN
          WHEN YEAR (S-IDX,Y-IDX) > 1000000
          SET SAVA TO S-IDX
          SET SAVB TO Y-IDX
          GO TO MILLION-PLUS.
TRY-AGAIN.
     SET S-IDX UP BY 1.
     SET Y-IDX TO 1.
     IF S-IDX > 50 GO TO CANT-FIND
          ELSE GO TO SRCH.
      .
      .
      .
```

Figure 6-9. Searching a Two-Dimensional Table

SAMPLE TABLE HANDLING PROGRAMS

Two sample programs are included in this section to show two different methods of table handling. The first program uses subscripting and the second program uses indexing and table searching. Both of these programs specify the table values in the Working-Storage Section of the Data Division. Refer to the sample program in section 5 (figure 5-8) for an example of entering values into a table during program execution and using the SEARCH statement to locate table elements.

TABLE-SUBSCRIPTING PROGRAM

The use of subscripts in table references is shown in the sample program illustrated in figure 6-10. This program reads a record (line 87), obtains the fare for the designated class from one table (line 92), obtains the city code for the destination from another table (line 94), and writes a line on the output report (line 95). The input data shown in figure 6-11 is used to create the output report shown in figure 6-12.

```
 1   IDENTIFICATION DIVISION.
 2   PROGRAM-ID.  TABLE-SUBSCRIPTING.
 3   ENVIRONMENT DIVISION.
 4   CONFIGURATION SECTION.
 5   SOURCE-COMPUTER.  CYBER-170.
 6   OBJECT-COMPUTER.  CYBER-170.
 7   INPUT-OUTPUT SECTION.
 8   FILE-CONTROL.
 9       SELECT CARDIN   ASSIGN TO INPUT.
10       SELECT PRINTOUT ASSIGN TO OUTPUT.
11   DATA DIVISION.
12   FILE SECTION.
13   FD  CARDIN
14           LABEL RECORDS ARE OMITTED
15           DATA RECORD IS CARD.
16   01  CARD.
17       03  NUMB                PICTURE IS 99.
18       03  NAME                PICTURE IS X(18).
19       03  CLSS                PICTURE IS 9.
20       03  DEST                PICTURE IS 9.
21       03  FILLER              PICTURE X(58).
22   FD  PRINTOUT
23           LABEL RECORDS ARE OMITTED
24           DATA RECORD IS PRINTLINE.
25   01  PRINTLINE               PICTURE IS X(136).
26   WORKING-STORAGE SECTION.
27   01  DISP.
28       03  FILLER              PICTURE IS X.
29       03  NUMBA               PICTURE IS 99.
30       03  FILLER              PICTURE IS X(10).
31       03  NAMEA               PICTURE IS X(18).
32       03  FILLER              PICTURE IS X(10).
33       03  CLASSA              PICTURE IS 9.
34       03  FILLER              PICTURE IS X(8).
35       03  DESTA               PICTURE IS XXX.
36       03  FILLER              PICTURE IS X(10).
37       03  FAREA               PICTURE IS $$$$.99.
38       03  FILLER              PICTURE IS X(66).
39   01  FARES.
40       03  FIRST-CLASS.
41           05  LAX             PICTURE IS 9(5)     VALUE 12750.
42           05  BUR             PICTURE IS 9(5)     VALUE 14500.
43           05  OAK             PICTURE IS 9(5)     VALUE 24000.
44           05  SFO             PICTURE IS 9(5)     VALUE 27250.
45           05  SEA             PICTURE IS 9(5)     VALUE 52500.
46       03  SECOND-CLASS.
47           05  LAX             PICTURE IS 9(5)     VALUE 10750.
48           05  BUR             PICTURE IS 9(5)     VALUE 12500.
49           05  OAK             PICTURE IS 9(5)     VALUE 22500.
50           05  SFO             PICTURE IS 9(5)     VALUE 25000.
51           05  SEA             PICTURE IS 9(5)     VALUE 50000.
52       03  TOURIST.
53           05  LAX             PICTURE IS 9(5)     VALUE  8750.
54           05  BUR             PICTURE IS 9(5)     VALUE 10500.
55           05  OAK             PICTURE IS 9(5)     VALUE 20000.
56           05  SFO             PICTURE IS 9(5)     VALUE 23000.
57           05  SEA             PICTURE IS 9(5)     VALUE 47500.
```

Figure 6-10. Sample Program Using Subscripts (Sheet 1 of 2)

```
 58   01  FARE-TABLE REDEFINES FARES.
 59       03  CLASS-CODE  OCCURS 3 TIMES.
 60           05  FARE    OCCURS 5 TIMES  PICTURE IS 999V99.
 61   01  CITY-TABLE.
 62       03  A                      PICTURE IS XXX      VALUE "LAX".
 63       03  B                      PICTURE IS XXX      VALUE "BUR".
 64       03  C                      PICTURE IS XXX      VALUE "OAK".
 65       03  D                      PICTURE IS XXX      VALUE "SFO".
 66       03  E                      PICTURE IS XXX      VALUE "SEA".
 67   01  CITY-ID REDEFINES CITY-TABLE.
 68       03  CITY OCCURS 5 TIMES PICTURE IS XXX.
 69   01  HEAD.
 70       03  FILLER                 PICTURE IS 9        VALUE 1.
 71       03  FILLER                 PICTURE IS XX       VALUE "ID".
 72       03  FILLER                 PICTURE IS X(17)    VALUE SPACES.
 73       03  FILLER                 PICTURE IS XXXX     VALUE "NAME".
 74       03  FILLER                 PICTURE IS X(15)    VALUE SPACES.
 75       03  FILLER                 PICTURE IS X(5)     VALUE "CLASS".
 76       03  FILLER                 PICTURE IS X(6)     VALUE SPACES.
 77       03  FILLER                 PICTURE IS XXXX     VALUE "CITY".
 78       03  FILLER                 PICTURE IS X(10)    VALUE SPACES.
 79       03  FILLER                 PICTURE IS XXXX     VALUE "FARE".
 80       03  FILLER                 PICTURE IS X(68)    VALUE SPACES.
 81   PROCEDURE DIVISION.
 82   START-UP.
 83       OPEN INPUT  CARDIN.
 84       OPEN OUTPUT PRINTOUT.
 85       PERFORM WRITE-HEAD.
 86   GET-FARE.
 87       READ CARDIN RECORD
 88           AT END GO TO CLOS-ROUTINE.
 89       MOVE SPACES TO PRINTLINE.
 90       MOVE NUMB TO NUMBA.
 91       MOVE NAME TO NAMEA.
 92       MOVE FARE (CLSS, DEST) TO FAREA.
 93       MOVE CLSS TO CLASSA.
 94       MOVE CITY (DEST) TO DESTA.
 95       WRITE PRINTLINE FROM DISP.
 96       GO TO GET-FARE.
 97   WRITE-HEAD.
 98       WRITE PRINTLINE FROM HEAD.
 99       MOVE SPACES TO PRINTLINE.
100       WRITE PRINTLINE.
101   CLOS-ROUTINE.
102       CLOSE CARDIN, PRINTOUT.
103       STOP RUN.
```

Figure 6-10. Sample Program Using Subscripts (Sheet 2 of 2)

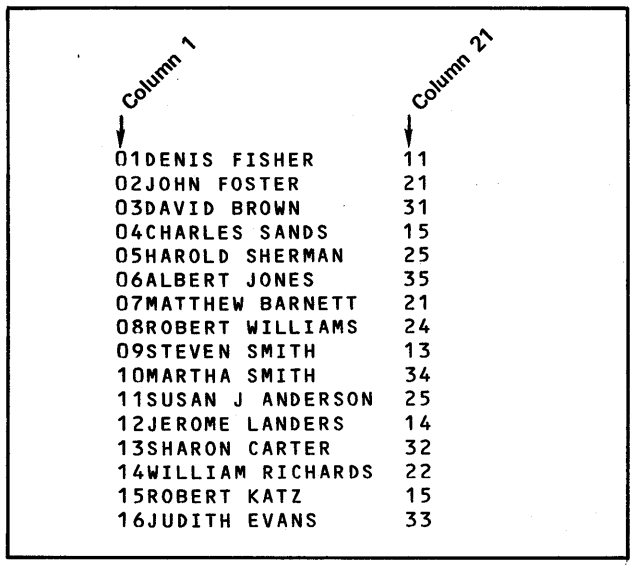


Figure 6-11. Input Data for Subscripting Program

Two tables are described in the Data Division of this program. Each table is redefined in order to reference the table elements by subscripting. The input record contains the data items CLSS and DEST (lines 19 and 20); these data items are used as subscripts to reference a specific element in the table FARE-TABLE (line 58). The table element FARE (line 60) requires two subscripts. The table CITY-ID (line 67) contains only one OCCURS clause and therefore uses only one subscript. The input data item DEST is used to reference a specific CITY element in this table.

TABLE-SEARCHING PROGRAM

Table reference by indexing is illustrated by the sample program shown in figure 6-13. This program reads an input record (line 92), searches a table for the part number in the input record (line 96), obtains the part description from a corresponding table (line 100), and writes the description on the output report (line 101). Sample input data and the resulting report are shown in figures 6-14 and 6-15, respectively.

```
ID          NAME              CLASS     CITY        FARE

01          DENIS FISHER        1       LAX        $127.50
02          JOHN FOSTER         2       LAX        $107.50
03          DAVID BROWN         3       LAX         $87.50
04          CHARLES SANDS       1       SEA        $525.00
05          HAROLD SHERMAN      2       SEA        $500.00
06          ALBERT JONES        3       SEA        $475.00
07          MATTHEW BARNETT     2       LAX        $107.50
08          ROBERT WILLIAMS     2       SFO        $250.00
09          STEVEN SMITH        1       OAK        $240.00
10          MARTHA SMITH        3       SFO        $230.00
11          SUSAN J ANDERSON    2       SEA        $500.00
12          JEROME LANDERS      1       SFO        $272.50
13          SHARON CARTER       3       BUR        $105.00
14          WILLIAM RICHARDS    2       BUR        $125.00
15          ROBERT KATZ         1       SEA        $525.50
16          JUDITH EVANS        3       OAK        $200.00
```

Figure 6-12. Output Report from Subscripting Program

```
 1   IDENTIFICATION DIVISION.
 2   PROGRAM-ID.  TABLE-SEARCHING.
 3   ENVIRONMENT DIVISION.
 4   CONFIGURATION SECTION.
 5   SOURCE-COMPUTER.  CYBER-170.
 6   OBJECT-COMPUTER.  CYBER-170.
 7   INPUT-OUTPUT SECTION.
 8   FILE-CONTROL.
 9       SELECT CARD-FILE ASSIGN TO INPUT.
10       SELECT LIST-FILE ASSIGN TO OUTPUT.
11   DATA DIVISION.
12   FILE SECTION.
13   FD  CARD-FILE
14           LABEL RECORD IS OMITTED
15           DATA RECORD IS CARD.
16   01  CARD.
17       03  PART-NO              PICTURE 999.
18       03  FILLER               PICTURE X(77).
19   FD  LIST-FILE
20           LABEL RECORD IS OMITTED
21           DATA RECORD IS LIST-LINE.
22   01  LIST-LINE        PICTURE X(136).
23   WORKING-STORAGE SECTION.
24   01  NUM              PICTURE 999.
25   01  PRT              PICTURE 999.
26   01  PART-NOS.
27       03  FILLER       PICTURE 999     VALUE 075.
28       03  FILLER       PICTURE 999     VALUE 212.
29       03  FILLER       PICTURE 999     VALUE 153.
30       03  FILLER       PICTURE 999     VALUE 609.
31       03  FILLER       PICTURE 999     VALUE 024.
32       03  FILLER       PICTURE 999     VALUE 030.
33       03  FILLER       PICTURE 999     VALUE 121.
34       03  FILLER       PICTURE 999     VALUE 174.
35       03  FILLER       PICTURE 999     VALUE 185.
36       03  FILLER       PICTURE 999     VALUE '186.
37       03  FILLER       PICTURE 999     VALUE 187.
38       03  FILLER       PICTURE 999     VALUE 205.
39       03  FILLER       PICTURE 999     VALUE 339.
40       03  FILLER       PICTURE 999     VALUE 216.
41       03  FILLER       PICTURE 999     VALUE 206.
```

Figure 6-13. Sample Program Using Index-Names (Sheet 1 of 2)

```
42        03  FILLER       PICTURE 999      VALUE 159.
43        03  FILLER       PICTURE 999      VALUE 157.
44        03  FILLER       PICTURE 999      VALUE 163.
45        03  FILLER       PICTURE 999      VALUE 002.
46        03  FILLER       PICTURE 999      VALUE 010.
47    01  PART-TABLE REDEFINES PART-NOS.
48        03  PART-ITEM    PICTURE 999
49                OCCURS 20 TIMES
50                INDEXED BY INDX-P.
51    01  TABLE-DESCRIPT.
52        03  FILLER       PICTURE X(20)    VALUE "CLOTH SHADES         ".
53        03  FILLER       PICTURE X(20)    VALUE "TEXTURED SHADES      ".
54        03  FILLER       PICTURE X(20)    VALUE "WOODEN ROMAN SHADES  ".
55        03  FILLER       PICTURE X(20)    VALUE "WASHABLE SHADES      ".
56        03  FILLER       PICTURE X(20)    VALUE "DECORATOR SHADES     ".
57        03  FILLER       PICTURE X(20)    VALUE "LIGHTPROOF SHADES    ".
58        03  FILLER       PICTURE X(20)    VALUE "OPAQUE SHADES        ".
59        03  FILLER       PICTURE X(20)    VALUE "WOVEN ALUMINUM SHADE".
60        03  FILLER       PICTURE X(20)    VALUE "TRANSPARENT SHADES   ".
61        03  FILLER       PICTURE X(20)    VALUE "DECORATOR AWNINGS    ".
62        03  FILLER       PICTURE X(20)    VALUE "METAL-STRIP AWNINGS  ".
63        03  FILLER       PICTURE X(20)    VALUE "WOODEN SHUTTERS      ".
64        03  FILLER       PICTURE X(20)    VALUE "ALUMINUM SHUTTERS    ".
65        03  FILLER       PICTURE X(20)    VALUE "VENETIAN BLINDS WOOD".
66        03  FILLER       PICTURE X(20)    VALUE "VENITIAN BLINDS,ALUM".
67        03  FILLER       PICTURE X(20)    VALUE "VENETIAN BLINDS WHIT".
68        03  FILLER       PICTURE X(20)    VALUE "VENETIAN BLINDS,GREY".
69        03  FILLER       PICTURE X(20)    VALUE "STRIPED CANVAS SHADE".
70        03  FILLER       PICTURE X(20)    VALUE "LAMINATED SHADES     ".
71        03  FILLER       PICTURE X(20)    VALUE "DRAPERY HARDWARE     ".
72    01  PART-DESC REDEFINES TABLE-DESCRIPT.
73        03  PART-LIST    PICTURE X(20)
74                OCCURS 20 TIMES
75                INDEXED BY INDX-D.
76    01  HEAD.
77        03  FILLER       PICTURE X(10)    VALUE SPACES.
78        03  HEADLINE     PICTURE X(20)    VALUE "NAME OF PART ORDERED".
79        03  FILLER       PICTURE X(106)   VALUE SPACES.
80    01  LINE-OUT.
81        03  FILLER       PICTURE X(10).
82        03  LIST-NAME    PICTURE X(20).
83        03  FILLER       PICTURE X(106).
84    PROCEDURE DIVISION.
85    START-UP.
86        OPEN INPUT  CARD-FILE.
87        OPEN OUTPUT LIST-FILE.
88        MOVE HEAD TO LIST-LINE.
89        WRITE LIST-LINE
90            BEFORE ADVANCING 2 LINES.
91    FIND.
92        READ CARD-FILE RECORD
93            AT END GO TO CLOS.
94        MOVE PART-NO TO PRT.
95        SET INDX-P TO 1.
96        SEARCH PART-ITEM
97            AT END GO TO NOT-FOUND
98            WHEN PART-ITEM (INDX-P) = PRT NEXT SENTENCE.
99        SET INDX-D TO INDX-P.
100       MOVE PART-LIST (INDX-D) TO LIST-NAME.
101       WRITE LIST-LINE FROM LINE-OUT.
102       GO TO FIND.
103   NOT-FOUND.
104       DISPLAY "PART " PRT " NOT FOUND".
105       GO TO FIND.
106   CLOS.
107       CLOSE CARD-FILE, LIST-FILE.
108       STOP RUN.
```

Figure 6-13. Sample Program Using Index-Names (Sheet 2 of 2)

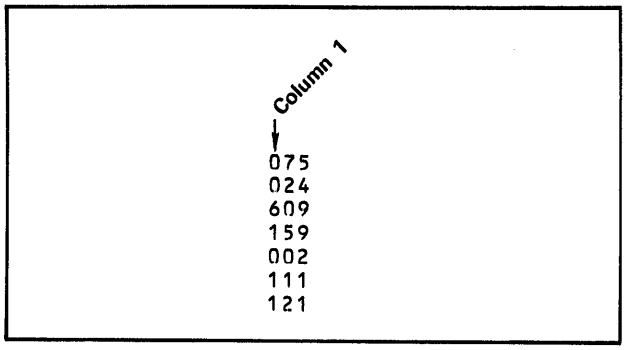


Figure 6-14. Input Data for Indexing Program

The first table described in this program specifies 20 different part numbers (lines 26 through 46). The second table specifies the corresponding description for each of the 20 part numbers (lines 51 through 71). Each table is redefined and assigned an index-name.

```
                NAME OF PART ORDERED


                CLOTH SHADES
                DECORATOR SHADES
                WASHABLE SHADES
                VENETIAN BLINDS WHIT
                LAMINATED SHADES
      PART   111 NOT FOUND
                OPAQUE SHADES
```

Figure 6-15. Output Report from Indexing Program

The table PART-TABLE is searched for the PART-ITEM element that is the same as the part number in the input record (line 98). When the table element is found, the value of the index-name (INDX-P) is transferred to the index-name (INDX-D) for the description table (line 99). The PART-LIST element that corresponds to the PART-ITEM element found by the search is then moved to the line for the output report (line 100).

Character handling is a term that is used to define special operations that can be performed on selected data items. Four types of special operations are provided in COBOL 5: setting the value of a data item, inspecting characters in a data item, transferring characters between data items, and referencing a part of a data-item. These operations are performed by the INITIALIZE, INSPECT, STRING, and UNSTRING statements, and with the reference modification structure.

SETTING THE VALUE OF A DATA ITEM

A data item can be set to a predetermined value by the INITIALIZE statement. The value that is used depends on the format of the statement and the category of the data item. One or more data items are specified as receiving items for the value.

An elementary item or a group item can be specified as a receiving item. When a group item is specified, subordinate elementary items are considered receiving items. The specified item cannot be an index data item, a renamed data item (level 66), or a data item described with the OCCURS DEPENDING ON clause. An item that is subordinate to the specified receiving item and that contains the REDEFINES clause, or any item subordinate to such an item, is not allowed. However, the receiving item itself can have a REDEFINES clause or be subordinate to a data item with a REDEFINES clause. When a group item is specified, subordinate index data items and elementary FILLER data items are not affected. Named data items subordinate to a FILLER group item, however, are receiving items.

In the simplest format of the INITIALIZE statement, receiving items are set to zeros or spaces. Numeric and numeric-edited receiving items are set to zeros; all other data items are set to spaces.

```
INITIALIZE COUNTER, TOTALS, LINE-OUT.
```

When this statement is executed, the numeric items COUNTER and TOTALS are given values of zero. The data item LINE-OUT is an alphanumeric item and is set to spaces.

The REPLACING phrase is included in the INITIALIZE statement to set the value of the receiving item (or items) to a specified value. The value of the sending item is indicated by specifying a literal or the data-name of the item that contains the value. An index data item cannot be specified as the sending item. The category of the sending item and each elementary receiving item must be consistent with the category specified in the REPLACING phrase. If a receiving item is a group item, only those elementary items in the specified category are affected by the INITIALIZE statement; all occurrences of table items in the group item are affected.

The INITIALIZE statement in figure 7-1 illustrates the use of the REPLACING phrase. The sending item is specified by the figurative constant ALL and a nonnumeric literal. The receiving item is the group item LINE-OUT. When the statement is executed, the elementary alphanumeric data items COL-1, COL-2, and COL-3 are set to asterisks. The FILLER data items are not affected by the INITIALIZE statement. An output line produced from LINE-OUT would then print three groups of five asterisks.

```
    .
    .
    .
DATA DIVISION.
    .
    .
    .
01  LINE-OUT.
    03  FILLER PICTURE X(10) VALUE SPACE.
    03  COL-1  PICTURE X(5).
    03  FILLER PICTURE X(10) VALUE SPACE.
    03  COL-2  PICTURE X(5).
    03  FILLER PICTURE X(10) VALUE SPACE.
    03  COL-3  PICTURE X(5).
    03  FILLER PICTURE X(91) VALUE SPACE.
    .
    .
    .
PROCEDURE DIVISION.
    .
    .
    .
    INITIALIZE LINE-OUT REPLACING
        ALPHANUMERIC DATA BY ALL "*".
    .
    .
    .
```

Figure 7-1. Initializing a Group Data Item

INSPECTING CHARACTERS IN A DATA ITEM

A data item can be inspected for the occurrences of one or more characters in order to perform tallying and/or replacing operations. The character string for which the data item is inspected and the operation to be performed are specified in the INSPECT statement. When the statement is executed, the inspection occurs as a series of cycles. The number of inspection cycles performed depends on the inspection criteria specified in the INSPECT statement.

INSPECTION CYCLE

An inspection cycle consists of comparing the character string to be tallied or replaced with an equal number of characters in the data item. The first inspection cycle begins at the first character position of the data item or at the first character position after a specified character string.

When a single character is being tallied or replaced, successive inspection cycles begin at the next character in sequence. When a group of characters is being tallied or replaced, the beginning position for successive inspection cycles depends on whether or not the preceding comparison resulted in a match:

- If a match occurred, the beginning position is the next position to the right of the matching characters.
- If a match did not occur, the beginning position is the next position to the right of the previous beginning position.

INSPECTION LIMITATION

The inspection cycles for tallying or replacing can be limited to the portion of the data item preceding or following the initial occurrence of a specified character string. The limiting character string can be specified as a nonnumeric literal, a figurative constant, or the data-name of a data item containing the character string. A figurative constant is considered to be one character in length. If a data item contains the limiting character string, it must be an elementary alphabetic, alphanumeric, or numeric data item; a numeric data item must be described with display usage.

The BEFORE INITIAL phrase limits the inspection cycles to the characters before the first occurrence of the limiting character string. The inspection cycles begin with the first character and include all characters preceding the first character of the limiting character string.

Only those characters following the first occurrence of the limiting character string are included in the inspection cycles when the AFTER INITIAL phrase is specified. The inspection cycles begin with the character immediately following the limiting character string and continue to the end of the data item.

TALLYING OPERATION

Inspection of a data item counts the number of occurrences of the character string when the keyword TALLYING is specified in the INSPECT statement. The data item to be tallied can be a group item or an elementary item with display usage; implicit display usage exists for a group item or for an elementary item without the USAGE clause. An elementary numeric item must be specified to hold the tally. This data item is not initialized by the INSPECT statement; one is added to the current value of the tally data item each time the character string occurs in the data item being inspected.

Character positions in the data item are tallied when the keyword CHARACTERS is specified. If the BEFORE/AFTER INITIAL phrase is included, only those character positions within the specified limit are tallied.

```
INSPECT ITEM-1 TALLYING ACCUM-1
    FOR CHARACTERS BEFORE INITIAL "X".
```

This statement causes the character positions in ITEM-1 to be tallied; the inspection begins with the first character and continues until the character X is encountered. One is added to ACCUM-1 for each character position preceding the X.

The data item is inspected for all occurrences of a character string when the keyword ALL is specified. If the inspection is limited by the BEFORE/AFTER INITIAL phrase, only those occurrences within the specified limit are tallied.

```
INSPECT ITEM-2 TALLYING ACCUM-2
    FOR ALL "G5" AFTER INITIAL "A".
```

When this statement is executed, the first inspection cycle begins with the character immediately following the first A in ITEM-2. Each occurrence of the character string G5 during the inspection cycles causes one to be added to the current value of ACCUM-2.

The keyword LEADING specifies that the data item is inspected for consecutive occurrences of the character string beginning with the first inspection cycle. The tally data item is incremented by one for each consecutive inspection cycle where the comparison results in a match. The characters to be inspected can be limited by the BEFORE/AFTER INITIAL phrase.

```
INSPECT ITEM-3 TALLYING ACCUM-3
    FOR LEADING ZEROS.
```

This statement inspects ITEM-3 for leading zeros. The inspection begins with the first character position and continues until a nonzero character is encountered. One is added to ACCUM-3 for each leading zero.

REPLACING OPERATION

The replacing operation is performed when the keyword REPLACING is specified in the INSPECT statement. This operation inspects the data item for occurrences of the character string and replaces each occurrence with another character string of equal length. A group item or an elementary item with display usage can be inspected for the replacing operation; implicit display usage exists for a group item or for an elementary item without the USAGE clause.

The replacing character string can be specified as a nonnumeric literal, a figurative constant, or the data-name of a data item containing the character string; it must have the same number of characters as the character string to be replaced. If a figurative constant is specified for the replacing character string, the character string to be replaced must be one character in length. When a data-name is specified for the replacement character string, the storage areas of the replacing string and the string to be replaced should not overlap; if the areas do overlap, unpredictable results might occur during program execution.

Each character in the data item is replaced by another character when the keyword CHARACTERS is specified. If the BEFORE/AFTER INITIAL phrase is included, only those characters within the specified limit are replaced. The replacing character string must be a single character when the keyword CHARACTERS is specified.

```
INSPECT ITEM-4 REPLACING CHARACTERS
    BY SPACE BEFORE INITIAL "X".
```

When this statement is executed, the replacing operation begins with the first character and continues until the character X is encountered. Each character preceding the initial X is replaced by a space.

The keyword ALL is specified when each occurrence of the character string is to be replaced. The BEFORE/AFTER INITIAL phrase can be specified to place a limit on the characters being inspected for the replacement.

```
INSPECT ITEM-5 REPLACING ALL "ABC"
    BY "XYZ".
```

Execution of this statement causes ITEM-5 to be inspected for the character string ABC. The character string XYZ replaces each occurrence of ABC within the data item ITEM-5.

Consecutive occurrences of the character string are replaced by another character string when the keyword LEADING is specified. The first inspection cycle must result in a matching comparison for any replacement to take place; only consecutive occurrences of the character string are replaced. If the BEFORE/AFTER INITIAL phrase is included, inspection cycles are only within the specified limits.

```
INSPECT ITEM-6 REPLACING LEADING "W"
    BY SPACE AFTER INITIAL "A".
```

When this statement is executed, the first inspection cycle begins in the character position immediately following the first A in ITEM-6. Each leading W is then replaced by a space; once an inspection cycle does not result in a matching comparison, the replacing operation is terminated.

Only the first occurrence of the character string is replaced when the keyword FIRST is specified. The inspection cycles are performed within the limit of the BEFORE/AFTER INITIAL phrase, if specified, and are terminated when a match is found.

```
INSPECT ITEM-7 REPLACING FIRST "M"
    BY SPACE BEFORE INITIAL "-".
```

This statement inspects ITEM-7 for the first occurrence of the letter M. The inspection cycles begin with the first character position and continue until the letter M or a hyphen is found. If the letter M is found, it is replaced by a space.

TALLYING AND REPLACING OPERATION

The INSPECT statement can specify that the operation to be performed on the data item includes both tallying and replacing. Each operation is specified in the same manner as described in the preceding paragraphs. Tallying can be performed before or after replacing; before is assumed by default.

```
INSPECT ITEM-8 TALLYING ACCUM-8 FOR
    ALL ZEROS BEFORE INITIAL "X"
    BEFORE REPLACING ALL SPACES BY "*".
```

When this statement is executed, ITEM-8 is inspected for tallying and replacing; tallying is performed before replacing. For each zero before the first X in ITEM-8, one is added to the current value of ACCUM-8. All spaces in ITEM-8 are then replaced by asterisks.

```
INSPECT ITEM-9 TALLYING ACCUM-9 FOR
    LEADING "*" AFTER REPLACING
    ALL SPACES BY "*".
```

This statement specifies that tallying is performed after all spaces in ITEM-9 have been replaced by asterisks. One is then added to ACCUM-9 for each leading asterisk in ITEM-9.

TRANSFERRING CHARACTERS BETWEEN DATA ITEMS

The STRING and UNSTRING statements provide the capability to join and separate the contents of data items. All or part of a sending item can be transferred to a receiving item.

STRING STATEMENT

The characters in two or more data items are transferred to a receiving data item by the STRING statement. The transfer begins with the first sending item and continues with the remaining sending items in the order specified. The transfer terminates when all sending items have been transferred or when the receiving item is filled.

The sending items can be data items, nonnumeric literals, and figurative constants. Data items must be described with implicit or explicit display usage. A figurative constant specified as a sending item is considered to be one character in length. Sending items cannot be boolean data items.

The receiving item can be a group item or an elementary item without editing symbols; the usage of the item must be display. Receiving items cannot be boolean data items and cannot be reference modified. Only those character positions that receive a sending character are affected by execution of the STRING statement.

The sending item and the receiving item should not share any part of their storage areas; if the items are not uniquely defined, unpredictable results might occur when the program is executed.

Transfer of characters begins with the first character in the sending item and terminates as specified in the DELIMITED BY phrase. All characters are transferred when the keyword SIZE is specified.

```
STRING ITEM-A, ITEM-B DELIMITED BY SIZE
    INTO GROUP-1.
```

When this statement is executed, all characters in ITEM-A are transferred to GROUP-1 followed by all characters in ITEM-B. Character transfer terminates when ITEM-A and ITEM-B are exhausted or GROUP-1 is filled.

Transfer of characters can be terminated by the occurrence of a specific character string in the sending item. The character string is a literal or the contents of a data item specified in the DELIMITED BY phrase. The literal can be a nonnumeric literal or a figurative constant; a figurative constant is considered to be one character in length. If a data item is used to delimit the transfer of characters, it must be a display data item.

```
STRING ITEM-C, ITEM-D, ITEM-E
    DELIMITED BY SPACE INTO GROUP-2.
```

This statement specifies that characters are to be transferred from ITEM-C, ITEM-D, and ITEM-E to GROUP-2. Character transfer in each sending item begins with the first character and terminates when a space is encountered; the space is not transferred to the receiving item.

Sending items can have different delimiting character strings. The DELIMITED BY phrase is specified for each sending item or group of sending items to which it applies.

```
STRING ITEM-F DELIMITED BY "*"
    ITEM-G DELIMITED BY TEMP
    INTO GROUP-3.
```

Execution of this statement transfers characters from ITEM-F and ITEM-G to GROUP-3. All characters in ITEM-F up to the first asterisk are transferred to the receiving item, and then all characters in ITEM-G up to a character string equal to the contents of TEMP are transferred to the receiving item.

The beginning position within the receiving item for the transfer of characters can be other than the first character position. The POINTER phrase specifies an elementary integer data item that contains the number of the character position to which a character is transferred. The data item is incremented by one each time a character is transferred to the receiving item. The initial value of the pointer data item must be set by the program before the STRING statement is executed.

```
STRING ITEM-H, ITEM-I DELIMITED BY SIZE
    INTO GROUP-4 WITH POINTER COUNT-4.
```

When this statement is executed, all characters in ITEM-H and ITEM-I are transferred to GROUP-4. The value of COUNT-4 specifies the character position within GROUP-4 in which character transfer begins. When all characters have been transferred, the value of COUNT-4 points to the position immediately following the last character transferred.

The ON OVERFLOW phrase is included in the STRING statement to specify the action to be taken if the receiving item is filled before the transfer operation is completed. The statement specified in this phrase is also executed if the value of the pointer data item does not indicate a position within the receiving item (less than one or greater than the number of character positions in the receiving item).

```
STRING ITEM-J, ITEM-K
    DELIMITED BY SPACE INTO GROUP-5
    ON OVERFLOW GO TO GROUP-FULL.
```

Execution of this statement transfers characters in ITEM-J and ITEM-K preceding the first space in each sending item to GROUP-5. If an overflow condition is encountered, control is passed to the paragraph named GROUP-FULL.

UNSTRING STATEMENT

The UNSTRING statement provides the means to separate data from a sending item into one or more receiving items. Characters are transferred to a receiving item until the item is filled or until a specified delimiter is encountered in the sending item. The transfer of characters occurs according to the COBOL MOVE rules.

The sending item must be an elementary alphanumeric data item or a group data item and must not be reference modified. Each receiving item must be an elementary data item with display usage or a group data item. The PICTURE clause for a receiving item can describe the item as alphabetic without the symbol B, alphanumeric, or numeric without the symbol P. The sending item and the receiving item should not share any part of their storage areas; if the items are not uniquely defined, unpredictable results might occur during program execution.

The UNSTRING statement specifies the sending item and one or more receiving items. When no optional phrases are included in the statement, character transfer begins with the first character position in the sending item and continues until all sending item characters are transferred or all receiving items are filled.

```
UNSTRING GROUP-6
    INTO ITEM-L, ITEM-M, ITEM-N.
```

When this statement is executed, character transfer begins from the first character position in GROUP-6 to the first character position in ITEM-L. When ITEM-L is filled, the next character in GROUP-6 is transferred to the first character position in ITEM-M. Characters are transferred to ITEM-N after ITEM-M is filled. Transfer of characters terminates when all characters have been transferred from GROUP-6 or when ITEM-N has been filled.

A delimiter for the transfer of characters from the sending item to a receiving item is specified by the DELIMITED BY phrase. A literal or the contents of a data item can be specified for the delimiter. The data item must be an elementary or group alphanumeric data item. If a literal is specified, it can be either a nonnumeric literal or a figurative constant; a figurative constant represents a single-character delimiter.

Characters are transferred from the sending item to the first receiving item beginning with the first character position. Transfer to subsequent receiving items begins with the first character following the delimiter for the previous transfer operation. Transfer to a receiving item terminates when the delimiter is encountered or when the receiving item is filled. The following events occur when transfer terminates due to the receiving item being filled:

- The sending item is searched for a delimiter; a delimiter is located.
- The sending item is moved to the receiving item, according to the COBOL MOVE rules.
- The receiving item is filled; truncation of characters occurs; transfer stops.

If the keyword ALL is specified preceding the delimiter, character transfer resumes following all consecutive occurrences of the delimiter; otherwise, each consecutive occurrence of the delimiter causes a subsequent receiving item to be zero or space filled, according to the description of the individual receiving item.

```
UNSTRING GROUP-7 DELIMITED BY "*"
    INTO ITEM-O, ITEM-P.
```

Execution of this statement transfers characters from GROUP-7 into ITEM-O until an asterisk is encountered or ITEM-O is filled. If the character following the asterisk is another asterisk, ITEM-P is zero or space filled; otherwise, the character and succeeding characters are transferred until another asterisk is encountered or ITEM-P is filled. Character transfer is terminated and control is immediately passed to the next statement if the end of GROUP-7 is reached.

More than one delimiter can be specified in the DELIMITED BY phrase. The occurrence of any specified delimiter terminates transfer to the current receiving item. If the DELIMITER IN phrase is specified for a receiving item, the delimiter that terminated transfer to the receiving item is moved to a separate data item. The data item to receive the delimiter must be an alphanumeric elementary or group data item.

```
UNSTRING GROUP-8 DELIMITED BY ALL ZEROS,
    OR ALL SPACES, OR ALL "*"
    INTO ITEM-Q, DELIMITER IN Q-SEP
    ITEM-R, DELIMITER IN R-SEP.
```

This statement transfers characters from GROUP-8 to ITEM-Q and ITEM-R. The transfer to ITEM-Q is terminated when a zero, a space, or an asterisk is encountered in GROUP-8; the actual delimiter is moved to Q-SEP. The next character following all consecutive appearances of the delimiter is transferred to ITEM-R; character transfer continues until one of the three delimiters is encountered. The delimiter that terminates transfer to ITEM-R is then moved to R-SEP. If the transfer of characters to either receiving item is terminated by reaching the end of the sending item, the data item to receive the delimiter is set to spaces.

When a delimiter is specified for the sending item, a count of the characters preceding the delimiter can be stored in a data item. The count indicates the number of characters between the preceding delimiter and the current delimiter, whether or not all characters are transferred to the receiving item. The COUNT IN phrase specifies the data item to receive the character count for the associated receiving item. The data item must be an elementary numeric integer data item.

```
UNSTRING GROUP-9
    DELIMITED BY ALL ZEROS
    INTO ITEM-S, COUNT IN S-CNTR
    ITEM-T, COUNT IN T-CNTR.
```

Execution of this statement transfers characters from GROUP-9 to ITEM-S and ITEM-T. Transfer to the receiving items is terminated when zeros are encountered in the sending item. The count of characters up to the first delimiter is stored in S-CNTR. The character count between the first and second delimiters is stored in T-CNTR; if the sending item does not contain a second delimiter, T-CNTR contains the number of characters between the first delimiter and the end of GROUP-9.

The POINTER phrase is specified when the transfer of characters from the sending item is to begin in a position other than the first character position. For the first character to be transferred, the data item indicates the beginning position. It is incremented by one each time a character is transferred. The data item must be an elementary numeric integer data item. The initial value of the pointer data item must be established by the program before the UNSTRING statement is executed.

```
UNSTRING GROUP-10 INTO ITEM-U,
    ITEM-V WITH POINTER COUNT-10.
```

When this statement is executed, the current value of COUNT-10 indicates the position of the first character within GROUP-10 to be transferred to ITEM-U. As each character is transferred, COUNT-10 is updated to reflect the next character position in GROUP-10.

The TALLYING phrase is used to maintain a count of the receiving items to which characters are actually transferred when the UNSTRING statement is executed. The data item specified in this phrase must be an elementary numeric integer data item and the initial value must be set by the program.

```
UNSTRING GROUP-11
    DELIMITED BY ALL SPACES
    INTO ITEM-W, ITEM-X
    TALLYING IN COUNTER.
```

Each time this statement is executed, COUNTER is incremented by one for each receiving item to which characters are transferred. Character transfer from GROUP-11 to ITEM-W and to ITEM-X is terminated by the occurrence of spaces in GROUP-11. The initial value of COUNTER is set before the first execution of the UNSTRING statement.

Action to be taken if the transfer terminates before the end of the sending item is reached is specified in the ON OVERFLOW phrase. The statement in this phrase is executed when the receiving items are filled and the sending item is not exhausted, or when the value of the pointer data item does not indicate a position within the sending item.

```
UNSTRING GROUP-12 INTO ITEM-Y,
    ITEM-Z WITH POINTER COUNT-12
    ON OVERFLOW GO TO SEND-AGAIN.
```

This statement transfers characters from GROUP-12 to ITEM-Y and ITEM-Z; transfer begins at the position indicated by the current value of COUNT-12. If transfer to the two receiving items terminates before the end of GROUP-12 is encountered, or if the value of COUNT-12 is less than one or greater than the number of character positions in GROUP-12, control is passed to the paragraph named SEND-AGAIN.

REFERENCING PART OF A DATA ITEM

Reference modification allows the referencing of a portion of a data item without predefining the item in the Data Division (with level number, name, size, and usage). Data item usage can only be DISPLAY. With the exception of boolean items, which remain in the boolean class, the class of reference modified items is always alphanumeric.

Reference modification can be used to eliminate complex REDEFINES clauses and to unstring data into areas of variable length.

In its simplest structure, an item is described in the Working-Storage Section. A portion of the characters in the item is then referenced in a Procedure Division statement.

```
WORKING-STORAGE SECTION.
01 NAMES  PICTURE X(7) VALUE "JOHNSON".
    .
    .
    .
PROCEDURE DIVISION.
    .
    .
    .
    MOVE NAMES (1:5) TO HOLD
```

This MOVE statement moves five characters, beginning with the first character of NAMES. The characters JOHNS are moved to the data item HOLD.

The example in figure 7-2 illustrates an out-of-bounds reference that is undetected by the COBOL compiler. Unless DB=RF is specified on the COBOL5 control statement, the values that are used in referencing a data

item are not checked to be within the range of the data item's length. If DB=RF is specified, bounds checking is performed. If illegal reference modification is used, a message appears in the dayfile at execution time and the job aborts. Program output (without the DB parameter specified) is also shown.

The first ordinal within the parentheses specifies the position of the leftmost character of interest within the data item. This ordinal must be a positive non-zero integer not greater than the number of characters in the data item. An arithmetic expression is also allowed.

The second ordinal within the parentheses specifies the total number of consecutive characters of interest. This ordinal must be a positive non-zero integer. Alternatively, END can be used to include all characters to the end of the data item. An arithmetic expression is also allowed.

The sum of the first and the second ordinals, minus one, cannot exceed the number of characters in the data item.

Figure 7-3 illustrates valid statements involving reference modification. The subscript is evaluated first in the final MOVE statement.

A. **Program Listing**

```
IDENTIFICATION DIVISION.
PROGRAM-ID.  REFMOD.
DATA DIVISION.
WORKING-STORAGE SECTION.
01 STRS.
    02 STRT PIC X(10) VALUE "ABCDEFGHIJ".
    02      PIC X(10) VALUE ALL "9".
    02 A    PIC 9 VALUE 3.
    02 B    PIC 9 VALUE 3.
PROCEDURE DIVISION.
STR.
    DISPLAY "**OUTPUT IS -"
    DISPLAY STRT (3 : A + B )
    DISPLAY STRT (4 : A + B )
    DISPLAY STRT ( 4 : 9 )
    IF STRT ( 3 : A + B ) = "CDEFGH"
      DISPLAY "GOOD".
    STOP RUN.
```

B. **Program Output**

```
**OUTPUT IS -
CDEFGH
DEFGHI
DEFGHIJ99
GOOD
```

Figure 7-2. Out-of-Bounds Reference Modification

```
WORKING-STORAGE SECTION.
01 NUMERALS PICTURE X(10) VALUE "0123456789".
01 CONST PICTURE 9 VALUE IS 3.
    .
    .
    .
PROCEDURE DIVISION.
    .
    .
    .
    MOVE NUMERALS (2 : 6) TO N(1).
    MOVE NUMERALS (3 : I + K) TO N(2).
    MOVE N(3) (4 : END) TO ITEM (CONST : 3).
```

Figure 7-3. Reference Modification Examples

Records in files can be sorted or merged automatically by internal routines that are executed as a result of a SORT or MERGE statement. The sort or merge operation uses one or more keys to process the records; the data items to be used as keys are specified in the SORT or MERGE statement.

Sorting or merging records is a three-phase operation:

1. Input phase - Transfers records from the input file or files to the sort/merge file.
2. Sort/merge phase - Sorts or merges the records in the sort/merge file.
3. Output phase - Transfers sorted or merged records to the output file.

The sort operation causes records from one or more files to be arranged in order according to a specified sequence. The merge operation combines records from two or more identically sequenced files.

SORT/MERGE FILE

The sort/merge file is not an actual file although it is logically treated as a file during sort/merge operation. It must be specified in SELECT and ASSIGN clauses in the FILE-CONTROL paragraph of the Environment Division; no other clauses are allowed for a sort/merge file.

A Sort-Merge Description (SD) entry must be specified in the Data Division for the sort/merge file. At least one Record Description entry must be included in the SD entry to describe the key items that are used to sort or merge the files. When a key item is a nonnumeric data item, the collating sequence used to determine the order of records in the output file can be specified for the sort/merge operation.

SORT-MERGE DESCRIPTION ENTRY

Whenever a sort or merge operation is to be performed during program execution, the file to be used for the sort or merge procedure is described in the File Section by a Sort-Merge Description entry (SD entry). The file-name from the SELECT clause is specified in the SD entry. Two optional clauses can be specified. The RECORD clause indicates the size of the sort/merge record; however, actual record size is determined by the Record Description entries for the file. The DATA RECORDS clause documents the names of the record formats for the sort/merge file.

At least one record format must be described for the sort/merge file. The size of the largest record described establishes the maximum record length. If a record processed during the sort/merge operation exceeds the maximum record length, the record is truncated. When fixed-length records are being merged, records should be at least 10 characters in length; otherwise, an extra record exists at the end of the output file for each input file used in the merge operation.

A file described in an SD entry can be referenced in the Procedure Division only by sort/merge statements. An SD entry for a sort file is illustrated in figure 8-1.

```
      .
      .
      .
DATA DIVISION.
FILE SECTION.
FD   GEN-FILE
         LABEL RECORD IS OMITTED
         DATA RECORD IS GEN-REC.
01   GEN-REC.
     03   IDENT-A   PICTURE 9(8).
     03   IDENT-B   PICTURE 99.
     03   IDENT-C   PICTURE X(20).
SD   SORT-FILE
     DATA RECORD IS SORT-REC.
01   SORT-REC.
     03   IDENT-1   PICTURE 9(8).
     03   IDENT-2   PICTURE 99.
     03   IDENT-3   PICTURE X(20).
     .
     .
     .
PROCEDURE DIVISION.
     .
     .
     .
     SORT SORT-FILE ON ASCENDING KEY
              IDENT-1, IDENT-2, IDENT-3
         USING GEN-FILE
         GIVING SORTED-FILE.
     .
     .
     .
```

Figure 8-1. SD Entry and Key Items

KEY ITEMS

One or more data items in the Record Description entry for a sort/merge file are specified as key items for the sort/merge operation. The actual value of a key item is used to determine the order of records in the output file. When more than one Record Description entry is specified for a sort/merge file, each key item must be described in at least one record.

Key item values are sequenced in either ascending or descending order as specified in the SORT or MERGE statement. Ascending order causes the values to be sequenced from the lowest value to the highest value; descending order is from the highest value to the lowest value. The position of a key item value in the sorted or merged order is determined according to the rules for comparison of operands in relational conditions; in this instance, the full range of COMPUTATIONAL-2 items can be properly compared. Relational conditions are discussed in section 5, Conditional Operations.

Data items used as keys for a sort/merge operation must be fixed-length items. A key item cannot be described by or be subordinate to an item described by the OCCURS clause. A key data item cannot be described with COMPUTATIONAL-4 usage.

When two or more key items are specified for the sort/merge operation, the order of the key items determines the order of significance. The first key specified is the most significant key and the last key specified is the least significant key. Comparisons for sorting or merging records proceed from the most significant to the least significant key.

For a sort operation, records with duplicate values for all specified keys are sequenced in the order the records were released to the sort file. This is called initial sequence and requires 10 extra characters for each record sorted. A significant amount of overhead can be involved when initial sequence is maintained. If the sequencing of records with duplicate key values is not important, the overhead can be reduced by executing the following statement to remove the initial sequence option:

```
ENTER "C.SORTP".
```

After this statement is executed, initial sequence is not maintained for a sort operation. Refer to the discussion of memory allocation for the effect on the initial sequence option when this statement is used to change the central memory block size for a sort operation.

The SD entry shown in figure 8-1 describes three elementary items. The SORT statement in the Procedure Division specifies each of these items as a key item for the sort operation. The most significant key is the data item IDENT-1. If two records contain the same value for IDENT-1, the values for the IDENT-2 data items are compared. When the first two key data items have identical values, the third key items (IDENT-3) are compared. The record with the lower key item value (on the first, second, or third comparison) is the first of the two records in the sorted file.

MEMORY ALLOCATION

A block of 9216_{10} words of central memory is allocated for the sort or merge operation. For most operations, this default size is sufficient; however, more efficiency can be gained by increasing or decreasing the size of the memory block when a large or a small sort/merge operation is performed. Memory size is changed by executing the following statement:

```
ENTER "C.SORTP" USING data-name-1, data-name-2.
```

Data-name-1 specifies a COMPUTATIONAL-1 data item that contains the memory size to be used for all subsequent sort and merge operations. It cannot be omitted or equal to zero; if only data-name-2 is of interest, data-name-1 must specify the default value.

Data-name-2 refers to the initial sequence option for duplicate sort keys. If it is omitted or is equal to zero, the initial sequence option is removed; a value other than zero in the COMPUTATIONAL-1 data item causes initial sequence to be maintained.

SORT/MERGE OPERATION

The input and output phases of a sort/merge operation are performed automatically when input and output files are specified in the SORT or MERGE statement. For a sort operation, the input phase can be program-controlled by providing an input procedure. The output phase is program-controlled for either a sort or a merge operation when an output procedure is specified in the SORT or MERGE statement.

INPUT/OUTPUT FILES

Input files are specified for the sort/merge operation when the records to be sorted or merged reside on files. The MERGE statement must indicate at least two input files. If an output file is specified, the sorted or merged records are automatically written on the output file.

The USING phrase of the SORT or MERGE statement specifies the input files. The following functions are then performed automatically:

1. The input files are opened for input and the sort/merge file is opened for output.
2. All the input records are transferred to the sort/merge file.
3. The input files and the sort/merge file are closed.

At this point in the sort/merge operation, the records are available in sorted or merged sequence. When an output file is specified in the GIVING phrase, the following functions are performed automatically:

1. The sort/merge file is opened for input and the output file is opened for output.
2. The sorted or merged records are transferred to the output file.
3. The sort/merge file and the output file are closed.

INPUT PROCEDURE

The input phase of a sort operation is controlled by the program when the INPUT PROCEDURE phrase is specified instead of the USING phrase. An input procedure, which must be written in one or more contiguous sections in the Procedure Division, allows the user to control the release of records to the sort file. It can include statements that select, create, or modify records for the sort operation. At least one RELEASE statement must be included to transfer records to the sort file; records are transferred one at a time. Execution of a SORT statement with the INPUT PROCEDURE phrase proceeds as follows:

1. The sort file is opened for output.
2. Control is passed to the input procedure until the last statement of the procedure has been executed.
3. The sort file is closed.

Processing then continues with the sort and output phases.

A SORT or MERGE statement cannot be specified within the input procedure sections of the source program. Control cannot be explicitly transferred outside the input procedure; however, implicit transfer of control to declarative procedures is allowed.

OUTPUT PROCEDURE

An output procedure provides the means for the user to control the use of records returned from the sort/merge file during the output phase of the sort/merge operation. The OUTPUT PROCEDURE phrase replaces the GIVING phrase in the SORT or MERGE statement and specifies the section or range of sections that contains the statements to be executed for the output phase.

The output procedure can include statements that select, modify, or copy the sorted or merged records. At least one RETURN statement must be specified to return a record from the sort/merge file for subsequent processing by the output procedure. Records are returned one at a time in sorted or merged sequence.

The output phase of a sort/merge operation that specifies an output procedure is as follows:

1. The sort/merge file is opened for input.
2. Control is transferred to the output procedure until all sorted or merged records have been returned; the AT END phrase of the last RETURN statement is executed.
3. The sort/merge file is closed.

A SORT or MERGE statement cannot be specified within the output procedure sections of the source program. Control cannot be explicitly transferred outside the output procedure; however, implicit transfer of control to declarative procedures is allowed.

SORT/MERGE STATEMENTS

Five Procedure Division statements are applicable to the sort/merge operation. The SORT statement sequences records according to specified key items. The MERGE statement combines two or more identically sequenced files. An input procedure for a sort operation requires the RELEASE statement; the RETURN statement must be used in an output procedure for a sort or merge operation. The collating sequence for sort/merge operations can be established through the SET statement.

SORT STATEMENT

The SORT statement causes records from one or more files to be sorted on a set of specified keys. Records are transferred to the sort file during the input phase, sorted on the key items during the sort phase, and returned from the sort file during the output phase. Only one file on a multifile tape reel can be specified in the SORT statement.

The sort file designated in the SORT statement is described in an SD entry in the File Section of the Data Division. This file receives the records to be sorted during the input phase, contains the records in sorted order at the end of the sort phrase, and provides the records to be returned during the output phase.

One or more key items are specified in the SORT statement. A record is placed in the sorted sequence according to the contents of the key items. Key values are sequenced as specified by the keyword ASCENDING or DESCENDING.

The collating sequence for a nonnumeric sort can be specified in the SORT statement. The alphabet-name in the COLLATING SEQUENCE phrase is defined by an ALPHABET clause in the SPECIAL-NAMES paragraph of the Environment Division. A collating sequence established by a SET statement before execution of the SORT statement overrides the COLLATING SEQUENCE phrase.

The USING phrase specifies one or more input files for the sort operation. Input files can have any file organization and are described by FD entries in the File Section. Automatic transfer of input records to the sort file occurs as if sequential READ statements were being executed.

If the USING phrase is not specified, the INPUT PROCEDURE phrase must specify the Procedure Division section or range of sections containing the statements to process and transfer records to the sort file. Control is passed to the input procedure through the SORT statement; the procedure must not be entered directly.

The sorted records are automatically written on the output file when the GIVING phrase is specified. The output file can have sequential, relative, indexed, or actual-key file organization. If the output file has indexed file organization, the most significant key for the sort operation must be the primary key and the key values must be in ascending sequence. The output file is described by an FD entry in the File Section; record size must be the same size described by the SD entry for the sort file.

The OUTPUT PROCEDURE phrase replaces the GIVING phrase when the output phase of the sort operation is program-controlled. The phrase specifies the section or range of sections that contains the statements to return and process the sorted records. The output procedure must not be entered directly; it receives control through the SORT statement.

Example 1

```
SORT SORT-FILE ON ASCENDING KEY IDENT-1
     ON DESCENDING KEY IDENT-3
     COLLATING SEQUENCE IS SORT-SEQ
     USING FILE-1, FILE-2
     GIVING FILE-3
```

Example 2

```
SORT SORT-FILE ON ASCENDING KEY IDENT-1
     INPUT PROCEDURE IS INP-1
     OUTPUT PROCEDURE IS OUT-1 THRU OUT-3.
```

Figure 8-2. Examples of the SORT Statement

Two sample SORT statements are shown in figure 8-2. Example 1 specifies two key items for the sort operation. The most significant key, IDENT-1, is sequenced in ascending order. The second key item, IDENT-3, is used when two or more records contain the same value for IDENT-1; it is sequenced in descending order. The collating sequence for the nonnumeric key, IDENT-3, is specified as SORT-SEQ; SORT-SEQ is defined by the ALPHABET clause in the Environment Division. Two files are designated for input; records from FILE-1 and FILE-2 are automatically transferred to the sort file. The sorted records are written on the file named FILE-3.

Example 2 in figure 8-2 illustrates a SORT statement that uses input and output procedures. One key item, IDENT-1, is specified and the records are to be sequenced with the values of IDENT-1 in ascending order. The input procedure, which is contained in a section named INP-1, receives control during the input phase of the sort operation. Execution of the statements in INP-1 causes records to be transferred to the sort file. The output procedure is contained in the sections beginning with OUT-1 and ending with OUT-3. Execution of the statements in these sections includes the return of sorted records from the sort file.

MERGE STATEMENT

The MERGE statement causes records from two or more identically sequenced files to be combined based on the values of specified key items. During the input phase, the records are transferred to the merge file. The merge phase merges the records into a single file with the records in order according to the key items. The output phase returns the records from the merge file.

The merge file specified in the MERGE statement is described in an SD entry in the File Section of the Data Division. This file receives the records from the input files, contains the merged records at the end of the merge phase, and provides the records in merged order to the output file or output procedure.

At least one key item is specified for the merge operation. Additional key items are specified for use when duplicate values can exist for a key item. The merged sequence for a key item is as specified by the keyword ASCENDING or DESCENDING.

When a key item for the merge operation is a nonnumeric data item, the collating sequence to be used for the comparison can be specified in the MERGE statement. The COLLATING SEQUENCE phrase specifies an alphabet-name that is defined by an ALPHABET clause in the SPECIAL-NAMES paragraph of the Environment Division. A SET statement executed before the MERGE statement can also specify the collating sequence for the merge operation. The collating sequence established by the SET statement overrides a collating sequence specified in the MERGE statement.

Two or more input files are specified in the USING phrase. These files must have sequential file organization and must be described by FD entries in the File Section of the Data Division. Record sizes for all input files must be the same as described in the SD entry. The records in the input files are transferred automatically to the merge file as if sequential READ statements were being executed.

The output phase of the merge operation proceeds automatically when an output file is specified in the GIVING phrase. The output file is described by an FD entry in the File Section and must have the same record size as described in the SD entry for the merge file. The file organization for the output file can be sequential, relative, indexed, or actual-key. For indexed file organization, the most significant key for the merge operation must be the primary key and the sequence must be ascending.

If the GIVING phrase is not specified, the OUTPUT PROCEDURE phrase must specify the Procedure Division section or range of sections containing the statements to return and process the merged records. Control is passed to the output procedure through the MERGE statement; the procedure must not be entered directly.

The use of the MERGE statement is illustrated in figure 8-3. Example 1 shows a statement that merges the records from two files (INFILE-1 and INFILE-2) and transfers the merged records to a third file (OUTFILE). The key field ITEM-A is used to merge the records with the key values in ascending sequence.

Example 1

```
MERGE MERGE-FILE ON ASCENDING KEY ITEM-A
    USING INFILE-1, INFILE-2
    GIVING OUTFILE.
```

Example 2

```
MERGE MERGE-FILE ON ASCENDING KEY ITEM-A
    ON DESCENDING KEY ITEM-B
    USING INFILE-1, INFILE-2
    OUTPUT PROCEDURE IS MERGE-OUT.
```

Figure 8-3. Examples of the MERGE Statement

Example 2 in figure 8-3 shows a MERGE statement that specifies an output procedure to be executed during the output phase. The input records are merged with the values of the key item ITEM-A in ascending sequence; the key item ITEM-B, which is used when more than one record contains the same value for ITEM-A, is sequenced in descending order. The output procedure is contained in the section named MERGE-OUT; the statements in the section return and process the records from the merge file.

RELEASE STATEMENT

The RELEASE statement is used in an input procedure for the SORT statement. At least one RELEASE statement must be included in the input procedure. When this statement is executed, a record is transferred to the sort file.

The record-name specified in the RELEASE statement is a record-name in the Data Division SD entry for the sort file. After the RELEASE statement is executed, the sort record is no longer available in the sort file record area.

The FROM phrase is included in the RELEASE statement to move the contents of a data area to the sort record area before transferring the sort record to the sort file. This phrase can be effectively used when the data for a sort record is created in a Working-Storage area; the FROM phrase eliminates the need to move the data into the sort record area.

Two examples using the RELEASE statement are shown in figure 8-4. In both examples, the section named INP-1 has been specified as the input procedure in a SORT statement. Example 1 reads an input record and moves the corresponding data to the sort record; the RELEASE statement then transfers the sort record to the sort file. Example 2 creates a record (TEMP-REC) from an input data item and an item resulting from an ADD statement; the RELEASE statement moves the data from TEMP-REC to the sort record (SORT-REC) and then releases the sort record to the sort file.

RETURN STATEMENT

The RETURN statement is used in an output procedure for a SORT or MERGE statement. At least one RETURN statement must be included in an output procedure. Execution of this statement causes the next record in sorted or merged sequence to be made available to the output procedure for processing.

The file-name of the sort/merge file is specified in the RETURN statement. An AT END phrase is included to specify the action to be taken after the last record has been returned from the sort/merge file. After the RETURN statement is executed, the record is available for processing by the output procedure. When the INTO phrase is included in the RETURN statement, the sort/merge record is moved from the sort/merge file into the specified storage area as well as into the sort/merge record area.

The use of the RETURN statement in output procedures is illustrated in figure 8-5. In both examples, the section named OUT-1 has been specified as the output procedure in a SORT or MERGE statement. Example 1 returns the sorted record, moves the corresponding data to OUT-REC, and writes the record on the output file. Example 2 returns a record from the merge file and at the same time stores it in an area named TEMP-REC; the value of an additional item in TEMP-REC is computed and the record is then written on the output file.

Example 1

```
     .
     .
     .
PROCEDURE DIVISION.
     .
     .
     .
INP-1 SECTION.
IN-RECORDS.
     READ CARD-IN RECORD
          AT END GO TO IN-END.
     MOVE CORRESPONDING CARD TO SORT-REC.
     RELEASE SORT-REC.
     GO TO IN-RECORDS.
     .
     .
     .
```

Example 2

```
     .
     .
     .
DATA DIVISION.
     .
     .
     .
01   TEMP-REC.
     03   ITEM-A   PICTURE 9(5).
     03   ITEM-B   PICTURE 9(7).
     .
     .
     .
PROCEDURE DIVISION.
     .
     .
     .
INP-1 SECTION.
IN-RECORDS.
     READ CARD-IN RECORD
          AT END GO TO IN-END.
     MOVE ITM-A TO ITEM-A.
     ADD ITM-C, ITM-B GIVING ITEM-B.
     RELEASE SORT-REC FROM TEMP-REC.
     .
     .
     .
```

Figure 8-4. Examples of the RELEASE Statement

Example 1

```
     .
     .
     .
PROCEDURE DIVISION.
     .
     .
     .
OUT-1 SECTION.
OUT-RECORDS.
     RETURN SORT-FILE RECORD
          AT END GO TO OUT-END.
     MOVE CORRESPONDING ST-REC TO OUT-REC.
     WRITE OUT-REC.
     GO TO OUT-RECORDS.
     .
     .
     .
```

Example 2

```
     .
     .
     .
DATA DIVISION.
     .
     .
     .
01   TEMP-REC.
     03   ITEM-A   PICTURE 9(5).
     03   ITEM-B   PICTURE 9(7).
     03   ITEM-C   PICTURE 9(5).
     .
     .
     .
PROCEDURE DIVISION.
     .
     .
     .
OUT-1 SECTION.
OUT-RECORDS.
     RETURN MERGE-FILE RECORD INTO TEMP-REC
          AT END GO TO OUT-END.
     COMPUTE ITEM-C = ITEM-A - ITEM-B.
     WRITE TEMP-REC.
     GO TO OUT-RECORDS.
     .
     .
     .
```

Figure 8-5. Examples of the RETURN Statement

SET STATEMENT

The SET statement can be used to establish the collating sequence for sort or merge operations. The collating sequence of a SET statement executed prior to the SORT or MERGE statement overrides any other collating sequence. A SET statement in a subprogram does not affect the collating sequence of any other program.

The alphabet-name specified in the SET statement must be defined in the SPECIAL-NAMES paragraph in the Environment Division. Depending on the keyword used in the SET statement, the specified collating sequence can apply to sort operations only, merge operations only, or both sort and merge operations.

The SET statement shown in figure 8-6 establishes the collating sequence for the subsequent SORT statement. The collating sequence S-SEQ is defined in the SPECIAL-NAMES paragraph as CDC-64; the collating sequence of the CDC 64-character code set is used for the sort operation.

```
      .
      .
      .
ENVIRONMENT DIVISION.
      .
      .
      .
SPECIAL-NAMES.
      ALPHABET S-SEQ IS CDC-64.
      .
      .
      .
PROCEDURE DIVISION.
      .
      .
      .
      SET SORT COLLATING SEQUENCE TO S-SEQ.
      SORT SORT-FILE
          ON ASCENDING KEY IDENT-3
          USING FILE-1
          GIVING FILE-2.
```

Figure 8-6. Establishing a Collating Sequence

SAMPLE SORT PROGRAM

The sample program shown in figure 8-7 illustrates a sort operation that is controlled by input and output procedures. The program reads an input deck, selects records for the sort file, and produces an output listing. The input data shown in figure 8-8 is used to create the output report shown in figure 8-9.

The input phase of the sort operation is controlled by the input procedure contained in the section INP-1 (lines 76 through 87). Records are read from the input file CARD-FILE; only those records that contain the letter A in the data item FLAG are released to the sort file.

The sort phase sorts the records in the sort file in ascending order according to the value of the key item S-NAME (line 72). Because no collating sequence is specified for the sort operation, the program collating sequence is used.

The output phase of the sort operation is controlled by the output procedure contained in the section OUT-1 (lines 88 through 105). Records are returned from the sort file and stored in the data area TEMP-REC. Output lines are created and written on the output file PRINT-FILE.

SAMPLE MERGE PROGRAM

The sample program shown in figure 8-10 illustrates a merge operation. The program reads two input files and merges them into a single output file. The first five characters of each record represent the merge key and are used to merge the records in ascending order.

```
 1   IDENTIFICATION DIVISION.
 2   PROGRAM-ID.  SORT-EXAMPLE.
 3   ENVIRONMENT DIVISION.
 4   CONFIGURATION SECTION.
 5   SOURCE-COMPUTER.  CYBER-170.
 6   OBJECT-COMPUTER.  CYBER-170.
 7   INPUT-OUTPUT SECTION.
 8   FILE-CONTROL.
 9       SELECT CARD-FILE   ASSIGN TO INPUT.
10       SELECT PRINT-FILE  ASSIGN TO OUTPUT.
11       SELECT SORT-FILE   ASSIGN TO SORTFLE.
12   DATA DIVISION.
13   FILE SECTION.
14   FD  CARD-FILE
15           LABEL RECORD IS OMITTED
16           DATA RECORD IS CARD-REC.
17   01  CARD-REC.
18       03  CUST-ID          PICTURE 999.
19       03  CUST-NAME        PICTURE X(20).
20       03  CUST-ADDRESS     PICTURE X(45).
21       03  FILLER           PICTURE X.
22       03  LIST-DATE        PICTURE X(8).
23       03  FILLER           PICTURE XX.
24       03  FLAG             PICTURE X.
```

Figure 8-7. Sample Sort Program (Sheet 1 of 3)

```
25    SD  SORT-FILE
26            DATA RECORD IS SORT-REC.
27    01  SORT-REC.
28        03  S-NAME           PICTURE X(20).
29        03  S-ADDRESS        PICTURE X(45).
30    FD  PRINT-FILE
31            LABEL RECORDS ARE OMITTED
32            DATA RECORD IS PRINTLINE.
33    01  PRINTLINE            PICTURE X(136).
34    WORKING-STORAGE SECTION.
35    01  CNTR                 PICTURE 99.
36    01  TEMP-REC.
37        03  T-NAME           PICTURE X(20).
38        03  T-ADDRESS.
39            05  T-STREET     PICTURE X(20).
40            05  T-CITY       PICTURE X(20).
41            05  T-ZIP        PICTURE 9(5).
42    01  OUT-LINE-1.
43        03  FILLER           PICTURE 9       VALUE 1.
44        03  FILLER           PICTURE X(27)   VALUE SPACES.
45        03  FILLER           PICTURE X(16)   VALUE "CUSTOMER LIST ON".
46        03  FILLER           PICTURE X       VALUE SPACES.
47        03  DATE-OUT         PICTURE X(8).
48        03  FILLER           PICTURE X(84)   VALUE SPACES.
49    01  OUT-LINE-2.
50        03  FILLER           PICTURE X(5)    VALUE SPACES.
51        03  NAME-OUT         PICTURE X(20).
52        03  FILLER           PICTURE X(5)    VALUE SPACES.
53        03  STREET-OUT       PICTURE X(20).
54        03  FILLER           PICTURE X(5)    VALUE SPACES.
55        03  CITY-OUT         PICTURE X(20).
56        03  FILLER           PICTURE X(5)    VALUE SPACES.
57        03  ZIP-OUT          PICTURE 9(5).
58        03  FILLER           PICTURE X(51)   VALUE SPACES.
59    PROCEDURE DIVISION.
60    READ-CARD SECTION.
61    READ-IN.
62        OPEN INPUT CARD-FILE.
63        OPEN OUTPUT PRINT-FILE.
64        MOVE ZEROS TO CNTR.
65        READ CARD-FILE RECORD
66            AT END GO TO ERROR-1.
67        IF LIST-DATE NOT EQUAL TO SPACES
68            MOVE LIST-DATE TO DATE-OUT
69            ELSE GO TO ERROR-1.
70    SORT-CARD SECTION.
71    SORTING.
72        SORT SORT-FILE ON ASCENDING KEY S-NAME
73            INPUT PROCEDURE IS INP-1
74            OUTPUT PROCEDURE IS OUT-1.
75        GO TO END-SORT.
76    INP-1 SECTION.
77    IN-1.
78        READ CARD-FILE RECORD
79            AT END GO TO IN-2.
80        IF FLAG NOT EQUAL TO "A"
81            GO TO IN-1.
82        MOVE CUST-NAME TO S-NAME.
83        MOVE CUST-ADDRESS TO S-ADDRESS.
84        RELEASE SORT-REC.
85        GO TO IN-1.
86    IN-2.
87        CLOSE CARD-FILE.
88    OUT-1 SECTION.
89    OT-1.
90        WRITE PRINTLINE FROM OUT-LINE-1.
91        MOVE SPACES TO PRINTLINE.
92        WRITE PRINTLINE
93            BEFORE ADVANCING 2 LINES.
```

Figure 8-7. Sample Sort Program (Sheet 2 of 3)

```
 94   OT-2.
 95       RETURN SORT-FILE RECORD INTO TEMP-REC
 96           AT END GO TO OT-3.
 97       MOVE T-NAME TO NAME-OUT.
 98       MOVE T-STREET TO STREET-OUT.
 99       MOVE T-CITY TO CITY-OUT.
100       MOVE T-ZIP TO ZIP-OUT.
101       WRITE PRINTLINE FROM OUT-LINE-2.
102       ADD 1 TO CNTR.
103       GO TO OT-2.
104   OT-3.
105       CLOSE PRINT-FILE.
106   END-SORT SECTION.
107   CLOSING.
108       DISPLAY SPACES.
109       DISPLAY " CUSTOMER LIST CONTAINS " CNTR " NAMES".
110       STOP RUN.
111   ERROR-1.
112       DISPLAY " BAD INPUT DECK ".
113       STOP RUN.
```

Figure 8-7. Sample Sort Program (Sheet 3 of 3)

Column 1 ↓ Column 24 ↓ Column 70 ↓ 01/31/76 Column 80 ↓

```
259ABC DISTRIBUTORS    5820 MARKET ST       ARCADIA, CA          91006          A
029ACME DISTRIBUTING   9802 VENICE BLVD     MAR VISTA, CA        90066          A
046JONES COMPANY       4156 WARNER AVE      CULVER CITY, CA      90230          A
44619   04683   55025   17802
089PREMIUM PRODUCTS    3691 SPRING ST       LOS ANGELES, CA      90012          A
73294   50721   64325   73815
134XYZ COMPANY         7708 WILSHIRE BLVD   LOS ANGELES, CA      90046          A
04513   97625   43581   44300
178MASON MERCHANDISERS 2764 ROSECRANS AVE   HAWTHORNE, CA        90250          A
88303   46165   73259   90707
263COURTESY SALES CORP 2700 W MAGNOLIA BLVDBURBANK, CA           91506          A
33450   79165   11950   73259
339RETAILERS INC       14391 E BROADWAY     WHITTIER, CA         90604          A
22870   94059   61667   53123
372SMITH AND SONS      1163 N ANAHEIM BLVD  ANAHEIM, CA          92801          A
99735   82701   00567   39462
428WORLD SALES COMPANY 3930 LANKERSHIM BLVDNORTH HOLLYWOOD, CA   91604          A
90921   77345   64521   50640
485DAY AND NIGHT INC   8529 BELLFLOWER AVE  BELLFLOWER, CA       90706          A
5110AKVILLE CORP       1744 LINCOLN BLVD    SANTA MONICA, CA     90404          A
76456   28904   20164   65077
620SELECT SALES COMPANY9635 SANTA MONICA BLBEVERLY HILLS, CA     90210          A
644YOUNG AND YOUNG     20125 DEVONSHIRE     CHATSWORTH, CA       91311          A
97894   35610   27059   08431
656QUALITY SALES CO    1276 W VICTORY BLVD  BURBANK, CA          91502          A
729DARRELL BROTHERS    5509 WESTMINSTER BL  SANTA ANA, CA        92703          A
45612   64302   50189   79532
747MERCHANTS INC       2268 E ORANGETHORPE  ANAHEIM, CA          92806          A
67943   52146   76285   90431
788IDEAL SALES COMPANY 7125 SEPULVEDA BLVD  VAN NUYS, CA         91405          A
805HOUSEHOLD PRODUCTS  802 N LA BREA AVE    INGLEWOOD, CA        90302          A
73158   62490   05137   44630
846EXECUTIVE SALES INC 5893 S FIGUEROA ST   LOS ANGELES, CA      90003          A
863ROYAL SALES COMPANY 11601 PARAMOUNT      DOWNEY, CA           90241          A
929MORGAN BROTHERS INC 8523 W OLYMPIC BLVD  LOS ANGELES, CA      90035          A
951A-1 PRODUCTS        16053 S CRENSHAW BL  TORRANCE, CA         90506          A
976INTERNATIONAL SALES 1049 ATLANTIC BLVD   ALHAMBRA, CA         91803          A
210MI CORPORATION      1732 CALIFORNIA AVE  LONG BEACH, CA       90813          A
```

Figure 8-8. Input Data for Sample Sort Program

```
                        CUSTOMER LIST ON 01/31/76

    A-1 PRODUCTS              16053 S CRENSHAW BL      TORRANCE, CA          90506
    ABC DISTRIBUTORS          5820 MARKET ST           ARCADIA, CA           91006
    ACME DISTRIBUTING         9802 VENICE BLVD         MAR VISTA, CA         90066
    COURTESY SALES CORP       2700 W MAGNOLIA BLVD     BURBANK, CA           91506
    DARRELL BROTHERS          5509 WESTMINSTER BL      SANTA ANA, CA         92703
    DAY AND NIGHT INC         8529 BELLFLOWER AVE      BELLFLOWER, CA        90706
    EXECUTIVE SALES INC       5893 S FIGUEROA ST       LOS ANGELES, CA       90003
    HOUSEHOLD PRODUCTS        802 N LA BREA AVE        INGLEWOOD, CA         90302
    IDEAL SALES COMPANY       7125 SEPULVEDA BLVD      VAN NUYS, CA          91405
    INTERNATIONAL SALES       1049 ATLANTIC BLVD       ALHAMBRA, CA          91803
    JONES COMPANY             4156 WARNER AVE          CULVER CITY, CA       90230
    MASON MERCHANDISERS       2764 ROSECRANS AVE       HAWTHORNE, CA         90250
    MERCHANTS INC             2268 E ORANGETHORPE      ANAHEIM, CA           92806
    MI CORPORATION            1732 CALIFORNIA AVE      LONG BEACH, CA        90813
    MORGAN BROTHERS INC       8523 W OLYMPIC BLVD      LOS ANGELES, CA       90035
    OAKVILLE CORP             1744 LINCOLN BLVD        SANTA MONICA, CA      90404
    PREMIUM PRODUCTS          3691 SPRING ST           LOS ANGELES, CA       90012
    QUALITY SALES CO          1276 W VICTORY BLVD      BURBANK, CA           91502
    RETAILERS INC             14391 E BROADWAY         WHITTIER, CA          90604
    ROYAL SALES COMPANY       11601 PARAMOUNT          DOWNEY, CA            90241
    SELECT SALES COMPANY      9635 SANTA MONICA BL     BEVERLY HILLS, CA     90210
    SMITH AND SONS            1163 N ANAHEIM BLVD      ANAHEIM, CA           92801
    WORLD SALES COMPANY       3930 LANKERSHIM BLVD     NORTH HOLLYWOOD, CA   91604
    XYZ COMPANY               7708 WILSHIRE BLVD       LOS ANGELES, CA       90046
    YOUNG AND YOUNG           20125 DEVONSHIRE         CHATSWORTH, CA        91311

  CUSTOMER LIST CONTAINS  25 NAMES
```

Figure 8-9. Output Report From Sample Sort Program

A. Program Listing

```
IDENTIFICATION DIVISION.
PROGRAM-ID.    MERGE-FILES.
ENVIRONMENT DIVISION.
INPUT-OUTPUT SECTION.
FILE-CONTROL.
    SELECT IFILE1 ASSIGN TO INP
           USE "RT=Z"
    .
    SELECT IFILE ASSIGN TO INO
           USE "RT=Z"
    .
    SELECT MFILE ASSIGN TO M
    .
    SELECT OFILE ASSIGN TO OT
           USE "RT=Z"
    .
DATA DIVISION.
FILE SECTION.
FD IFILE1 LABEL RECORDS OMITTED.
01 IREC1 PIC X(30).
FD IFILE LABEL RECORDS OMITTED.
01 IREC PIC X(30).
SD MFILE.
01 MREC.
    02 MKEY PIC X(5).
    02     PIC X(25).
FD OFILE LABEL RECORDS OMITTED.
01 OREC  PIC X(30).
PROCEDURE DIVISION.
STRT.
    MERGE MFILE ON ASCENDING KEY MKEY
          USING IFILE, IFILE1
          GIVING OFILE.
    STOP RUN.
```

B. Program Input

FILE INP:

```
00010AAAAAAAAAAAAAAAAAAAAAAAA
00015BBBBBBBBBBBBBBBBBBBBBBBB
00020CCCCCCCCCCCCCCCCCCCCCCCC
```

FILE INO:

```
00009XXXXXXXXXXXXXXXXXXXXXXXX
00014YYYYYYYYYYYYYYYYYYYYYYYY
00017ZZZZZZZZZZZZZZZZZZZZZZZZ
```

C. Program Output: Merged file OT

```
00009XXXXXXXXXXXXXXXXXXXXXXXX
00010AAAAAAAAAAAAAAAAAAAAAAAA
00014YYYYYYYYYYYYYYYYYYYYYYYY
00015BBBBBBBBBBBBBBBBBBBBBBBB
00017ZZZZZZZZZZZZZZZZZZZZZZZZ
00020CCCCCCCCCCCCCCCCCCCCCCCC
```

Figure 8-10. Sample Merge Program

When a COBOL 5 program is executing, segmentation provides a way to conserve memory space by overlaying sections of the program in central memory. The entire Procedure Division in a segmented program is written in sections that are separated into fixed and independent segments. The segment number assigned to a section determines whether it is a fixed or an independent segment.

If any reordering of the object program is required to handle the flow from segment to segment, the compiler provides the control transfers to maintain the logic flow specified in the source program. An automatic jump is made when the succeeding section is in a different segment. Control can be transferred to any paragraph in a section; it is not mandatory to transfer control to the beginning of a section.

The use of the Fast Dynamic Loader (FDL) can serve as an alternative to segmentation (see section 10).

TYPES OF SEGMENTS

A segment of the Procedure Division is either a fixed segment or an independent segment. The segment number designates the type of segment; segment numbers 0 through 49 identify fixed segments, and numbers 50 through 99 identify independent segments. Sections with the same segment number are part of the same segment.

FIXED SEGMENTS

Fixed segments are logically treated as if they are always in memory; therefore, those segments should contain the sections that are referenced most frequently. A fixed segment is either permanent or overlayable.

A fixed permanent segment is always available for reference. It cannot be overlaid by another segment. Sections that must be available at all times are included in the fixed permanent segments.

A fixed overlayable segment can overlay and can be overlaid by another overlayable segment (fixed or independent). If it is overlaid, it is returned in its last used state; last used state refers to modified GO TO statements.

The range of segment numbers for fixed segments (0 through 49) is divided into numbers that define permanent segments and numbers that define overlayable segments. The SEGMENT-LIMIT clause in the Environment Division specifies the lowest number that can be used for fixed overlayable segments. Segment numbers up to, but not including, the specified number are used for fixed permanent segments; segment numbers from the specified number up to and including 49 are used for fixed overlayable segments. When the SEGMENT-LIMIT clause is not specified, all segments with segment numbers 0 through 49 are treated as fixed permanent segments.

INDEPENDENT SEGMENTS

Independent segments, which are identified by segment numbers 50 through 99, are considered overlayable. These segments can overlay and can be overlaid by another overlayable segment (fixed or independent). In future standards, independent segments might be deleted; therefore, it is best to avoid using independent segments, if possible.

When control is transferred to an independent segment, it is in its initial state only under the following conditions:

- Execution of a PERFORM, USE, SORT, MERGE, or GO TO statement transfers control to the independent segment.
- Execution of the last statement in the preceding segment implicitly transfers control to the independent segment.

An independent segment is in its last used state when it receives control under any other circumstances. Initial state and last used state refer to modified GO TO statements.

OVERLAYS

Both fixed overlayable segments and independent segments are called overlays. Only one overlay can be present in memory with the fixed permanent segments. As each overlay is referenced, it is loaded into memory over the last overlay. This conserves memory because it need be only as large as the fixed permanent segments plus the largest overlay.

If the BBH field of the file information table (FIT) is set to YES through the USE clause or through a FILE control statement, the associated file must be opened and closed with no intervening call to a fixed overlayable or independent segment.

SUBPROGRAMS AND OVERLAYS

Subprograms should not be written in segments because they are not actually broken up into segments; therefore, no advantage in memory space is gained and some is lost.

A subprogram can be called from a segmented main program. If the statement calling the subprogram is in an overlayable segment and it specifies a literal that is not a program-name in the FDL file, the subprogram must reside on a user library. The user library must be made available to the job before the main (calling) program is loaded. The subprogram is initialized each time the overlay is loaded. To prevent this, the subprogram must be loaded with the fixed permanent segments. An unexecuted ENTER or CALL statement in a fixed permanent segment or an LDSET,USE control statement can be used to load the subprogram with the fixed permanent segments.

STRUCTURING SEGMENTS

The Procedure Division of a segmented program is organized into sections. Each section is assigned to a specific segment and is part of either a fixed segment or an independent segment. More than one section can be included in a segment.

Segmentation does not affect the logical sequence of the program as specified in the source program. Control can be transferred to any paragraph in a segment; it need not be transferred to the beginning of a section.

A section is assigned to a segment by specifying a segment number in the section header. The segment number further identifies the segment as a fixed or independent segment. Fixed segments are assigned segment numbers 0 through 49; independent segments are assigned segment numbers 50 through 99.

```
READ-INPUT SECTION 4.
.
.
.
UPDATE-MASTER SECTION 53.
```

The first section header specifies that the section named READ-INPUT is assigned to segment number 4, which is a fixed segment. The section named UPDATE-MASTER is assigned to segment number 53, which is an independent segment.

The SEGMENT-LIMIT clause is used to divide the fixed segment numbers (0 through 49) into permanent and overlayable segments. This clause is specified in the OBJECT-COMPUTER paragraph of the Environment Division. The designated number is the lowest segment number for fixed overlayable segments. Segment numbers that are less than the designated number identify fixed permanent segments.

```
SEGMENT-LIMIT IS 45
```

This clause specifies that segment numbers 0 through 44 are used for fixed permanent segments. Segment numbers 45 through 49 are used for fixed overlayable segments.

When segment numbers are being assigned, consideration should be given to the frequency with which the section is used.

- Sections that must be available at all times should be assigned to fixed permanent segments.
- Frequently referenced sections should be assigned to fixed permanent segments.
- Sections that frequently reference each other should be assigned to the same segment.
- Declarative sections must be assigned to permanent segments.

The coding requirements for a segmented program are summarized in the following rules:

- Overlay segments (fixed or independent) are made available when:

  A PERFORM statement references a procedure within the segment.

  A GO TO statement references a procedure within the segment.

  The segment logically follows the previously executed statement.

  A SORT or MERGE statement references a procedure within the segment.

- A PERFORM statement in a fixed segment can reference one of the following:

  Procedures in fixed segments (permanent or overlayable).

  Procedures in any one independent segment.

- A PERFORM statement in an independent segment can reference one of the following:

  Procedures in fixed segments (permanent or overlayable).

  Procedures in the independent segment that contains the PERFORM statement.

- An ALTER statement that changes the procedure-name for a GO TO statement to a procedure in an independent segment must be within that independent segment.

- A SORT or MERGE statement in a fixed segment can reference one of the following:

  Input or output procedures in fixed segments (permanent or overlayable).

  Input or output procedures in any one independent segment.

- A SORT or MERGE statement in an independent segment can reference one of the following:

  Input or output procedures in fixed segments (permanent or overlayable).

  Input or output procedures in the independent segment that contains the SORT or MERGE statement.

An independently compiled subprogram can be used within the structure of a COBOL 5 source program. The subprogram can be written in COBOL, COMPASS, or FORTRAN and is compiled and tested as an independent program. Two typical instances in which a subprogram could be used are:

- A subroutine written in one of the acceptable languages has already been coded, compiled, and tested.
- A square root is needed and can be calculated more efficiently with a FORTRAN routine.

Control is transferred to a subprogram when an ENTER or CALL statement in the main program is executed. The ENTER statement is used for subprograms that are not written in COBOL. For COBOL subprograms, the CALL statement is specified. (Mode errors might occur if CALL is used with FORTRAN subprograms.)

Data can be passed between the main program and the subprogram through the USING phrase of the ENTER or CALL statement. The Common-Storage Section can also be used for passing data. External files and the CYBER Database control system (CDCS) data base files can be shared between programs.

Fast Dynamic Loader (FDL) processing allows COBOL subprograms to be dynamically loaded and unloaded during execution of the main program. Fast Dynamic Loader is not required for subprograms to access data base files under CDCS 2.

TRANSFERRING CONTROL TO A SUBPROGRAM

Depending on the language used to write the subprogram, either the ENTER statement or the CALL statement is specified in the COBOL 5 program (main program). When the statement is executed, control is transferred to the specified subprogram. Control returns to the main program at the statement immediately following the ENTER or CALL statement.

ENTERING NON-COBOL SUBPROGRAMS

A subprogram written in COMPASS or FORTRAN can be entered from the COBOL 5 main program to perform specific functions. Common data can be passed between the main program and the subprogram through the USING phrase of the ENTER statement or the Common-Storage Section in the Data Division.

The ENTER statement is specified in the main program to transfer control to a non-COBOL subprogram. This statement specifies the language of the subprogram and the entry point into the subprogram. COMPASS is the default. If COMPASS or FTN 5 is not specified, COMPASS is assumed. When the ENTER statement is executed, control is transferred to the subprogram at the designated entry point. The entry point is defined in the subprogram; it can be a program name, subroutine name, function name, or statement label. The entry point must be specified as a nonnumeric literal if it contains characters other than letters and digits or if it duplicates a COBOL reserved word.

```
ENTER COMPASS TEST1.
```

This statement specifies that the subprogram is written in COMPASS. The entry point into the subprogram is named TEST1.

The USING phrase is included in the ENTER statement to designate data that is common to the main program and the subprogram. The parameter list specified in this phrase can include data-names, file-names, procedure-names, and literals.

```
ENTER FTN5 HYP USING LEG1,
    LEG2, DIAG.
```

When this statement is executed, the FORTRAN subprogram is entered at the entry point named HYP. The data items LEG1, LEG2, and DIAG are shared between the main program and the subprogram.

When a FORTRAN subprogram is entered, best results are obtained by passing the following types of data items:

- An item described as COMPUTATIONAL-1
- An item described as COMPUTATIONAL-2
- A level 01 item that is a multiple of 10 characters in length
- Literals

COMPUTATIONAL-1 and COMPUTATIONAL-2 data items in COBOL 5 correspond to integer and real items, respectively, in FORTRAN. Data items with any other COBOL usage are treated as character strings. A numeric literal without a decimal is treated as an integer item in FORTRAN. A numeric literal with a decimal is treated as a real item. An alphanumeric literal is treated as a character string.

Data to be passed between the main program and the subprogram can also be specified in the Common-Storage Section. The data is allocated to a labeled common block named CCOMMON. The FORTRAN or COMPASS subprogram accesses this data by referencing the common block CCOMMON.

CALLING COBOL SUBPROGRAMS

A subprogram written in COBOL is called into execution by a CALL statement in the COBOL main program. Data to be passed between the main program and the subprogram can be specified in the CALL statement or it can be passed through the Common-Storage Section. External files and data base files can also be shared between the programs.

The CALL statement in the main program specifies the program name from the PROGRAM-ID paragraph in the subprogram. The program name is specified as a nonnumeric literal of no more than seven characters when

the FDL parameter is not included in the COBOL5 control statement. When the FDL parameter is specified, the program name can be contained in an alphanumeric data item and can be up to 30 characters in length. FDL processing is discussed later in this section.

```
CALL "COMPAY".
```

Execution of this statement transfers control to the COBOL subprogram named COMPAY. The program name is specified as a nonnumeric literal and must be enclosed in quotation marks. The program name must begin with an alphabetic character and must not contain a space or a hyphen.

The subprogram can also contain CALL statements; however, it must not call the main program or call another subprogram that calls the main program. When the subprogram is called, all data fields and alterable switches are the same as when the subprogram was last exited. The status and positioning of all local files used by the subprogram are also unchanged; however, External files might have been changed by other programs.

Data items that are common to the main program and the subprogram are specified in one of two ways:

- The USING phrase can be included in the CALL statement to name the common data items.
- The Common-Storage Section can be used to describe the common data items.

Sharing External files and data base files is discussed later in this section.

The USING phrase of the CALL statement specifies the data items that are to be passed between the main program and the subprogram. Each data item specified must be a level 01 or level 77 item that is described in either the File Section or the Working-Storage Section of the main program. If the subprogram is called by another subprogram, the data items can also be described in the Linkage Section of the calling program. When the USING phrase is specified in the CALL statement to identify common data, the subprogram must identify the common data items in the Procedure Division header and describe the data items in the Linkage Section of the Data Division. The data-names in the CALL statement correspond by position to the data-names in the Procedure Division header; therefore, the names can differ while the number of names must be identical. Both data items in each corresponding pair should be defined at the same level (level 01 or level 77) to ensure proper synchronization. If items of different levels are in corresponding positions, the ANSI=77LEFT parameter must be specified in the COBOL5 control statement. When the parameter is specified for the main program, it should be specified for all subprograms to avoid conflicts between level 77 items.

```
CALL "SUBPRO3" USING SUB-REC.
```

When this statement is executed, control is transferred to the COBOL subprogram named SUBPRO3. The data item SUB-REC, which is described in the Working-Storage Section of the main program as a level 01 data item, is made available to the subprogram. Items subordinate to SUB-REC pass values to be used by the subprogram and receive values determined through execution of the subprogram.

Data to be passed between the main program and the COBOL subprogram can be specified in the Common-Storage Section of the Data Division. This section, in both the main program and the subprogram, describes the shared data. The data-names and descriptions need not be the same in both sections; however, the data in each section must be identical. For example, a table can be fully described in one Common-Storage Section and be described with an OCCURS clause in the other Common-Storage Section; the complete size of the table must be the same in both sections.

The initial value of a data item shared between the main program and the COBOL subprograms can be set only in the main program by the VALUE clause; the initial value cannot be set in the subprograms.

SHARING FILES

Files declared in the main program can be shared with any subprogram in the same run unit. Various types of files can be shared between programs. External files and data base files are two special types of files with particular requirements and considerations that are discussed in the following paragraphs. Any other type of file that is shared must be described in each subprogram that references it. The file must be closed before the subprogram is exited so that another program using the file can open it for processing. This method of sharing files should be avoided; External files provide more efficient processing.

EXTERNAL FILES

External files can be referenced by any program in the run unit. File information exists external to the programs. The record areas are shared in the same manner as Common-Storage Section items are shared. External files do not have to be closed by one program and reopened by another program processing the files.

All External files must be declared in the main program. The File Description (FD) entry for an External file includes the EXTERNAL clause. The following restrictions apply to an External file:

- Data items specified in the File-Control and FD entries must be defined in the Common-Storage Section. (This includes data-names in clauses such as the BLOCK COUNT and RECORD KEY clause, but does not include the Record Description entries of levels 01 and subordinate entries.)
- The LABEL RECORDS clause in the FD entry must specify OMITTED.
- The Report Writer feature cannot be used.
- The RERUN and SAME AREA clauses in the I-O-CONTROL paragraph cannot be specified.

A subprogram describes only those External files it references. The file description must be exactly the same as it is in the main program; therefore, COPY statements or UPDATE common decks should be used for External file descriptions.

DATA BASE FILES

Data base files are accessed through the CYBER Database Control System (CDCS). The interface with CDCS is described in detail in section 14. When data base files are shared between programs in a run unit, the main program specifies the name of the subschema describing the data base files in the SUB-SCHEMA clause in the SPECIAL-NAMES paragraph. A subprogram that accesses at least one data base file must also include the SUB-SCHEMA clause in the SPECIAL-NAMES paragraph. There are no other requirements.

PROCESSING WITH FAST DYNAMIC LOADER

Fast Dynamic Loader (FDL) processing provides additional capabilities during execution of COBOL subprograms. The information on the FDL file, which must be created before FDL processing can be used, allows the main program and any subprogram in the same run unit to perform operations related to the usage of program names.

PROGRAM NAME USAGE

The usage of program names affects the CALL statement and allows the CANCEL statement to be executed for dynamic subprograms. With FDL processing, program names can be up to 30 characters in length. The CALL and CANCEL statements can also specify a data item that contains the program name instead of specifying a literal. The FDL file indicates whether a subprogram is static or dynamic. All static subprograms are loaded with the base module. A dynamic subprogram is not loaded until a CALL statement for that subprogram is executed. After the subprogram has been executed, the CANCEL statement can be used to release the memory space occupied by the subprogram.

For a dynamic subprogram, the ON OVERFLOW phrase can be included in the CALL statement. This phrase specifies a statement that is executed when there is not enough room to load the dynamic subprogram. This could occur if the maximum field length for the job or the field length specified in the job statement is reached. If the ON OVERFLOW phrase is not specified and an overflow condition occurs, the run is aborted. The phrase is ignored if it is specified for a static program.

```
CALL "DEDUCTIONS"
    USING DEDUCT-REC
    ON OVERFLOW GO TO CANT-LOAD.
```

The dynamic subprogram DEDUCTIONS is called for the first time. If the subprogram cannot be loaded within the program field length, control is transferred to the paragraph named CANT-LOAD.

FDL FILE CREATION

The FDL file, which must be made available to the COBOL 5 compiler to initiate Fast Dynamic Loader processing, consists of a series of card images containing FDL processing information. The Program Equivalence section of the file contains statements related to program name usage.

A program equivalence statement equates the program name from the PROGRAM-ID paragraph of a subprogram with the internal name used by the system. The program name, which is also used in CALL and CANCEL statements, can be up to 30 characters in length. The internal name cannot exceed seven characters and must be unique within the run unit. If the subprogram is static, the key word STATIC must be included in the program equivalence statement; otherwise, the subprogram is considered to be dynamic.

COMPILATION WITH FDL PROCESSING

When COBOL 5 programs that use FDL processing are compiled, the FDL parameter must be specified in the COBOL5 control statement. This parameter designates the FDL file to be used during compilation. The main program and all subprograms are compiled as one compilation job. The programs are input to the compiler in the following order: main program, static subprograms, and dynamic subprograms.

During compilation, overlay capsules are generated for the dynamic subprograms as part of the relocatable binary file. The capsule format allows subprograms to be dynamically loaded and unloaded during execution. Operating system control statements must then be specified to load the binary file. Once the load file is created, the overlay capsules and the main program must be maintained as a unit. The unit can be executed from a sequential file or it can be placed on a user library.

Another subprogram can be compiled and added to the user library. The COBOL5 control statement must include the SB parameter and the FDL parameter, which must specify the same FDL file as when the library was created. The COPYL utility can then be used to replace the subprogram capsule on the original binary file. The load sequence must then be repeated in order to create a new library.

CANCELING A SUBPROGRAM

When Fast Dynamic Loader processing is used, a dynamic subprogram can be canceled. This releases the memory space occupied by the subprogram. Once the subprogram has been called and dynamically loaded, it cannot be canceled until an EXIT PROGRAM statement has been executed.

The CANCEL statement specifies one or more dynamic subprograms to be canceled. A subprogram is indicated by specifying either the program name as a nonnumeric literal or an alphanumeric data item containing the program name. If the subprogram to be canceled is not currently loaded, no action takes place and control is passed to the next executable statement.

```
CANCEL "DEDUCTIONS", PROG-NAME.
```

The subprogram DEDUCTIONS and the subprogram identified by the current contents of the data item PROG-NAME are canceled when this statement is executed. If either subprogram has not been loaded or has already been canceled, no action takes place for that subprogram.

WRITING A COBOL SUBPROGRAM

A COBOL subprogram is written the same as any other COBOL program. When data is shared between the main program and the subprogram, certain requirements must be met. The specific requirements depend on the method used to specify the common data in the main program.

PROCEDURE DIVISION HEADER

The USING phrase must be included in the Procedure Division header when the CALL statement in the main program specifies shared data in a USING phrase. Each data item specified in the main program CALL statement must be referenced in the subprogram Procedure Division header.

The data-names in the subprogram need not match the data-names in the main program; the data items correspond by position in the USING phrases rather than by data-names. The use of the Procedure Division header to identify shared data is shown in the second sample program at the end of this section of the guide.

LINKAGE SECTION

The Linkage Section is included in the subprogram when shared data is specified through the USING phrases in the Procedure Division header of the subprogram and in the CALL statement of the main program. This section describes the common data items for processing by the subprogram.

The data-names specified in the Procedure Division header are described in the Linkage Section as level 77 or level 01 entries. The data descriptions of these data items must match the corresponding data descriptions in the main program. Values cannot be assigned to Linkage Section items; if a VALUE clause is specified, it is ignored and causes a trivial diagnostic to be issued.

Results are unpredictable if a Procedure Division statement references a Linkage Section item that is not specified in the Procedure Division header or is not subordinate to one of those items.

An item defined in the Linkage Section should be moved to a Working-Storage item if it is referenced frequently. This eliminates the additional overhead created by Linkage Section references.

COMMON-STORAGE SECTION

The Common-Storage Section provides the most efficient method for sharing data between programs. This section must be included in the subprogram when it is called by a COBOL 5 program that specifies shared data in the Common-Storage Section. It is also used when the subprogram is called by a FORTRAN or COMPASS program that shares data through the common block CCOMMON. If an External file is used by the subprogram, data items specified in the File-Control entry and the FD entry are defined in the Common-Storage Section. For example, if data-names are used in the BLOCK COUNT clause or FILE STATUS clause, they must be defined in the Common-Storage Section. However, level 01 and subordinate Record Description entries must not be defined in the Common-Storage Section.

Data-names and descriptions in the main program and the subprogram can be different, but the storage allocations must be the same. For example, a table described with the OCCURS clause in the main program can be described as individual elements in the subprogram as long as the table size is not changed. For COBOL 5 programs, it is best to use the COPY statement or UPDATE common decks to ensure that the descriptions are the same.

Level 77 entries and Record Description entries can be specified in the Common-Storage Section. The initial value of a data item can be set by the VALUE clause in the main program but not in the subprogram. If an initial value is specified in the Common-Storage Section of the subprogram, the value is ignored and a warning diagnostic is issued.

RETURN OF CONTROL

The subprogram returns control to the main program when the EXIT PROGRAM statement is executed. Control is returned to the statement immediately following the CALL statement in the main program.

SAMPLE PROGRAMS

Three examples are included in this section to illustrate the use of subprograms. The first example shows a COBOL 5 main program that enters a subprogram written in FORTRAN 5. The second and third examples call a COBOL 5 subprogram; the programs are the same application, but the method of describing shared data differs.

ENTERING A FORTRAN SUBPROGRAM

The use of a FORTRAN subprogram by a COBOL 5 main program is illustrated in figure 10-1. The COBOL 5 program enters the subprogram (which is written in FORTRAN) to determine the diagonal of a right triangle.

The ENTER statement in the COBOL 5 program (line 24) specifies three data items to be used for passing data between the main program and the subprogram. When the ENTER statement is executed, the data items LEG1 and LEG2 contain the triangle dimensions that are passed to the FORTRAN subprogram; the data item DIAG is used to receive the result computed by the subprogram. The subprogram identifies the three data items as A, B, and C, respectively.

CALLING A COBOL SUBPROGRAM

Two different ways to specify the data to be passed between a COBOL 5 main program and a COBOL 5 subprogram are shown in the programs illustrated in figures 10-2 and 10-3. In figure 10-2, the main program specifies the shared data in the CALL statement; the subprogram specifies the data in the USING phrase of the Procedure Division and describes the data in the Linkage Section. The main program and the subprogram in figure 10-3 specify common data in the Common-Storage Section.

COBOL 5 Main Program

```
 1   IDENTIFICATION DIVISION.
 2   PROGRAM-ID.  FIND-DIAGONAL.
 3   ENVIRONMENT DIVISION.
 4   CONFIGURATION SECTION.
 5   SOURCE-COMPUTER.  CYBER-170.
 6   OBJECT-COMPUTER.  CYBER-170.
 7   DATA DIVISION.
 8   WORKING-STORAGE SECTION.
 9   01  DIAG          USAGE IS COMPUTATIONAL-2.
10   01  LEG1          USAGE IS COMPUTATIONAL-2.
11   01  LEG2          USAGE IS COMPUTATIONAL-2.
12   01  DISPLAY-ITEM        PICTURE 9(4).
13   01  CARD-IN.
14       03  AA              PICTURE 9(4).
15       03  BB              PICTURE 9(4).
16       03  FILLER          PICTURE X(72).
17   PROCEDURE DIVISION.
18   START-UP.
19       ACCEPT CARD-IN.
20       DISPLAY "LEG1 = " AA.
21       DISPLAY "LEG2 = " BB.
22       MOVE AA TO LEG1.
23       MOVE BB TO LEG2.
24       ENTER FTN5 HYP USING LEG1, LEG2, DIAG.
25       MOVE DIAG TO DISPLAY-ITEM.
26       DISPLAY "DIAG = " DISPLAY-ITEM.
27       STOP RUN.
```

FORTRAN Subprogram

```
      SUBROUTINE HYP (A,B,C)
      C=SQRT(A*A + B*B)
      RETURN
      END
```

Input

```
00030004
```

Output

```
LEG1 =      3
LEG2 =      4
DIAG =      5
```

Figure 10-1. Entering a FORTRAN Subprogram

In both examples, the same application and the same data-names are used. The main program reads a record from the input file PAY-FILE, accumulates the total of merchandise charges to be deducted, and then calls the subprogram to compute the net pay. The subprogram performs four calculations to determine the net pay and then returns control to the main program. The result of the computations is output by the main program and the process is repeated for the next record in the input file.

The main program in figure 10-2 indicates the shared data in the USING phrase of the CALL statement (lines 54 and 55). Three level 01 data items are specified: INPUT-CARD, OUT-GOING, and WORK-REC. The first two data items are described in the File Section; WORK-REC is described in the Working-Storage Section. The Procedure Division header in the subprogram designates the same three data items by the data-names to be used in the subprogram (line 23). The data items are described as level 01 items in the Linkage Section.

The main program in figure 10-3 specifies the data to be shared in the Common-Storage Section. Eight individual level 01 data items and one group item are described in this section. The subprogram describes the same eight elementary data items and the group item in its Common-Storage Section.

COBOL 5 Main Program

```
IDENTIFICATION DIVISION.
PROGRAM-ID.  PAYROLL.
ENVIRONMENT DIVISION.
CONFIGURATION SECTION.
SOURCE-COMPUTER.  CYBER-170.
OBJECT-COMPUTER.  CYBER-170.
INPUT-OUTPUT SECTION.
FILE-CONTROL.
    SELECT PAY-FILE ASSIGN TO INPUT.
    SELECT PAY-LIST ASSIGN TO OUTPUT.
DATA DIVISION.
FILE SECTION.
FD  PAY-FILE
        LABEL RECORDS ARE OMITTED
        DATA RECORD IS INPUT-CARD.
01  INPUT-CARD.
    03  ID-NUMBER                PICTURE 9(8).
    03  NAME-IN                  PICTURE A(30).
    03  GROSS-PAY                PICTURE 9(5).
    03  NO-OF-CHARGES            PICTURE 9.
    03  MONTHLY-CHARGES          PICTURE 999V99
            OCCURS 7 TIMES.
FD  PAY-LIST
        LABEL RECORDS ARE OMITTED
        DATA RECORD IS OUT-GOING.
01  OUT-GOING.
    03  FILLER                   PICTURE X.
    03  ID-NUMBER                PICTURE 9(8)BB.
    03  NAME-OUT                 PICTURE A(30).
    03  NET-PAY                  PICTURE $$999.99BCR.
WORKING-STORAGE SECTION.
01  I                            PICTURE 9       VALUE 1.
01  WORK-REC.
    03  FICA-RATE                PICTURE V9(4)   VALUE .0300.
    03  FED-RATE                 PICTURE V9(4)   VALUE .0775.
    03  MERCH-DEDUCT             PICTURE 9(4)V99.
    03  TOTAL-DEDUCT             PICTURE 999V99.
    03  FED-DEDUCT               PICTURE 999V99.
    03  FICA-DEDUCT              PICTURE 999V99.
PROCEDURE DIVISION.
INIT SECTION.
INIT-PARA.
    OPEN INPUT PAY-FILE.
    OPEN OUTPUT PAY-LIST.
    MOVE "  NUMBER   NAME                              NET-PAY"
        TO OUT-GOING.
    WRITE OUT-GOING BEFORE ADVANCING 2 LINES.
PROCESS-CARD SECTION.
PROCESS-CARD-PARA.
    READ PAY-FILE RECORD
        AT END GO TO END-OF-PROG.
    COMPUTE MERCH-DEDUCT = 0.
    PERFORM ACCUM-CHARGES NO-OF-CHARGES TIMES.
    CALL "COMPAY"
        USING INPUT-CARD, OUT-GOING, WORK-REC.
    MOVE ID-NUMBER OF INPUT-CARD TO ID-NUMBER OF OUT-GOING.
    MOVE NAME-IN TO NAME-OUT.
    WRITE OUT-GOING AFTER ADVANCING 1 LINE.
    COMPUTE I = 1.
    GO TO PROCESS-CARD.
ACCUM-CHARGES SECTION.
ACCUM-CHARGES-PARA.
    COMPUTE MERCH-DEDUCT = MERCH-DEDUCT + MONTHLY-CHARGES (I).
    COMPUTE I = I + 1.
END-OF-PROG SECTION.
END-OF-PROG-PARA.
    CLOSE PAY-FILE, PAY-LIST.
    STOP RUN.
```

Figure 10-2. Calling a COBOL Subprogram that Uses the Linkage Section (Sheet 1 of 2)

COBOL 5 Subprogram

```
 1   IDENTIFICATION DIVISION.
 2   PROGRAM-ID.  COMPAY.
 3   ENVIRONMENT DIVISION.
 4   CONFIGURATION SECTION.
 5   SOURCE-COMPUTER.  CYBER-170.
 6   OBJECT-COMPUTER.  CYBER-170.
 7   DATA DIVISION.
 8   LINKAGE SECTION.
 9   01  IN-REC.
10       03  FILLER                  PICTURE X(38).
11       03  GROSS-PAY               PICTURE 9(5).
12       03  FILLER                  PICTURE X(36).
13   01  OUT-REC.
14       03  FILLER                  PICTURE X(41).
15       03  NET-PAY-AMT             PICTURE $$999.99BCR.
16   01  WORK-REC.
17       03  FICA-RATE               PICTURE V9(4).
18       03  RATE-FED                PICTURE V9(4).
19       03  MERCH-DEDUCT            PICTURE 9(4)V99.
20       03  TOTAL-DEDUCT            PICTURE 999V99.
21       03  DEDUCT-FED              PICTURE 999V99.
22       03  FICA-DEDUCT             PICTURE 999V99.
23   PROCEDURE DIVISION USING IN-REC, OUT-REC, WORK-REC.
24   PARA.
25       COMPUTE DEDUCT-FED = RATE-FED * GROSS-PAY.
26       COMPUTE FICA-DEDUCT = FICA-RATE * GROSS-PAY.
27       COMPUTE TOTAL-DEDUCT = DEDUCT-FED + FICA-DEDUCT +
28           MERCH-DEDUCT.
29       COMPUTE NET-PAY-AMT = GROSS-PAY - TOTAL-DEDUCT.
30   GO-BACK.
31       EXIT PROGRAM.
```

Input

```
Column 1                                  Column 39
↓                                         ↓
12345678JOHN SMITH                        00500000000000000000000000000000000000000000
50449786WILLIAM JOHNSON                   00580703575009250045500820021000069500510
98765432PAUL JONES                        00400202460005200000000000000000000000000000
41639890ROBERT CARRINGTON                 00475401050016100048000615000000000000000
39065152CHARLES RUTHERFORD                00615600380002900124502035009650115000000
```

Output

```
 NUMBER    NAME                           NET-PAY

12345678   JOHN SMITH                     $446.25
50449786   WILLIAM JOHNSON                $426.85
98765432   PAUL JONES                     $327.20
41639890   ROBERT CARRINGTON              $386.39
39065152   CHARLES RUTHERFORD             $488.24
```

Figure 10-2. Calling a COBOL Subprogram that Uses the Linkage Section (Sheet 2 of 2)

COBOL 5 Main Program

```
1    IDENTIFICATION DIVISION.
2    PROGRAM-ID.  PAYROLL.
3    ENVIRONMENT DIVISION.
4    CONFIGURATION SECTION.
5    SOURCE-COMPUTER.  CYBER-170.
6    OBJECT-COMPUTER.  CYBER-170.
7    INPUT-OUTPUT SECTION.
8    FILE-CONTROL.
9        SELECT PAY-FILE ASSIGN TO INPUT.
10       SELECT PAY-LIST ASSIGN TO OUTPUT.
11   DATA DIVISION.
12   FILE SECTION.
13   FD  PAY-FILE
14           LABEL RECORDS ARE OMITTED
15           DATA RECORD IS INPUT-CARD.
16   01  INPUT-CARD.
17       03  ID-NUMBER                PICTURE 9(8).
18       03  NAME-IN                  PICTURE X(30).
19       03  GROSS-PAY                PICTURE 9(5).
20       03  NO-OF-CHARGES            PICTURE 9.
21       03  MONTHLY-CHARGES          PICTURE 999V99
22               OCCURS 7 TIMES.
23   FD  PAY-LIST
24           LABEL RECORDS ARE OMITTED
25           DATA RECORD IS OUT-GOING.
26   01  OUT-GOING.
27       03  FILLER                   PICTURE X.
28       03  ID-NUMBER                PICTURE 9(8)BB.
29       03  NAME-OUT                 PICTURE X(30).
30       03  NET-PAY                  PICTURE $$999.99BCR.
31   COMMON-STORAGE SECTION.
32   01  I                            PICTURE 9       VALUE 1.
33   01  FICA-RATE                    PICTURE V9(4)   VALUE .0300.
34   01  FED-RATE                     PICTURE V9(4)   VALUE .0775.
35   01  MERCH-DEDUCT                 PICTURE 9(4)V99 VALUE 0.
36   01  TOTAL-DEDUCT                 PICTURE 999V99.
37   01  FED-DEDUCT                   PICTURE 999V99.
38   01  FICA-DEDUCT                  PICTURE 999V99.
39   01  NET-PAY-AMT                  PICTURE 9(4)V99.
40   01  IN-REC.
41       03                           PICTURE X(38).
42       03  GROSS-IN                 PICTURE 9(5).
43       03                           PICTURE X(36).
44   PROCEDURE DIVISION.
45   INIT SECTION.
46   INIT-PARA.
47       OPEN INPUT PAY-FILE.
48       OPEN OUTPUT PAY-LIST.
49       MOVE "  NUMBER   NAME                                NET-PAY"
50           TO OUT-GOING.
51       WRITE OUT-GOING BEFORE ADVANCING 2 LINES.
52   PROCESS-CARD SECTION.
53   PROCESS-CARD-PARA.
54       READ PAY-FILE RECORD INTO IN-REC
55           AT END GO TO END-OF-PROG.
56       COMPUTE MERCH-DEDUCT = 0.
57       PERFORM NO-OF-CHARGES TIMES
58           COMPUTE MERCH-DEDUCT = MERCH-DEDUCT + MONTHLY-CHARGES (I)
59           COMPUTE I = I + 1
60       END-PERFORM.
61       CALL "COMPAY"
62       MOVE ID-NUMBER OF INPUT-CARD TO ID-NUMBER OF OUT-GOING.
63       MOVE NAME-IN TO NAME-OUT.
64       MOVE NET-PAY-AMT TO NET-PAY.
65       WRITE OUT-GOING AFTER ADVANCING 1 LINE.
66       COMPUTE I = 1.
```

Figure 10-3. Calling a COBOL Subprogram that Uses the Common-Storage Section (Sheet 1 of 2)

COBOL 5 Main Program (Cont'd)

```
67          GO TO PROCESS-CARD.
68      END-OF-PROG SECTION.
69      END-OF-PROG-PARA.
70          CLOSE PAY-FILE, PAY-LIST.
71          STOP RUN.
```

COBOL 5 Subprogram

```
1       IDENTIFICATION DIVISION.
2       PROGRAM-ID.  COMPAY.
3       ENVIRONMENT DIVISION.
4       CONFIGURATION SECTION.
5       SOURCE-COMPUTER.  CYBER-170.
6       OBJECT-COMPUTER.  CYBER-170.
7       DATA DIVISION.
8       COMMON-STORAGE SECTION.
9       01  I                              PICTURE 9.
10      01  FICA-RATE                      PICTURE V9(4).
11      01  RATE-FED                       PICTURE V9(4).
12      01  MERCH-DEDUCT                   PICTURE 9(4)V99.
13      01  TOTAL-DEDUCT                   PICTURE 999V99.
14      01  DEDUCT-FED                     PICTURE 999V99.
15      01  FICA-DEDUCT                    PICTURE 999V99.
16      01  NET-PAY-AMT                    PICTURE 9(4)V99.
17      01  IN-REC.
18          03                              PICTURE X(38).
19          03  GROSS-PAY                  PICTURE 9(5).
20          03                              PICTURE X(36).
21      PROCEDURE DIVISION.
22      PARA.
23          COMPUTE DEDUCT-FED = RATE-FED * GROSS-PAY.
24          COMPUTE FICA-DEDUCT = FICA-RATE * GROSS-PAY.
25          COMPUTE TOTAL-DEDUCT = DEDUCT-FED + FICA-DEDUCT +
26              MERCH-DEDUCT.
27          COMPUTE NET-PAY-AMT = GROSS-PAY - TOTAL-DEDUCT.
28      GO-BACK.
29          EXIT PROGRAM.
```

Input

```
Column 1                                Column 39
↓                                       ↓
12345678JOHN SMITH                      00500000000000000000000000000000000000000000
50449786WILLIAM JOHNSON                 00580703575009250045500820021000069500510
98765432PAUL JONES                      00400202460005200000000000000000000000000000
41639890ROBERT CARRINGTON               00475401050016100048000615000000000000000
39065152CHARLES RUTHERFORD              00615600380002900124502035009650115000000
```

Output

```
 NUMBER    NAME                          NET-PAY

12345678   JOHN SMITH                    $446.25
50449786   WILLIAM JOHNSON               $426.85
98765432   PAUL JONES                    $327.20
41639890   ROBERT CARRINGTON             $386.39
39065152   CHARLES RUTHERFORD            $488.24
```

Figure 10-3. Calling a COBOL Subprogram that Uses the Common-Storage Section (Sheet 2 of 2)

A COBOL 5 source program is coded on standard COBOL coding sheets according to the applicable format specifications. Coded information is punched on 80-column cards or entered through a terminal. The resulting source program is input to the COBOL 5 compiler to compile it into executable code.

Compilation and subsequent execution of a COBOL 5 program are controlled by a set of control statements preceding the source program. The control statements provide information for the operating system and for the COBOL 5 compiler. The set of control statements begins with a job statement and ends with a 7/8/9 card or its equivalent.

COMPILING A PROGRAM

Source program compilation is controlled by the COBOL5 control statement. Optional parameters can be specified in this statement to designate the files to be used for input and output and the compiler options to be selected for compilation of the source program. The format and number of control statements preceding the source program depend on the operating system and the specific requirements of the program. The input file can contain more than one source program to be compiled; the COBOL 5 compiler recognizes the Identification Division header as the start of a new source program. Compilation can be followed immediately by execution of the object program.

COBOL5 CONTROL STATEMENT

The COBOL5 control statement calls the COBOL 5 compiler and provides information related to compilation of the source program. This statement must be included in the set of control statements preceding the source program to be compiled.

The control statement consists of the word COBOL5 optionally followed by a parameter list. The word COBOL5 and the specified parameters are separated by any valid control statement separator. The complete control statement is terminated by either a period or a right parenthesis. The control statement cannot be continued from one card or card image to another.

A wide range of compiler options can be specified in the parameter list. Each option is identified by a parameter-name. The option is selected by specifying the parameter-name; in some instances, the parameter-name is followed by an equal sign and a value. Parameters can be specified in any order; any parameter can be omitted. System default values are in effect for omitted parameters. The default values discussed in this guide are the system release values; default values might be changed by individual installations.

Input/Output File Parameters

Four input/output files are used during compilation of a source program: the input source file, the output binary file, the output error file, and the output listing file. One additional file, the update file, is used when the program is to be stored in a COBOL 5 source library using the UPDATE utility program. Any of these files can be specified in the COBOL5 control statement.

The input file containing the source program is specified by the I parameter. If this parameter is omitted, the source program must reside on the file INPUT. The I parameter is interpreted as follows:

omitted	Source program resides on the system file INPUT.
I	Source program resides on the file COMPILE.
I=lfn	Source program resides on the file with the designated logical file name.

The file on which the binary output from compilation is written is specified by the B parameter. If this parameter is omitted, the binary output is written on the file LGO. The B parameter is interpreted as follows:

omitted	Binary output is written on the system file LGO.
B	Binary output is written on the file BIN.
B=0	No binary output is produced.
B=lfn	Binary output is written on the file with the designated logical file name.

The listing file is specified by the L parameter. This file contains the source listing, diagnostics, and any other listings selected in the COBOL5 control statement. The L parameter is interpreted as follows:

omitted	Source listing, diagnostics, and selected listings are written on the system file OUTPUT.
L	Source listing, diagnostics, and selected listings are written on the file LIST.
L=lfn	Source listing, diagnostics, and selected listings are written on the file with the designated logical file name.
L=0	No listing is produced.

The output error file, which is specified by the E parameter, contains information related to errors encountered during compilation. If the error file is the same as the listing file, error information is written only on the listing file. If the error file is not the same as the listing file and full listing is selected by the LO (listing options) parameter, error information is written on both files. The E parameter is interpreted as follows:

omitted	Error information is written on the system file OUTPUT.
E=0	Error information is written on the system file OUTPUT.

E	Error information is written on the file ERRS.
E=lfn	Error information is written on the file with the designated logical file name.

The update file is specified by the U parameter. COMPASS line images of the generated program are written on this file in a format acceptable to the UPDATE utility program. The first image written on the file is a DECK directive with the first seven characters from the PROGRAM-ID paragraph as the deck name. The second image is an IDENT directive with the same deck name. The U parameter is interpreted as follows:

omitted	No update file is created.
U=0	No update file is created.
U	COMPASS line images of the source program are written on the file COMPS.
U=lfn	COMPASS line images of the source program are written on the file with the designated logical file name.

Error Processing Parameters

The level of errors to be listed, the error level that causes execution to be aborted, and the diagnosing of ANSI extensions as errors can be specified by COBOL5 control statement parameters. Parameters can also be used to diagnose and control ANSI extensions and to note language elements that do or do not conform to a specified Federal Information Processing Standard (FIPS) level. Another parameter indicates that only syntax checking of the source program is to be performed.

Four levels of errors can be detected during compilation. The lowest error level to be listed on the error file is determined by the EL parameter. The error levels in increasing order of severity are trivial (T), warning (W), fatal (F), and catastrophic (C). A trivial error indicates a suspicious usage; the syntax is correct, but the usage is questionable. A warning error indicates that the syntax is incorrect, but the compiler has made an assumption and continued compilation. A fatal error indicates an error that prevents compilation of the statement. A catastrophic error indicates a compiler error; compilation usually continues but a system analyst should be notified. The EL parameter is interpreted as follows:

omitted	Errors of levels W, F, and C are listed.
EL=W	Errors of levels W, F, and C are listed.
EL	Errors of levels F and C are listed.
EL=F	Errors of levels F and C are listed.
EL=T	Errors of levels T, W, F, and C are listed. FIPS errors are also listed.
EL=C	Errors of level C are listed.

The action taken by the compiler after compilation of the source program is determined by the ET parameter. The error level (T, W, F, or C) indicated by the ET parameter is the lowest error level that causes the compiler to abort execution of the object program. Level T or W errors usually produce good binary output that can be executed. The binary output with level F or C errors will abort the loader unless the DB (debugging) parameter specifies the B option. The ET parameter is interpreted as follows:

omitted	When compilation terminates, the next control statement in the job is executed, regardless of errors diagnosed during compilation.
ET=T	Errors of level T, W, F, or C abort the compiler.
ET=W	Errors of level W, F, or C abort the compiler.
ET=F	Errors of level F or C abort the compiler.
ET=C	Errors of level C abort the compiler.

If the compiler is aborted, the job resumes after any EXIT control statement in the job stream.

Language extensions that do not conform to ANSI standard X3.23-1974 can be diagnosed and treated as errors by specifying the ANSI parameter in the COBOL5 control statement. These errors can be detected as either trivial or fatal errors. Non-ANSI errors are not listed unless the error level (EL) parameter specifies that trivial errors are to be listed (EL=T); the errors are listed as level N errors.

Editing of numeric display items is normally performed when the DISPLAY statement is executed, but can be suppressed by specifying the ANSI parameter; items with embedded decimal points and/or overpunched signs are displayed without editing. The ANSI parameter must be included in the COBOL5 control statement when the corresponding data items in the USING phrase of a CALL statement and the USING phrase of the Procedure Division header are not both level 77 items or level 01 items. When the parameter is used, level 77 items, which are normally synchronized right within computer words, are synchronized left to agree with the level 01 items. The ANSI parameter is interpreted as follows:

omitted	Non-ANSI extensions are allowed in the program, numeric display items are edited by the DISPLAY statement, and level 77 items are right justified within computer words. If non-ANSI reserved words are used as user-defined words, diagnostics result.
ANSI	Non-ANSI extensions are diagnosed and treated as trivial errors.
ANSI=T	Non-ANSI extensions are diagnosed and treated as trivial errors.
ANSI=F	Non-ANSI extensions are diagnosed and treated as fatal errors.
ANSI=NOEDIT	Numeric display items are not edited by the DISPLAY statement; non-ANSI extensions are diagnosed and treated as trivial errors.
ANSI=77LEFT	Level 77 items are synchronized left within the computer word; non-ANSI extensions are diagnosed and treated as trivial errors.
ANSI=AUDIT	Non-ANSI reserved words are not recognized as reserved words. Also, the conditions described for ANSI=NOEDIT and ANSI=77LEFT are in effect.

When multiple options are selected in the ANSI parameter, the options are separated by slashes.

Support of selected language features at a particular level of the Federal Information Processing Standard (FIPS) can be diagnosed by specifying the FIPS parameter in the COBOL5 control statement. Four levels of usage can be indicated. FIPS diagnostics are not listed unless the ANSI parameter specifies that non-ANSI extensions are to be diagnosed, and the EL parameter specifies that trivial errors are to be listed (EL=T). The FIPS parameter is interpreted as follows:

omitted	No FIPS diagnostics are issued.
FIPS=1	Support of language features at levels 1, 2, 3, and 4 is diagnosed.
FIPS=2	Support of language features at levels 2, 3, and 4 is diagnosed.
FIPS=3	Support of language features at levels 3 and 4 is diagnosed.
FIPS	Support of language features at level 4 is diagnosed.
FIPS=4	Support of language features at level 4 is diagnosed.

Generation of executable code can be inhibited during compilation through the SY parameter in the COBOL5 control statement. This option is useful when syntax checking without execution is desired. If executable code is not generated, compilation time is greatly reduced. The SY parameter is interpreted as follows:

omitted	Source program is compiled and executable code is generated.
SY	Source program is checked for correct syntax, but executable code is not generated.

Source Program Parameters

Special source program processing can be specified through several COBOL5 control statement parameters. These parameters are provided mainly for processing existing programs.

The PSQ parameter is specified when sequence numbers are to be processed by the compiler. Sequence numbers containing digits and spaces only must be specified for every line in the source program; the sequence number cannot be all spaces, however. Compilation and execution diagnostics then reference the sequence numbers. If this parameter is omitted, sequence numbers are optional and can contain any character in the computer character set. The PSQ parameter is interpreted as follows:

omitted	Compiler-generated line numbers are referenced in all diagnostics; sequence numbers are optional and are not processed by the compiler.
PSQ	Sequence numbers must be specified and are referenced in all diagnostics.

An illustration of the PSQ parameter with a program created through a terminal, using a NOS text editor, is shown in section 16.

The apostrophe character can be specified as the delimiting character for nonnumeric literals by the APO parameter. This option is the same as specifying the QUOTE IS APOSTROPHE clause in the SPECIAL-NAMES paragraph of the Environment Division. When this option is selected, the apostrophe and not the quotation mark delimits nonnumeric literals and the quotation mark character can then be used within the literals the same as any other character. The APO parameter is interpreted as follows:

omitted	Nonnumeric literals in the source program are delimited by the quotation mark character.
APO	Nonnumeric literals in the source program are delimited by the apostrophe character.

The CC1 parameter provides the means to convert data items described in the source program as COMPUTATIONAL to COMPUTATIONAL-1 items. This option allows programs written for other compilers to gain the efficiencies of COMPUTATIONAL-1 processing. The CC1 parameter is interpreted as follows:

omitted	Data items described as COMPUTATIONAL are stored and processed as COMPUTATIONAL items.
CC1	Data items described as COMPUTATIONAL are stored and processed as COMPUTATIONAL-1 items.

The UC1 parameter should be used only when files created by COBOL 4 are being processed under COBOL 5. In COBOL 4, COMPUTATIONAL-1 data items have a different format than COMPUTATIONAL-1 data items under COBOL 5. COBOL 4 COMPUTATIONAL-1 data items that contain more than 14 digits are represented as two COMPUTATIONAL-2 data items; the UC1 parameter does not apply to these data items. Specifying UC1 in the COBOL5 control statement causes all COMPUTATIONAL-1 items to be converted to integer format during processing. This results in a larger and slower object program. The UC1 parameter is interpreted as follows:

omitted	All COMPUTATIONAL-1 data items are processed in integer format.
UC1	All COMPUTATIONAL-1 items are converted to integer format before processing.

When arithmetic statements and comparisons are performed, numeric fields cannot have leading blanks. For these operations, the LBZ parameter specifies that leading blanks are treated as zeros. Selection of this option significantly slows execution time and increases the size of the object program. The LBZ parameter is interpreted as follows:

omitted	Numeric fields that contain leading blanks are in error.
LBZ	Leading blanks in numeric fields are treated as zeros in arithmetic statements and comparisons.

Output Listing Parameters

The listings that are produced when the source program is being compiled and the format of the output pages are determined by several COBOL5 control statement parameters. All listings are written on the listing file specified by the L parameter; if this parameter is omitted, the listings are written on the system file OUTPUT.

Output listings are selected by the LO parameter. One or more of four listings can be specified: source program, cross reference, object code, and data map. The LO parameter is interpreted as follows:

omitted	Source program listing is produced.
LO=S	Source program listing is produced.
LO=-S	Source program listing is not produced; other listings can be selected.
LO=R	Source program listing and cross reference listing of program entities and locations of definitions and use within the program are produced.
LO=O	Source program listing and generated object code with COMPASS mnemonics are produced.
LO=M	Source program listing and data map listing that correlates program entities, attributes such as data class and size, and physical storage are produced.
LO	Source program, cross reference, and data map listings are produced.
LO=0	None of the listings that can be selected are produced.

When multiple listings are selected, slashes are used to separate the option letters (LO=S/R).

The spacing between sections of listings produced by the compiler is controlled by the BL parameter. Normally (parameter omitted), a triple space separates the listings; however, a page eject can be specified to start each section on a new page. A page eject occurs before printing the Procedure Division, the cross reference, the generated object code, the data map, and the diagnostics. The BL parameter is interpreted as follows:

omitted	Triple space separates the sections.
BL	Page eject occurs between the sections.

The density of output print lines is determined by the PD parameter. Lines are printed at either six or eight lines per inch and are either single or double spaced. The PD parameter applies to the listings written on the listing file and the error file, which are specified by the L and E parameters, respectively. This parameter is ignored for connected interactive terminal listings. The option specified by this parameter must be supported by the printer on which the listings will be output. The PD parameter is interpreted as follows:

omitted	Listings are printed according to the job default print density (user changeable, installation parameter).
PD	Listings are printed single spaced at eight lines per inch.
PD=8	Listings are printed single spaced at eight lines per inch.
PD=3	Listings are printed double spaced at six lines per inch.
PD=4	Listings are printed double spaced at eight lines per inch.
PD=6	Listings are printed single spaced at six lines per inch.

The number of lines printed on an output page can be specified by the PS parameter.

The PS parameter is interpreted as follows:

omitted	Number of lines on a printed output page is determined by the job default page size (user changeable, installation parameter).
PS=n	Number of lines on a printed output page is the specified number (n).

The width of an output printed page is designated by the PW parameter. The PW parameter is interpreted as follows:

omitted	Length of lines printed output is determined by the job default print width (user changeable, installation parameter).
.PW	Lines of printed output are 72 characters in length.
PW=n	Lines of printed output are the specified number of characters in length; listing lines are reformatted to this length.

Debugging Parameters

In addition to the debugging features that can be included in the source program, two debugging aids available under COBOL 5 can be selected by parameters in the COBOL5 control statement. The DB parameter indicates one or more debugging options. The TDF parameter can be included to obtain a formatted dump of the contents of program data items.

Debugging options are selected by the DB parameter. One or more of four options can be specified: compilation of debugging lines, production of binary executable code, tracing of program flow, and checking of subscript and index references. The DB parameter is interpreted as follows:

omitted	None of the DB parameter options are performed.
DB=0	None of the DB parameter options are performed.
DB=B	Binary executable code is produced regardless of all errors in the source program.
DB=DL	Debugging lines in the source program are compiled.
DB=ID	Debugging lines in the source program (lines with a D in column 7) are compiled as executable code. When the DB=ID parameter is specified at compilation time, DB=RF and DB=SB are automatically selected.
DB=SB	Subscript and index references are checked during execution for out-of-bounds references.
DB=TR	Execution flow of the program is traced.
DB=RF	Reference modification values are checked for out-of-bound references.

DB — Debugging lines are compiled, subscript and index references are checked, and binary executable code is produced. This option is the same as specifying DB=DL/SB/B.

When multiple options are selected in the DB parameter, the options are separated by slashes.

When the TDF parameter is included in the COBOL5 control statement, a termination dump can be obtained by specifying the C5TDMP control statement after the COBOL5 control statement. The dump is produced regardless of whether the program terminates normally or abnormally. The dump consists of a formatted map of the contents of all data items in the program. The TDF parameter is interpreted as follows:

omitted — No termination dump processing information is written.

TDF — Termination dump processing information is written on the file TDFILE.

TDF=lfn — Termination dump processing information is written on the file with the designated logical file name.

COPY Statement Parameter

When COPY statements are included in the source program, the COBOL 5 source library containing the text to be copied into the source program is specified by the X parameter. This is the default library for COPY statements that do not specify a library-name. The source library must be an UPDATE random program library. The X parameter is interpreted as follows:

omitted — COBOL source library resides on the file OLDPL.

X=0 — COBOL source library resides on the file OLDPL.

X — COBOL source library resides on the file NEWPL.

X=lfn — COBOL source library resides on the file with the designated logical file name.

COBOL Subprogram Parameters

When a COBOL 5 subprogram is being compiled, the SB parameter must be specified in the COBOL5 control statement. If the main program is not a COBOL program, the MSB parameter must also be specified. The FDL parameter is specified when fast dynamic loader processing is used.

The SB parameter is used only if the subprogram is being compiled independently; if the subprogram follows a COBOL 5 main program in the input file, this parameter is not required. The SB parameter is interpreted as follows:

omitted — Source program is compiled as a main program.

SB — Source program is compiled as a subprogram.

When the subprogram being compiled is called by a main program that is written in a language other than COBOL, the MSB parameter must also be specified. Only the first COBOL subprogram called in a group of independently compiled subprograms should specify the MSB parameter. The MSB parameter is interpreted as follows:

omitted — Source program is compiled normally.

MSB — Source program is compiled as a subroutine that includes COBOL initiation.

Fast dynamic loader (FDL) processing is activated when the FDL parameter is specified. This parameter designates the FDL file that contains the information required for fast dynamic loader processing. When the FDL parameter is specified, subprograms can be canceled; the CALL statement can specify a literal or an identifier and the program name can be longer than seven characters. If the FDL parameter is not specified in the COBOL5 control statement, program names must be seven characters or less and must be unique within the run unit. If the TDF parameter is specified to obtain a termination dump of all programs in the FDL file, the FDL parameter must also be specified; if the FDL parameter is not specified, a dump of only the main program is produced. The FDL parameter is interpreted as follows:

omitted — Subprograms cannot be canceled.

FDL=lfn — FDL processing information is contained on the file with the designated logical file name.

FDL — FDL processing information is contained on the file FDLFILE.

Sub-Schema File Parameter

When a program in the run unit accesses a data base file through a subschema, the COBOL5 control statement must include the D parameter. This parameter specifies the file that contains the subschema. If the D parameter is not specified and the SUB-SCHEMA clause appears in the program, a fatal error occurs during compilation. The D parameter is interpreted as follows:

omitted — A CDCS subschema is not used.

D=0 — A CDCS subschema is not used.

D=lfn — The CDCS subschema resides on the file with the designated logical file name.

D — The CDCS subschema resides on the file with the name specified in the SUB-SCHEMA clause.

COMPILATION OUTPUT LISTINGS

Various listings can be produced when a source program is being compiled. The LO parameter in the COBOL5 control statement is used to select the desired listings.

Source Program Listing

The source program listing is the normal listing desired when compiling a COBOL 5 program. This listing includes the source program images, diagnostics, the load map, and the job dayfile.

Each line of the source program is printed on the output listing, unless the OFFSOURCE or OFFALLLIST command is specified in the source program. The format and order of each line on the listing is identical to that of the COBOL coding sheet.

The source listing commands can be used to control the printing of the source program listing. The commands are specified on comment lines (lines with an asterisk in column 7) and must begin in column 8; they can appear anywhere a comment line is legal. When the command ONSOURCE or ONALLLIST is specified, the source program listing is turned on. When the command OFFSOURCE or OFFALLLIST is specified, printing of the source listing is suppressed; however, the OFFSOURCE and OFFALLLIST commands themselves are always printed. The listing is initially turned on for every program in the compilation. The listing can be turned off and on again as many times as desired. When the L=0 or LO=-S parameter is included in the COBOL5 control statement, the source listing commands are ignored. Figure 11-1 illustrates a source listing produced by the COBOL 5 compiler.

Diagnostics are printed immediately following the source program listing. The error level (N, T, W, F, or C), the source line number and the column number in which the error occurs, the diagnostic number, and the message are listed for each error. If the PSQ parameter is specified in the COBOL5 control statement, the sequence number of the source program line is referenced in the diagnostic instead of the compiler-generated line number. Only those errors designated by the EL parameter in the COBOL5 control statement are printed. After the last error message is printed, the number of listed errors and the number and type of unlisted errors are printed. If the program compiles with no errors diagnosed, a diagnostic listing is not produced. Figure 11-2 illustrates sample diagnostics produced by the COBOL 5 compiler.

The load map is a printed listing that contains the names and locations of program and block entry points. (A system-defined default can produce the load map or it can be selected by the MAP control statement.) The address and length for each entry point are expressed in terms of octal numbers. All addresses are relative addresses, not absolute addresses. The addresses listed on the load map can be helpful in debugging a program when an error address is listed in the dayfile. Figure 11-3 illustrates a load map produced for a compilation run.

If the compilation terminates abnormally, an exchange package dump (DMPX) is printed following the load map. This dump lists the exchange jump package, the contents of the first 100 words of field length, and the contents of the 100 words preceding and following the address where the job terminated. Figure 11-4 illustrates the standard dump that is printed when a job terminates abnormally.

```
      SOURCE LISTING OF


       1    IDENTIFICATION DIVISION.
       2    PROGRAM-ID.  LST-IND.
       3    ENVIRONMENT DIVISION.
       4    CONFIGURATION SECTION.
       5    SOURCE-COMPUTER.  CYBER-170.
       6    OBJECT-COMPUTER.  CYBER-170.
       7    INPUT-OUTPUT SECTION.
       8    FILE-CONTROL.
       9        SELECT EMP-FILE ASSIGN TO EMPFLE, INDFLE
      10            ORGANIZATION IS INDEXED
      11            ACCESS MODE IS DYNAMIC
      12            RECORD KEY IS EMP-ID
      13            ALTERNATE RECORD KEY IS HIRE-DATE
      14                WITH DUPLICATES ASCENDING
      15            ALTERNATE RECORD KEY IS JOB-ID
      16                WITH DUPLICATES ASCENDING.
      17        SELECT PRINTOUT ASSIGN TO OUTPUT.
      18    DATA DIVISION.
      19    FILE SECTION.
                      .
                      .
                      .
      81        MOVE JOB-ID TO ID-OUT.
      82        MOVE EMP-ID TO EMP-ID-OUT.
      83        WRITE PRINTLINE FROM LINE-OUT.
      84        GO TO READING.
      85    PRINT-HEAD.
      86        WRITE PRINTLINE FROM HEAD-OUT.
      87        MOVE SPACES TO PRINTLINE.
      88        WRITE PRINTLINE.
      89    BAD-DATE.
      90        DISPLAY "NO EMPLOYEES HIRED FROM " DATE-IN.
      91    CLOSE-OUT.
      92        CLOSE EMP-FILE, PRINTOUT.
      93        STOP RUN.

    COLUMN   1         2         3         4         5         6         7         8
    12345678901234567890123456789012345678901234567890123456789012345678901234567890
```

Figure 11-1. Source Listing

```
CDC COBOL 5.3 - LEVEL 507          DIAGNOSTICS IN NAME-RP
SEV LINE COL ERROR

 W     4   8  1026  THIS ELEMENT MAY NOT BEGIN IN THE A AREA.

 W    24  NA  4079  THE MINIMUM RECORD LENGTH DEFINED BY THIS RECORD DESCRIPTION IS LESS THAN THE MINIMUM RECORD LENGTH
                    SPECIFIED IN THE RECORD CONTAINS CLAUSE FOR THIS FILE.

 F    28  32  2002  THE RIGHT PARENTHESIS, REQUIRED HERE, IS MISSING IN THIS REPETITION COUNT.
                .
                .
                .
 F    57  46  3049  A LITERAL IS REQUIRED IN A VALUE CLAUSE.

 W    57  47  1001  THIS CHARACTER MAY NOT FOLLOW THE PRECEDING CHARACTER.  AN INTERVENING SPACE IS ASSUMED.

 W    57  47  3212  A DUPLICATE PERIOD HAS BEEN ENCOUNTERED. THE PRECEDING ONE IS IGNORED.

 F    73  33  7225  -ADVANCING- MUST BE FOLLOWED BY AN IDENTIFIER, AN INTEGER LITERAL, A MNEMONIC NAME, OR -PAGE-.
                .
                .
                .
 F    82  35  7994  UNDEFINED DATA NAME REFERENCE.

                     **     14 ERRORS LISTED              **

                     **      1 UNLISTED TRIVIAL ERROR     **
065000B CM,     .984 CPU SECS, 0000000B  ECS USED
```

Figure 11-2. COBOL 5 Diagnostics

```
LOAD MAP - SPL                                  CYBER LOADER 1.5-642      85/10/10. 09.55.38.      PAGE     1


FWA OF THE LOAD            111
LWA+1 OF THE LOAD        10603

TRANSFER ADDRESS -- SPL              112

PROGRAM ENTRY POINTS --        SPL              112


PROGRAM AND BLOCK ASSIGNMENTS.

BLOCK        ADDRESS   LENGTH    FILE      DATE      PROCSSR VER LEVEL  HARDWARE   COMMENTS

/CCOMMON/        111        0
/C.HASHV/        111        1
SPL              112      167     LGO      85/10/10  COBOL  5.3 642     64   I     COBOL PROGRAM
C$DSPLY          301      171  SL-COBOL5   85/08/24  COMPASS 3.6 642          I     DISPLAY A LINE ON OUTPUT FILE
C$EDIT           472      406  SL-COBOL5   85/08/24  COMPASS 3.6 642          I     MOVE AN ITEM TO AN EDITED FIELD
/STP.END/       1100        1
C$INIT          1101      211  SL-COBOL5   85/08/24  COMPASS 3.6 642          I     INITIALIZATION AND CONSTANTS

  .
  .
  .

ENTRY POINTS.

ENTRY      ADDRESS   PROGRAM            REFERENCES

AAM$CTL      *WEAK*                     CTRL$AA        6525
AAM$ENV      *WEAK*                     CTRL$AA        6526
AAM.CTL      *WEAK*                     CTRL$AA        6527
CHEK$SQ      *WEAK*                     CTL$RM         5671
DF$CRM       *WEAK*                     CTL$RM         5702
  .
  .
  .
CMM.GOA      10412   CMM.GOA            CMF.ALF        4761      4771
                                        CMF.POE        7502
CMM.MEM      10444   CMM.MEM            CMM.R          7521      7552      7612      7613      7632
CMM.POA      10453   CMM.POA            CMM.GOA       10426     10432     10436
CMM.SV       10567


     .172 CP SECONDS              30100B CM STORAGE USED                    13 TABLE MOVES
```

Figure 11-3. Load Map

```
DMPX.

 P   006756  A0  004045  B0  000000
 RA  062700  A1  000001  B1  000001    C(A1)= 0102 2420 0000 0000 0000    C(B1)= 0102 2420 0000 0000 0000
 FL  042200  A2  004062  B2  000002    C(A2)= 0000 0000 0000 0000 3716    C(B2)= 1404 0000 0000 0000 0017
 EM  700700  A3  004070  B3  004141    C(A3)= 0000 0000 0000 0000 0000    C(B3)= 4360 0516 0004 2154 6000
 RE  000036  A4  036174  B4  000000    C(A4)= 6000 0000 0350 5503 6300    C(B4)= 0000 0000 0000 0000 0000
 FE  000000  A5  011764  B5  037006    C(A5)= 1725 2420 2524 0000 0027    C(B5)= 5120 0056 3443 1733 6621
 MA  000600  A6  000001  B6  004141    C(A6)= 0102 2420 0000 0000 0000    C(B6)= 4360 0516 0004 2154 6000
             A7  004054  B7  002541    C(A7)= 0000 0000 0000 0000 0000    C(B7)= 0000 0000 0000 0000 0000
 X0  0104  0361  7500  0000  0000
 X1  0000  0000  0000  0000  0000
 X2  0000  0007  6340  0000  0000
 X3  0000  0000  0000  0000  0000
 X4  0000  0000  0000  0000  0000
 X5  1725  2420  2524  0000  0027
 X6  0102  2420  0000  0000  0000
 X7  0000  0000  0000  0000  0000

    00000   00000 00000 00000 00000        01022 42000 00000 00000        14040 00000 00000 00017        00000 00000 00000 00000
    00054  #56110 03110 00054 54710        51100 00001 03110 00055        64550 02550 00000 46000        00000 00000 00000 00000
    00061  #61447 77776 03040 00056        07040 00060 51600 00001        04000 00063 00000 00000
    00064   04021 52324 22040 00001        40000 00000 00007 64404        40000 00000 01000 12425        00000 00000 40000 00000
    00070   04021 52324 22045 11404        52570 00000 00000 00000        00000 00000 00000 00000
    00100  #54000 00000 01000 00001

    06656   00000 00000 00000 00000        00000 00000 00000 00001
    06660   00000 00000 00000 00000        00000 00000 00000 00001        00000 00000 00000 00000        00000 00000 00000 00001
    06664   00000 00000 00000 00000  06666#00000 00000 00000 00001        00000 00000 00000 00000
    06670   00000 00000 00000 00001        00030 00000 10000 41766        10722 51200 00065 46000        03220 06674 51400 06676
    06674   10644 22207 51600 06717        02000 06717 61000 46000        37423 10711 10100 20236        15617 37116 03240 06704
    06700   03060 06702 63710 53427        10644 53637 61000 46000        36221 36331 61707 77775        02000 06707 61000 46000
    06704   61707 77775 73227 73337        66711 03060 06707 53427        36226 10744 53737 36336        03010 06716 53227 54421
    06710   10622 22704 53637 54761        76311 37113 03010 06716        54227 54447 61000 46000        10622 54227 22704 54447
    06714   37113 46000 54667 54777        03110 06713 61000 46000        04000 11104 00000 00000        37423 10711 10100 20236
    06720   03070 06716 72677 76314        10044 03340 06722 14044        36007 11606 03260 06725        10077 12223 13777 43470
    06724   61700 06745 02000 06747        71600 06672 03340 06736        12226 20636 12336 46000        10077 72777 77753 43470
    06730   03370 06731 71000 00024        61700 06732 02000 06747        46400 06671 61000 46000        63770 10011 07710 06716
    06734   71407 77753 37334 20436        37224 02000 06727 46000        20636 12676 36336 20636        36226 46000 61000 46000
    06740   71600 00024 73070 37776        43470 03370 06742 10066        10600 37330 20636 37226        61700 06744 02000 06747
    06744   46400 06671 61000 46000        63770 10011 07710 06716        02000 06740 61000 46000        22610 20003 36606 15064
    06750   11446 20032 20454 12040        12620 51600 06671 12630        46400 06671 61000 46000        54660 02770 00000 46000
    06754   04000 06767 00000 00000        01300 00000 00000 00000        04000 01042 00000 00000        51100 00001 03110 06757
    06760   54610 04000 06755 46000        51100 00066 03310 06763        51100 06754 04000 06764        71100 00130 20160 46000
    06764   13661 13161 13661 46000        51600 06755 10611 46000        51100 00001 01000 06754        20652 01000 06756 46000
    06770   51100 00001 03110 06770        04004 06771 61000 46000        51100 00001 03110 06771        71602 20314 04000 06767
    06774   20150 36661 01000 06756        04004 06775 61000 46000        71602 20314 20652 36662        53160 20173 03310 06775
    07000   03010 06775 51100 00001        03110 06777 71100 00001        04000 06774 61000 46000        71603 24616 12661 20651
    07004   01000 06756 61000 46000        04000 03574 00000 00000        73660 20630 12161 73610        20123 03210 07003 20151
    07010   13116 20636 51600 07013        74660 36116 20123 46000        04000 07003 61000 46000        00000 03634 00000 00001
    07014   04000 12252 00000 00000        61100 00001 01000 10607        71600 11406 50607 77746        51100 07014 20136 10611
    07020   50607 77747 10622 43766        50607 77744 21306 15737        21306 73630 50607 77741        50707 77740 73721 46000
    07024   03060 07025 73771 46000        50707 77742 01000 07046        03010 07030 50207 77736        53020 04000 07014 46000
    07030   71204 07030 03220 07037        76110 01000 10560 46000        01000 07147 61000 46000        03110 07035 76111 46000
    07034   01000 10663 61000 46000        50207 77736 50307 77747        53020 63730 02770 00000        71100 07046 71200 07147
    07040   50307 77742 01004 07040        04000 07035 61000 46000        77777 77777 77777 77777        77777 77777 77770 00000
    07044   00000 00000 00000 00000        00000 00000 00000 00003        04000 07026 00000 00000        10600 22705 51600 07042
    07050   54761 77620 77730 54671        54761 43600 76700 46000        50207 77742 50307 77741        03030 07060 50107 77771
    07054   63610 43401 61000 46000        56160 67661 74010 20022        12610 11741 21144 73110
```

Figure 11-4. Standard Dump

The final printed output for any job is the dayfile. The dayfile records the history of job execution. Each control statement is listed with the time it was encountered. Other information on the dayfile includes equipment assignments, operating system diagnostics, job accounting information, and job statistics. The time of day associated with each processing event is printed on the dayfile. Figure 11-5 illustrates the dayfile produced at the end of job execution.

Cross Reference Listing

The cross reference listing is an optional listing that provides alphabetic lists of data-names and procedure-names used in the source program. This listing is divided into four parts: referenced data-names, unreferenced data-names, referenced procedure-names, and unreferenced procedure-names.

Each list contains the applicable program names and the line number and column number where the name is defined. The referenced data-names and referenced procedure-names parts of the listing also list all line numbers that reference the name. Figure 11-6 illustrates a cross reference listing produced by the COBOL 5 compiler.

Object Code Listing

The object code listing can be selected for output. This listing contains the object code generated during compilation along with the corresponding COMPASS mnemonics. Since all corrections should be made at the source program level, the entire listing is probably of little value to the COBOL 5 programmer. Selected portions of the object listing, can be printed through the LIST/NOLIST commands. The commands, which are specified on comment lines beginning in column 8 (immediately following the asterisk), can appear anywhere within the Procedure Division that a comment line is legal; commands specified outside the Procedure Division are ignored. The OFFOBJLIST and OFFALLLIST commands suppress printing of the listing; however, the OFFOBJLIST and OFFALLIST commands themselves are always printed. The listing is initially turned on for every program in the compilation if the LO=O parameter is specified in the COBOL5 control statement; the listing can be turned off and on again as often as desired. If the LO=O parameter is not specified or the L=O parameter is specified, the object code listing commands are ignored. Figure 11-7 illustrates the object code listing produced during a COBOL 5 compilation run.

Data Map Listing

The data map listing contains an entry for each item defined in the Data Division. The data items are listed in the order of section appearance. The data map shows the level number, name, size, usage or class, and other relevant information. Figure 11-8 illustrates the data map listing generated during a COBOL 5 compilation run.

EXECUTING A PROGRAM

Once a COBOL 5 source program has been compiled and an object program has been generated, the program is ready for execution. Execution can be initiated immediately following compilation as part of the same job. When the source program is completely debugged, the object program can be punched on cards or stored on disk as a permanent file for execution at a later time.

The program is called into execution by a program call control statement. This statement specifies the logical file name of the file containing the object program, optionally followed by a parameter list. The statement is specified according to operating system syntax and cannot be continued from one card image to another. The program call control statement can be used when execution immediately follows compilation and when execution is initiated in a separate run. The file that contains the object program is specified by the B parameter in the COBOL5 control statement.

When execution of a COBOL 5 program follows compilation in the same job, the program call control statement immediately follows the COBOL5 control statement. The program call control statement specifies either the logical file name in the COBOL5 control statement B parameter or, if the B parameter is omitted, the system file LGO.

```
          FULL  DAYFILE.  85/10/10. 09.55.39.*09.55.34* PAGE    1

09.55.34.CINDX.
09.55.34.USER,USR1765,.
09.55.34.ABSC,  B.
09.55.34.CHARGE,8347,ACCT175.
09.55.34.MAP(OFF)
09.55.35.DEFINE,INVNTRY
09.55.35.DEFINE,INVIDX
09.55.35.COBOL5.
09.55.37.066400B CM,      .231 CPS, 000000B ECS
09.55.37.LGO.
10.19.03.UCLP, 03, 116,       0.896KLNS.
```

Figure 11-5. Dayfile

```
CDC COBOL 5.3 - LEVEL 518     CROSS REFERENCE FOR NEW-IND     AOPT= 66/CDC/CDCS2
  REFERENCED DATA-NAMES        LINE COLUMN           REFERENCE(S)

CARD-IN                        20    12             9     67    70    74    92
CARD-1                         23    12            22     73
CARD-2                         34    12            22     80
CITY                           29    20            73
CITY                           57    20            73
DEPT                           39    16            80

  REFERENCED DATA-NAMES        LINE COLUMN           REFERENCE(S)
CDC COBOL 5.3 - LEVEL 518     CROSS REFERENCE FOR NEW-IND     AOPT= 66/CDC/CDCS2
  REFERENCED DATA-NAMES        LINE COLUMN           REFERENCE(S)

DEPT                           61    16            80
DIV                            41    16            80
DIV                            62    16            80
EMP-FILE                       47    12            10     68    92
EMP-ID                         53    16            13     72    76
EMP-ID-1                       24    16            72
EMP-ID-2                       35    16            76
EMP-NAME                       26    16            73
EMP-NAME                       54    16            73
EMPLOYEE                       52    12            51     73    80    81    86    89
HIRE-DATE                      63    16            14     79
HIRE-DATE-IN                   43    16            79
JOB-ID                         60    16            16     78
JOB-ID-IN                      37    16            78
LOCATION                       64    16            80
LOCATION                       45    16            80
STATE                          30    20            73
STATE                          58    20            73
STREET                         28    20            73
STREET                         56    20            73
ZIP-CODE                       32    20            73
ZIP-CODE                       59    20            73

  REFERENCED DATA-NAMES        LINE COLUMN           REFERENCE(S)

CDC COBOL 5.3 - LEVEL 518     CROSS REFERENCE FOR NEW-IND     AOPT= 66/CDC/CDCS2
  UNREFERENCED DATA-NAMES      LINE COLUMN

EMP-ADDRESS                    27    16
EMP-ADDRESS                    55    16

  UNREFERENCED DATA-NAMES      LINE COLUMN

CDC COBOL 5.3 - LEVEL 518     CROSS REFERENCE FOR NEW-IND     AOPT= 66/CDC/CDCS2
  REFERENCED PROCEDURE-NAMES   LINE COLUMN           REFERENCE(S)

BAD-RECORD                     88    8             82
CLOSE-FILES                    91    8             71
INPUT-ERROR                    84    8             75     77
READ-CARDS                     69    8             83     87    90

  REFERENCED PROCEDURE-NAMES   LINE COLUMN           REFERENCE(S)

CDC COBOL 5.3 - LEVEL 518     CROSS REFERENCE FOR NEW-IND     AOPT= 66/CDC/CDCS2
  UNREFERENCED PROCEDURE-NAMES LINE COLUMN

OPEN-FILES                     66    8

  UNREFERENCED PROCEDURE-NAMES LINE COLUMN
```

Figure 11-6. Cross Reference Listing

```
                                OBJECT LISTING OF NEW-IND

                                                IDENT   NEW-IND
                                                LDSET   LIB=COBOL5/BAMLIB
                                                LDSET   USE=C.GETBK
                                                LDSET   OMIT=C.DMOPN/C.DMCLS/C.DMDEL/C.DMSTR/C.DMWR1/C.DMWR2
                                                LDSET   OMIT=C.DMRD1/C.DMRD2/C.DMREW/C.DMSKP/DM$CKP/C.KRORD
                                                LDSET   OMIT=C.RORD/C.SVSKP
                                                LDSET   PRESET=55555555555555555555B
                                                USE     RECAREA
000175                                  #RECAREA BSS    0
000175                                  L000001  BSS    0
000175                          000011           BSS    00011B
000206                                  L000002  BSS    0
000206                          000012           BSS    00012B
                                                USE     /C.HASHV/
000000                                  #C.HASHV BSS    0
000000                          000001           BSS    00001B
                                                USE     SREG
000220                                  #SREG    BSS    0
000220  00000000000000000000            DATA    00000000000000000000B
000221                          000010           BSS    00010B
                                                USE     FITS
000231                                  #FITS    BSS    0
000231  11162025240000000000            DATA    11162025240000000000B
000232  00000001000000000000            DATA    00000001000000000000B
    .
    .
    .
000146  0100000000 *                    RJ      =XC.MSG                 93
                  0000000135        -     PS      000135B                 93
000147                             P000013  BSS    0                       93
                                                USE     LITERAL         93
000170  16052746111604555555            DATA    16052746111604555555B   93
000171  55555555555555555555            DATA    55555555555555555555B   93
000172  55555555555555555555            DATA    55555555555555555555B   93
    .
    .
    .

                                                EXT      C.OPSQ         93
                                                EXT      C.RDNSQ        93
                                                EXT      C.STOPR        93
                                                EXT      C.STRIN        93
                                                EXT      C.WRRIS        93
                                                EXT      C.CLOSF        93
                                                EXT      C.OPIN         93
                                                EXT      C.OPOUT        93
                                                EXT      C.OUTPT        93
                                000000 +        END     NEW-IND         93
```

Figure 11-7. Object Code Listing

```
                              MAP OF NEW-IND


                          *** DATA MAP (ADDR/BCP IN OCTAL, SZ IN DECIMAL) ***

                 FILE SECTION
  FD     CARD-IN                BLOCK=PROGRAM    ADDR/BCP=000231/                          LNR=20
           (INPUT)
*  01    CARD-1                                                 000175/00 SZ=80      GROUP       23
     03    EMP-ID-1                                             000175/00    3       NUMERIC     24
     03    FILLER                                               000175/03    1       AN          25
     03    EMP-NAME                                             000175/04   20       AN          26
     03    EMP-ADDRESS                                          000177/04   48       GROUP       27
     05    STREET                                               000177/04   20       AN          28
     05    CITY                                                 000201/04   20       AN          29
     05    STATE                                                000203/04    2       ALPHA       30
     05    FILLER                                               000203/06    1       AN          31
     05    ZIP-CODE                                             000203/07    5       NUMERIC     32
     03    FILLER                                               000204/02    8       AN          33
*  01    CARD-2                                                 000175/00   80       GROUP       34
     03    EMP-ID-2                                             000175/00    3       NUMERIC     35
     03    FILLER                                               000175/03    1       AN          36
     03    JOB-ID-IN                                            000175/04    5       AN          37
     03    FILLER                                               000175/11    5       AN          38
     03    DEPT                                                 000176/04    3       NUMERIC     39
     03    FILLER                                               000176/07    2       AN          40
     03    DIV                                                  000176/11    3       NUMERIC     41
     03    FILLER                                               000177/02    2       AN          42
     03    HIRE-DATE-IN                                         000177/04    6       NUMERIC     43
     03    FILLER                                               000200/00    4       AN          44
     03    LOCATION                                             000200/04    3       NUMERIC     45
     03    FILLER                                               000200/07   43       AN          46
  FD     EMP-FILE                                               000303/                         47
           (EMPFLE)
*  01    EMPLOYEE                                               000206/00   90       GROUP       52
     03    EMP-ID                                               000206/00    3       NUMERIC     53
     03    EMP-NAME                                             000206/03   20       AN          54
     03    EMP-ADDRESS                                          000210/03   47       GROUP       55
     05    STREET                                               000210/03   20       AN          56
     05    CITY                                                 000212/03   20       AN          57
     05    STATE                                                000214/03    2       ALPHA       58
     05    ZIP-CODE                                             000214/05    5       NUMERIC     59
     03    JOB-ID                                               000215/00    5       AN          60
     03    DEPT                                                 000215/05    3       NUMERIC     61
     03    DIV                                                  000215/10    3       NUMERIC     62
     03    HIRE-DATE                                            000216/01    6       NUMERIC     63
     03    LOCATION                                             000216/07    3       NUMERIC     64

                          *** PROCEDURE MAP (ADDR IN OCTAL) ***
                 OPEN-FILES                    ADDR=000004                              LNR=66
                 READ-CARDS                         000014                                69
                 INPUT-ERROR                        000115                                84
                 BAD-RECORD                         000124                                88
                 CLOSE-FILES                        000134                                91

                                        *** END MAP ***
```

Figure 11-8. Data Map Listing

If a program is not compiled and executed in the same job, execution can be accomplished in one of two ways. A program call control statement that specifies the logical file name of the file containing the object program causes the program to be executed. The second way to execute an object program requires the LOAD and EXECUTE control statements; the LOAD control statement specifies the logical file name of the file containing the object program and the EXECUTE control statement causes the program to be executed. More than one LOAD control statement is required when subprograms are loaded from more than one file. When the object program has been stored as a permanent disk file, the file must be attached by an ATTACH control statement before a program call control statement or a LOAD control statement can be executed.

Three of the parameters that can be specified in the program call control statement cause informative messages to be printed on the job dayfile following program execution. The file equivalence parameter allows one file name to be substituted for another file name during OPEN statement processing. Any characters in the program call control statement that are not recognized by the compiler are assumed to be user parameters; user parameters are processed within the logic of the COBOL 5 program.

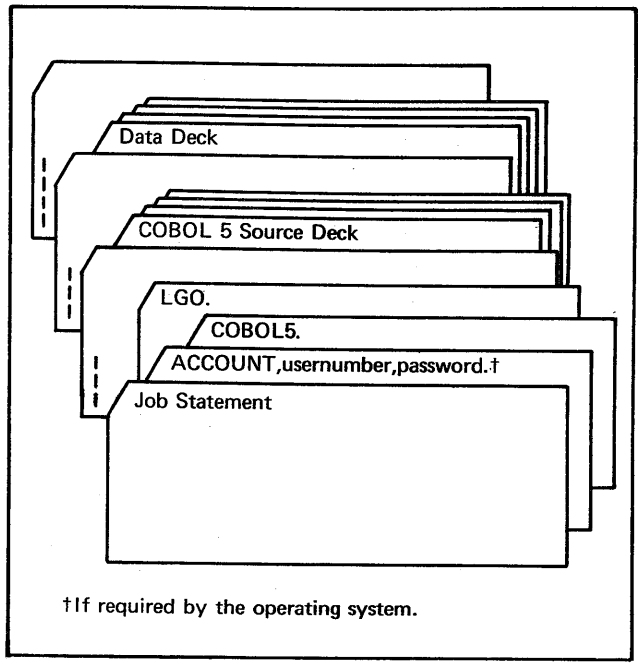
The *CORE, *MSGS, and *TIME parameters cause three different types of information to be displayed on the job dayfile. The maximum amount of central memory used during execution is displayed in octal when *CORE is specified. Messages issued by the Sort/Merge facility indicating the number of records processed are printed when *MSGS is specified. The amount of central processing time used during execution is displayed on the job dayfile when *TIME is specified.

An alternate file name can be specified for a file name already defined in the program; two file names are specified, on an operating system control statement, with a separating equals sign. The name to the left of the equals sign is the implementor-name specified in the ASSIGN clause of a File-Control entry. When the file is opened, the name to the right of the equals sign becomes the new logical file name of the file information table (FIT). Subsequent operating system control statements must specify the alternate file name. Parameters can be passed to a COBOL 5 program at execution time through the program call control statement.

The entire statement is considered as one parameter string of 80 characters. The string is made available to the COBOL program by entering the C.GETEP routine. When the routine is entered, the entire 80-character field is returned to the data item specified in the ENTER statement. The data item must be alphanumeric and should be 80 characters in length. Truncation of the parameter string on the right occurs when the item is less than 80 characters. If the item is larger than 80 characters, it is blank filled. Once the parameter string has been transferred to the data item, it can be separated into smaller data items by the UNSTRING statement and can be used within the program logic. The UNSTRING statement is discussed in section 7, Character Handling.

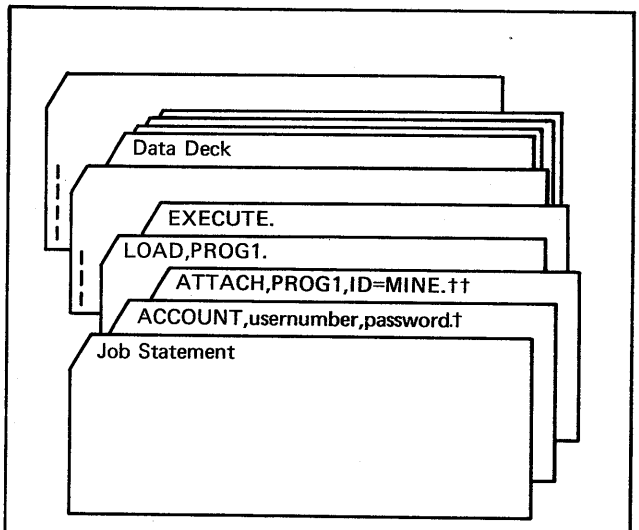
SAMPLE DECK STRUCTURES

Four sample deck structures are shown in figures 11-9 through 11-12. The control statements other than the COBOL5 control statement are described in detail in the applicable operating system reference manual.



†If required by the operating system.

Figure 11-9. Compiling and Executing a COBOL 5 Source Program



†If required by the operating system.
††Use the permanent file ATTACH format appropriate to the operating system.

Figure 11-10. Executing a COBOL 5 Object Program

The NOS/BE job deck structure shown in figure 11-9 is used to compile a source program and execute the resulting object program. The binary object program is written on the system file LGO. The program call control statement (LGO.) calls the object program into execution.

Figure 11-10 illustrates a NOS/BE deck structure that executes an object program generated by a previous compilation run. At compilation time, the object program was stored as a permanent disk file with the logical file name PROG1. The ATTACH and LOAD control statements specify the logical file name for the object program. The EXECUTE control statement causes the object program to be executed. The LOAD and EXECUTE control statements can be replaced by a program call control statement that specifies PROG1.

Compilation and execution of a COBOL 5 main program and a COBOL 5 subprogram are accomplished by the NOS job deck structure shown in figure 11-11. The COBOL5 control statement controls compilation of the main program and the subprogram. The binary output is written on the system file LGO.

The NOS job deck structure illustrated in figure 11-12 shows the compilation and execution of a COBOL 5 main program where the subprogram has been previously compiled. The LOAD control statement specifies that the binary object subprogram is to be loaded from the system file INPUT because the object subprogram resides on punched cards. The main program is called into execution by the program call control statement (LGO.).

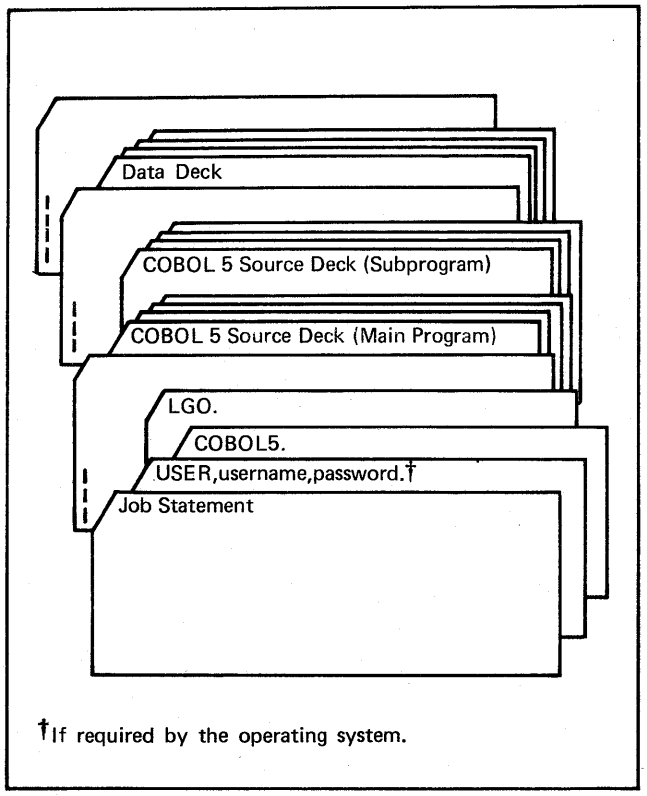


Figure 11-11. Compiling and Executing a COBOL 5 Main Program and a COBOL 5 Subprogram

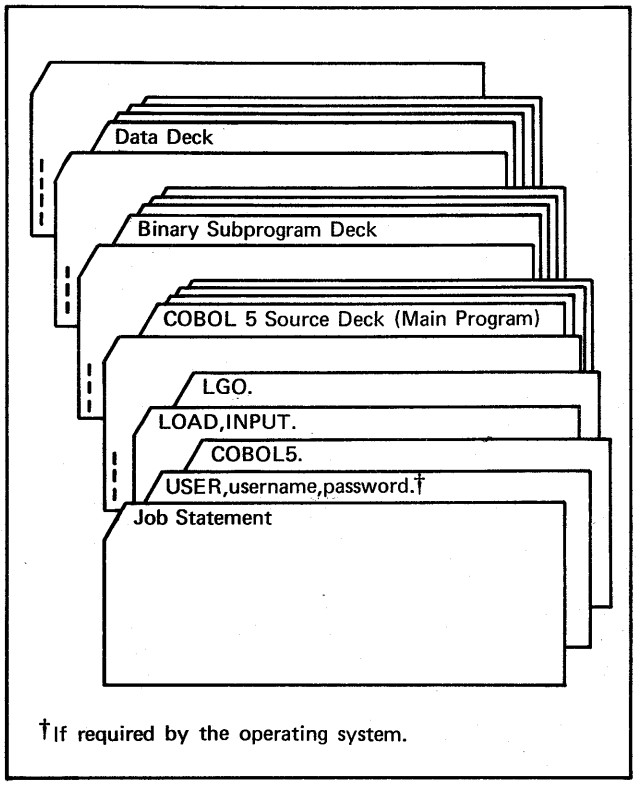


Figure 11-12. Compiling and Executing a COBOL 5 Main Program with a Previously Compiled Subprogram

A COBOL 5 source library contains source program entries and statements that can be copied into a COBOL 5 program when it is being compiled. Copying portions of a program from the library into a source program eliminates duplication of effort. Code that is identical in more than one COBOL 5 program, or that is easily modified for use in more than one program, can be stored on the library for subsequent access by the COBOL 5 compiler.

The Update utility program is used to create and maintain a COBOL 5 source library. The library generated by UPDATE is a random program library that is structured in groups of COBOL 5 source entries or statements. Each group is referred to as a deck and is identified by a unique name.

The COPY statement is used in the COBOL 5 source program to indicate the source library deck to be copied into the program. This statement can also specify certain modifications to the library deck. The entries or statements in the library deck replace the COPY statement when the source program is compiled.

CREATING A COBOL SOURCE LIBRARY

A COBOL 5 source library is created as a random program library through the Update utility program. Source decks are stored on the library in compressed symbolic format. A source deck consists of an entire program or a portion of a program.

The COBOL 5 source program entries and statements that are to be stored on the source library are grouped into decks. A deck usually contains only a portion of a program because an entire program is not usually copied. Each deck to be stored on the library is preceded by a DECK or COMDECK directive. This directive introduces a new source deck and specifies the name by which the deck is identified. The name can consist of up to nine characters that form a unique identifier for the deck. The source deck is terminated by the next DECK or COMDECK directive or by a 7/8/9 card (or its equivalent).

```
*DECK EMPREC1
```

This directive specifies that the name EMPREC1 is assigned to the source deck following the DECK directive. A COBOL 5 source program COPY statement that references EMPREC1 causes the source deck to be copied into the program.

The set of control statements preceding the library source file includes a job control statement, an UPDATE control statement, and the control statements necessary to make the library a permanent file. The specific control statements used for permanent files depend on the particular operating system and are described in the operating system reference manual. The UPDATE control statement is described in detail in the Update reference manual. The F and N parameters must be specified for creating the library; other parameters are included in the UPDATE control statement as applicable.

MAINTAINING A COBOL SOURCE LIBRARY

After a COBOL 5 source library has been created, the contents of decks can be changed, new decks can be added, and existing decks can be deleted through the Update utility program. Correction directives are used to update the source library; these directives are described in detail in the Update reference manual.

Correction directives that update the source library on a card basis are grouped into correction sets. A correction set begins with an IDENT directive, which specifies the identifier for the correction set. The Update program uses the identifier and adds a sequence number to uniquely identify each card affected by the correction set.

The source library can also be updated on a deck or correction set basis. Correction directives are provided to add new decks to the library and to temporarily or permanently remove decks or correction sets from the library.

ADDING NEW DECKS

The ADDFILE directive is used when a deck is being added to the source library. This directive specifies the file containing the new deck and the placement of the deck in the source library. If the new deck follows the ADDFILE directive in the Update input file, the file name can be omitted from the directive. The deck is placed after a specific card or deck in the source library by specifying in the ADDFILE directive a comma and either the identifier of the card or the name of the deck; if this parameter is not specified, the deck is added at the end of the file.

```
*ADDFILE FILEA, XYZ

*ADDFILE, XYZ
```

The first ADDFILE directive causes the deck (or decks) on the file named FILEA to be added to the source library immediately following the deck named XYZ. The second ADDFILE directive indicates that the UPDATE input file contains the deck to be added after the deck named XYZ.

INSERTING NEW CARDS

New cards are inserted into existing decks by the INSERT and BEFORE directives. The directive designates the card before or after which the new cards are to be inserted by specifying the card identifier. The INSERT directive causes cards to be inserted after the specified card while the BEFORE directive causes cards to be inserted before the specified card. The new cards must immediately follow the INSERT or BEFORE directive on the input file.

```
*INSERT XYZ.12
```

This directive causes the cards following it to be inserted into the source library immediately after the card with the identifier XYZ.12.

```
*BEFORE SET1.5
```

The cards following this directive are inserted into the source library immediately preceding the card with the identifier SET1.5.

DELETING CARDS FROM DECKS

One or more cards are deleted from a deck by the DELETE directive. This directive causes the temporary removal of cards; the cards can be restored to the deck at a later time. The deleted cards can be replaced by new cards; if this is desired, the new cards must immediately follow the DELETE directive in the input file.

A single card is deleted by specifying the identifier of the card in the DELETE directive. A block of cards is deleted by specifying the identifiers of the first and last cards to be deleted; all cards with identifiers in the specified range are deleted. Any new cards following the DELETE directive replace the deleted cards.

```
*DELETE XYZ.5, XYZ.10
```

This directive causes the cards with identifiers XYZ.5 through XYZ.10 to be deleted from the deck. Because this is a temporary deletion, the cards can be restored to the deck by a subsequent correction set.

RESTORING CARDS TO DECKS

Cards that have been deleted by the DELETE directive are restored to the deck by the RESTORE directive. This directive causes a deleted (inactive) card or block of cards to be reactivated. If new cards are to be inserted immediately after the last reactivated card, the new cards must follow the RESTORE directive.

The RESTORE directive specifies the identifier of a single card to be reactivated or the range of identifiers for a block of cards to be reactivated. When a range of identifiers is specified, any active cards within the range are not affected by the RESTORE directive. Any new cards following the RESTORE directive are inserted after the last card reactivated.

```
*RESTORE XYZ.5, XYZ.8
```

This directive causes the cards with identifiers XYZ.5 through XYZ.8 to be reactivated. New cards following the RESTORE directive are inserted after the card with identifier XYZ.8.

REMOVING CORRECTION SETS

The effects of a correction set are removed by the YANK, SELYANK, PURGE, and SELPURGE directives. The YANK and SELYANK directives are temporary removals; the correction sets can be reactivated at a later time. The PURGE and SELPURGE directives permanently remove correction sets; a purged correction set cannot be reactivated.

The YANK directive specifies the correction sets to be removed; individual correction sets or a range of correction sets can be specified in one directive. Cards that were deactivated by the correction sets are reactivated; cards that were activated by the correction set are deactivated. All source library cards affected by the correction set are affected by the YANK directive.

```
*YANK CSET1
```

This directive removes the effects of the correction set with the identifier CSET1. All cards with identifiers beginning with CSET1 are changed.

The SELYANK directive operates in the same way as the YANK directive except that the correction set is removed from the specified deck only. Other decks that were changed by the correction set are not affected by the SELYANK directive.

```
*SELYANK XYZ.CSET1
```

The effects of the correction set with the identifier CSET1 are removed from the deck named XYZ. Other decks in the source library are not changed.

The PURGE directive permanently removes the specified correction sets; individual correction sets or a range of correction sets can be specified. A correction set is purged from all decks. Once a correction set has been purged, it cannot be reactivated.

```
*PURGE CSET5
```

This directive causes the correction set with the identifier CSET5 to be permanently removed. It is no longer accessible.

The SELPURGE directive is the same as the PURGE directive except that a correction set is purged only from the specified deck. Other decks in the source library are not affected by the SELPURGE directive.

```
*SELPURGE ABC.CSET4
```

This directive purges from the deck named ABC all cards that belong to the correction set with the identifier CSET4. Only the deck ABC is affected by this SELPURGE directive.

REMOVING DECKS

Complete decks are removed from the source library by the YANKDECK and PURDECK directives. The YANKDECK directive temporarily removes decks while the PURDECK directive permanently removes decks.

The YANKDECK directive deactivates all cards within the specified decks. A deactivated deck can be reactivated at a later time.

```
*YANKDECK XYZ
```

The deck named XYZ is deactivated by this directive. All cards in the deck are deactivated regardless of the correction set to which they belong.

The PURDECK directive permanently removes a deck or group of decks from the source library. All cards in the deck are purged regardless of correction set identifiers.

```
*PURDECK ABC
```

This directive causes the deck named ABC to be permanently removed from the source library. It cannot be reactivated by a subsequent correction set.

USING A COBOL 5 SOURCE LIBRARY

Decks in a COBOL 5 source library are incorporated into the source program when COPY statements are executed. The library deck can be copied exactly as it exists on the library or with certain modifications specified in the COPY statement. The entire COPY statement, which can appear anywhere in the source program, is replaced by the copied deck.

The source library containing the decks to be copied must be attached before the source program is compiled. The X parameter in the COBOL5 control statement specifies the default source library for COPY statements that do not specify a library name.

When a COPY statement is encountered in the source program, the specified library deck is copied into the program. If the library deck does not reside on the default source library, the COPY statement must specify the library name.

```
COPY EMPREC1 IN MYLIB.
```

When this statement is encountered, the library deck named EMPREC1 is copied into the source program exactly as it appears on the library file. The deck is contained in the source library named MYLIB.

The REPLACING option is included in the COPY statement to specify the changes to be made when the library deck is copied into the program. This option contains one or more sets of operands separated by the keyword BY. The first operand in a set is the library text that is to be changed; the second operand is the text that is to replace the library text.

```
COPY EMPREC1 IN MYLIB
    REPLACING EMP-REC BY SORT-REC.
```

Execution of this statement causes the library deck named EMPREC1 to be copied into the source program. The data-name EMP-REC in the library deck is changed to SORT-REC in the source program.

An operand in the REPLACING option can be a series of data-names, reserved words, and literals called pseudo-text. In a COPY REPLACING statement, a complete COBOL word, literal, picture character-string, or comment entry must be referenced. This provides the means to change a complete entry or statement in the library deck. When pseudo-text is specified as the first operand in a set, library text that is the same as the pseudo-text is replaced by the second operand. The second operand in a set can also be specified as pseudo-text. Two contiguous equal signs are used to delimit pseudo-text.

```
COPY TOTALS REPLACING
    ==SUBTRACT DISCOUNT FROM ACCUM==
    BY ==SUBTRACT DISCOUNT FROM ACCUM
    GIVING AMOUNT==.
```

When this statement is executed, the library deck named TOTALS is copied into the source program with one modification. The library text SUBTRACT DISCOUNT FROM ACCUM is changed in the source program to SUBTRACT DISCOUNT FROM ACCUM GIVING AMOUNT.

COBOL 5 contains three features that aid the programmer in debugging a COBOL program. These features are:

- The debugging feature is used to monitor specific files, data items, and procedures during program execution. With this feature, debugging sections are executed when specific conditions occur.
- The paragraph trace feature is used to trace the flow of a program during execution. With this feature, information is written to a trace file that can later be printed.
- The termination dump feature is used to obtain a listing of the contents of all data items, or of selected data items in the program. With this feature, information is written to an output file which can be printed upon program execution.

Snap-shot dumps can also be taken at any time during execution. With this feature, an informative map is produced regardless of whether the program terminates normally or abnormally.

DEBUGGING FEATURE

The debugging feature provides two types of debugging procedures that can be included in the COBOL 5 source program. Debugging lines, which can appear in various parts of the program, are always executed if the lines are compiled as executable code. Debugging sections, which can appear only as declarative sections, are executed only when the debugging option is selected at execution time.

Executing debugging lines and debugging sections generally results in a longer execution time. The activation or deactivation of the debugging feature is specified through compile-time and execution-time switches that are controlled by the programmer.

DEBUGGING LINES

When a program is being coded, debugging lines are included anywhere in the program after the OBJECT-COMPUTER paragraph. A debugging line is indicated by the letter D in the Indicator Area (column 7). The content of a debugging line must be syntactically correct as a line of executable code whether or not the debugging feature is activated. For example, a debugging line in the Data Division must conform to the rules for a Data Division entry.

Debugging lines are compiled as executable code only when the debugging feature is activated during compilation. Whenever the object program is executed, the debugging lines are executed. If the debugging feature is not activated during compilation, debugging lines are compiled as comment lines.

DEBUGGING SECTIONS

Debugging sections are specified in the Declaratives portion of the Procedure Division. A debugging section is a procedure that is executed for the file, data item, or procedure being monitored. Each section has an associated USE FOR DEBUGGING statement.

The USE FOR DEBUGGING statement identifies what is being monitored. It can specify a file-name, a data-name, or a procedure-name, or it can specify that all procedures are to be monitored. The paragraphs following the USE statement specify the procedure to be executed at the appropriate time.

```
PROCEDURE DIVISION.
DECLARATIVES.
DEBUG-PROC SECTION.
    USE FOR DEBUGGING ON ALL PROCEDURES.
DPROC-1.
    .
    .
    .
END DECLARATIVES.
```

This USE statement specifies that the debugging section is to be executed for all procedures in the program. The statements in the paragraph DPROC-1 and any other paragraphs in the section DEBUG-PROC are executed whenever a procedure is executed.

The debugging feature must be activated during compilation in order to compile debugging sections as executable code. Subsequent execution of the debugging sections is then controlled by the setting of switch 6 at execution time; if switch 6 is not turned on, the debugging sections are not executed. If the debugging feature is not activated when the program is compiled, the debugging sections are compiled as comments and cannot be executed; the setting of switch 6 then has no effect on the debugging sections.

Monitoring Data Items

One or more data items can be specified in a USE FOR DEBUGGING statement. The debugging section is executed in conjunction with the execution of a statement referencing the data item. The order of execution depends on the statement referencing the data item and whether or not the ALL REFERENCES OF phrase is specified in the USE statement:

- RELEASE, REWRITE, or WRITE statement; USE statement with or without the ALL REFERENCES OF phrase.

  The debugging section is executed before the statement is executed; if the statement includes the FROM phrase, the debugging section is also executed after the move operation is performed.

- PERFORM statement with the VARYING, AFTER, or UNTIL phrase; USE statement with or without the ALL REFERENCES OF phrase.

  The debugging section is executed after each initialization, modification, or evaluation of the data item.

- GO TO statement with the DEPENDING ON phrase; USE statement with the ALL REFERENCES OF phrase.

  The debugging section is executed before control is transferred to the procedure and before any debugging section associated with that procedure is executed.

- All other statements; USE statement with the ALL REFERENCES OF phrase.

  The debugging section is executed after the statement is executed.

- All other statements; USE statement without the ALL REFERENCES OF phrase.

  The debugging section is executed after the statement is executed only if the content of the data item changed during execution.

Monitoring Procedures

The USE FOR DEBUGGING statement can name one or more procedures to be monitored or it can specify that all procedures are to be monitored. When an individual procedure is monitored, the debugging section is executed before the specified procedure is executed; an ALTER statement that references the procedure causes the debugging section to be executed after the ALTER statement is executed.

If the USE statement specifies ALL PROCEDURES, the debugging section is executed before each procedure is executed. (Procedures in debugging sections are not monitored.) The debugging section is also executed after an ALTER statement referencing a procedure is executed. When ALL PROCEDURES is specified, individual procedures cannot be named for monitoring in a USE statement.

Monitoring Files

One or more files can be specified for monitoring in a USE FOR DEBUGGING statement. The execution of a statement that references a file being monitored causes the debugging section to be executed.

- OPEN, CLOSE, DELETE, or START statement.

  The debugging section is executed after the statement is executed.

- RETURN statement not resulting in an at end condition.

  The debugging section is executed after the statement and any other specified USE procedures are executed.

- READ statement not resulting in an at end or invalid key condition.

  The debugging section is executed after the statement and any other specified USE procedures are executed.

DEBUGGING REGISTER

When debugging is activated during program execution, the special register DEBUG-ITEM is automatically generated by the compiler. Each time a debugging section is executed, DEBUG-ITEM is updated with information related to the condition that caused the debugging section to execute.

The special register DEBUG-ITEM contains four types of information:

- The line number of the source statement causing execution of the debugging section.
- The file-name, procedure-name, or data-name causing execution of the debugging section.
- If the item being monitored requires subscripting or indexing, the occurrence number for each applicable level of subscripting or indexing.
- The content of the data item being monitored or a phrase related to the procedure being monitored (for example, SORT INPUT).

The information in DEBUG-ITEM can be output to the printer. The printed information can be inspected to determine whether or not the data is being processed properly.

ACTIVATING DEBUGGING AT COMPILE TIME

The debugging feature is activated during compilation in one of two ways. The DEBUGGING MODE clause can be specified in the SOURCE-COMPUTER paragraph of the Environment Division or the DL option of the DB parameter can be specified in the COBOL5 control statement. If neither the clause nor the parameter is specified, debugging lines and debugging sections are compiled as comments rather than as executable code.

ACTIVATING DEBUGGING AT EXECUTION TIME

The debugging feature cannot be activated at execution time unless it was activated at compilation time. Debugging lines that have been compiled as executable code are always executed when the program executes. Execution of debugging sections, however, depends on the setting of a compile-time switch, switch 6. The switch must be turned on to activate execution of debugging sections.

Switch 6 is set either within the program by a SET statement or external to the program by a control statement. If switch 6 is controlled internally, it must be assigned a name in the SPECIAL-NAMES paragraph of the Environment Division. The switch can then be set on or off by a SET statement.

```
SPECIAL-NAMES.
    SWITCH-6 IS DEBUG-SWITCH.
    .
    .
    .
PROCEDURE DIVISION.
    SET DEBUG-SWITCH TO ON.
```

When this SET statement is executed, switch 6 is set to on and debugging sections are executed.

The SWITCH control statement can also be used to set switch 6 before the program executes. Execution of a SWITCH control statement reverses the current setting (on or off) of the switch. At the start of the job, switch 6 is off.

```
SWITCH,6.
```

The first time the SWITCH control statement is executed, switch 6 is turned on. Execution of a second SWITCH control statement turns off switch 6 (reverses the setting).

PARAGRAPH TRACE FEATURE

The paragraph trace feature is selected through the DB parameter in the COBOL5 control statement (DB=TR). This feature produces object code that traces the flow of the program during execution. Use of the paragraph trace feature results in slightly slower program execution speeds and larger program field length; however, the default field length is sufficiently large to accommodate loading.

When the paragraph trace feature is selected, messages are written on a trace file. These messages indicate the name of a paragraph and the time the paragraph was executed.

SOURCE PROGRAM STATEMENTS

The control statement parameter selects the paragraph trace feature and causes the object code to be generated; however, the trace must be initiated by an ENTER statement in the source program. The ENTER statement can be used to turn the paragraph trace feature on, turn it off, or reinitialize it.

The paragraph trace feature is turned on by the following statement in the source program:

```
ENTER "C.ONTR".
```

When this statement is executed, the trace is turned on. The first record written on the trace file contains the following message:

```
**** TRACE ON FROM   LINE nnnnn
```

The next record written on the trace file contains the name of the next paragraph entered.

The paragraph trace feature is turned off by the following statement in the source program:

```
ENTER "C.OFFTR".
```

Execution of this statement turns off the trace. A record is written on the trace file with the following message:

```
**** TRACE OFF FROM LINE nnnnn
```

No further tracing is performed until the trace feature is turned on again.

The trace file is closed by the following statement in the source program:

```
ENTER "C.STPTR".
```

When this statement is executed, a final record is written on the trace file and the file is then closed and rewound. The record written on the trace file contains the following message:

```
**** TRACE CLOSED    LINE nnnnn
```

If the trace file is not closed by an ENTER statement in the source program, execution of the STOP RUN statement automatically calls the routine C.STPTR. The trace file can be processed by the source program after the ENTER "C.STPTR" statement is executed.

TRACE FILE

Two types of records are written on the trace file, which has the logical file name COBTRFL. Each record contains 50 characters; the CYBER Record Manager record type is Z with C type blocking. The file is described in the source program only if it is processed by the program. The job is responsible for preserving the file when the program ends.

A COBOL 5 description of the trace file is shown in figure 13-1. The first record described is a paragraph trace message that is written on the file each time a paragraph is executed. The second record is a status message that is written on the file whenever the trace feature is turned on or off, or when the trace file is closed.

```
.
.
.
SELECT TRFILE ASSIGN TO COBTRFL
    USE "RT=Z".
.
.
.
FD  TRFILE LABEL RECORDS ARE OMITTED.
01  TRREC.
    02  PARA-NAME   PICTURE X(30).<----Paragraph name.
    02  CP-TIME     PICTURE 9(10).<----Central processor time used since start of job.
    02  TR-NUMBER   PICTURE 9(8).<----Consecutive number.
    02  FILLER      PICTURE XX.<----Zero bytes.
01  TRMSREC.
    02  STARS       PICTURE XXXX.<----Four asterisks, if this is a message record from a
                                      directive.
    02  FILLER      PICTURE X(26).<----Message.
    02  CP-TIME     PICTURE 9(10).<----Central processor time used since start of job.
    02  TR-NUMBER   PICTURE 9(8).<----Consecutive number.
    02  FILLER      PICTURE XX.<----Zero bytes.
```

Figure 13-1. Trace File Format

TERMINATION DUMP FEATURE

The termination dump feature, along with the C5TDMP control statement, is selected through the TDF parameter in the COBOL5 control statement. This feature produces a listing of every data item within the COBOL program, as well as the value and data type of each item. A dump can be taken regardless of whether the program terminates normally or abnormally.

When the termination dump feature is selected, information needed for the dump is written to a termination dump file at compilation time. Following execution, this file is used along with a system file to produce the dump listing.

OBTAINING A TERMINATION DUMP

The TDF parameter in the COBOL5 control statement selects the termination dump feature and causes extra information required for the dump to be generated by the compiler; however, the dump is produced only if the C5TDMP control statement is specified after program execution.

At compilation time, tables to be used in producing a dump are written to the file specified by the TDF parameter; if no file is specified, the tables are written to the file TDFILE. At execution time, the system file ZZZZZ4P, which contains the data values at any given time, is produced. When the program completes execution, the C5TDMP utility uses both TDFILE (or the file specified) and the system file to produce a dump listing of the program data items. The dump is taken of the main program and any COBOL 5 subprograms that are specified on an FDL file and are in memory when execution terminates. The system file is returned at the completion of C5TDMP. If successive dumps are desired, the file must be saved prior to execution of the C5TDMP control statement.

Four optional parameters can be included in the C5TDMP control statement:

T Specifies the file from which compiler information is to be obtained. The file named by the T parameter must be the same as that named by the TDF parameter; if omitted the default file is TDFILE.

L Indicates the file on which the termination dump is to be written; if omitted, the default file is OUTPUT.

I Specifies the directive file to be used to select specific data-names or the number of occurrences of a data-name to be included in the dump. If I is specified without a logical file name, the file INPUT is assumed to contain the directives. If C5TDMP is entered through a terminal, a prompt (question mark) appears and the directives can then be entered.

NA Specifies that no array items are to be listed in the dump. Duplicate items within an array are indicated.

Figure 13-2 illustrates a COBOL program that uses the termination dump. The COBOL5 control statement used to compile the program includes the TDF and the DB=TR parameters. In the example in figure 13-2, the ENTER "C.ONTR" statement turns the trace on, and the ENTER "C.SNAP" statement calls the snap dump routine. Without the ENTER "C.ONTR" statement, the PARAGRAPH TRACE-BACK portion of the output shown in figure 13-2 is not produced. Instead, the dump shows the last I-O statement that executed before the dump.

TERMINATION DUMP LISTING

The termination dump listing contains an entry for each elementary item defined in the Data Division. The data items are listed in the order of section appearance. The termination dump shows the data name, usage or class, and value of each item. Figure 13-2 illustrates the termination dump listing generated (following execution of the COBOL 5 program) by using the C5TDMP utility from a terminal. In the C5TDMP control statement, the characters =INPUT can be omitted since INPUT is the default file. (Underlining, in figure 13-2, indicates terminal user input.)

CONTROL STATEMENT DEBUGGING OPTIONS

Two optional parameters in the COBOL5 control statement can be used to select certain debugging features. The termination dump feature is chosen by the TDF parameter, which has been previously described in this section. One or more of the following values can be specified in the DB parameter:

B Causes binary output to be produced regardless of errors in the source program.

DL Causes debugging lines and sections to be compiled as object code; has the same effect as the WITH DEBUGGING MODE clause.

SB Causes subscripts and indexes to be checked for values within the designated bounds.

TR Causes execution flow of the program to be traced.

If the parameter name (DB) is specified alone, the DL, SB, and B values are selected by default.

The DL and TR options have been previously discussed in this section. The B and SB parameters are described in the following paragraphs.

BINARY OUTPUT

Binary executable code is not produced when level F or C errors are encountered during compilation. For debugging purposes, the B option of the DB parameter causes this code to be produced regardless of errors in the source program. Level C or F errors in the source program result in compilation of a call to an execution time abort routine; execution of the code aborts the program.

SUBSCRIPT AND INDEX CHECKING

The SB option of the DB parameter causes subscript and index references to be checked during execution. These references are checked to ensure that they are within the declared bounds. Detection of out-of-bounds references aborts the program; a dayfile message identifies the line of incorrect reference.

A. **COBOL5 Compiler Call Statement**

```
cobol5,i=tdum5p,tdf,db=tr
```

B. **Program Listing**

```
1                IDENTIFICATION DIVISION.
2                PROGRAM-ID.  TDUMP.
3                ENVIRONMENT DIVISION.
4                CONFIGURATION SECTION.
5                SOURCE-COMPUTER. CYBER-170.
6                OBJECT-COMPUTER. CYBER-170.
7                DATA DIVISION.
8                WORKING-STORAGE SECTION.
9                01 STOR.
10                   02 ST1    PIC X(5) VALUE "AB-15".
11                   02 ST2    PIC 9 VALUE 0.
12                   02 ST3    PIC 9 VALUE 0.
13               01 TBL.
14                   02 TABL OCCURS 5 PIC 9.
15               PROCEDURE DIVISION.
16               STRT.
17                   ENTER "C.ONTR".          <-- Turns on trace.
18                   PERFORM A1.
19                   PERFORM A2 5 TIMES.
20                   ENTER "C.SNAP" USING "SNAP 1".   <-- Causes snap dump.
21                   MOVE "AB-17" TO ST1.
22                   PERFORM A3
23                       VARYING ST3 FROM 1 BY 1
24                       UNTIL ST3 GREATER 3.
25                   MOVE "56789" TO TBL.
26                   STOP RUN.
27               A1.
28                   MOVE "AB-16" TO ST1.
29               A2.
30                   ADD 1 TO ST2.
31                   MOVE ST2 TO TABL ( ST2 ).
32               A3.
33                   DISPLAY "TABLE IS-" TABL ( ST3 ).
```

C. **Program Output**

```
TABLE IS- 1
TABLE IS- 2
TABLE IS- 3
```

D. **C5TDMP Control Statement and Selection Criteria**

```
/c5tdmp,i=input
? select st1 st2 occurrences 2 to 3 of tabl
```

Figure 13-2. COBOL Program with Termination Dump (Sheet 1 of 2)

E. Termination Dump Output

```
1CDC COBOL 5.3 - LEVEL 518  TERMINATION DUMP


 SELECT ST1 ST2 OCCURRENCES 2 TO 3 OF TABL
1CDC COBOL 5.3 - LEVEL 518  TERMINATION DUMP


 SNAP DUMP -         SNAP 1

 --------- PARAGRAPH TRACE-BACK --------
           A1
           A2
           A2
           A2
           A2
           A2

 LAST LINE EXECUTED          20


 --------- P R O G R A M - I D -------- TDUMP


 DATA NAME                        TYPE      CONTENTS

 ST1                               AN   AB-16
 ST2                               CP   5
 TABL
  OCCURRENCE      2                CP   2
                  3                     3




1CDC COBOL 5.3 - LEVEL 518  TERMINATION DUMP    OF TDUMP


 --------- PARAGRAPH TRACE-BACK --------
           A1
           A2
           A2
           A2
           A2
           A2
           A3
           A3
           A3

 LAST LINE EXECUTED          26


 --------- P R O G R A M - I D -------- TDUMP


 DATA NAME                        TYPE      CONTENTS

 ST1                               AN   AB-17
 ST2                               CP   5
 TABL
  OCCURRENCE      2                CP   6
                  3                     7
```

Figure 13-2. COBOL Program with Termination Dump (Sheet 2 of 2)

COBOL 5 interfaces with the CYBER Database Control System (CDCS) through the subschema facility. This allows the program to perform input/output operations on data base files. Data and data descriptions are shared by applications programs that utilize the CDCS interface.

Entering the data base environment presents some new terminology and processing concepts. These are described briefly before discussing COBOL 5 processing of data base files. Detailed information is contained in the CDCS 2 reference manual and the DDL 3 reference manual, volume 2. Refer to section 10 for detailed information on the subprogram interface with CDCS.

DATA BASE CONCEPTS

A centralized data base allows multiple applications programs to access the same data, thus eliminating redundant storage of data. Several files are consolidated into the data base. An applications program can then process one or more of the files (areas, in data base terminology), which are described in a manner that meets the requirements of the program. The words file and area are used interchangeably in this section.

The COBOL 5 program uses conventional input/output statements to process data base files. The data base access requests are monitored and interpreted by CDCS, which is the controlling interface between the program and the data base. CDCS executes calls to Advanced Access Methods (AAM) to perform input/output operations.

The schema describes the complete data base; a COBOL subschema describes the portion of the data base to be processed by an applications program. The creation of the schema and subschemas, which is not normally performed by the applications programmer, is not discussed in this guide.

THE SCHEMA

The entire data base is described by a schema saved in a permanent file. The schema defines the structure and organization of the files (areas in data base terminology) and describes the characteristics of the data in each record type applicable to an area. In addition, the schema can join areas that have common data items.

The file organization for an area can be extended indexed, extended direct, or extended actual-key. All areas have a primary key and can have one or more alternate keys.

COBOL SUB-SCHEMAS

The portion of the data base to be made available to a COBOL 5 program is described in a COBOL subschema. The program can process only those areas specified in the subschema. For a specified area, the subschema describes the structure and content of each applicable record type. The description does not necessarily have to be the same as the schema description; data format can be changed to satisfy program requirements. The subschema also indicates the relations that can be accessed by the applications program.

Any number of subschemas can exist for a data base; however, a COBOL 5 program can utilize only one subschema. The availability of multiple subschemas allows record structures and individual data items to be defined according to the needs of different programs. The subschemas reside on a permanent file subschema library that must be attached by the user for program compilation.

The subschema, as well as the schema, is created through the Data Description Language (DDL) compiler. The DDL source listing for the subschema can be used to obtain data-names and other information needed to process data base files. The listing is similar to the Data Division in a COBOL source listing. The COBOL programmer uses the RECORD DIVISION of the subschema listing to obtain data item descriptions. The RELATION DIVISION is used to obtain the names of areas in a relation. If a subschema source listing is not available, data item descriptions can be obtained by using the LO=M parameter on the COBOL5 control statement. Data items in the REPORT SECTION are not included when the parameter is used.

RELATIONS

Areas that have a common data item can be joined together in a relation. This allows the COBOL 5 program to access several areas with one read operation. All relations are defined in the schema; the subschema indicates the specific relations that can be accessed by the program. Records retrieved in a relational read operation can be restricted to those records that meet the qualification criteria specified in the subschema.

An area in a relation is joined to another area through an identical data item in both areas. A relation can have several areas joined together where one area is joined by a data item to another area that, in turn, is joined by a different data item to still another area. When a relational read operation is performed, a record from each area in the relation is available to the COBOL 5 program. Relational reading is discussed in detail later in this section.

The subschema can specify qualification criteria for a relation. Record retrieval from any of the areas in the relation can be restricted by qualification. When qualification is specified, only those records that satisfy the specified criteria are read from the area. This eliminates the need for the COBOL 5 program to test for various conditions before processing the records.

PROCESSING AN AREA

Input/output processing of a data base area is the same as it is for conventional files. Before the area can be processed, it must be opened. Records in the area can then be read, written, deleted, and rewritten with the standard COBOL 5 statements. File access can be controlled by defining specific passwords in the schema; the COBOL

program must include the proper password to access specific files. The START statement can be used to position the area for subsequent sequential reading. The area must be closed before the program terminates.

THE DATA BASE STATUS BLOCK

The DB$DBST routine can be used to obtain extensive data base status and error information. (The DB$DBST routine combines the features of the C.DMRST and the C.IOST routines.) The COBOL application program defines a central memory area that is automatically updated by CDCS after every operation on a data base file. One call to DB$DBST establishes the status block location in memory; no further calls are needed. After any input/output operation, the information returned to the data base status block is available for use by the COBOL programmer.

Figure 14-1 illustrates a COBOL description of a data base status block. Items must appear in the order shown. If any item is defined in the data base status block, all previous items must be defined and sufficient length must be specified. The status block length must be at least one word, and not greater than 11 words. COMP-1 must be used for the fields indicated in figure 14-1.

```
01 DATABASE-STATUS-BLOCK.
    02 DATABASE-STATUS PIC 9(5) COMP-1.
    02 AUX-STATUS.
      03 ITEM-ORDINAL PIC 9(5) COMP-1.
      03 FILE-POSITION PIC 9(3) COMP-1.
      03 FILLER PIC X(10).
    02 DB-FUNCTION PIC A(10).
    02 REL-RANK-STATUS.
      03 DB-REL-ERROR PIC 9(3) COMP-1.
      03 DB-REL-CTLBK PIC 9(3) COMP-1.
      03 DB-REL-NULL PIC 9(3) COMP-1.
    02 DB-REALM PIC X(30).
```

Figure 14-1. Data Base Status Block Description

- The item in the first word, DATABASE-STATUS, is defined for the return of the CRM or CDCS error code. Codes of 384 or greater can be found in the CDCS 2 reference manual. Codes lower than 384 can be found in the AAM reference manual. The decimal codes returned by DB$DBST must be converted to octal to correspond to the codes listed in the AAM reference manual.

- The next three-word item, AUX-STATUS, is defined for the return of CDCS subschema item level errors and for file position codes. For example, if a RECORD MAPPING ERROR (code 445) occurs, the item ordinal in the first word of AUX-STATUS indicates the item causing the error. The item ordinals appear on the subschema listing.

  File position codes are taken from the FP field of the file information table (FIT). For example, FILE-POSITION contains the decimal codes:

  16 when a record is sucessfully returned during a read operation.

  8 when the end of a keylist is reached on a read by alternate key.

  64 when end-of-information is reached.

  All three words must be defined for any information to be returned to the AUX-STATUS field.

- The one-word item DB-FUNCTION is defined for the return of the characters indicating the file function being performed (such as OPEN, CLOSE, DELETE, LOCK).

- All three words of REL-RANK-STATUS must be defined for any information to be returned to the field.

- The one-word item DB-REL-ERROR is defined for the return of the rank (in a relation operation) on which a CYBER Record Manager (CRM) or CDCS error occurred.

- The one-word item DB-REL-CTLBK item is defined for the return of the lowest numbered rank (in a relation operation) on which a control break occurred.

- The one-word item DB-REL-NULL is defined for the return of the lowest numbered rank (in a relation operation) for which there is a null record.

- The three-word item DB-REALM is defined for the return of the name of the realm or area on which an error occurred.

The DB$DBST routine can be called at any point in a COBOL program as follows:

```
MOVE 11 TO BLK-LENGTH.
.
.
.
ENTER "DB$DBST" USING DB-STATUS-BLOCK,
    BLK-LENGTH
```

The ENTER statement communicates the length (BLK-LENGTH) of the field and the location of the field (DB-STATUS-BLOCK) in which the data base status information is to be returned. BLK-LENGTH must be defined as a COMP-1 data item. In the preceding example, the value 11 is moved to BLK-LENGTH prior to the ENTER statement.

COMMON CDCS DIAGNOSTICS

Table 14-1 contains some common CDCS non-fatal diagnostic codes (shown in decimal for the COBOL user) and corresponding messages. The codes are returned in the first word of the data base status block. The following paragraphs discuss the significance of the 426, 428, and 435 codes to the user:

- The 426 diagnostic code indicates that an open was attempted on a file already opened. This diagnostic can occur on the statements OPEN or CLOSE of a relation or a file.

- The 428 diagnostic code indicates that a close was attempted on a file already closed. This diagnostic can occur on the statements OPEN or CLOSE of a relation or a file.

- The 435 code indicates a deadlock condition; that is, a situation arising in concurrent data base access when two or more applications programs are contending for a resource that is locked by one of the other programs. The 435 code indicates that CDCS has unlocked all locks held by the COBOL program. The user should provide appropriate code to handle recovery from a deadlock. Refer to the CDCS 2 reference manual for more information on deadlocks.

The 423 diagnostic, PFM ERROR 002 AREA XXX, is a common fatal error that occurs while attaching files for data base processing under the NOS operating system. The user should refer to volume 2 of the NOS operating system reference manual for code 002.

TABLE 14-1. NON-FATAL CDCS DIAGNOSTIC CODES

Decimal Code	Message
385	VIOLATION OF CONSTRAINT xxx ON xxx OF RECORD xxx
386	CRM ERROR xxx ON AREA xxx IN CONSTRAINT PROCESSING
426	AREA xxx ALREADY OPEN
427	CRM ERROR xxx DURING xxx ON AREA xxx
428	AREA xxx NOT OPEN
431	INCORRECT RECORD TYPE DURING xxx, AREA xxx
432	KEY MAPPING ERROR DURING xxx, AREA xxx
435	DEADLOCK ON AREA xxx
436	ILLEGAL REQUEST ON AREA xxx, READ OR FILE LOCK REQUIRED BEFORE xxx
440	CANNOT CHANGE RECTYPE ON xxx, AREA xxx
443	xxx ABORT BY DBPROC xxx
445	RECORD MAPPING ERROR DURING xxx ON RECORD xxx

PROCESSING A RELATION

Relational processing greatly simplifies the programming when several related areas are required by the applications program. A single READ statement can be executed to make available a record from each area in the relation. In addition, selective retrieval of records in the relation can be performed by CDCS. Before discussing relational processing within the COBOL 5 program, it is necessary to examine the structure of a relation and the manner in which CDCS returns records to the program.

STRUCTURE OF A RELATION

Areas that are logically related can be joined together through common data items. The structure of a relation is defined when the schema is created. The subschema listing provides the names of the areas in the relation in the order they are joined.

A relation can be described as a hierarchical tree structure. The root of the tree is the area through which the relation is entered; this is the first area listed for a relation. A data item in the area at the root of the structure joins the area to a common data item in the next area listed for the relation. When the relation is entered, the value of the data item in the root area record leads to a record in the second area. More than one record in the second area can contain the same value; thus, one record in the root area can lead to several records in the second area.

The second area in the relation can be joined to a third area through a common data item. Once again, a record in the second area can lead to several records in the third area. This branching out from the root of the tree continues through each area in the relation.

Parent-child relationships exist. All records in one area that branch from the same record in another area are called children. A record that has one or more branches into a numerically higher ranked file is called the parent to the children records. The root area is numerically the lowest rank of the tree structure (rank 1); the last area in the relation is numerically the highest rank of the tree structure.

Each area joined in a relation is assigned a rank in the relation. Figure 14-2 illustrates the tree structure of a relation that joins three areas. The first area, which is the root of the structure, consists of a master record for each customer. The second area contains a summary record for each current order. The third area, the highest rank in the structure, contains a record for each line item in an order. The common data item joining the first area to the second area is the customer number; the order number is used to join the second area to the third area.

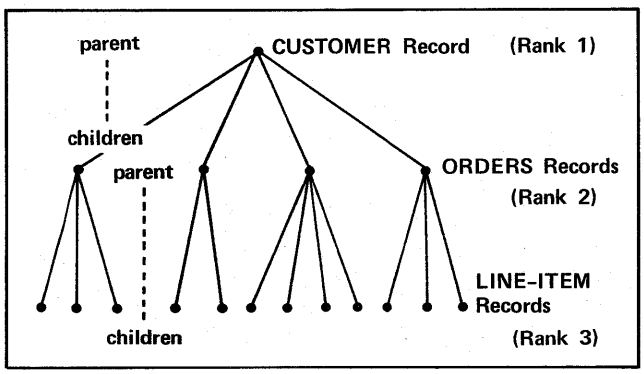


Figure 14-2. Tree Structure for a Three-Area Relation

CDCS RELATION PROCESSING

CDCS returns records to the COBOL 5 program when relational read operations are executed. One record from each area in the relation is available to the program for processing. All normal read operations can be performed on a relation.

A random read of a relation accesses the root area randomly by primary or alternate key. A record is retrieved from each area in the relation. The relation can then be read sequentially to retrieve additional records related to the record accessed by the random read.

As sequential relation reads are executed, CDCS retrieves records only from the area with the highest rank until all records related to the current record in the next lower rank have been read. In the example shown in figure 14-2, CDCS returns three records to the program for the first read of the relation: a CUSTOMER record, an ORDER record, and a LINE-ITEM record. The next two sequential reads cause CDCS to retrieve only the two LINE-ITEM records joined to the ORDERS record. For the next sequential read, CDCS retrieves the next ORDERS record

joined to the CUSTOMER record and the first LINE-item record joined to that ORDERS record. This process continues until all records related to the CUSTOMER record have been retrieved.

The status block created by the ENTER DB$DBST statement can be used to determine whether or not an error occurred for any area in the relation. Two conditions encountered during relational reads are also diagnosed by CDCS: a null record condition, and a control break condition. Information on a null record condition is returned by DB$DBST in the eighth word of the data base status block. Information on a control break condition is returned by DB$DBST in the seventh word of the data base status block.

A null record condition exists when no record in the specified area can be retrieved by CDCS. This occurs when either no record in that area is joined to the record at the next lower rank in the relation or no record in that area qualifies for retrieval. The null record condition can occur at any rank from the lowest (root area) to the highest.

If the null record condition exists in an area that is not the highest rank, the null record condition automatically exists in all areas higher in rank. If the relation shown in figure 14-2 is read and no ORDERS record is joined to the CUSTOMER record, a null record condition exists for both the ORDERS area and the LINEITEMS area. This is also true when ORDER records are joined to the CUSTOMER record but none of the records qualify for retrieval.

When a null record condition occurs, CDCS returns a record to the program consisting of a display code right bracket (]) in each character position.

The second condition that is encountered while reading a relation is a control break condition. During sequential reading of a relation, a control break occurs when CDCS retrieves a new child record for any area other than the highest ranked area. If the records in the relation shown in figure 14-2 are read sequentially, a control break condition occurs when the first and second ORDERS records are returned to the program.

The DB$DBST routine can be used to obtain extensive data base status and relation error information. Refer to the previous subsection entitled Processing an Area. An example of the DB$DBST routine is illustrated later in this section.

RECORD QUALIFICATION

Record qualification is the method by which the records returned to the program are restricted to records that satisfy specific requirements. This can greatly limit the number of records that CDCS returns to the program. Qualification criteria can be specified for records in any area in the relation.

The subschema specifies the record qualification criteria that must be satisfied in order to return a record to the COBOL 5 program. The Relation Division in the subschema listing indicates the requirements for record retrieval. A RESTRICT clause is included for each record type to be qualified for retrieval.

Qualification is specified in much the same way a relational condition is specified in a COBOL 5 program. The two operands, which are separated by a relational operator, are compared by CDCS. If the condition indicated by the relational operator exists for the record to be retrieved, CDCS returns the record to the program. The first operand is a data item in the record being qualified. The second operand can be another data item in the record, a literal, or a data item defined in the COBOL 5 program. If the second operand is a program data item, it must be initialized before reading the relation.

Qualification can be specified for more than one data item in the record. The logical operator connecting the qualification criteria determines the complete requirement for record retrieval. Both data items must qualify (logical operator AND), at least one data item must qualify (logical operator OR), or only one data item must qualify (logical operator XOR).

CDCS checks record qualification from the lowest ranked area to the highest ranked area. An unqualified record at any rank automatically causes all records joined to it in higher ranking areas to be disqualified.

The Relation Division of a COBOL subschema is shown in figure 14-3. Two RESTRICT clauses are included for the CREDIT-RISK relation. When the relation is read, a SLS-CHK record, which is in the root area, is retrieved only if the value of the AMOUNT-DUE data item is greater than the value of the AMOUNT-REC data item. For a qualified SLS-CHK record, a CUSTOMER-REC record is retrieved only if the value of the CURRENT-ORDERS data item is greater than 1000.

PROGRAM RELATION PROCESSING

All files joined in a relation can be opened with a single relational OPEN statement.

```
OPEN INPUT CREDIT-RISK
```

This statement is performed as if a separate OPEN were executed for each file within the relation CREDIT-RISK, in the order of the rank of the files. A relation is normally opened for input processing only. However, the files can be opened for I-O if locking of records or updating of individual files is desired.

All files joined in a relation can be closed with a single relational CLOSE statement.

```
CLOSE CREDIT-RISK.
```

This statement is performed as if a separate CLOSE were executed for each file within the relation CREDIT-RISK, in the order of the rank of the files.

```
RELATION DIVISION.
    RN IS CREDIT-RISK
    RESTRICT SLS-CHK WHERE AMOUNT-DUE GR AMOUNT-REC
    RESTRICT CUSTOMER-REC WHERE CURRENT-ORDERS GR 1000.
```

Figure 14-3. Record Qualification in the Subschema

The COBOL 5 program reads a relation by executing a READ statement that specifies a relation-name instead of a file-name. The root area can be positioned for subsequent sequential reading of the relation by a random relational READ statement or a START statement. Relation processing involves only the retrieval of records from the areas in the relation. Update operations can be performed on an individual record retrieved by a relational read; however, care must be exercised to maintain the integrity of the relation.

A random relational READ statement accesses the root area randomly. The primary key or an alternate key for the root area is specified in the KEY IS phrase. The record retrieved from the root area is one with the key value equal to the current value of the specified key data item. A record is also retrieved from each area joined in the relation.

```
READ CREDIT-RISK RECORD
    KEY IS CUSTOMER-ID
    INVALID KEY GO TO NO-SALES.
```

Execution of this statement accesses the root area in the CREDIT-RISK relation randomly. The current value of the alternate key CUSTOMER-ID is used to retrieve a record from the root area. If no record in the root area contains an alternate key of equal value, control is transferred to the paragraph named NO-SALES.

The relation can be read sequentially to retrieve all records related to a root area record. A sequential READ statement can be executed for the first read of the relation, after a random relational READ statement is executed, or after the root area is positioned by the START statement. If the root area-name is specified in a random READ statement, a sequential read of the relation then returns the next record in the root area and records joined to it. Qualification criteria specified in the subschema and the number of records joined at each rank in the relation determine the actual number of records retrieved by a sequential read.

```
READ CUST-ORDERS NEXT RECORD
    AT END GO TO FINISHED.
```

When this statement is executed, the CUST-ORDERS relation is read sequentially. Control is transferred to the paragraph named FINISHED when the end of the root area is reached. The status block created by the ENTER DB$DBST statement can be used after a relational read to determine the status of the areas in the relation.

The root area in a relation can be positioned to a specific record before reading the relation sequentially. File positioning is extremely important during sequential reading; therefore, areas must not be repositioned while reading a relation sequentially in order to ensure accurate record retrieval.

Records returned to the program by a relational read can be updated; however, precaution must be taken to ensure that existing relationships are not destroyed. Changing the common data items that join areas in the relation can destroy the positioning within existing relationships. When a record is to be deleted and records in a higher rank area are joined to it, the records in the area with higher rank should be deleted first.

Two or more relations can be read in parallel as long as no common areas exist between the relations being read. Results can be unpredictable if the relations have common areas.

CODING THE PROGRAM

Coding a COBOL 5 program that interfaces with CDCS is basically the same as coding any other COBOL 5 program. Changes in program coding occur in the Environment Division and the Data Division. No new Procedure Division statements are required; however, some statements cannot reference data base areas. Coding requirements for subprograms that access data base areas are discussed in section 10, Subprogram Interface.

ENVIRONMENT DIVISION

The SPECIAL-NAMES paragraph in the Configuration Section must include the SUB-SCHEMA clause. This clause identifies the COBOL subschema that provides the interface between the program and CDCS. The name of the subschema can be found in the Title Division of the subschema source listing or can be provided by the data administrator.

The FILE-CONTROL paragraph need not include a File-Control entry for each subschema area that is accessed by the program. The SELECT and ASSIGN clauses are required for data base files only when the FILE STATUS clause is used. No other clause is allowed in the entry. If used, the SELECT clause specifies an area-name from the Realm Division of the subschema source listing. The ASSIGN clause specifies a logical file name for the area and for any associated alternate key index file. Logical file names are user defined.

DATA DIVISION

The File Description entries and associated Record Description entries are not specified in the File Section for data base areas. The descriptions of these areas are obtained from the COBOL subschema during compilation. Other files, such as input and output files, are described in the File Section. The section header must be specified even if the only files used by the program are data base areas.

PROCEDURE DIVISION

Conventional input/output statements are used in processing data base areas. The COBOL subschema source listing provides the relation-names, record-names, and data-names to be used in the Procedure Division statements. The names are found in the Record Division and the Relation Division of the listing.

The USE FOR ACCESS CONTROL statement can be used to provide file passwords; the statement defines the optional keys required to gain access to files with passwords. If a file requires passwords (defined in the schema) and the correct passwords are not provided in the COBOL program, the PRIVACY BREACH ATTEMPT error occurs and the job aborts.

Figure 14-4 illustrates the USE FOR ACCESS CONTROL statement. Access control locks are defined within the schema. In the COBOL statements, read access is specified by the ON INPUT phrase. The KEY IS PART-KEY phrase identifies the data item (PART-KEY) containing the access control key. The FOR PARTS-FILE phrase identifies the file (PARTS-FILE) for which the access control key applies. The MOVE statement designates the value 4320 for the key. The MOVE statement must be specified within a paragraph in the same section that contains the USE FOR ACCESS CONTROL statement.

```
PROCEDURE DIVISION.
DECLARATIVES.
ACCESS-CTL SECTION.
    USE FOR ACCESS CONTROL ON INPUT
    KEY IS PART-KEY FOR PARTS-FILE.
ACC-CTL-PARAGRAPH.
    MOVE '4320' TO PART-KEY.
END DECLARATIVES.
```

Figure 14-4. USE FOR ACCESS CONTROL Example

The USE FOR DEADLOCK statement can be used to identify the procedure to be executed when a deadlock situation occurs. When the statement is included and a deadlock situation occurs, a non-fatal error (435) is returned to the data base status block. If the USE FOR DEADLOCK statement is omitted and a deadlock situation occurs for a given file, the job aborts. A deadlock situation is one that arises in concurrent data base access when two or more applications programs are contending for a resource that is locked, and none of the programs can proceed without that resource. An example of the USE FOR DEADLOCK statement is shown in figure 14-5.

```
WORKING-STORAGE SECTION.
01 DEADLOCK-FLAG PICTURE 99.
.
.
.
PROCEDURE DIVISION.
DECLARATIVES.
ADEADLOCK SECTION.
    USE FOR DEADLOCK ON ANY-FILE.
BEGIN-DEADLOCK.
    MOVE 99 TO DEADLOCK-FLAG.
END DECLARATIVES.
.
.
.
OPEN-PAR.
    MOVE 0 TO DEADLOCK-FLAG.
    OPEN I-O ANY-FILE.
.
.
.
READ-PAR.
    READ ANY-FILE KEY IS ANY-KEY.
    IF DEADLOCK-FLAG NOT EQUAL 0
        MOVE 0 TO DEADLOCK-FLAG
        GO TO READ-PAR.
.
.
.
```

Figure 14-5. USE FOR DEADLOCK Example

The ACCEPT statement and the DISPLAY statement cannot reference data base areas. The SORT statement cannot specify a data base area in the GIVING phrase. A data base area cannot be specified in either the USING phrase or the GIVING phrase of the MERGE statement. Any other Procedure Division statement can reference a data base area.

COMPILING THE PROGRAM

Compilation of a COBOL 5 source program that accesses the data base has two special requirements. The permanent file subschema library that contains the subschema specified in the SUB-SCHEMA clause must be attached before compilation can begin. The COBOL5 control statement must include the D parameter. This parameter specifies the logical file name of the subschema library.

EXECUTING THE PROGRAM

Execution of a program compiled for data base processing requires no file attachments except when using the CDCS Batch Facility; attaching the master directory and any required log files are then necessary. Otherwise, errors result if the data base files, log files, master directory, or the data base procedure library are attached.

Permanent file names and specific requirements of the schema (such as the necessary access control keys) should be provided by the individual responsible for controlling and monitoring the data base. The logical file name for the subschema library must be the same name specified for the D parameter in the COBOL5 control statement when the program was compiled.

SAMPLE PROGRAM

Accessing data base files is illustrated by the sample program included in this section. The program reads a relation that joins together three areas. The subschema listing shown in figure 14-6 contains information needed to code the source program. Source listings of the schema, the master directory, and the program that creates the data base files are in appendix C.

Figure 14-7 shows the source program CBILLS. To compile the program, the subschema library containing the subschema BILLING is attached. To execute the program, no file attachments are needed.

The subschema to be used by the program is identified by the SUB-SCHEMA clause (line 8). The only File Description entry and File-Control entry needed are for the output file.

The tree structure of the CUST-ORDERS relation read by this program is the same as the one illustrated in figure 14-2. The area ADDRESSES is the root area in the relation and is joined to the area ORDERS, which in turn is joined to the area LINEITEMS. When the file is opened, the root area is positioned at the first record.

The DB$DBST routine is entered (line 80) to establish automatic updating of the data base status items described under the group item DB-STATUS-BLOCK. All areas in the relation CUST-ORDERS are opened with a single OPEN statement (line 81).

A sequential read of the relation (lines 87 and 88) accesses the root area record and records joined to it. A value other than zero in DATABASE-STATUS indicates that an error occurred during the relational read. If DATABASE-STATUS contains zero, control passes to the paragraph ITEM-WRITE to generate a detailed output line. The last statement in the paragraph ITEM-WRITE transfers control to the paragraph READ-RELATION to execute the next sequential read of the relation.

DB-REL-NULL is tested for a null record condition (line 92). This means that at least one area in the relation does not have a record joined to the root area record. Additional processing is not desired when the null record condition occurs at either level joined to the root area; therefore, control is transferred to read the relation for the next record in the root area.

DB-REL-CTLBK is tested for a control break condition (line 93). When this occurs, the paragraphs NEW-ORDER through NEW-EXIT are performed before control is transferred to the paragraph ITEM-WRITE. The first statement in the paragraph NEW-ORDER checks the control block status item for a 3. DB-REL-CTLBK contains a 3 when a new ORDERS record is retrieved for the same customer. Notice that the control break condition occurs on rank 3 (child record) due to a change in value of record items at rank 2 (parent record). The user should always test for a value that is numerically one larger than the rank that causes the control break.

Figure 14-8 illustrates the output report generated by this program.

```
                00001                TITLE DIVISION.
                00002                SS  BILLING WITHIN DBFILES.
                00003                ALIAS DIVISION.
                00004                AD  RECORD VENDOR-CUSTOMER BECOMES CUSTOMER.
                00005                AD  DATA VEN-CUST-NO BECOMES CUST-NUM.
                00006                AD  DATA VEN-CUST-NAME BECOMES CUST-NAME.
                00007                REALM DIVISION.
                00008                RD  ADDRESSES, ORDERS, LINEITEMS.
                00009                RECORD DIVISION.
                00010                01  CUSTOMER.                                                  ** WITHIN ADDRESSES
                00011                    03  CUST-NUM            PICTURE 9(6).                     ** ORDINAL 1
                00012                    03  REC-TYPE            PICTURE X.                        ** ORDINAL 2
                00013                    03  STREET-NO           PICTURE X(5).                     ** ORDINAL 3
                00014                    03  STREET              PICTURE X(15).                    ** ORDINAL 4
                00015                    03  CITY                PICTURE X(15).                    ** ORDINAL 5
                00016                    03  STATE               PICTURE XX.                       ** ORDINAL 6
                00017                    03  ZIP-CODE            PICTURE 9(5).                     ** ORDINAL 7
                00018                    03  CUST-NAME           PICTURE X(30).                    ** ORDINAL 8
                00019                01  ORDERED.                                                   ** WITHIN ORDERS
                00020                    03  ORDER-NO            PICTURE X(6).                     ** ORDINAL 1
                00021                    03  CUST-NO             PICTURE 9(6).                     ** ORDINAL 2
                00022                    03  ORDER-DATE          PICTURE 9(6).                     ** ORDINAL 3
                00023                    03  BILL-DATE           PICTURE 9(6).                     ** ORDINAL 4
                00024                    03  BILL-AMOUNT         PICTURE 9(6)V99.                  ** ORDINAL 5
                00025                    03  AUTOKEY             PICTURE 9(6)    USAGE IS COMP-1.  ** ORDINAL 6
                00026                01  LINE-ITEM.                                                 ** WITHIN LINEITEMS
                00027                    03  PART-NUM            PICTURE X(6).                     ** ORDINAL 1
                00028                    03  PART-DESC           PICTURE X(20).                    ** ORDINAL 2
                00029                    03  ORD-NUMBER          PICTURE X(6).                     ** ORDINAL 3
                00030                    03  QTY-ORDER           PICTURE 9(4).                     ** ORDINAL 4
                00031                    03  QTY-SHIP            PICTURE 9(4).                     ** ORDINAL 5
                00032                    03  UNIT-PRICE          PICTURE 9(4)V99.                  ** ORDINAL 6
                00033                    03  AUTO-KEY            PICTURE 9(6)    USAGE IS COMP-1.  ** ORDINAL 7
PRIMARY KEY     00011           CUST-NUM FOR AREA ADDRESSES
PRIMARY KEY     00025           AUTOKEY FOR AREA ORDERS
PRIMARY KEY     00033           AUTO-KEY FOR AREA LINEITEMS
                *****           RECORD MAPPING IS NOT NEEDED FOR REALM -   ADDRESSES
                *****           RECORD MAPPING IS NOT NEEDED FOR REALM -   ORDERS
                *****           RECORD MAPPING IS NOT NEEDED FOR REALM -   LINEITEMS
                00034                   RELATION DIVISION.
                00035                       RN IS CUST-ORDERS
                00036                       RESTRICT CUSTOMER WHERE REC-TYPE EQ "C"
                00037                       RESTRICT LINE-ITEM WHERE QTY-SHIP GR 0.
                *****           END OF SUB-SCHEMA SOURCE INPUT

                *****                   RELATION   STATISTICS                   *****
RELATION 001                    CUST-ORDERS JOINS             AREA - ADDRESSES
                                                              AREA - ORDERS
                                                              AREA - LINEITEMS

                -----           BEGIN SUB-SCHEMA FILE MAINTENANCE               -----

              SUBSCHEMA                              CHECKSUM
              BILLING                                1556715225044313270O

                -----           END OF FILE MAINTENANCE                 -----
```

Figure 14-6. Source Listing for Subschema BILLING

```
1     IDENTIFICATION DIVISION.
2     PROGRAM-ID.  CBILLS.
3     ENVIRONMENT DIVISION.
4     CONFIGURATION SECTION.
5     SOURCE-COMPUTER.  CYBER-170.
6     OBJECT-COMPUTER.  CYBER-170.
7     SPECIAL-NAMES.
8         SUB-SCHEMA IS BILLING.
9     INPUT-OUTPUT SECTION.
10    FILE-CONTROL.
11        SELECT PRINT-FILE ASSIGN TO OUTPUT.
12    DATA DIVISION.
13    FILE SECTION.
14    FD  PRINT-FILE
15            LABEL RECORDS ARE OMITTED
16            DATA RECORD IS PRINT-LINE.
17    01  PRINT-LINE                PICTURE X(136).
18    WORKING-STORAGE SECTION.
19    01 READ-PASSWORD-ORD PIC X(30).
20    01 READ-PASSWORD-ADDR PIC X(30).
21    01 BLK-LENGTH PIC 99 COMP-1.
22    01  DB-STATUS-BLOCK.
23        02  DATABASE-STATUS PIC 9(5) COMP-1.
24        02  AUX-STATUS OCCURS 3 TIMES PIC 9(10) COMP-1.
25        02  DB-FUNCTION PIC A(10).
26        02  DB-REL-ERROR PIC 9(3) COMP-1.
27        02  DB-REL-CTLBK PIC 9(3) COMP-1.
28        02  DB-REL-NULL PIC 9(3) COMP-1.
29        02  DB-REALM PIC X(30).
30    01  HLINE-1.
31        03  FILLER                PICTURE X(5)    VALUE "1    ".
32        03  NAME-OUT              PICTURE X(30).
33        03  FILLER                PICTURE X(101)  VALUE SPACES.
34    01  HLINE-2.
35        03  FILLER                PICTURE X(5)    VALUE SPACES.
36        03  NO-OUT                PICTURE X(5).
37        03  FILLER                PICTURE X       VALUE SPACE.
38        03  STREET-OUT            PICTURE X(15).
39        03  FILLER                PICTURE X(110)  VALUE SPACES.
40    01  HLINE-3.
41        03  FILLER                PICTURE X(5)    VALUE SPACES.
42        03  CITY-OUT              PICTURE X(15).
43        03  FILLER                PICTURE X       VALUE SPACE.
44        03  STATE-OUT             PICTURE XX.
45        03  FILLER                PICTURE XX      VALUE SPACES.
46        03  ZIP-OUT               PICTURE 9(5).
47        03  FILLER                PICTURE X(106)  VALUE SPACES.
48    01  HLINE-4.
49        03  FILLER                PICTURE X(5)    VALUE "-    ".
50        03  FILLER                PICTURE X(12)   VALUE "INVOICE NO. ".
51        03  ORDER-OUT             PICTURE X(6).
52        03  FILLER                PICTURE X(5)    VALUE SPACES.
53        03  FILLER                PICTURE X(13)  VALUE "CUSTOMER NO. ".
54        03  CUST-OUT              PICTURE 9(6).
55        03  FILLER                PICTURE X(5)    VALUE SPACES.
56        03  FILLER                PICTURE X(11)   VALUE "ORDER DATE ".
57        03  DATE-OUT              PICTURE 9(6).
58        03  FILLER                PICTURE X(5)    VALUE SPACES.
59        03  FILLER                PICTURE X(11)   VALUE "AMOUNT DUE ".
60        03  TOTAL-OUT             PICTURE $$$$,999.99.
61        03  FILLER                PICTURE X(40)   VALUE SPACES.
62    01  ITEMS.
63        03  FILLER                PICTURE X(13)   VALUE SPACES.
64        03  QTY-OUT               PICTURE ZZZ9.
```

Figure 14-7. Sample Program for Reading a Data Base Relation (Sheet 1 of 2)

```
65          03  FILLER                 PICTURE X(21)   VALUE SPACES.
66          03  DESC-OUT               PICTURE X(20).
67          03  FILLER                 PICTURE X(21)   VALUE SPACES.
68          03  AMT-OUT                PICTURE Z,ZZ9.99.
69          03  FILLER                 PICTURE X(49)   VALUE SPACES.
70      PROCEDURE DIVISION.
71      DECLARATIVES.
72      CUST-ACC-CONTROL SECTION.
73          USE FOR ACCESS CONTROL ON INPUT
74          KEY IS READ-PASSWORD-ADDR FOR ADDRESSES.
75      ACC-1.
76          MOVE "READ ADDRESSES" TO READ-PASSWORD-ADDR.
77      END DECLARATIVES.
78      OPENING.
79          MOVE 11 TO BLK-LENGTH.
80          ENTER "DB$DBST" USING DB-STATUS-BLOCK, BLK-LENGTH.
81          OPEN INPUT CUST-ORDERS.
82          OPEN OUTPUT PRINT-FILE.
83          IF DATABASE-STATUS NOT = 0
84              DISPLAY "FILE OPEN ERROR =  " DATABASE-STATUS
85              GO TO FINISHED.
86      READ-RELATION.
87          READ CUST-ORDERS NEXT RECORD
88              AT END GO TO FINISHED.
89          IF DATABASE-STATUS NOT EQUAL TO ZERO
90              DISPLAY "ERROR OCCURRED ON FILE " DATABASE-STATUS
91              "  " DB-REALM   "  " DB-FUNCTION.
92          IF DB-REL-NULL NOT EQUAL TO ZERO GO TO READ-RELATION.
93          IF DB-REL-CTLBK NOT EQUAL TO ZERO PERFORM NEW-ORDER
94              THRU NEW-EXIT, GO TO ITEM-WRITE.
95      ITEM-WRITE.
96          MOVE QTY-SHIP TO QTY-OUT.
97          MOVE PART-DESC TO DESC-OUT.
98          COMPUTE AMT-OUT = QTY-SHIP * UNIT-PRICE.
99          WRITE PRINT-LINE FROM ITEMS.
100         GO TO READ-RELATION.
101     NEW-ORDER.
102         IF DB-REL-CTLBK EQUALS 3 GO TO SAME-CUSTOMER.
103         MOVE CUST-NAME TO NAME-OUT.
104         WRITE PRINT-LINE FROM HLINE-1.
105         MOVE STREET-NO TO NO-OUT.
106         MOVE STREET TO STREET-OUT.
107         WRITE PRINT-LINE FROM HLINE-2.
108         MOVE CITY TO CITY-OUT.
109         MOVE STATE TO STATE-OUT.
110         MOVE ZIP-CODE TO ZIP-OUT.
111         WRITE PRINT-LINE FROM HLINE-3.
112     SAME-CUSTOMER.
113         MOVE ORDER-NO TO ORDER-OUT.
114         MOVE CUST-NO TO CUST-OUT.
115         MOVE ORDER-DATE TO DATE-OUT.
116         MOVE BILL-AMOUNT TO TOTAL-OUT.
117         WRITE PRINT-LINE FROM HLINE-4
118             AFTER ADVANCING 2 LINES.
119         MOVE SPACES TO PRINT-LINE.
120         WRITE PRINT-LINE.
121     NEW-EXIT.
122         EXIT.
123     FINISHED.
124         CLOSE CUST-ORDERS, PRINT-FILE.
125         STOP RUN.
```

Figure 14-7. Sample Program for Reading a Data Base Relation (Sheet 2 of 2)

```
INTERNATIONAL TRIKE PARTS
1236  INDUSTRIAL DR
SAN JOSE         CA  95151

INVOICE NO. 030667      CUSTOMER NO. 216800      ORDER DATE 041176      AMOUNT DUE    $1,445.00

         100                       BLACK EXTENSION TUBE                        45.00
          25                       8 INCH TIRE                                237.50
          50                       HANDLE BAR                               1,037.50
         100                       CHAIN LINKS                                125.00

INVOICE NO. 149020      CUSTOMER NO. 216800      ORDER DATE 042576      AMOUNT DUE    $1,540.00

          30                       BRAKE ASSEMBLY                             675.00
          50                       REAR WHEEL                                 865.00




ALL AMERICAN BIKES
11185 MAIN STREET
FAIRFIELD        CA  94533

INVOICE NO. 095188      CUSTOMER NO. 649025      ORDER DATE 040776      AMOUNT DUE      $672.50

         125                       BRAKE PAD LEFT                             118.75
         125                       BRAKE PAD RIGHT                            118.75
          75                       REAR WHEEL BRACE                           435.00

INVOICE NO. 206193      CUSTOMER NO. 649025      ORDER DATE 042176      AMOUNT DUE    $4,063.75

          50                       REAR WHEEL HUB ASSY                        625.00
          50                       BACK WHEEL ASSEMBLY                      1,297.50
         250                       FLAT SEAT COVER                            487.50
         125                       MASTER CHAIN LINK                        1,031.25
          30                       HANDLE BAR                                 622.50
```

Figure 14-8. Output Report Generated by Program CBILLS

CYBER Record Manager (CRM) is the input/output processor that provides an interface between a COBOL program and the operating system routines that process files on hardware devices. CRM file processing capabilities are divided into two categories: the Basic Access Methods (BAM) and the Advanced Access Methods (AAM). BAM is a group of routines that process sequential and word-addressable files. AAM includes the Multiple-Index Processor and the CRM routines that process indexed sequential, direct access, and actual-key files. MIP provides the means to access records by more than one field (that is, alternate access paths are available.

Six different file organizations are defined in COBOL 5 and supported by CRM to allow the COBOL user flexibility in file structure and usage. The COBOL file organizations (identified through the ORGANIZATION IS clause) and the equivalent CRM file organizations are shown in table 15-1. The terms are used interchangeably in this section.

TABLE 15-1. FILE ORGANIZATIONS

COBOL	CRM	CRM Mnemonic
Sequential	Sequential	SQ
Word-address	Word Addressable	WA
Relative	(No corresponding file organization; Word Addressable implemented)	
Direct	Direct Access	DA
Indexed	Indexed Sequential	IS
Actual-key	Actual Key	AK

The COBOL language provides for virtually all types of file processing. Most interaction between COBOL and CRM is transparent to the COBOL programmer. CRM is of interest when designing new files, when changing the structure of an existing file, and when attempting to obtain file status or input/output error information. As long as the file is read by the same method used to write it, the user is not concerned with physical representation. However, physical record format is important when reading an existing file from another computer vendor or through a language different from that used is creating the file.

It is assumed that the reader understands the file processing concepts in section 3 of this guide. This section discusses CRM features that are most useful to the experienced COBOL programmer. For further details about CRM, refer to the BAM and AAM reference manuals and user's guides.

FILE INFORMATION TABLE

The file information table (FIT) is the most important element in the COBOL interface with CYBER Record Manager; all communication between COBOL and CRM is performed through the FIT. The FIT is a 35-word table that contains descriptions of an individual file's organization, record size and type, blocking structure, and processing options; the logical file name by which the file is known to the operating system; and other pertinent data such as labeling information, buffer size, and record area location.

To establish communication with CRM, the COBOL compiler creates a FIT for each file. The compiler places file information in the table for each file specified in the program as it encounters applicable source program statements. In response to read or write requests, CRM uses the information in this table to request file processing action by the operating system. CRM updates file information, such as processing direction and file position, during program execution. Figure 15-1 illustrates COBOL input/output interfaces.

The file characteristics in the FIT can be specified by the COBOL user in three ways:

- By source program statements other than the USE clause
- By the USE clause
- By FILE control statements

The three methods are listed in order of override. That is, a FILE control statement parameter can override a USE clause parameter. A USE clause parameter can override other source-statement-generated parameters. Caution must be used in all FIT field overrides. The USE clause should be used in preference to a FILE card, whenever possible.

FIT FIELDS SET WITH SOURCE STATEMENTS

Most FIT fields are set automatically by the COBOL compiler. For example:

- SELECT CREDIT ASSIGN TO ISFILE sets the LFN field to ISFILE.
- ORGANIZATION IS INDEXED sets the FO field to IS.
- BLOCK CONTAINS 5100 CHARACTERS sets the MBL field to 5100 and the BT field to K.

Figure 15-2 illustrates other FIT fields set through the COBOL language. Table 15-2 lists FIT fields set for each record type. Table 15-3 lists FIT fields set for each file organization.

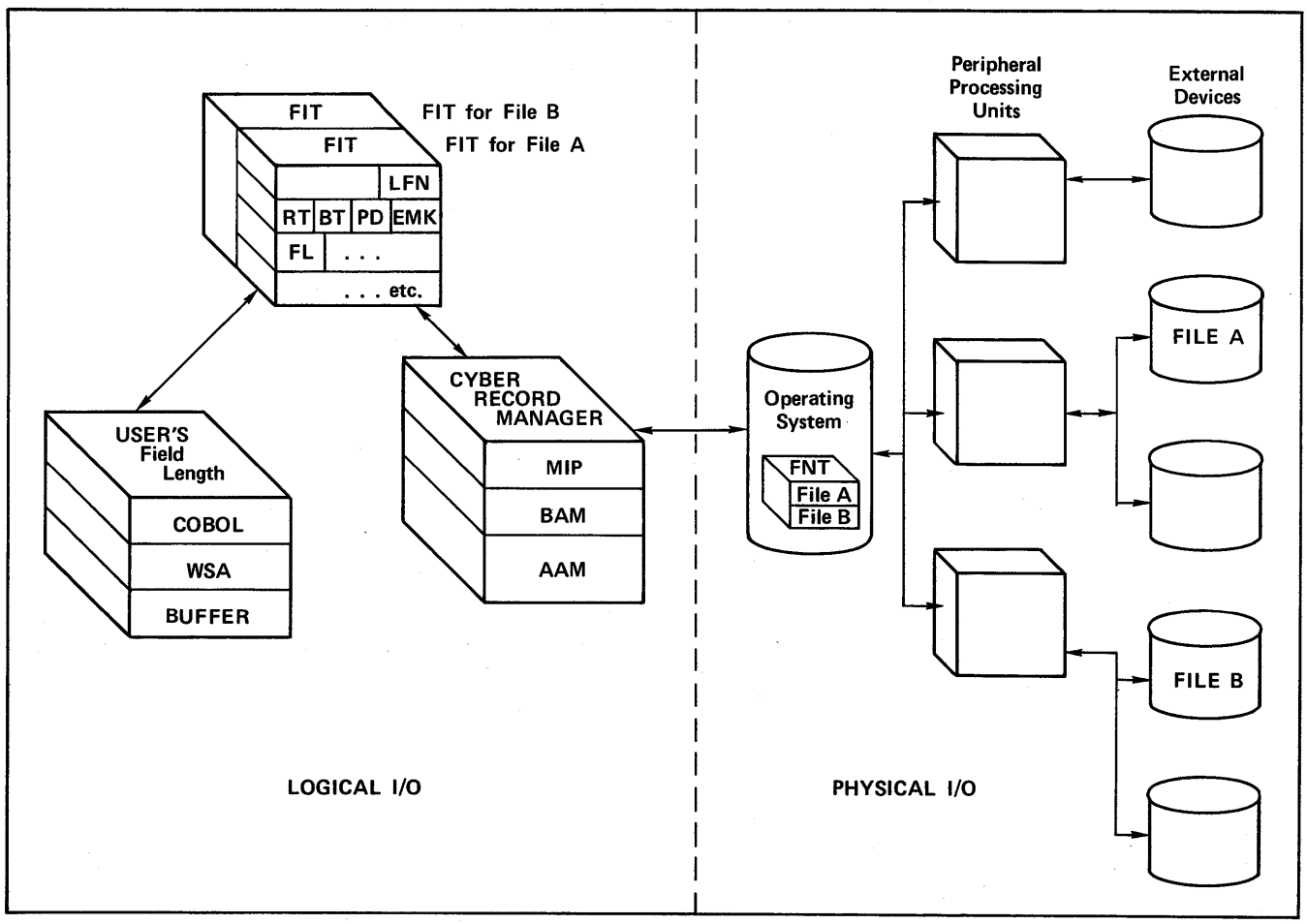


Figure 15-1. COBOL INPUT/OUTPUT INTERFACES

FIT FIELDS SET WITH THE USE CLAUSE

The USE clause in the Environment Division can specify file characteristics that cannot otherwise be specified through COBOL language statements. The USE clause can also be used to override some file characteristics that are specified through COBOL statements. Certain FIT fields cannot be overridden, and the USE clause parameter is then ignored.

The USE clause can set the following FIT fields:

BBH	DFC	ERL	MUL	RMK
BCK	DP	FLM	NL	RT
BFS	EFC	FWI	ORG	SBF
BT	EO	IP	PC	SPR

Only the most commonly used parameters are discussed in this section. The following paragraphs discuss the use of the USE clause for the BT, IP, RT and ORG parameters. DFC, EFC, and ERL are discussed in the subsection entitled CRM Debugging Tools.

Setting the Index Block Padding

Index block padding is set by the IP field. For indexed sequential files, index blocks can be padded to minimize the automatic block splitting and creation of new index levels that can result from file growth.

```
SELECT PMASTER ASSIGN TO "PMASTER"
    USE "IP=5".
```

In this example, the index block padding factor is set to 5 percent. Each index block in the file PMASTER has 5 percent of its available space free for adding new index entries; this allows for a 10 percent growth in a two-level file.

Changing the Record and Block Type

Files used by COBOL input/output statements are organized into logical records. CRM recognizes eight record types. All record types are available to the COBOL user:

D	Decimal character count
F	Fixed length
S	System
R	Record mark
T	Trailer count
U	Undefined
W	Control word
Z	Zero byte terminated

The record type specification defines the format of every record in a file and enables CRM to determine the length of a record on a read or write. The COBOL compiler can derive the record type indirectly from the RECORD clause or the OCCURS...DEPENDING ON clause, and the lengths of the 01 level entries.

```
IDENTIFICATION DIVISION.
PROGRAM-ID.  DELINQUENT.
ENVIRONMENT DIVISION.
INPUT-OUTPUT SECTION.
FILE-CONTROL.
    SELECT CREDIT ASSIGN TO ISFILE <-------------- Sets LFN to ISFILE
        ORGANIZATION IS INDEXED <---------------- Sets FO to IS
        RECORD KEY IS CREDIT-ID.<---------------- Used to set RKW and RKP
DATA DIVISION.
FILE SECTION.
FD CREDIT
    LABEL RECORD OMITTED
    BLOCK CONTAINS 5710 CHARACTERS.<------------- Sets MBL to 5740 (9 PRUs) and
01  BAD-RISK.                                     BT to K
    02            PIC X(10).
    02 CREDIT-ID PIC X(10). <--------------------- Sets RKW to 1 and RKP to zero
    02            PIC X(40).
    02 KOUNT      PIC 99.
    02 PAYMENT  OCCURS 1 TO 12 TIMES DEPENDING ON KOUNT.  }
       03  MONTH-LATE  PIC X(3).                  }  Sets RT=T, HL=62, TL=3, CP=60,
PROCEDURE DIVISION.                                   CL=2, MRL=98
OPEN-IT.
    OPEN INPUT CREDIT.
READ-IT.
    READ CREDIT NEXT AT END GO TO CLOSE-IT.
    WRITE BAD-RISK.  GO TO READ-IT.
CLOSE-IT.
    CLOSE CREDIT.
    STOP RUN.
```

Corresponding FILE control statements:

```
FILE(ISFILE,FO=IS,BT=K,RT=T,HL=62,TL=3,CP=60,CL=2,MRL=98,)
FILE(ISFILE,RKW=1,RKP=0,KL=10,KT=S)
```

Notes:

Notice that both the RECORD KEY clause and the record description are required to set RKP and RKW. AAM will expect the key for each record to be located in word 1, position 0.

The OCCURS . . . DEPENDING ON clause (and the fact that KOUNT is included in the record description) cause the RT field of the FIT to be set to T. The fixed portion of the record description (HL) is 62 characters. The trailer length portion (TL) is three characters. The beginning character position (CP) of the trailer count field of the record is position 60 (counting from 0), determined by location of KOUNT. The maximum number of characters in the record (MRL) is 98.

Figure 15-2. COBOL File Processing

TABLE 15-2. FIT FIELDS BY RECORD TYPE

Record Type	FIT Field Mnemonic	Description	Setting
D	RT	Record type	D (decimal character count)
	MNR	Minimum record length	integer-1 from RECORD clause
	MRL	Maximum record length	integer-2 from RECORD clause
	LP	Start of count field	Length of items preceding data-name in DEPENDING ON option
	LL	Length of count field	Length of data-name in DEPENDING ON option
F	RT	Record type	F (fixed length)
	FL	Fixed length	Length of longest Record Description entry
R	RT	Record type	R (record mark)
	RMK	Record mark character	Right bracket character] (62B)
	MRL	Maximum record length	Calculated from Record Description
T	RT	Record type	T (trailer count)
	MNR	Minimum record length	Calculated from integer-1 in OCCURS option
	MRL	Maximum record length	Calculated from integer-2 in OCCURS option
	CP	Start of trailer count field	Length of items preceding data-name in DEPENDING ON option
	CL	Length of trailer count field	Length of data-name in DEPENDING ON option
	HL	Header length	Length of items preceding subject of OCCURS clause
	TL	Length of trailer item	Length of subject of OCCURS clause
W	RT	Record type	W (control word)
	MRL	Maximum record length	Length of longest Record Description entry plus 10 characters
	RL	Record length	Length of specified Record Description entry (reset for each WRITE)
Z	RT	Record type	Z (zero byte terminated)
	FL	Full length	Length of longest Record Description entry

NOTES:

If the file name is INPUT, OUTPUT, or PUNCH, RT is set to Z.

If the RECORD clause is omitted and all record description entries are the same length, CRM sets RT to F. If RECORD CONTAINS...CHARACTERS is specified, RT is set to F.

If there exists an OCCURS...DEPENDING ON clause in the record description and the data-name is defined within the record, RT is set to T.

If the RECORD clause (FD entry) specifies RECORD CONTAINS x TO x CHARACTERS DEPENDING ON data-name in records, RT is set to D.

If none of the above conditions exist, RT is set to W.

TABLE 15-3. FIT FIELDS SET FROM SOURCE CODE

FIT Field Set	Symbolic Value	COBOL Clause or Statement Used
All Organizations		
FO		ORGANIZATION Clause
	SQ	SEQUENTIAL
	WA	RELATIVE
	IS	INDEXED
	DA	DIRECT
	AK	ACTUAL-KEY
	WA	WORD-ADDRESS
RT, MNR, MRL		RECORD clause, Record Description entry, WRITE with ADVANCING phrase, FD entry, LINAGE clause. See table 15-2.
RL		RECORD clause, Record Description Description entry (See the SIZE and TYPE table in section 4 of the COBOL 5 reference manual).
Other record description fields		Record Description
PD		OPEN statement
	INPUT	INPUT
	OUTPUT	OUTPUT
	I-O	I-O
	EXTEND	EXTEND
LFN		ASSIGN clause first implementor name
LT		LABEL RECORDS clause
	UL	OMITTED
	S	STANDARD
ULP		VALUE OF phrase in LABEL RECORDS clause
Sequential files		
BT, MNB, MBL		BLOCK CONTAINS clause (for K and E type blocks)
BFS		RESERVE clause
CM		RECORDING MODE clause (overrides COBOL default, CM=YES)
	YES	DECIMAL
	NO	BINARY
MFN, PNO		MULTIPLE FILE TAPE clause
OF	R,N,E	OPEN statement
Indexed files		
IP		Installation parameter
DP		Installation parameter
KT, KA, KP, KL		RECORD KEY clause, ALTERNATE RECORD KEY clause, and item description
IBL		Installation parameter

TABLE 15-3. FIT FIELDS SET FROM SOURCE CODE (Contd)

FIT Field Set	Symbolic Value	COBOL Clause or Statement Used
MBL		BLOCK CONTAINS clause
XN		ASSIGN clause second implementor-name
CDT/DCT		Program collating sequence
ON	NEW OLD	OPEN statement OUTPUT Other
REL	EQ, GE, LE	START statement
Direct files		
RKP, RKW, KA, KL		RECORD KEY clause, ALTERNATE RECORD KEY clause, and item description
HMB		BLOCK COUNT clause
MBL		BLOCK CONTAINS clause
HRL		USE FOR HASHING declarative
XN		ASSIGN clause second implementor-name
ON	NEW OLD	OPEN statement OUTPUT Other
REL	EQ, GE, LE	START statement
Actual-key files		
KA, KL		RECORD KEY clause, ALTERNATE RECORD KEY clause, and item description
MBL		BLOCK CONTAINS clause
REL	EQ, GE, LE	START Statement
RB		BLOCK CONTAINS clause and Record Description
RKP, RKW		Item description (for AK files with alternate keys)
ON	NEW OLD	OPEN statement OUTPUT Other
XN		ASSIGN clause second implementor - name
Word-address files		
BFS		RESERVE clause
OF	R, N	OPEN statement
Relative files		
BFS		RESERVE clause
OF	R, N	OPEN statement

Table 3-2 in section 3 shows the record types selected by the compiler according to RECORD clause and File Description entry interactions, for all except relative and word-address files. Table 15-4 shows the possible record types for all COBOL file organizations.

TABLE 15-4. RECORD TYPE AND FILE ORGANIZATION COMBINATIONS

File Organization	Record Types	
	COBOL Generated	Requires USE Clause or FILE Statement
Sequential	D, F, T, W	R, S, U, Z
Word-address	U	F, W
Relative	F	-
Direct	D, F, T, W	R, S, U, Z
Indexed	D, F, T, W	R, S, U, Z
Actual key	D, F, T, W	R, S, U, Z

During input/output operations, logical records on sequential files are grouped into physical blocks for transfer between a buffer and an external storage device. The number of logical records in the block depends on the file organization and structure. Sequential files can have one of four block types:

K Fixed number of records per block
C Character count
E Exact records
I Internal control word

Record type and block type are set by the RT and BT fields, respectively. The COBOL-generated block type and record type can be overridden as follows:

```
SELECT MYFILE ASSIGN TO CZFILE
    USE "BT=C, RT=Z".
```

These clauses can be used when processing a file that has been created through a terminal. (The file, therefore, has CZ format.)

Setting the Old/New File Organization

Direct, indexed, and actual-key file organizations can be of two types: initial or extended. If both types are available, the user should be aware of differences in processing. Extended AAM file organization is generally more efficient. If extended files have been installed, they are the default for COBOL programs; ORG=OLD must be specified in the USE clause if initial file organization is desired.

```
SELECT MYFILE ASSIGN TO "MYFILE"
    USE "ORG=OLD".
```

This statement specifies initial file organization for the file MYFILE.

FIT FIELDS SET WITH THE FILE CONTROL STATEMENT

At execution-time, the FILE control statement establishes FIT field values that have not already been established at compile-time. The FILE control statement is used to override FIT parameter values normally assigned by COBOL. Any parameter can be used on a FILE control statement; however, the following fields will have no effect:

EX	KL	LBL	MKL
FO	KP	LFN	ULP
HRL	KT	LT	WSA
KA	LA	LX	

In general, parameters used on a FILE control statement are confined to those permitted by the USE clause.

Every file processed through CRM must have a valid FIT at the time the file is opened. For files referenced in COBOL programs, the COBOL compiler establishes a table for each file during compilation and sets fields in the table to appropriate values based on the clauses and statements within the program. These tables become part of the compiled object program. Most of the FIT fields needed for a given file are set during compilation; some can be reset by the COBOL execution time routines. For example, the block type (BT) field for a sequential file is reset if the COBOL execution time routines determine that the block type originally selected is inappropriate for the device on which the file actually resides. In most cases, however, the values selected during compilation remain in effect throughout program execution.

If one or more FILE control statements have been provided for a file, the values they specify are placed in the FIT when the file is opened during program execution.

The FILE control statement has two uses of interest to the COBOL programmer. The first use enables other processors or languages to use files created through COBOL programs. For example, a FORTRAN program can read a COBOL-created file if a FILE control statement specifies the same file structure produced by the original COBOL program. Even file structures not provided as part of standard FORTRAN input/output (such as D, R, or T type records, or K or E type blocks) can be specified through the FILE control statement.

The second use of the FILE control statement alters or supplements default COBOL processing. FIT settings provided by the FILE control statement take effect when the file is opened at execution time, and override values established by COBOL during compilation. Thus, file structures other than those defined in the source program result when the program is executed.

NOTE

The COBOL language contains adequate provision for most file processing likely to be of value to the programmer; unexpected results can occur from incorrect FIT field overrides.

The following example illustrates the format of the FILE control statement:

```
FILE(PMASTER,BT=C,RT=Z)
```

This statement places a value of Z in the RT field of the FIT, and a value of C in the BT field of the FIT.

CRM DEBUGGING TOOLS

The system error file (ZZZZZEG) is a local mass storage file that disappears at job termination. The CRMEP control statement can be used to read the error file and list its contents.

CRM automatically performs certain checks and error processing, and maintains file status and error information in certain FIT fields. The following FIT fields can be set by the COBOL user to obtain error information:

EFC	Error file control
DFC	Dayfile control
ERL	Trivial error limit

The following FIT field is automatically set by CRM and can be displayed by the COBOL user:

ES	Error Status

A file status code area can be defined by the COBOL user. CRM puts file status codes into the area during input/output operations. The codes can then be displayed by the COBOL user.

ACCESSING FILE STATUS CODES

The COBOL compiler generates a status code each time an input/output statement is executed. If this code is to be used by the program, the FILE STATUS clause in the Environment Division specifies the data item to receive the status code. Refer to figure 15-3.

```
SELECT PMASTER ASSIGN TO "PMASTER"
FILE STATUS IS CODE-RETURN.
.
.
.
WORKING-STORAGE SECTION.
01 CODE-RETURN PIC XX.
.
.
.
READ PMASTER NEXT RECORD
    AT END GO TO END-ROUTINE.
IF CODE-RETURN EQUALS "90"
    DISPLAY "STATUS CODE IS" CODE-RETURN.
```

Figure 15-3. Accessing the File Status Code

The status code data item must be described as a two-character alphanumeric data item; it cannot be described in the File Section or Report Section. Status code values that can be received by the data item are as shown in table 15-5.

When status code 90 is returned to the status code data item, the specific CRM error can be determined by executing the ENTER "C.IOST" statement. Similarly, when status code 99 is returned, the specific COBOL-detected error can be determined by using the C.IOST routine. COBOL-detected error codes are listed in section 3 of the COBOL5 reference manual. (Refer to the following subsection for an example of the C.IOST routine usage.)

For data base files, the DB$DBST routine can be used to obtain file status or errors. Refer to section 14 for further discussion on data base files.

TABLE 15-5. FILE STATUS CODES

Status Code	Meaning
00	The statement executed successfully.
02	The statement executed successfully; a duplicate alternate key value is involved. If a record was read by alternate key, the next record in that alternate key sequence contains the same value for the alternate key. For a WRITE or REWRITE statement, the record written created a duplicate value for one or more alternate keys.
10	An at end condition occurred for a READ, RETURN, or SEARCH statement.
21	An invalid key condition occurred due to a primary key sequence error. For an indexed file with sequential access, the primary key value in the record being written is not in ascending sequence. For a REWRITE statement, the primary key value changed between execution of the sequential READ statement and the REWRITE statement.
22	An invalid key condition occurred because the record being written or rewritten creates a duplicate key value when duplicates are not allowed.
23	An invalid key condition occurred because no record in the file contains the specified key value.
24	An invalid key condition occurred for a relative or indexed file due to a boundary violation.
30	A permanent error occurred (parity error, transmission error, mass storage not available, and so forth).
34	A permanent error occurred (boundary violation, file limit established by FILE control statement parameter is reached, and so forth).
90	A CYBER Record Manager error other than those indicated by a specific status code value occurred.
99	An I/O error occurred and was detected by the COBOL compiler.

ACCESSING THE CRM ERROR STATUS CODE

CRM stores a three-digit octal error status code into the ES field of the FIT when a trivial or fatal input/output error occurs. This is the value returned to a user-specified field in a COBOL program when the C.IOST routine is used. The C.IOST routine multiplies each octal digit by its corresponding power of ten and returns a decimal representation of the octal code. The code returned by C.IOST can be used, without conversion, when referencing the codes listed in the BAM and AAM reference manuals. For example, a 142 decimal number returned by C.IOST is listed in the manuals as 142 octal.

The following COBOL statements can be used to access the CRM error status code and severity level when an I/O error occurs on the file named PMASTER:

```
WORKING-STORAGE SECTION.
01 CRM-CODE PIC 9999 COMP-1.
01 SEVERITY PIC X.
.
.
.
ENTER "C.IOST" USING PMASTER CRM-CODE
    TRAGEDY.
```

The CRM error code for the error encountered while processing the file PMASTER is returned to CRM-CODE, which must be a four-digit integer described with COMP-1 usage. TRAGEDY is defined as one alphanumeric character and receives one of the following letters:

F Indicates that any I-O statement, other than CLOSE, causes the job to abort.

T Indicates that the job does not abort.

These indicators are not the same as the CRM severity indicators.

USING THE SYSTEM ERROR FILE AND FIT DUMP

The error file control (EFC) field of the FIT controls the CRM messages that are written to a special system error file named ZZZZZEG. The possible values of the EFC field are as follows:

0 No errors or notes written (default)

1 Errors written

2 Notes written

3 Errors and notes written

The COBOL user can set the EFC field with the USE clause as follows:

```
SELECT PMASTER ASSIGN TO "PMASTER"
    USE "EFC=3".
```

If a non-zero value exists in the EFC field, the CRMEP control statement can be used to read the error file, and list its contents. An example of the CRMEP control statement format is:

```
CRMEP,LO,RU
```

This statement selects notes, fatal and trivial errors, and data manager messages. It also returns and unloads the error file after processing.

The FIT dump is a useful debugging tool. The contents of the FIT can be written to the system error file as a note. The COBOL user then sets a value of 2 or 3 in the EFC field and calls the FITDMP macro as follows:

```
SELECT ISDATA ASSIGN TO ISFILE
    USE "EFC=3".
.
.
.
OPEN OUTPUT ISDATA.
ENTER FITDMP USING ISDATA.
```

The CRMEP control statement can then list the error file. A sample FIT dump is shown in figure 15-4.

CONTROLLING CRM MESSAGES ON THE DAYFILE

The DFC field of the FIT controls the listing of CRM messages written to the user's job dayfile. The possible values of the DFC field are as follows:

0 Fatal messages only (default)

1 All error messages

2 Notes only

3 Error messages and notes

The COBOL user can set the DFC field with the USE clause as follows:

```
SELECT PMASTER ASSIGN TO "PMASTER"
    USE "DFC=3".
```

In this example, all CRM error messages and notes are to be written to the job dayfile.

SETTING THE TRIVIAL ERROR LIMIT

The trivial error limit (ERL) field of the FIT places a limit on the number of trivial CRM errors allowed. When the limit is reached, a fatal error occurs. If the value in the field is zero, no error count is performed and an infinite number of trivial errors is permitted. If a value is specified, the job aborts when the value of the ERL field equals the value of the trivial error count (ECT) field. The ECT field of the FIT is automatically incremented by CRM whenever a trivial error occurs. The field is decremented by COBOL when errors producing INVALID KEY or AT END are encountered.

The COBOL user can set the ERL field with the USE clause as follows:

```
SELECT PMASTER ASSIGN TO "PMASTER"
    USE "ERL=5".
```

In this example, program execution is terminated when five trivial CRM errors occur.

```
1        CRMEP,LO,RU,L=OFILE.
 RM NOTE  1001 ON LFN PMASTER  FILE OPENED
 RM NOTE  1000 ON LFN PMASTER  FIT DUMP              (FIT AT 000440)
                    0 ASCII                0000000001 EOIWA                    0 LCR                00000000 PNO
                    0 BAL                       000 ERL           20150123240522 LFN                   00 POS
                    0 BBH                       000 ES                   000000 LGX                00000000 PTL
                    0 BCK                    000705 EX                       01 LL                     0031 RB
                000000 BFS                        1 EXD                      02 LNG              0000000000 RC
            0000000000 BN                         0 FF                       01 LOP                    0 RDR
                    3 BT                   00000121 FL                       01 LOP5                   0 REL
                000440 BZF                 7777777777 FLM                00000062 LP                   00 RKP
                    0 B8F                         0 FNF                      2 LT                   0000 RKW
                000000 CDT                        3 FO                      15 LVL                00000000 RL
                    0 CF                        020 FP                  000000 LX                   01 RMK
                   01 CL                          0 FPB                00011754 MBL                  05 RT
                    0 CM                         36 FTS           000000000000 MFN                    0 SB
                    1 CMPLT                  027364 FWB                  000 MKL                    0 SBF
                    0 CNF                         0 FWI                00000051 MNB                  0 SDS
                00000062 CP                       0 HB                 00000063 MNR                  00 SES
                000000 CPA                 00000063 HL                 00000121 MRL                   0 SOL
                    0 C1                   00000000 HMB                      00 MUL                   0 SPR
                   00 DC                     000000 HRL                      0 NDX                00000005 TL
                000000 DCA                 00000000 IBL                     03 NL                     00 TRC
                005155 DCT                      000 IP                      0 NOFCP                  0 ULP
                    0 DFC                       000 IRS                     1 OC                     0 VF
                    0 DFLG                   000340 KA                      1 OF                    03 VNO
                    0 DKI                       005 KL                      1 ON                0000000000 WA
                  000 DP                          0 KNE                     1 ORG                    0 WPN
                 0411 DVT                        00 KP                      0 OVF                00000000 WSA
                000000 DX                      0000 KR                     00 PC                  000000 XBS
                   000 ECT                        3 KT                      2 PD          00000000000000 XN
                     3 EFC                  000000 LA                       0 PEF
                     1 EMK                      00 LAC                 000340 PKA
                     0 EO                 00000000 LBL                      0 PM

 RM NOTE  1002 ON LFN PMASTER  FILE CLOSED
 RM NOTE  1003 ON LFN PMASTER  NUMBER OF INDEX LEVELS          0
 RM NOTE  1004 ON LFN PMASTER  ***NUMBER OF GETS THIS OPEN.                0
 RM NOTE  1005 ON LFN PMASTER  ***NUMBER OF PUTS THIS OPEN                 42
 RM NOTE  1006 ON LFN PMASTER  ***NUMBER OF REPLACES THIS OPEN              0
 RM NOTE  1007 ON LFN PMASTER  ***NUMBER OF DELETES THIS OPEN               0
 RM NOTE  1033 ON LFN PMASTER  ***NUMBER OF GET NEXTS THIS OPEN             0
 RM NOTE  1010 ON LFN PMASTER      ***TOTAL DISKAREA***          640  WORDS
```

Figure 15-4. Example of a FIT Dump

MULTIPLE-INDEX FILES

All AAM files must have a key associated with each record. This key, called the primary key, is used by CRM to locate the records in the file when the file is read or written randomly.

Additional keys, called alternate keys, can be defined through COBOL statements when the data file is created, or through an index generation utility. The MIPGEN utility is used to define alternate keys for an existing file. It is usually more efficient to use MIPGEN to add alternate keys to an existing file than to define the alternate keys when the data file is created. MIPGEN is also used to redefine or delete existing alternate keys.

AAM creates an index file consisting of entries for each alternate key defined for a data file. AAM updates the index file whenever the data file is updated, or whenever the MIPGEN utility is used to add, delete, or replace alternate keys.

DEFINING ALTERNATE KEYS WITH MIPGEN

MIPGEN is a utility used for AAM files to:

- Define alternate keys and create an index file for an existing file that does not have them.
- Define or modify alternate keys for an existing multiple index file.

The MIPGEN utility can only be called by control statements. It reads directives from a file and creates or modifies an index file according to those directives and the contents of the data file.

If MIPGEN is being used to define a new alternate key and no index file currently exists, permanent file space must be allocated for the index file (with write permission). If MIPGEN is being used to redefine existing alternate keys, the existing index file must be attached (with write permission).

The MIPGEN utility is the only method by which the user can change alternate key definitions after an index file has been created. The COBOL user cannot add, replace, or delete alternate keys through source language statements.

Once an index file has been associated with a data file, the index file must be attached to any job that either:

- Attaches and updates the data file

or

- Retrieves records by alternate key.

Figure 15-5 illustrates the MIPGEN utility on the NOS operating system.

POSITIONING AND READING A FILE BY ALTERNATE KEY

When the data file is read by alternate key, the index entries for that alternate key are searched for the desired alternate key value. The first primary key in the list of primary keys associated with that alternate key value is used to retrieve a record in the data file containing the desired alternate key value. Subsequent records also containing that alternate key value can be retrieved by executing a sequential read by alternate key.

```
MOVE "PNE44" TO WHERE-USED
START PMASTER KEY EQUALS WHERE-USED
    INVALID KEY PERFORM SEE-CODE.
    .
    .
    .
READ PMASTER NEXT RECORD AT END GO TO
    START-IT.
```

These statements cause the index file to be positioned at the alternate key entry for WHERE-USED with a value of PNE44. The index file can then be read sequentially with READ...NEXT. Figure 15-6 illustrates a COBOL program that accesses the file LIBCONG by its alternate keys (STACK and WROTE-IT).

FILE STRUCTURE AND EFFICIENCY CONSIDERATIONS

The structure of a file and the method in which a file is accessed involves:

- The physical format of its records (record type).
- The physical grouping of records into blocks.
- The record length (and whether fixed or variable).
- Any record compression.
- The amount of padding, if any, used within blocks of records.
- The existence and composition of keys (and whether or not embedded).
- The existence of indexes used to locate records.
- The sequential or random nature of storing records.
- The size of a file.
- Any labels attached to the beginning of a file.
- File boundaries.

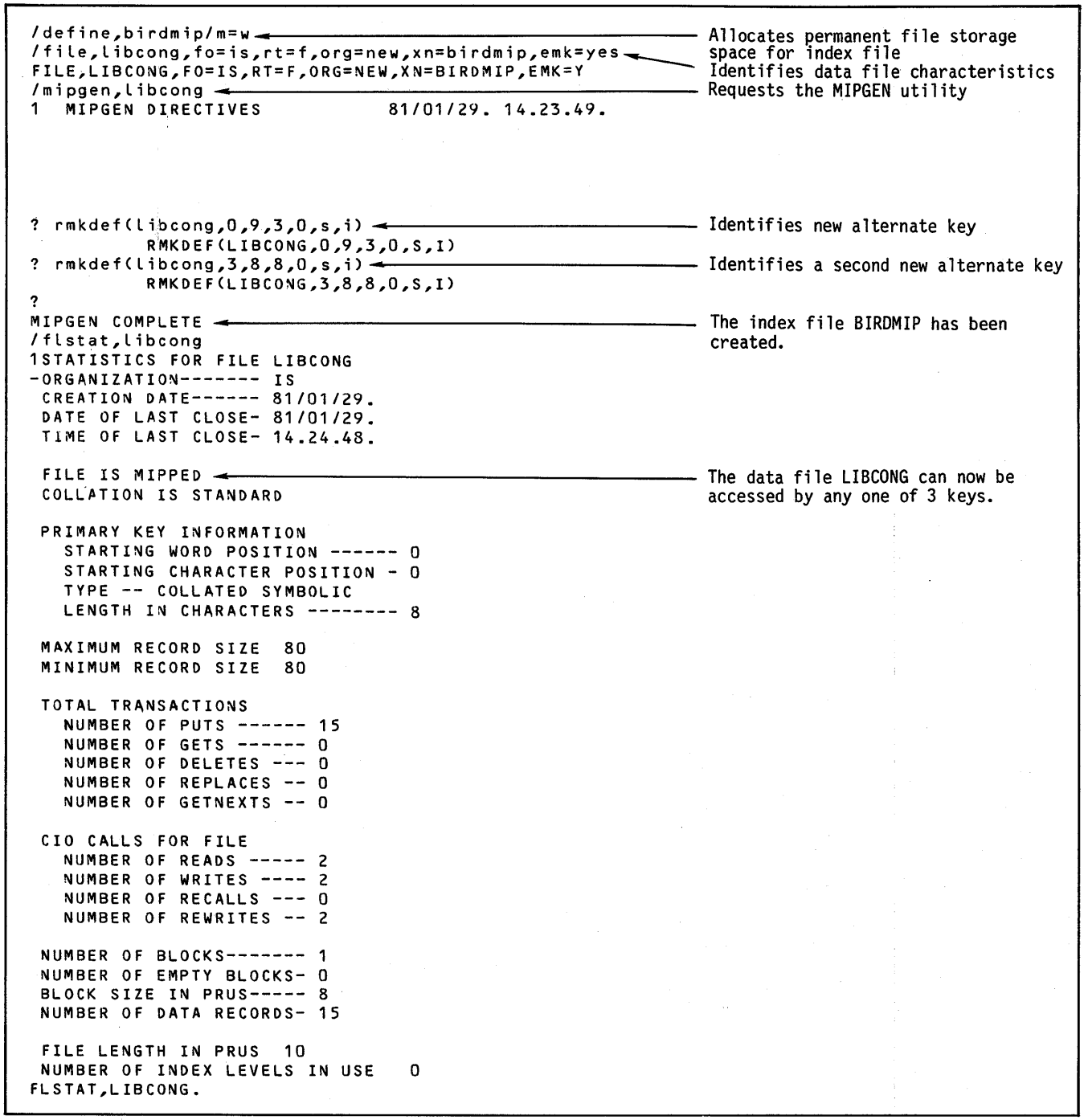


Figure 15-5. MIPGEN Example - NOS

Control Statements

```
/attach,libcong/m=w
/attach,birdmip/m=w
/file,libcong,fo=is,rt=f,org=new,xn=birdmip,emk=yes
FILE,LIBCONG,FO=IS,RT=F,ORG=NEW,XN=BIRDMIP,EMK=Y
/lgo
```

Source Program

```
IDENTIFICATION DIVISION.
PROGRAM-ID. BIRD-BOOKS.
ENVIRONMENT DIVISION.
INPUT-OUTPUT SECTION.
FILE-CONTROL.
    SELECT LIBCONG ASSIGN TO "LIBCONG" BIRDMIP
    USE "EFC=3, DFC=3"
    FILE STATUS IS CODE-RETURN
    ORGANIZATION IS INDEXED
    ACCESS MODE IS DYNAMIC
    RECORD KEY IS LIBRARY-CONG
    ALTERNATE RECORD KEY IS WROTE-IT WITH DUPLICATES ASCENDING
    ALTERNATE RECORD KEY IS STACK WITH DUPLICATES ASCENDING.
DATA DIVISION.
FILE SECTION.
FD  LIBCONG
    LABEL RECORD OMITTED
    DATA RECORD IS B-REC.
01  B-REC.
    05  LIBRARY-CONG PICTURE X(8).
    05  FILLER PICTURE X.
    05  STACK PICTURE X(3).
    05  TITLE PICTURE X(26).
    05  WROTE-IT PICTURE X(8).
    05  FILLER PICTURE X(12).
    05  REC-NO PICTURE XX.
    05  FILLER PICTURE X(20).
WORKING-STORAGE SECTION.
01 CODE-RETURN PICTURE XX.
01 ERR-CODE PICTURE 9999 USAGE IS COMP-1.
01 SEVERITY PICTURE X.
PROCEDURE DIVISION.
OPEN-EM.
    OPEN I-O LIBCONG.
    DISPLAY SPACES.
READ-PRIME.
    READ LIBCONG NEXT RECORD
        AT END GO TO START-IT.
    DISPLAY B-REC.
    GO TO READ-PRIME.
START-IT.
    DISPLAY SPACES.
    DISPLAY SPACES.
    DISPLAY "----PRINTED FROM START-IT PARAGRAPH---".
        DISPLAY SPACES.
    MOVE "13A" TO STACK.
    START LIBCONG KEY EQUALS STACK
        INVALID KEY PERFORM SEE-CODE.
SEE-STACK-A.
    READ LIBCONG NEXT RECORD
        AT END GO TO END-INDEX.
    DISPLAY B-REC, "  " CODE-RETURN.
    IF CODE-RETURN NOT EQUAL "02" THEN
        DISPLAY "END OF RECORDS WITH ALTERNATE KEY    " STACK
        GO TO NEXT-REC
        END-IF.
    GO TO SEE-STACK-A.
```

Defines data file LIBCONG and index file BIRDMIP

Identifies primary key

Identifies 2 alternate keys: WROTE-IT and STACK

Identifies items to contain error information when ENTER C.IOST is executed

Processes the entire file

Processes all records with key value 13A in the STACK field

Figure 15-6. Reading a File by Alternate Key (Sheet 1 of 3)

Source Program (Contd)

```
NEXT-REC.
    READ LIBCONG NEXT RECORD AT END DISPLAY SPACES.
    DISPLAY " STACK VALUE AFTER SEQUENTIAL READ IS " STACK
        " PRIME RECORD KEY IS " LIBRARY-CONG.
AUTHOR-SEE.
    DISPLAY SPACES.
    MOVE "A       " TO WROTE-IT.
    START LIBCONG KEY GREATER THAN WROTE-IT
        INVALID KEY DISPLAY " BAD AUTHOR-SEE START".
    DISPLAY SPACES.
    DISPLAY "----AUTHOR-SEE EXECUTING---".
SEE-INDEX.
    READ LIBCONG NEXT RECORD AT END GO TO DONE.
    DISPLAY B-REC.
    GO TO SEE-INDEX.
SEE-CODE.
    DISPLAY " ERROR OCCURED".
    DISPLAY "     CODE IN CODE-RETURN IS " CODE-RETURN.
    ENTER "C.IOST" USING LIBCONG ERR-CODE SEVERITY.
    DISPLAY "     CRM CODE IS "
         ERR-CODE " SEVERITY IS " SEVERITY.
END-INDEX.
    DISPLAY " END OF INDEX REACHED IN SEE-STACK-A".
DONE.
     CLOSE LIBCONG.
     STOP RUN.
```

Processes all records alphabetically by the WROTE-IT key

Output

```
618998   13B ICELAND SUMMER            SUTTON          03

6216745  12B TREASURY OF BIRDLORE      KRUTCH          02

6352622  12A HNDBK BIRDS INDIA & PAK   RIPLEY          15

6823181  12A BIRD NAVIGATION           MATTHEWS        07

6855824  12A COURTSHIP HABITS GREBES   HUXLEY          13

72143678 13A BIRD STUDY                BERGER          04

72151434 13B HIGH ARCTIC               SUTTON          09

7283709  12B BEND IN A MEXICAN RIVER   SUTTON          06

7317833  13A ORNITHOLOGY OF US & CAN.  NUTTALL         10

736826   12B HNDBK OF WESTERN BIRDS    BROWN           11

7415911  13A PORTRAITS MEXICAN BIRDS   SUTTON          14

745811   12B BIRDS OF THE SEYCHELLES   PENNY           12

75122251 12A HUNGRY BIRD BOOK          ARBIB           01

7520430  12A FUND. OF ORNITHOLOGY      VAN TYNE        05

75619273 13A RAILS OF THE WORLD        RIPLEY          08


----PRINTED FROM START-IT PARAGRAPH---

72143678 13A BIRD STUDY                BERGER          04
           02
7317833  13A ORNITHOLOGY OF US & CAN.  NUTTALL         10
           02
```

Figure 15-6. Reading a File by Alternate Key (Sheet 2 of 3)

Output (Contd)

```
7415911   13A PORTRAITS MEXICAN BIRDS  SUTTON              14
            02
75619273  13A RAILS OF THE WORLD       RIPLEY              08
            00
END OF RECORDS WITH ALTERNATE KEY   13A
 STACK VALUE AFTER SEQUENTIAL READ IS 13B PRIME RECORD KEY IS 618998


----AUTHOR-SEE EXECUTING---
75122251  12A HUNGRY BIRD BOOK         ARBIB               01

72143678  13A BIRD STUDY               BERGER              04

736826    12B HNDBK OF WESTERN BIRDS   BROWN               11

6855824   12A COURTSHIP HABITS GREBES  HUXLEY              13

6216745   12B TREASURY OF BIRDLORE     KRUTCH              02

6823181   12A BIRD NAVIGATION          MATTHEWS            07

7317833   13A ORNITHOLOGY OF US & CAN. NUTTALL             10

745811    12B BIRDS OF THE SEYCHELLES  PENNY               12

6352622   12A HNDBK BIRDS INDIA & PAK  RIPLEY              15

75619273  13A RAILS OF THE WORLD       RIPLEY              08

618998    13B ICELAND SUMMER           SUTTON              03

72151434  13B HIGH ARCTIC              SUTTON              09

7283709   12B BEND IN A MEXICAN RIVER  SUTTON              06

7415911   13A PORTRAITS MEXICAN BIRDS  SUTTON              14

7520430   12A FUND. OF ORNITHOLOGY     VAN TYNE            05

 RM NOTE  1010 ON LFN LIBCONG         640
/
```

Figure 15-6. Reading a File by Alternate Key (Sheet 3 of 3)

Since the terminology for file components and boundaries is not consistent, table 15-6 shows the general terms used by CRM and the equivalent terms for COBOL, operating system, disk, card, and tape.

Several CRM utilities can be helpful in determining the most efficient file structure when designing new files, and in processing and maintaining existing files. A brief description of some of the utilities most useful to the experienced COBOL user are discussed in the following paragraphs. Refer to the BAM or AAM reference manuals or user's guides for more detail and usage. File reformatting is described in the FORM reference manual.

DETERMINING THE BEST BLOCK SIZE

If a file is usually processed sequentially, large data blocks are most efficient because the next record to be accessed usually exists in the block currently in the buffer and transfer of another block into central memory is not required. If a file is usually processed randomly, small data blocks are most efficient because the next record to be accessed usually does not exist in the block currently in the buffer in central memory. Therefore, fewer records are handled in the block transfer.

The system uses the BLOCK CONTAINS clause to determine block size for sequential, indexed, direct, and actual-key file organizations. For sequential files, blocking is dependent on the block type and the device on which the file resides. For indexed, direct, and actual-key files, block size is always a multiple of PRU size. For any block size specified by the program, CRM adds five words (50 characters) for variable length records or four words (40 characters) for fixed length records, and rounds upward to the next PRU. For example, if the block contains 590 characters, one PRU is involved. A 591-character block involves two PRUs.

NOTE

If records are of variable length, the number of records specified in the BLOCK CONTAINS clause might not fit in one block because CRM requires an additional one-half word for each record in an indexed, direct, or actual-key file.

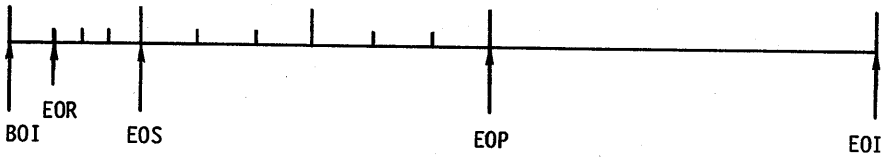
Buffer length can then be specified directly through the BFS parameter in the USE clause or on the FILE control statement.

TABLE 15-6. CRM FILE STRUCTURE TERMS AND EQUIVALENTS

General File Terminology	COBOL Term	CRM Term	NOS and NOS/BE Terms	Disk	Card	SI Tape
Beginning of physical file	BOI	BOI	BOI	-	-	-
A single record	Line image	Data record	Line image (NOS) Unit record (NOS/BE)	-	Card	Line image
End of record	-	EOR	-	-	-	-
End of group of records (end of system logical record)	EOF (for file named INPUT only)	EOS	EOR	Short PRU	7/8/9	Short PRU
End of larger group of records (end of logical file)	EOF	EOP	EOF	Short PRU	6/7/9 (NOS only)	Short PRU
End of physical file	EOF	EOI	EOF	End of table	6/7/8/9	Tape mark

BOI = Beginning-of-information
EOR = End-of-record
EOF = End-of-file

EOS = End-of-section
EOP = End-of-partition
EOI = End-of-information




```
SELECT MY-FILE ASSIGN TO MYFILE
    USE "BFS=3000"
```

This statement sets the buffer length to 3000 characters for the sequential file MYFILE. For sequential files, one PRU is the default. However, a larger buffer size reduces the number of times I-O is performed and increases overall performance.

Indexed Sequential Block Size

Block size has a major effect on both the physical structure and the performance of an indexed sequential file. AAM uses the values in various FIT fields when data blocks are created. When creating data blocks, the COBOL user has three options:

- Specify the BLOCK CONTAINS and the RECORDS CONTAIN clauses. The numbers specified in these clauses are then used by COBOL to calculate MBL. (Refer to the discussion in section 3 entitled File Description Entry for an example.)
- Define the MBL directly through the FILE control statement.
- Accept the default block size calculated by AAM.

MBL is the maximum number of characters in the data block. The block size specified by the MBL field must be large enough to hold at least one maximum-length record plus enough words to hold the primary key (if not embedded). The maximum record length (MRL) is determined by COBOL from the record description entries.

The FLBLOK utility should be used to help select the appropriate value for the MBL field. FLBLOK is described in the AAM reference manual.

AAM increases the specified MBL to use mass storage efficiently. The resulting data block size (MBL) will be:

[(Specified MBL + 50 characters) rounded to the next PRU multiple] - 20 characters

If the user wants to set MBL to specify a certain number of PRUs per block, the BLOCK CONTAINS clause values should be set to 590, 1230, 1870, 2510, and so on (that is 50 characters less than the PRU multiple desired).

Actual-Key Block Size

Block size has a major effect on both the physical structure and the performance of an actual-key file. AAM uses the values in various FIT fields when data blocks are created. When creating data blocks, the COBOL user has three options:

- Specify the BLOCK CONTAINS and the RECORDS CONTAIN clauses. The numbers specified in these clauses are then used by COBOL to calculate MBL. (Refer to the discussion in section 3 entitled File Description Entry for an example.)
- Define the MBL directly through the FILE control statement.
- Accept the default block size calculated by AAM.

The user can determine the data block size by defining the maximum block length (MBL) field and the number of records per block (RB) field, as follows:

1. Set MBL by multiplying the average number of characters per record (RL) by an estimated number of records per block (RB). Start with 3 to 10 records per block to favor random processing and to keep central memory usage low.
2. Round RL times RB up to a PRU multiple and subtract 50 characters that are needed by AAM. The resulting MBL characters is the physical limitation on the size of data blocks.
3. Adjust the number of records per block used in step 1 so that RB times RL is as close to MBL as possible. By specifying an RB value in the BLOCK CONTAINS ... RECORDS clause, the user sets a maximum number of records that can be stored logically within each data block that has MBL characters.

AAM increases the specified MBL to use mass storage efficiently. The resulting data block size (MBL) will be:

[(Specified MBL + 50 characters) rounded to the next PRU multiple] - 20 characters

If the user wants to set MBL to specify a certain number of PRUs per block, the BLOCK CONTAINS clause should be set to 590, 1230, 1870, 2510, and so on (that is, 50 characters less than the PRU multiple desired).

REDUCING DIRECT FILE CREATION TIME

CREATE is a utility that can be called through a COBOL program to create a direct file with embedded keys. The default hashing routine or a user-supplied hashing routine can be used. CREATE should be used for large files (more than 1000 records). It significantly reduces creation time, since all records that hash to a given home block can be written in one mass storage access; otherwise, a home block must be transferred from the central memory buffer to mass storage for each record written.

The CREATE utility hashes the key from an input record and prefixes the key to the record. Sort/Merge is then used to sort the hashed keys. After the sort operation, the prefixed keys are removed and the CREATE utility uses AAM to produce the direct file.

The following COBOL statements illustrate a call to the CREATE utility for creation of a direct file.

```
SELECT DAFILE ASSIGN TO "DAFILE"
ORGANIZATION IS DIRECT
    .
    .
    .
ENTER SDACRTU USING REC1
    CUSTOMER-ID REC-SIZE
    .
    .
    .
ENTER SDAENDC.
```

The ENTER SDACRTU statement must exist within a loop for each record read. The statement generates a hashed value for the primary key value for CUSTOMER-ID. The number of characters in the record is indicated by REC-SIZE. The first time this statement is executed, the CREATE directive is expected as the next unexecuted record in the file named INPUT. The following statement follows the end-of-record mark (or 7/8/9 card) at the end of the COBOL source program:

```
CREATE(DAFILE)
```

When the ENTER SDAENDC statement is executed, the records are sorted by the hashed primary key values. AAM then creates the direct file. The file DAFILE is not explicitly opened through COBOL statements.

A FILE control statement must be supplied before execution of the COBOL program that creates the direct file with CREATE. The following FILE control statements can be used to describe the direct file DAFILE that is to be created.

```
FILE(DAFILE,FO=DA,ORG=NEW,HMB=3,MBL=100,
    EFC=3)
FILE(DAFILE,RT=F,MRL=50,KT=S,KL=10,RKW=2,
    RKP=0,EMK=YES)
```

The SORT library must be available in the system when using the CREATE utility.

COBOL programs can be created, compiled, and executed through a terminal. Interactive processing enables a user at a remote site to enter a COBOL source program into a terminal. Compilation and execution results can be displayed at the terminal. This section illustrates interactive processing of COBOL programs in which a command is entered into a terminal, the command is processed, a response is displayed, another command is entered, and so on. Description of remote batch processing is not included.

It is assumed that the COBOL user is not familiar with any standard CDC communications product - such as INTERCOM, Interactive Facility (IAF), or Transaction Facility (TAF) - but is familiar with procedures for using permanent files and batch processing under either the NOS or the NOS/BE operating system. Specifically, it is assumed that the NOS user is familiar with the ATTACH, ASSIGN, DEFINE, and SAVE commands. It is assumed that the NOS/BE user is familiar with the ATTACH, RETURN, REWIND, COPY, and CATALOG commands. It is also assumed that the user is familiar with the system files INPUT, OUTPUT, and LGO.

This section includes the following:

- Some basic concepts of terminal usage
- A method of creating and executing COBOL programs through a terminal, using either IAF under the NOS operating system or INTERCOM under the NOS/BE operating system
- Some interactive uses of the COBOL verbs ACCEPT and DISPLAY

In all examples, underlining indicates terminal user input. All user input shown in the examples is to be terminated by pressing a carriage RETURN key (or its equivalent).

CONCEPTS OF TERMINAL OPERATION

Terminal operations require the use of commands. A command is a user entry that calls for action from the communication software product in use. Commands are identified by a keyword. Optional parameters can follow the keyword and are separated by commas. Commands are not complete until a RETURN key (or its equivalent) is pressed. Command formats and parameters shown in this section represent a subset of the commands and parameters available. Refer to the appropriate communications software product manual for a complete list.

Terminal operations require the establishment of a communication link between the terminal and the computer at the central site. The terminal might be hard-wired, or a dial-in procedure might be necessary. A login procedure is usually required to identify the terminal to the system, to identify the user's right to use the system, and to establish accounting information. These procedures differ according to the operating system, the interactive facility, and the installation. The user should have access to any procedure required to establish a connection with the computer, to log in, and to log out.

The period of time between the login and the logout is called a terminal session. Files available during a given terminal session are called local files. Most files used by the beginning terminal user are local files. Before a file can be used at a terminal it must be a local file; local file status results automatically from most user commands. A file's local status pertains only to a given terminal session.

Files created during a terminal session must be preserved (allocated to permanent mass storage) at the end of a terminal session if they are to be used in a future session. A permanent file can be attached for use in a terminal session, thus becoming a local file for the duration of the session. Local files can include temporary files as well as attached permanent files.

The following summarizes the characteristics of a local file under both the NOS and the NOS/BE operating system.

- A local file is immediately accessible from a terminal.
- A local file is a file that is created during the current terminal session or is an attached permanent file (or an accessible copy of a permanent file).
- A local file can be made permanent if it is not already permanent.
- A local file can be used during execution of a program.
- A local file is lost at logout unless it is has been made permanent.
- A local file must have a unique name; that is, only one local file can exist with a given file name.
- There is a limited number of local files for a terminal session. A message appears at the terminal when the limit for a particular installation is reached.

NOS TERMINAL OPERATIONS USING IAF

The Interactive Facility (IAF) is a network product that provides a terminal user with the interactive capabilities of NOS. The following paragraphs contain procedures that can be used to create and execute a COBOL program through IAF (Batch subsystem), under the NOS operating system.

Two commands are useful in most NOS terminal operations:

- ENQUIRE,F is used to obtain a list of local files.
- CATLIST is used to obtain a list of permanent files.

LOCAL FILES UNDER NOS

Local files in NOS terminal operations include the following:

- Temporary files created during the current terminal session.

- Accessible copies of indirect access permanent files.
- Attached direct access permanent files.
- Files assigned to a terminal through the ASSIGN,TT command.

Local file names are listed at the terminal with a file type of LO, PM, or PT when a ENQUIRE,F command is entered. A direct access permanent file is indicated by PM. A primary file is indicated by PT. All other local files are indicated by LO. Local file characteristics are listed in the earlier discussion on basic terminal concepts.

Temporary files created during the current terminal session are files that are used only during the terminal session. A temporary file is lost when the terminal user logs off. A file created through an interactive text editor facility is a temporary file until it is saved as a permanent file.

A local file can be assigned to the terminal through the ASSIGN,TT,lfn command, where lfn is a local file name. Upon login, the system files INPUT and OUTPUT are assigned by default to the terminal. This concept is equivalent to the concept of connected files under the NOS/BE system. When the term connected file is used (in this guide, under the NOS system), it means that a file has been assigned to TT with the ASSIGN command or that a file has been assigned to the terminal by default.

PROGRAM CREATION USING FSE AND IAF UNDER NOS

The Full Screen Editor can be used to create and modify a program. The program being created or modified is contained in a temporary file called an edit file. The edit file must be saved before the user logs off or it will be lost. Saving the edit file is discussed in a later subsection.

Figure 16-1 illustrates how to create, list, and alter a COBOL program by using FSE in line mode. The user entries in figure 16-1 are described below:

1. Enter the FSE command to call the Full Screen Editor and assign the file named EXPROG as the file to contain the source program:

   ```
   FSE,EXPROC
   ```

   FSE responds with:

   ```
   CREATE: EXPROG
   NOS FULL SCREEN EDITOR
   SCOPE TABS SET
   ??
   ```

2. Enter SET TAB 8,12,16,20 to set the tab stop positions corresponding to columns 8, 12, 16, and 20. FSE responds with the ?? prompt.
3. Enter SET CHAR SEMI to define the semicolon character as the tab control character. FSE responds with the ?? prompt.
4. Enter the directive INSERT to enter input mode. FSE responds with the 1 ? prompt.
5. Enter the source program, line by line, pressing RETURN after each line. After entering the last line, press RETURN again. FSE responds with the ?? prompt, indicating that you can again enter directives.
6. Enter the PA directive to list the program. This displays the entire program on the terminal screen.
7. For further editing, a pointer must be positioned to the line of text that is to be changed. Entering P16 positions the pointer to the line of text to be changed, in this case line 16.
8. Enter the ALTER directive. FSE responds by displaying line 16 again, followed by the prompt:

   ```
   A??
   ```

9. You then space over to where the change is to be made and enter the correction underneath the error. When you press RETURN, the corrected line is displayed.

You can make further changes to the program by using other FSE directives, some of which are described below.

- The INSERT directive allows you to insert new lines. The following example inserts a new line after existing line 14:

  ```
  i14
  ```

  FSE prompts you with:

  ```
  15 ?
  ```

  Enter the new line and then press RETURN twice. FSE responds with the ?? prompt.

- The DELETE directive allows you to delete a line. The following example deletes line 9:

  ```
  d9
  ```

  The deleted line is then displayed at your terminal, followed by the ?? prompt.

Entering the QUIT REPLACE directive terminates editing and saves the edit file as a permanent file. The old version of the file is replaced with the new version. Entering the QUIT directive terminates editing without saving the revisions to the edit file.

For further information on the NOS Full Screen Editor, refer to the NOS Full Screen Editor User's Guide.

PROGRAM COMPILATION AND EXECUTION UNDER NOS

The following discussion includes two aspects of interactive program execution: the compilation and execution of a program that has been created through FSE, and the execution of a program with local data files.

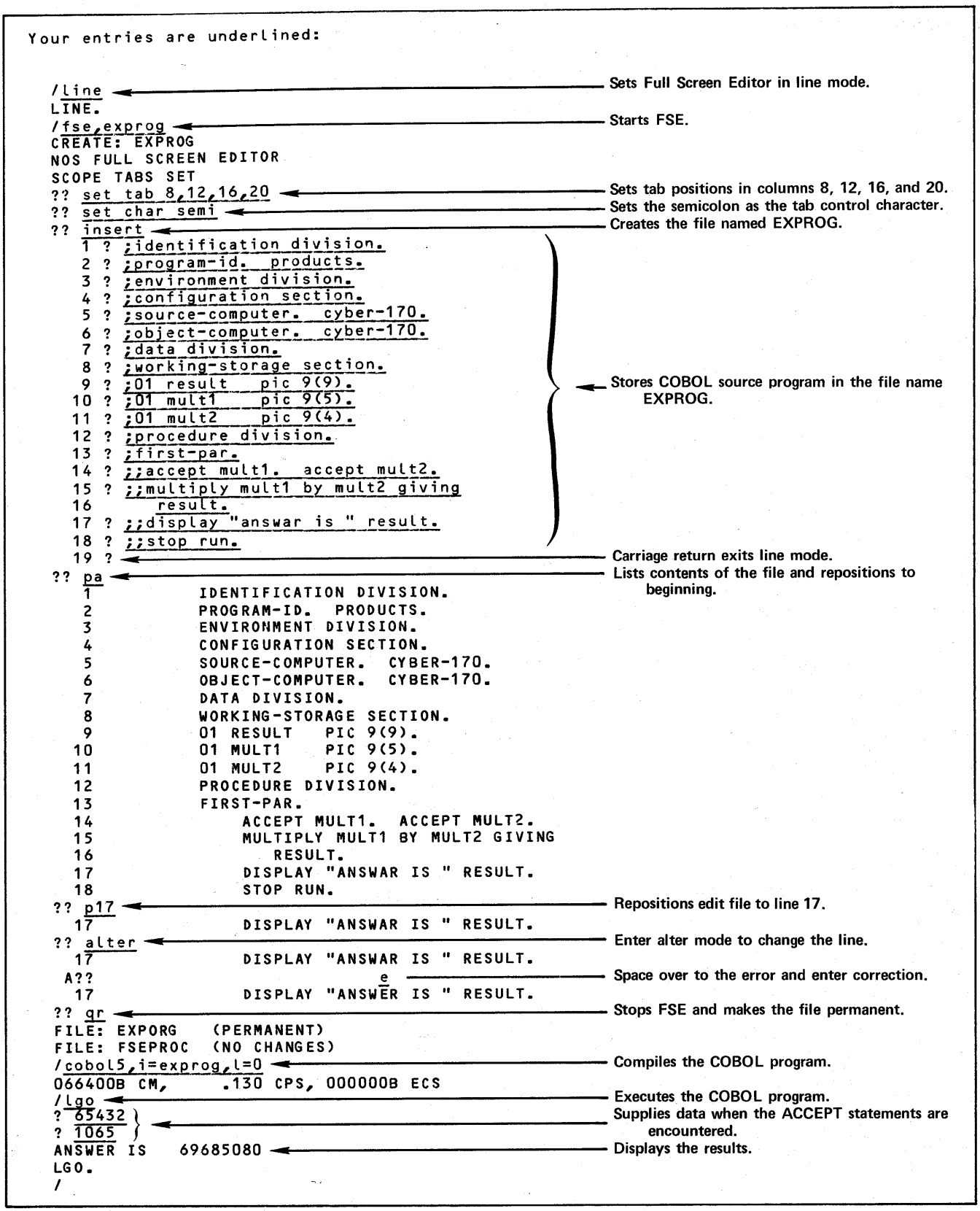


Figure 16-1. FSE Program Creation, Compilation, and Execution

Running the Program

The COBOL program can be compiled through the terminal by entering the COBOL5 control statement, specifying input (I parameter) as the local file name that contains the source program. Other parameters can also be specified, if desired.

```
COBOL5,I=EXPROG,L=LISTFIL.
```

This COBOL5 control statement causes relocatable binary instructions to be created on the file named LGO. The file EXPROG contains the source program and is used for compilation input. The file LISTFIL contains the source listing and any diagnostics after the statement has been executed. (If the L parameter is omitted, compilation results are displayed at the terminal, unless ASSIGN,MS,OUTPUT has been specified.) If compilation errors exist, the program must be changed, the file rewound, and the program compiled again.

When no errors exist, the executable instructions can be loaded and executed through the terminal by entering the command LGO. When execution is completed, the system responds by displaying the characters LGO and rewinds the file LGO.

Figure 16-1 illustrates program compilation and execution.

When a COBOL program has been created through FSE, with line sequence numbers and if the PSQ parameter is used on the COBOL5 control statement, the sequence numbers are used for any diagnostic message references. An example is shown in figure 16-2.

Executing With Local Data Files

A COBOL program can be run using local input data files. Program execution can involve one or more of the following possibilities:

- A data file that has been created through FSE is used by the program.
- A permanent direct access file is attached or a permanent indirect access file copy is used as input data.
- Data is input directly through the terminal.

Using Files Created Through FSE

The file entered through FSE does not need to be a program. Any type of character data can be input. In addition to creating a program, the user can create a file of data that will be read by a program.

All local data files created through FSE have system symbols to mark the end of the file. FSE automatically inserts end-of-record and end-of-file markers at the end of each edit file. When a file is read by a executing COBOL program, the end-of-file marker is sensed by the AT END clause of the COBOL READ statement.

By using the (EOR) and (EOF) directives, similar markers can be written to separate multirecord data files under construction in FSE:

- (EOR) corresponds to a 7/8/9 end-of-record card.
- (EOF) corresponds to a 6/7/8/9 end-of-file card.

Using Attached Permanent Files

A COBOL program can use an attached permanent file as input data. Record format can be any format allowed by COBOL.

Direct access permanent files can be altered through FSE only if they are attached in write mode; M=W must be specified on the ATTACH statement.

The use of the NOS system permanent file commands is similar in both an interactive environment and a batch environment. With interactive use, however, the user must exit FSE before issuing NOS permanent file commands.

Using Files Assigned to The Terminal

The COBOL ACCEPT and DISPLAY statements can be used to read and write small volume data on a file assigned to the terminal. Data can be input directly through the terminal to an executing program by using the ACCEPT verb. Data can be displayed upon (or written to) a file by using the DISPLAY verb. More detail is provided in the subsection that discusses interactive usage of COBOL ACCEPT and DISPLAY statements.

```
ASSIGN,TT,ANYFILE
```

This NOS command assigns the file ANYFILE to the terminal.

NOS/BE TERMINAL OPERATIONS USING INTERCOM

INTERCOM Version 5 provides the interactive terminal user with the time-sharing capabilities of NOS/BE. The following paragraphs contain procedures that can be used to create and execute a COBOL program through INTERCOM and the NOS/BE operating system.

Several commands are useful in most NOS/BE terminal operations:

- FILES is used to obtain a list of local files.
- AUDIT,AI=P,ID=USERID is used to obtain a list of the permanent files cataloged under the name USERID. The system file OUTPUT must be connected before the AUDIT command is entered if the response is to be displayed at the terminal.
- CONNECT,ANYFILE is used to connect the file ANYFILE to the terminal.
- DISCONT,ANYFILE is used to disconnect the file ANYFILE from the terminal.

LOCAL FILES UNDER NOS/BE

Local files, in NOS/BE terminal operations, include the following:

- Temporary files created during the current terminal session.
- Permanent mass storage files that have been attached.
- Connected files.

```
/cobol5,i=inter,psq
```

Compiles COBOL program with the PSQ parameter.

```
CDC COBOL 5.3 - LEVEL 642       SOURCE LISTING OF INTERAC
```

Sequence line numbers can be entered through the FSE command SET NUMBER B.

```
00100  IDENTIFICATION DIVISION.
00110  PROGRAM-ID. INTERACTIVE
00120  ENVIRONMENT DIVISION.
00130  CONFIGURATION SECTION.
00140  INPUT-OUTPUT SECTION.
00150  FILE-CONTROL.
00160  DATA DIVISION.
00170  WORKING-STORAGE SECTION.
00180  01 REC-NO PIC 99.
00190  PROCEDURE DIVISION.
00200  FIRST-PAR.
00210      MOVE 10 TO RC-NO.
00220      DISPLAY "ANSWER IS REC-NO.
00230      STOP RUN.

COLUMN   1         2         3         4         5         6         7         8
12345678901234567890123456789012345678901234567890123456789012345678901234567890


CDC COBOL 5.3 - LEVEL 642          DIAGNOSTICS IN INTERAC      AOPT= 64/CDC/CDCS2
SEV LINE COL ERROR
```

Sequence line numbers used in diagnostic messages.

```
 W   120   8  1027  A PERIOD IS REQUIRED AFTER THE PRECEDING ELEMENT.

 F   210  23  7994  UNDEFINED DATA NAME REFERENCE.

 F   220  20  1011  THIS NON-NUMERIC LITERAL HAS NO TERMINAL QUOTE.  A QUOTE FOLLOWING COLUMN 72 IS ASSUMED.

                    **      3 ERRORS LISTED               **

065000B CM,      .314 CPS, 000000B ECS
/
```

Figure 16-2. PSQ Parameter Example

When a FILES command is entered, local files are listed at the terminal under the LOCAL category. The previous discussion on terminal concepts includes the characteristics of local files.

Files that are used only during a given terminal session are called temporary files. A temporary file is lost when the terminal user logs off. Until it is saved as a permanent file, a file that is created through the interactive text editing facility of INTERCOM is a temporary file.

A file created through a REWIND or a COPY command is automatically local and is a temporary file unless it is made permanent through a CATALOG command.

Permanent files must be made local by using the ATTACH command if they are to be used at the terminal. The file's permanent status is retained. The RETURN command is used to disassociate a permanent file from the terminal. On the list of local files displayed by using the FILES command, attached permanent file names are preceded by an asterisk.

Connected files are associated with the keyboard or terminal display. Information written to a file while it is connected goes to the executing program for use or to the terminal for immediate display. No mass storage copy exists for the information. On the list of local files displayed by using the FILES command, connected files are preceded by a dollar sign.

The CONNECT command connects a file; the DISCONT command disconnects a file. A connected file must be disconnected before it can be disassociated from the terminal by using a RETURN command. The system file INPUT must be connected by using the CONNECT command if input is expected from the keyboard (as with the interactive usage of the COBOL ACCEPT verb). The system file OUTPUT must be connected if the printed results of a program compilation or execution are to be displayed at the terminal.

PROGRAM CREATION USING INTERCOM EDITOR UNDER NOS/BE

The EDITOR facility of INTERCOM can be used to create and modify a COBOL program. EDITOR has its own set of commands. All NOS/BE operating commands can be performed while in EDITOR mode.

Upon successfully logging in to the terminal, INTERCOM responds with the word COMMAND followed by a hyphen. Additional commands can then be entered.

Figure 16-3 illustrates program creation and listing through INTERCOM EDITOR. The user entries are described below.

The source program can be created as follows:

1. Enter EDITOR to call the editor program. INTERCOM responds with two dots, indicating that EDITOR is ready to accept a command.

2. Enter FORMAT,COBOL to set the semicolon (;) as the tab character and to set tab stop positions in columns 8, 12, 16, 20, and 24.

3. Enter CREATE to begin the process of building an edit file containing the COBOL program. The system responds with the first line sequence number, 100, followed by an equals sign.

4. Enter the program line by line, pressing RETURN after each line. Use the semicolon as a tab character to position the text at the columns designated as the COBOL tab stops.

5. Enter the single character = and press RETURN to exit from CREATE. INTERCOM supplies an end-of-record marker (equivalent to a CYBER Record Manager end-of-section marker) to the file at this point.

The program can be saved as a local file with a local file name (lfn), if desired. MYPROG (the file containing the program) is saved, with no sequence numbers, by using the following command:

```
SAVE,MYPROG,NOSEQ
```

The program can be examined by the LIST command in either of the following ways:

- LIST,2,10 displays the portion of the program between line 2 and line 10.
- LIST,ALL lists the entire program.

The file containing the program can be changed as follows:

- The linenum=newtext command adds or modifies a single line. The following adds or replaces the text beginning in column 12 on line 150 with the characters ITEM-NUMBER PIC 9(5).

```
150=;;ITEM-NUMBER PIC 9(5).
```

- The DELETE command deletes one or more lines. The following deletes the text on line 180.

```
DELETE,180
```

- The /oldtext/=/newtext/ command changes the text string TEXT to CHANGED-TEXT on line number 210.

```
/TEXT/=/CHANGED-TEXT/,210
```

The edit file MYPROG can be made local and the corrected source program can be substituted for the previous version, with the SAVE command.

```
SAVE,MYPROG,NOSEQ,O
```

This statement causes the edit file to overwrite the previously saved file MYPROG. A CATALOG command can be used to save the file as a permanent file. A REQUEST,lfn,PF statement is not needed because INTERCOM always places the edit file on a permanent file device.

If the program is saved as a permanent file MYPROG, the following commands can be used to later attach the file and place a copy in the edit file for review or change.

```
ATTACH,MYPROG,ID=USERID
EDIT,MYPROG,S
```

PROGRAM COMPILATION AND EXECUTION UNDER NOS/BE

The following discussion includes two aspects of interactive program execution: the compilation and execution of a program that has been created through EDITOR, and the execution of a program with local data files.

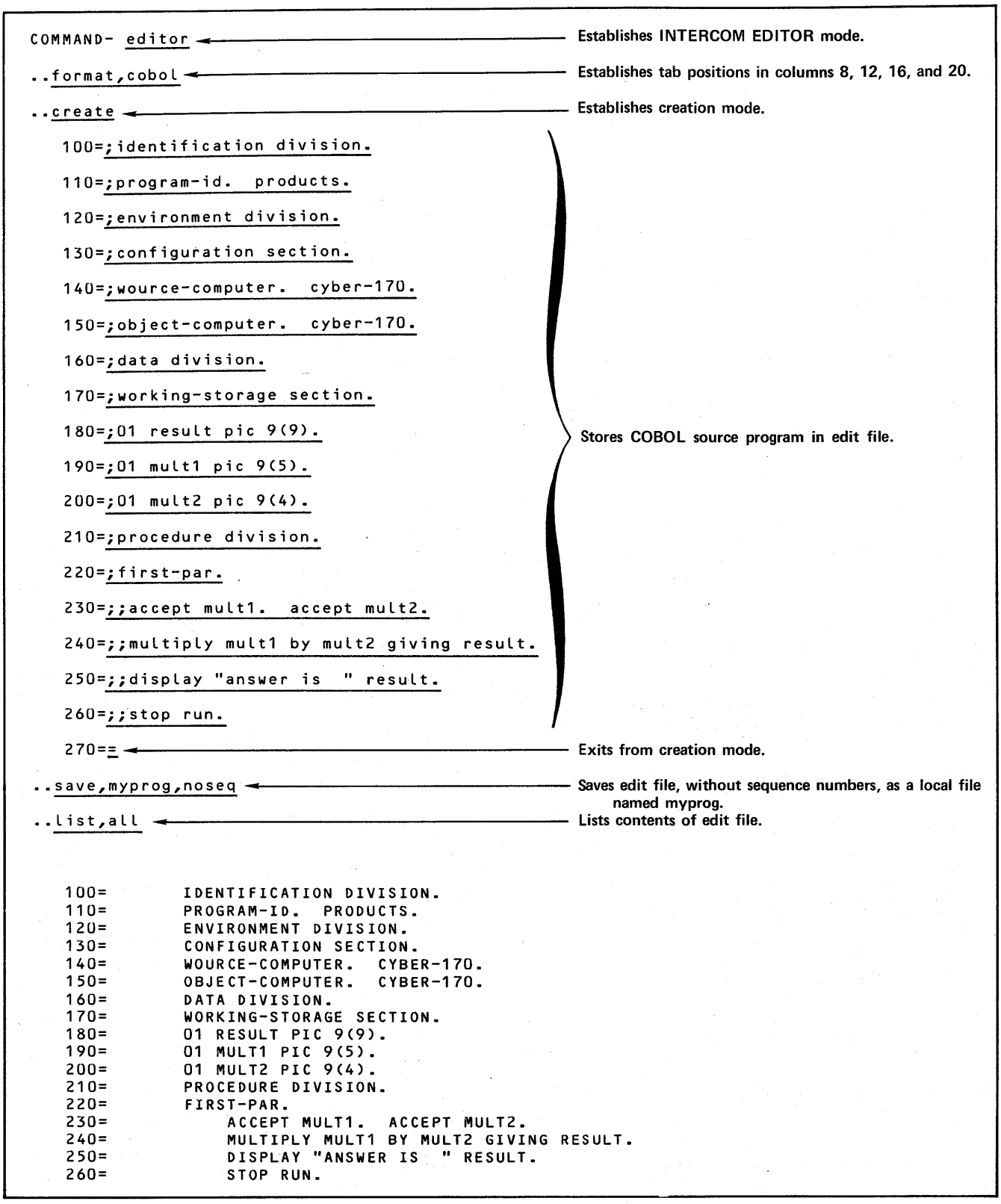


Figure 16-3. INTERCOM Program Creation, Compilation, and Execution (Sheet 1 of 2)

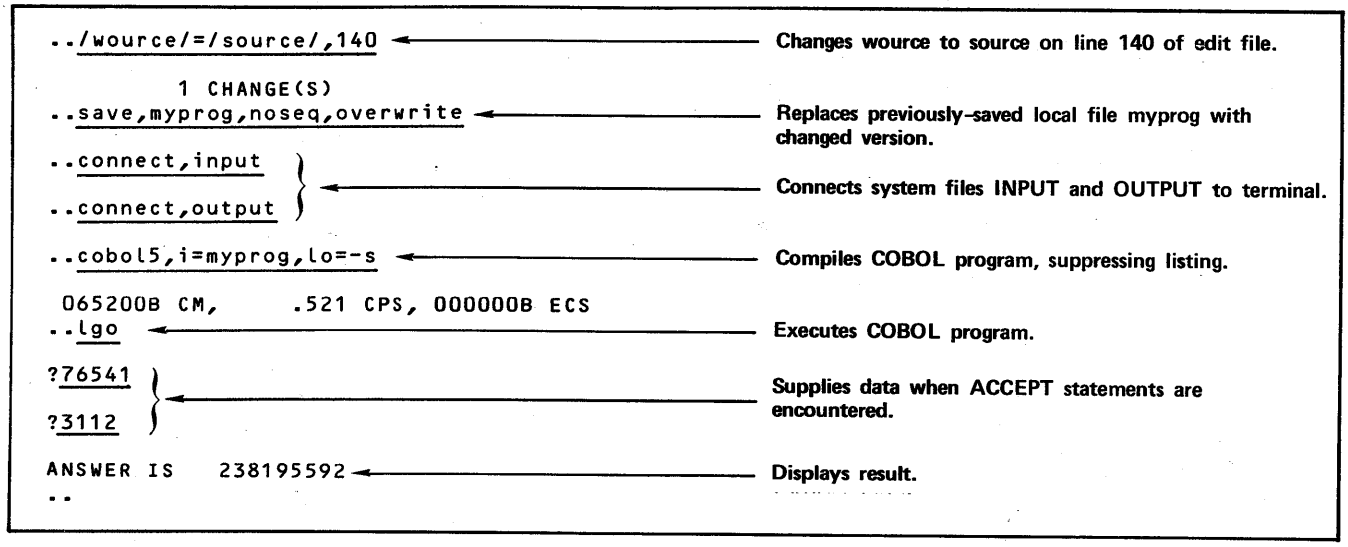


Figure 16-3. INTERCOM Program Creation, Compilation, and Execution (Sheet 2 of 2)

Running the Program

The COBOL program can be compiled through the terminal by entering the COBOL5 control statement, specifying input (I parameter) as the local file name that contains the saved source program. Other parameters can also be specified, if desired.

```
COBOL5,I=MYPROG,L=LISTFIL.
```

This COBOL5 control statement causes relocatable binary instructions to be created on the file named LGO. The file MYPROG contains the source program and is used for compilation input. The file LISTFIL contains the source listing and any diagnostics after the statement has been executed. (If the L parameter is omitted, and the OUTPUT file is connected, compilation results are displayed at the terminal.) If compilation errors exist, the program must be changed, the file rewound, and the program compiled again.

When no errors exist, the executable instructions can be loaded and executed through the terminal by entering the command LGO. When execution is complete, the system rewinds the file LGO.

Figure 16-3 illustrates program compilation and execution.

Execution With Local Data Files

A COBOL program can be run using local input data files. Program execution can involve one or more of the following possibilities:

- A data file that has been created through the EDITOR CREATE command is used by the program.
- A permanent file or files are attached for use as input data.
- A connected file is used to input data directly through the terminal.

COBOL rewinds data files whenever an OPEN or CLOSE statement is executed (unless the source program contains statements preventing the rewind); an exception exists for connected files, however. The system files INPUT and OUTPUT are always rewound by the software during interactive use. If a file needs to be repositioned to beginning-of-information, the REWIND command is used:

```
REWIND,FILENAME
```

This command rewinds the file FILENAME.

Using Files Created Through EDITOR

The file entered through EDITOR need not be a program. Any type of character data can be input. In addition to creating a program, the user can create a file of data to be read by a program.

All local data files created through EDITOR have end-of-record markers, designating the end of a system logical record; this is equivalent to a CYBER Record Manager end-of-section. When a file is read by an executing COBOL program, the file ending marker is sensed by the AT END imperative in the COBOL READ statement. Similar markers can be written to separate two data files under construction in EDITOR:

- *EOR corresponds to a 7/8/9 card.
- *EOI corresponds to a 6/7/8/9 card.

All local data files created through EDITOR have C-type blocking and Z-type record format. The COBOL compiler does not normally expect this format. Therefore, when using a file that has been created through EDITOR, the user must do one of the following:

- Specify RT=Z in the USE clause of the FILE-CONTROL paragraph.
- Specify RT=Z on a FILE control statement before execution.

Using Attached Permanent Files

A COBOL program can use an attached permanent file as input data. Record format can be any format allowed by COBOL. The following INTERCOM commands correspond to the CATALOG, ATTACH, and PURGE commands, respectively:

- STORE,FILENAME makes the local file FILENAME permanent.
- FETCH,FILENAME attaches the permanent file FILENAME.
- DISCARD,FILENAME deletes the local file FILENAME (which can be an attached permanent file).

Conceptually, there is no difference between the use of permanent files in interactive processing and the use of permanent files in batch processing.

Using Connected Files

Connected files are associated directly with the keyboard or terminal display. Information passed to or from a terminal leaves no mass storage copy.

Data can be input directly through the terminal to an executing program by using the COBOL ACCEPT verb. This is illustrated later in the section.

COBOL does not rewind a connected data file whenever an OPEN or CLOSE statement is executed.

INTERACTIVE USAGE OF COBOL ACCEPT AND DISPLAY STATEMENTS

The COBOL DISPLAY and ACCEPT statements can be used interactively to communicate with the program during execution. Data can be accepted from or displayed on a terminal during execution. The SPECIAL-NAMES paragraph is used in the COBOL program to equate a user-defined mnemonic-name with a particular implementor-name, such as "TERMINAL". The advantages of using connected files to accept and display data follow:

- ACCEPT and DISPLAY use system resources more efficiently than READ and WRITE, and are convenient for low volume files.
- SELECT, ASSIGN, and File Description (FD) entries are not needed when using ACCEPT and DISPLAY. Data is described in the Working-Storage Section.
- Messages can be displayed directly from a program executing at the terminal; this capability is useful as a debugging tool.

A disadvantage of using connected files to accept and display data is that files are limited to block type C, record type Z, and sequential file organization.

The SPECIAL-NAMES paragraph can be used to equate a user-defined mnemonic-name with one of the following:

- "INPUT"
- "OUTPUT"
- "OUTPUT-C"
- "TERMINAL"
- "TERMINAL-C"
- Any file connected to a terminal

With the exception of "INPUT" and "OUTPUT", these special names cannot be used in a SELECT clause of a File-Control entry. The user-defined mnemonic-name is then used in ACCEPT and DISPLAY statements in the Procedure Division.

In all ACCEPT statements, FROM "INPUT" is the default if the FROM phrase is omitted. In all DISPLAY statements, UPON "OUTPUT" is the default when the UPON phrase is omitted.

ACCEPTING DATA FROM THE TERMINAL

The ACCEPT FROM statement can be used to accept data from a terminal as the program executes. The SPECIAL-NAMES paragraph is required.

```
SPECIAL-NAMES
    "TERMINAL" IS TERML.
.
.
.
ACCEPT KEY-IN FROM TERML.
```

These statements are used to associate terminal input with the special name "TERMINAL". TERML is the user-defined mnemonic-name to be equated with "TERMINAL".

- When the ACCEPT statement is encountered, execution pauses and a question mark is displayed at the terminal. The pause continues until the user enters data and presses a carriage RETURN key.
- KEY-IN specifies the data item to receive data from a terminal and should be defined with USAGE IS DISPLAY. See figure 16-4 for an example of the ACCEPT statement with the special name "TERMINAL".
- The input data must be terminated by a carriage return. If the receiving data item exceeds 300 characters, a prompt (question mark) is displayed for each multiple of 300. The data item named KEY-IN is blank filled if the input data is less than the size of the record. Input characters exceeding the size of KEY-IN are lost.

ACCEPTING DATA FROM A CONNECTED FILE

The ACCEPT FROM statement can be used to accept data from a terminal as the COBOL program executes. The SPECIAL-NAMES paragraph is required.

```
SPECIAL-NAMES
    "MY-FILE" IS MY-FILE.
.
.
.
ACCEPT MORE-KEYS FROM MY-FILE.
```

The ACCEPT statement specifies that data is to be accepted from the file named MY-FILE, which is connected to the terminal, during execution. MY-FILE must be explicitly connected under the NOS/BE system with the CONNECT command. MY-FILE must be assigned to TT under the NOS system with the ASSIGN command.

```
define,ifile                                  <-- Defines permanent file ifile, to be created in COBOL program.
/cobol5,i=fig164                              <-- Compiles COBOL program.


CDC COBOL 5.3 - LEVEL 518       SOURCE LISTING OF DATA-RE


 1               IDENTIFICATION DIVISION.
 2               PROGRAM-ID. DATA-READ.
 3               ENVIRONMENT DIVISION.
 4               CONFIGURATION SECTION.
 5               SOURCE-COMPUTER.  CYBER-170.
 6               OBJECT-COMPUTER.  CYBER-170.
 7               SPECIAL-NAMES.
 8                   "TERMINAL" IS TERML.
 9               INPUT-OUTPUT SECTION.
 10              FILE-CONTROL.
 11                  SELECT IFILE ASSIGN TO "IFILE".
 12              DATA DIVISION.
 13              FILE SECTION.
 14              FD IFILE
 15                  LABEL RECORD OMITTED
 16                      DATA RECORD IS I-REC.
 17              01 I-REC PIC X(18).
 18              WORKING-STORAGE SECTION.
 19              01 INREC.
 20                  02 FIELD1 PIC X(5).
 21                  02 FILLER PIC X(8).
 22                  02 FIELD2 PIC X(5).
 23              PROCEDURE DIVISION.
 24              STRT.
 25                  OPEN OUTPUT IFILE.
 26                  ACCEPT FIELD1.
 27                  DISPLAY "RECORD WRITTEN FOR ID  " FIELD1.
 28                  PERFORM UNTIL FIELD1 = ZEROS
 29                      IF FIELD1 = FIELD2
 30                        MOVE INREC TO I-REC
 31                        WRITE I-REC
 32                        DISPLAY "RECORD WRITTEN FOR ID " FIELD1
 33                      END-IF
 34                      ACCEPT INREC FROM TERML
 35                  END-PERFORM
 36                  CLOSE IFILE.
 37                  STOP RUN.

          COLUMN   1         2         3         4         5         6         7         8
          12345678901234567890123456789012345678901234567890123456789012345678901234567890
065100B CM,      .944 CPS, 000000B ECS
/lgo
? 12345          12345      <--+
RECORD WRITTEN FOR ID  12345   |
? 22222          22222      <--+
RECORD WRITTEN FOR ID 22222    |-- Accepts data into INREC field from the terminal.
? 66666          66676      <--+
? 89898          89898      <--+
RECORD WRITTEN FOR ID 89898
? 00000          <-- Causes program termination.
LGO.
/
```

Figure 16-4. Accepting Data From a Terminal

The file from which data is accepted is assumed to have Z record type and C block type. The field length is set to the size of the receiving data item, up to a maximum of 300 characters. The USE clause and the FILE control statement are not needed for setting these fields. However, an FL value less than 300 on a FILE control statement can override the value of 300.

Figure 16-5 illustrates program execution with data being accepted from a connected file, INVENT, which was created with the Full Screen Editor.

DISPLAYING DATA UPON THE TERMINAL

The DISPLAY UPON statement transfers the contents of a data item to the terminal during execution of a COBOL program. A literal alone, or a combination of literals and data items, can also be transferred. Each DISPLAY statement produces one line (record) of terminal output. Multiple items in a single DISPLAY statement appear in the order specified, within the allowable character limits. The SPECIAL-NAMES paragraph is required.

```
SPECIAL-NAMES
    "TERMINAL" IS TERML.
.
.
.
DISPLAY MY-RECORD UPON TERML.
```

These statements are used to associate terminal output with the special implementor-name "TERMINAL". TERML is a user-defined mnemonic-name to be equated with "TERMINAL". The DISPLAY statement causes MY-RECORD to be displayed at a terminal during program execution.

The maximum number of characters that can be displayed on a single line on the terminal is 72 for both NOS and NOS/BE. Under NOS/BE, however, the SCREEN command can override the default value.

```
DISPLAY MY-RECORD UPON TERM-C.
```

This statement can be used under NOS/BE to display information at a terminal, beginning with the second character of MY-RECORD. (Normally, all output is single spaced.) The first character of the record is used for carriage control. TERM-C is equated with "TERMINAL-C" in the SPECIAL-NAMES paragraph.

```
DISPLAY MY-RECORD UPON TERML WITH NO
    ADVANCING.
```

This statement causes the data in MY-RECORD to be displayed at the terminal with no carriage return. A subsequent ACCEPT statement then accepts data beginning with the position following the last displayed character, under NOS/BE, and following the prompt, under NOS. Under NOS/BE, two successive DISPLAY WITH NO ADVANCING statements result in overprinting of the first displayed data. Under NOS, two successive DISPLAY WITH NO ADVANCING statements cause the second displayed data to print immediately following the first displayed data.

DISPLAYING DATA UPON A CONNECTED FILE

The DISPLAY UPON statement transfers the contents of a data item to a connected file during execution of a COBOL program. The SPECIAL-NAMES paragraph is required.

```
SPECIAL-NAMES
    "MY-FILE" IS MY-FILE
.
.
.
DISPLAY MY-RECORD UPON MY-FILE.
```

This DISPLAY statement illustrates data being received by MY-FILE. My-record is defined in the Working-Storage Section. MY-FILE must be explicitly connected under the NOS/BE system with the CONNECT command. MY-FILE must be assigned to TT under the NOS system with the ASSIGN command.

Unless the FL parameter on a FILE control statement is set to a value, a maximum of 72 characters can be received by the file from MY-RECORD.

The file that receives data is assumed to have sequential format with C type blocking and Z type records.

A. INTERACTIVE SESSION

```
//get,enterd2                    Attaches permanent file named invent.
/get,invent                      Attaches permanent file (enterd2) containing COBOL
                                 source program.

/cobol5,i=enterd2                Compiles COBOL program, using file enterd2; results are
                                 displayed at terminal.


CDC COBOL 5.3 - LEVEL 507        SOURCE LISTING OF ENTER-D

    1                IDENTIFICATION DIVISION.
    2                PROGRAM-ID.  ENTER-DATA.
    3                ENVIRONMENT DIVISION.
    4                CONFIGURATION SECTION.
    5                SPECIAL-NAMES.
    6                "TERMINAL" IS TERML
    7                "INVENT" IS IFILE.
    8                INPUT-OUTPUT SECTION.
    9                FILE-CONTROL.
    10               DATA DIVISION.
```

Figure 16-5. Accepting Data from a Connected File (Sheet 1 of 2)

```
11              WORKING-STORAGE SECTION.
12              01 REC-NO PIC 9.
13              01 ANSWER PIC X.
14              01 SAVE-REC.
15                  02 FIELD1 PIC X(5).
16                  02 FILLER PIC X(8).
17                  02 FIELD2 PIC X(5).
18              PROCEDURE DIVISION.
19              FIRST-PAR.
20                  MOVE ZERO TO REC-NO.
21              SECOND-PAR.
22                  ACCEPT SAVE-REC FROM IFILE.
23                  ADD 1 TO REC-NO.
24                  DISPLAY "RECORD NUMBER " REC-NO "   " SAVE-REC.
25                  IF FIELD1 EQUALS 99999 GO TO TOTAL-PAR.
26                  GO TO SECOND-PAR.
27              TOTAL-PAR.
28                  DISPLAY "TOTAL RECORDS " REC-NO.
29              MORE.
30                  DISPLAY "DO YOU WANT TO ADD ANOTHER RECORD?".
31                  ACCEPT ANSWER.
32                  IF ANSWER EQUALS "N" GO TO CLOSING.
33                  DISPLAY "ENTER 5-DIGIT NUMBER" WITH NO ADVANCING.
34                  ACCEPT SAVE-REC.  ADD 1 TO REC-NO.
35                  DISPLAY "NEW TOTAL IS " REC-NO.
36                  GO TO MORE.
37              CLOSING.
38                  STOP RUN.

          COLUMN   1         2         3         4         5         6         7         8
          12345678901234567890123456789012345678901234567890123456789012345678901234567890
107100B CM,    1.599 CPS, 000000B ECS
/lgo                                                  <- Executes COBOL program; results are displayed at terminal.
RECORD NUMBER  1  12345        12345
RECORD NUMBER  2  22222        22222
RECORD NUMBER  3  63654        63654
RECORD NUMBER  4  77777        77777                  <- Executes DISPLAY Statements.
RECORD NUMBER  5  87878        87878
RECORD NUMBER  6  99999        99999
TOTAL RECORDS  6
DO YOU WANT TO ADD ANOTHER RECORD?
? y                                                   <- Executes ACCEPT ANSWER statement.
ENTER 5-DIGIT NUMBER? 65543                           <- Accepts data for SAVE-REC.
NEW TOTAL IS  7
DO YOU WANT TO ADD ANOTHER RECORD?
? y
ENTER 5-DIGIT NUMBER? 54890                           <- Accepts data for SAVE-REC.
NEW TOTAL IS  8
DO YOU WANT TO ADD ANOTHER RECORD?
? n                                                   <- Terminates execution.
LGO.
/
```

B. RECORDS IN INVENT FILE

```
12345        12345
22222        22222
63654        63654
77777        77777
87878        87878
99999        99999
```

Figure 16-5. Accepting Data from a Connected File (Sheet 2 of 2)

The Message Control System (MCS) provides a means of sending and receiving data between COBOL programs and communication devices such as terminals. MCS is the network software, available under the NOS operating system only, that performs the following functions:

- Controls the routing of messages from a COBOL program to a terminal.
- Controls the routing of messages from a terminal to a COBOL program.
- Performs message queuing and maintains queue status information.

Entering the COBOL/MCS communications environment presents some new terminology and processing concepts. These are described briefly before discussing specific COBOL Communications Facility (CCF) features.

Figure 17-1 illustrates the COBOL/MCS communications environment.

GENERAL CONCEPTS

MCS provides the COBOL user with a telecommunications message handling capability. To use MCS, application definitions must be established through an Application Definition Language (ADL) to define relationships between the MCS application components:

- Terminals
- COBOL programs
- Queues
- Special message files (called journals)

The creation of the MCS application definition, which is not normally performed by the COBOL programmer, is not discussed in this section. An example of an MCS application is described in appendix D. (It is recommended that the reader become familiar with this application, named FINANCE, before proceeding; it will be referenced in later examples in this section.)

To use an MCS application, the COBOL programmer must be aware of the following information:

- The name of the MCS application.
- The names of any destinations (terminals) to which the COBOL program is to send data, using the SEND verb.
- The names of any input queues from which the COBOL program is to receive data, using the RECEIVE verb.

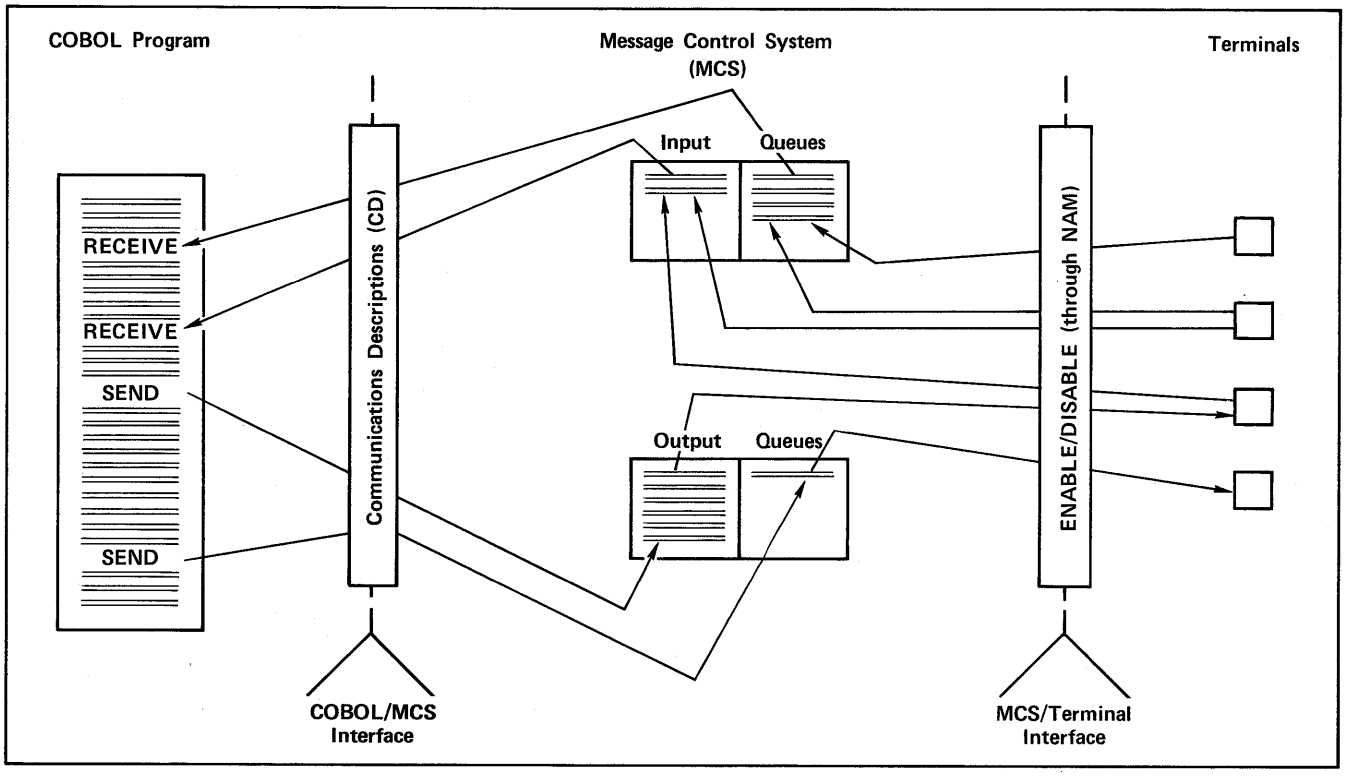


Figure 17-1. COBOL/MCS Communications Environment

- The hierarchal structure of the queues.
- The status and error codes that MCS returns to indicate an exception condition (such as, no data available or disk errors).

This information is usually provided by the individual responsible for defining the MCS application. Answers to the following questions can also be of use to the COBOL programmer using MCS:

- Is the COBOL program an on-line program or an off-line program?
- Is the COBOL program automatically initiated by MCS? If so, under what conditions?
- Is the COBOL program automatically terminated by MCS? If so, under what conditions? If not, should the COBOL program test for certain conditions to terminate processing?

MESSAGES AND MESSAGE QUEUES

Messages are the fundamental units of data transmitted and processed in a COBOL/MCS telecommunications environment. A message consists of an arbitrary amount of information (usually character data). Messages or message segments are transmitted through COBOL with the SEND and RECEIVE statements.

An input message queue contains messages that have been transmitted from terminals and are awaiting transmission to a COBOL program. An output message queue contains messages that have been transmitted from COBOL programs and are awaiting transmission to a terminal.

Message segments are delimited by end-of-segment indicators (ESI). Messages are logically separated from other messages by end-of-message indicators (EMI). Groups of messages are logically separated from other groups of messages by end-of-group indicators (EGI).

ENQUEUING AND DEQUEUING MESSAGES

The process by which complete messages are placed into a queue is called enqueuing. The process by which messages or segments are removed from a queue is called dequeuing. A selection algorithm can be specified by the user for certain messages to be placed in a given input queue, or for all messages for a given destination to be placed in a given output queue. Dequeuing can be performed on a first-in, first-out basis, or priorities can be established by the user.

QUEUE HIERARCHY

Four levels of input queues can be defined in the ADL: queue, sub-queue-1, sub-queue-2, and sub-queue-3. The four-level queue hierarchy allows MCS to route messages based on user-defined conditions.

A queue with no subqueues is a simple queue. A queue with subqueues is a compound queue. All incoming messages are tested at each level of the queue hierarchy, based on the specified conditions, and are then stored in the lowest level simple queue. Messages are stored only in simple queues. When a COBOL RECEIVE statement requests a message from a compound queue, MCS searches all levels of the named queue (from highest level to lowest level) until a nonempty simple queue is encountered; the next message or message segment in that queue is returned to the COBOL program.

ENABLING AND DISABLING QUEUES

An input queue must be enabled before MCS places messages in it. MCS can enable or disable queues based on time of day, message activity, or other factors. The COBOL programmer can also enable or disable queues with the ENABLE or DISABLE statements. Refer to the COBOL 5 reference manual for a description of these statements.

MESSAGE DESTINATION AND SOURCE

A message destination is the communication device to which the message is being sent. A message source is the communication device from which the message is being sent. Other COBOL programs can also be the source or destination of a message. Symbolic names for message sources and destinations are defined in the ADL description and referenced in the COBOL program.

DATA MODE AND COMMAND MODE

In data mode, MCS attempts to route all input from the terminal to the appropriate input queue where it becomes available to COBOL programs within the application. In command mode, the user can enter commands to control certain aspects of MCS operations.

COBOL COMMUNICATION FACILITY

COBOL programs perform message processing for a specific application through the following language features:

- A Communication Section in the Data Division that describes message queues and other MCS-related data.
- Message handling statements such as SEND and RECEIVE.

THE COMMUNICATION SECTION

The Communication Section is required in the Data Division of a COBOL program when MCS is used. The section must appear after the Linkage Section and before the Report Section (if applicable). Input Communications Descriptions (CD) entries define input queues; output CD entries define the destinations to receive messages. Figure 17-2 illustrates a Communication Section that can be used in a COBOL program for the FINANCE application defined in appendix D. The clauses in both the input CD and the output CD must be written in the order shown in the figure.

SENDING AND RECEIVING MESSAGES

Messages can be received from a terminal for use in a COBOL program by using the RECEIVE statement. After execution of the RECEIVE statement, the input CD area is updated. Messages can be sent from a COBOL program to a terminal by using the SEND statement. After execution of the SEND statement, the output CD area is updated.

```
COMMUNICATION SECTION.
CD   MSG-IN;      FOR INPUT
     SYMBOLIC QUEUE IS LOANQ-IN
     SYMBOLIC SUB-QUEUE-1 IS LOANQ-IN-1
     SYMBOLIC SUB-QUEUE-2 IS LOANQ-IN-2
     SYMBOLIC SUB-QUEUE-3 IS LOANQ-IN-3
     MESSAGE DATE IS TRAN-DATE
     MESSAGE TIME IS TRAN-TIME
     SYMBOLIC SOURCE IS IN-SOURCE
     TEXT LENGTH IS IN-LENGTH
     END KEY IS IN-KEY
     STATUS KEY IS STATUS-CODE
     MESSAGE COUNT IS IN-COUNT.
CD   MSG-OUT;      FOR OUTPUT
     DESTINATION COUNT IS OUT-DCOUNT
     TEXT LENGTH IS OUT-LENGTH
     STATUS KEY IS OUT-STATUS
     DESTINATION TABLE OCCURS 6 TIMES
         INDEXED BY D-TABLE
     ERROR KEY IS OUT-KEY
     SYMBOLIC DESTINATION IS OUT-DEST.
```

Figure 17-2. A COBOL Communication Section

After a RECEIVE statement is executed and MCS has removed a message from the input queue, the message is not available for further RECEIVES unless a journal file has been created.

Receiving Messages

In the FINANCE application, simple queues are used for each of the two savings transaction types. Figure 17-3 illustrates the queue structure for the queue named SAVEPMTQUE.

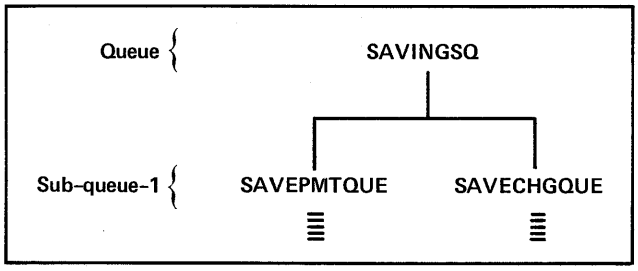


Figure 17-3. SAVINGSQ Structure

The RECEIVE statement is used in the Procedure Division of a COBOL program to acquire a message (or part of a message) from an input queue. The statements shown in figure 17-4 can be used to receive a message from the simple queue SAVEPMTQUE. The following should be noted:

- INBUF is defined in the Working-Storage Section and is used to hold messages from the queue named SAVEPMTQUE.
- If no messages exist in the queue, a program branch to PRINT-EMPTY is taken.
- Input fields for transaction type need not be tested (as would be necessary without MCS) because MCS has performed such testing in routing messages to the proper input queues.

```
WORKING-STORAGE SECTION.
01 INBUF.
     .
     .
     .
COMMUNICATION SECTION.
CD SV-MSG; FOR INPUT
SYMBOLIC QUEUE IS SAVINGSQ-IN
SYMBOLIC SUB-QUEUE-1 IS SAVQ-IN-1
     .
     .
     .
     MOVE "SAVINGSQ" TO SAVINGSQ-IN
     MOVE "SAVEPMTQUE" TO SAVSQ-IN-1
     RECEIVE SV-MSG MESSAGE INTO INBUF;
         NO DATA GO TO PRINT-EMPTY.
```

Figure 17-4. Receiving Messages from a 2-level Queue Structure

In the FINANCE application, a compound queue is used for accepting loan payment transactions. (The FASTLOANPMT queue is used before 5 o'clock; the SLOWLOANPMT queue is used after 5 o'clock.) Figure 17-5 illustrates the queue hierarchy of the compound queue LOANPMTQUE.

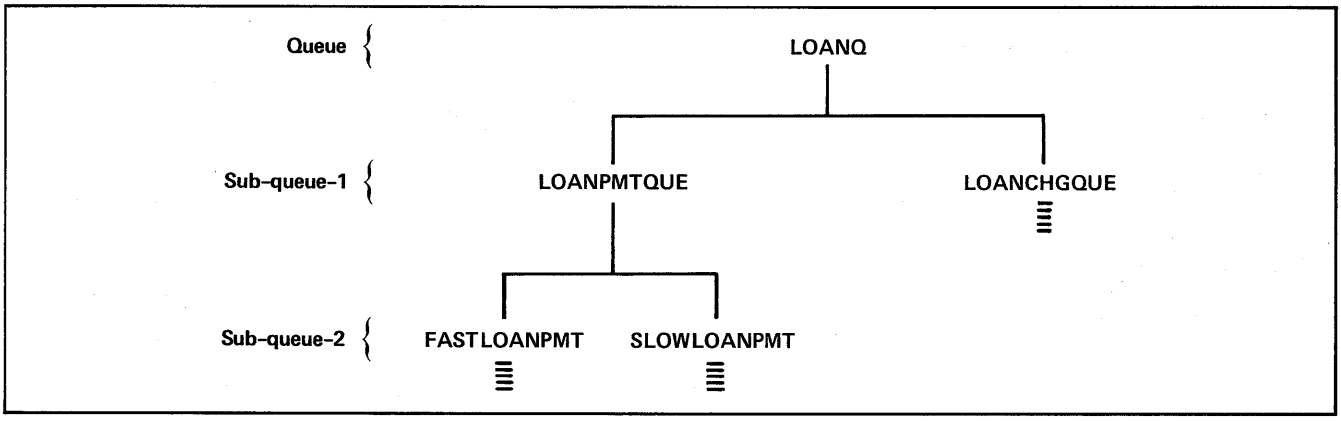


Figure 17-5. LOANQ Structure

The statements shown in figure 17-6 can be used to receive a message from the compound queue LOANPMTQUE. The following should be noted:

- The RECEIVE statement shown in figure 17-6 receives a message from either of two subqueues: FASTLOANPMT or SLOWLOANPMT.
- Before the RECEIVE statement is executed, the input CD area LN-MSG contains the symbolic queue name LOANPMTQUE that identifies the queue from which the next message is to be received.
- MCS returns the first message encountered by searching the FASTLOANPMT queue and then searching the SLOWLOANPMT queue.
- During the RECEIVE statement execution, MCS returns either the characters FASTLOANPMT or the characters SLOWLOANPMT to the data item LOANQ-IN-2; this indicates from which queue the message was received. If no data is available, MCS fills the CD area with blanks.

```
WORKING-STORAGE SECTION.
01 INBUF.
     02 MSG-ID PIC X(8).
     02 ACCT-NO PIC 9(11).
     02 TRAN-CODE PIC X(3).
     02 AMOUNT PIC 9(9).
     .
     .
     .
COMMUNICATION SECTION.
CD LN-MSG        FOR INPUT
SYMBOLIC QUEUE IS LOANSQ-IN
SYMBOLIC SUB-QUEUE-1 IS LOANQ-IN-1
SYMBOLIC SUB-QUEUE-2 IS LOANQ-IN-2
     .
     .
     .
     MOVE "LOANQ" TO LOANSQ-IN.
     MOVE "LOANPMTQUE" TO LOANQ-IN-1.
     RECEIVE LN-MSG MESSAGE INTO INBUF
         NO DATA GO TO PRINT-EMPTY.
```

Figure 17-6. Receiving Messages from a 3-level Queue Structure

Alternatively, if the following COBOL statements are used:

```
COMMUNICATION SECTION
CD LN-MSG; FOR INPUT
SYMBOLIC QUEUE IS LOANSQ-IN
     .
     .
     .
MOVE "LOANQ" TO LOANSQ-IN
RECEIVE LN-MSG MESSAGE INTO INBUF;
    NO DATA GO TO PRINT-EMPTY.
```

MCS returns the first message encountered by searching queues in the following order:

- FASTLOANPMT
- SLOWLOANPMT
- LOANCHGQUE

In the preceding COBOL statements, no queue name is moved into the data item LOANQ-IN-1. During the RECEIVE statement execution, MCS returns either the characters LOANPMTQUE or the characters LOANCHGQUE to the data item LOANQ-IN-1. If LOANPMTQUE is returned, either the characters FASTLOANPMT or the characters SLOWLOANPMT are returned to the data item LOANQ-IN-2.

Sending Messages

The SEND statement is used in the Procedure Division of a COBOL program to release a message (or part of a message) to MCS for enqueuing in an output queue. MCS determines when to transmit the message to a specific destination. The statements shown in figure 17-7 can be used to send a message from a COBOL program to a terminal. The following should be noted:

- OUTBUF is defined in the Working-Storage Section and is used to hold messages, before the messages are sent to terminals.
- A record is read from the loan account file LMASTER. Appropriate code follows to update account records according to the payment transaction code in the message previously received. This code is not shown in figure 17-7.
- The CLEAR-TABLE routine is executed to initialize the items in the output CD area.
- Before execution of the SEND statement, values are moved into the OUTBUF area and into the output CD area.
- The STATUS-CHECK-OUT routine is executed, after the SEND statement execution, to verify successful execution or to perform appropriate error recovery.

Updating the CD Areas

Whenever a RECEIVE statement is executed, updated values are returned to the input CD area. For example, referencing the CD in figure 17-2, the following occur when a RECEIVE statement is executed:

- MCS updates the contents of TRAN-DATE with the year, month, and day on which MCS received the complete message.
- MCS updates the contents of TRAN-TIME with the hours, minutes, and seconds that MCS received the complete message.
- MCS provides the symbolic name of the terminal that is the source of the message in the data item IN-SOURCE.
- MCS indicates the number of message character positions filled in the data item IN-LENGTH.
- MCS indicates a status condition that exists after the RECEIVE statement is executed by providing a code in the data item STATUS-CODE (code 00 indicates normal execution).

```
WORKING-STORAGE SECTION.
01 OUTBUF.
    02 MSGOUT PIC X(32).
    02 AMT-OUT PIC 9(9).
    02 AC PIC X(8) VALUE IS "   ACCT ".
    02 ACCT-OUT PIC 9(11).
     .
     .
     .
COMMUNICATION SECTION.
CD MSG-OUT;       FOR OUTPUT
   DESTINATION COUNT IS OUT-DCOUNT
   TEXT LENGTH IS OUT-LENGTH
   STATUS KEY IS OUT-STATUS
   DESTINATION TABLE OCCURS 6 TIMES
       INDEXED BY D-TABLE
   ERROR KEY IS OUT-KEY
   SYMBOLIC DESTINATION IS OUT-DEST.
    .
    .
    .
PROCEDURE DIVISION.
     .
     .
     .
     READ LMASTER INVALID KEY GO TO NO-ACCT.
     .
     .
     .
     PERFORM CLEAR-TABLE.
     MOVE "TRANSACTION COMPLETE NEW BAL IS " TO MSGOUT.
     MOVE ACCT-NUM TO ACCT-OUT.
     MOVE LBAL TO AMT-OUT.
     MOVE "LOAN200" TO OUT-DEST (1).
     MOVE 1 TO OUT-DCOUNT.
     MOVE 60 TO OUT-LENGTH.
     SEND MSG-OUT FROM OUTBUF WITH EMI.
     PERFORM STATUS-CHECK-OUT.
     .
     .
     .
```

Definition of area that holds output message before it goes to output queue.

Output CD area definition.

Reads a record from a data base file.

(Missing code updates data base file, etc.)

Moves values to OUTBUF area.

Tells MCS the destination to which the message is to be sent.

Tells MCS how many destinations are referenced.

Tells MCS the number of characters in message to be sent.

Sends message from COBOL program to output queue and updates output CD area.

Verifies status of previous operation.

Figure 17-7. Sending Messages from a COBOL Program

Whenever a SEND statement is executed, updated values are returned to the output CD area. For example, referencing the CD in figure 17-2, the following occurs when a SEND statement is executed:

- MCS indicates a status condition that exists after the SEND statement is executed by providing a code in the data item OUT-STATUS (code 00 indicates normal execution).
- MCS indicates more detailed status by providing a code in the data item OUT-KEY (code 0 indicates normal execution).

ACCESSING THE STATUS KEY

The STATUS KEY clause in the input CD is important to the COBOL user in error processing.

```
CD INPUT-AREA; FOR INPUT
STATUS KEY IS IN-CODE
     .
     .
     .
RECEIVE INPUT-AREA MESSAGE INTO INBUF.
IF IN-CODE NOT EQUAL "00"
    GO TO ERROR-ROUTINE.
```

When the preceding RECEIVE statement is executed, MCS returns a status code to IN-CODE. If a message has not been successfully received (indicated by code 00), a program branch is taken to ERROR-ROUTINE. The codes indicate conditions such as system errors, invalid password, disabled terminal, and invalid application name. Status codes are listed in table 17-1.

Similarly, the STATUS KEY clause in the output CD can be used for error processing.

```
CD OUTPUT-AREA; FOR OUTPUT
STATUS KEY IS OUT-CODE
     .
     .
     .
SEND OUTPUT-AREA FROM OUTBUF WITH EMI.
IF OUT-CODE NOT EQUAL "00"
    GO TO DEBUG-ROUTINE.
```

When the preceding SEND statement is executed, MCS returns a status code to OUT-CODE. If a message has not been successfully accepted (indicated by code 00), a program branch is taken to DEBUG-ROUTINE. Status codes are listed in table 17-1. Error key codes are listed in table 17-2.

TABLE 17-1. MCS STATUS KEY CODES

Code	Description
00	The CCF statement executed successfully.
10	The CCF statement executed successfully for all destinations referenced; however, one or more destinations are currently disabled.
15	One or more of the queue source paths were already enabled or disabled. Disable or enable action applied to other paths.
20	One or more symbolic queue names or destination names are invalid or unknown. Action taken only for valid destinations.
21	Symbolic source name invalid or unknown. No action taken while executing the CCF statement.
30	DESTINATION COUNT invalid or exceeds the maximum value possible. No action was taken while executing the CCF statement.
40	Password invalid. No action was taken while executing the CCF statement, except that DISABLE OUTPUT or ENABLE OUTPUT are applied to valid destinations.
50	TEXT LENGTH invalid or exceeds the maximum value possible. No action was taken while executing the SEND statement.
60	A portion of a segment or message is to be sent, but no message area is referenced and/or TEXT LENGTH is set to zero. No action was taken while executing the SEND statement.
70	One or more destinations do not have partial messages associated with them. PURGE statement successfully completed for other destinations.
80	Two or more of the conditions designated by the STATUS KEY codes 10, 15, 20, 40, 70, and 95 have occurred. Action successfully completed for those destinations for which no exception condition exists.
90	System error. The results of the CCF statement are unpredictable and the program should terminate processing.
91	MCS is not running or MCS is not defined as a system control point. No action was taken while executing the CCF statement.
92	No valid application name parameter was specified on the program call statement or the specified application is not running. No action was taken while executing the CCF statement. Code occurs only if the program was not initiated by MCS.
93	The program name is unknown or a duplicate program name has been established with MCS. No action was taken while executing the CCF statement.
94	MCS or the application is shutting down. No action was taken while executing the CCF statement.
95	Output queue threshold for one or more destinations exceeded. SEND statement successfully completed for other destinations.
96	Sufficient resources (such as central memory) are not available to satisfy the request. No action was taken while executing the CCF statement. The program may repeat the request.
97	MCS encountered a CIO error when accessing a mass storage queue. The results of the request are unpredictable and the program should terminate processing. Further attempts to access the queue by any part of the application results in the application being closed down by MCS.

EXECUTION OF COBOL PROGRAMS USING MCS

A COBOL program is executed in the MCS environment, under the NOS operating system, by using the *APPL parameter on the execution call statement. For example, all COBOL programs within the FINANCE application can use the following statement for program execution:

```
LGO,*APPL=FINANCE.
```

In an MCS environment, a COBOL program can be submitted as a batch job or can be stored on an invocation file and submitted with the MCS INVOKE command. Alternatively, a COBOL program can be automatically invoked upon specific conditions (defined in the ADL). Refer to the MCS reference manual for further discussion of invocation files and job submission.

AN INTERACTIVE SESSION

Figure 17-8 illustrates a terminal user session for the FINANCE application. The terminal operator enters three loan transactions that are processed by the COBOL program LNMONEY. The COBOL program updates loan account balances on the data base file LMASTER, prints a message to the operator that the transaction is complete, and indicates the new account balance. An invalid account number is then entered, and the COBOL program responds with an appropriate message.

TABLE 17-2. MCS ERROR KEY CODES

Code	Description
0	No exception condition for this destination.
1	Symbolic destination name invalid or unknown.
2	Destination disabled.
3	Password invalid for this destination.
4	No partial message associated with this destination.
5	Destination already disabled.
6	Output queue threshold for this destination exceeded.

```
MCS 1.0 80/01/18. 18.11.59.
MCS APPLICATION ?FINANCE,LNAOP                    <- Operator identifies the MCS application name and password.
SYMBOLIC-NAME ?LOAN200                            <- Operator identifies the terminal being used.
DATA MODE
HELLO - YOU'RE THE AOP
?PAYMENT,12345678544PRD0000500                    <- Operator enters a $5.00 disbursement to loan account 123-4567854-4.
SERIAL NUMBER =          1
TRANSACTION COMPLETE NEW BAL IS    6231455    ACCT 12345678544   <- Response from COBOL program.

?PAYMENT,14452221245PRR59500                      <- Operator enters a $595.00 receipt to loan account 144-5222124-5.
SERIAL NUMBER =          2
TRANSACTION COMPLETE NEW BAL IS    3785415    ACCT 14452221245   <- Response from COBOL program.

?PAYMENT,34555541247PRR69300                      <- Operator enters a $693.00 receipt to loan account 345-5554124-7.
SERIAL NUMBER =          3
TRANSACTION COMPLETE NEW BAL IS    2122544    ACCT 34555541247   <- Response from COBOL program.

?PAYMENT,00004236074PRD55555                      <- Operator enters an invalid account number.
SERIAL NUMBER =          4
INVALID ACCOUNT NO                 000000000    ACCT 00004236074   <- Response from COBOL program.
                                                                   <- Break-2 character to switch to command mode.
COMMAND MODE
?BYE                                              <- Operator terminates session.
MCS ENDED 80/01/18.    18.14.50.MCS      CONNECT TIME 00.03.05.
```

Figure 17-8. COBOL/MCS Interactive Terminal User Session

STANDARD CHARACTER SET A

CONTROL DATA operating systems offer the following variations of a basic character set:

CDC 64-character set

CDC 63-character set

ASCII 64-character set

ASCII 63-character set

The installation can specify the 63-character set option in either CDC or ASCII mode. In either case, the colon becomes 63_8 and collates properly. In the ALPHABET clause, the mnemonic-name CDC-64 and ASCII-64 refer to CDC-63 and ASCII-63 character sets, respectively.

The set in use at a particular installation was specified when the operating system was installed.

Depending on another installation option, the system assumes an input deck has been punched either in 026 or in 029 mode (regardless of the character set in use). Under NOS/BE 1, the alternate mode can be specified by a 26 or 29 punched in columns 79 and 80 of the job statement or any 7/8/9 card. The specified mode remains in effect through the end of the job unless it is reset by specification of the alternate mode on a subsequent 7/8/9 card.

Under NOS 2, the alternate mode can be specified by a 26 or 29 punched in columns 79 and 80 of any 6/7/9 card, as described above for a 7/8/9 card. In addition, 026 mode can be specified by a card with 5/7/9 multipunched in column 1, and 029 mode can be specified by a card with 5/7/9 multipunched in column 1 and a 9 punched in column 2.

Graphic character representation appearing at a terminal or printer depends on the installation character set and the terminal type. Characters shown in the CDC Graphic column of the standard character set table are applicable to BCD terminals; ASCII graphic characters are applicable to ASCII-CRT and ASCII-TTY terminals.

Character sets are shown in the following tables:

- Table A-1. COBOL and Standard Character Sets
- Table A-2. CDC Character Set Collating Sequence
- Table A-3. ASCII Character Set Collating Sequence
- Table A-4. EBCDIC 64-Character Subset Collating Sequence
- Table A-5. UNIVAC 1100 Series Collating Sequence (UNI)

TABLE A-1. COBOL AND STANDARD CHARACTER SETS

COBOL	Display Code (octal)	CDC			ASCII		
		Graphic	Hollerith Punch (026)	External BCD Code	Graphic Subset	Punch (029)	Code (octal)
	00†	: (colon)††	8-2	00	: (colon) ††	8-2	072
A	01	A	12-1	61	A	12-1	101
B	02	B	12-2	62	B	12-2	102
C	03	C	12-3	63	C	12-3	103
D	04	D	12-4	64	D	12-4	104
E	05	E	12-5	65	E	12-5	105
F	06	F	12-6	66	F	12-6	106
G	07	G	12-7	67	G	12-7	107
H	10	H	12-8	70	H	12-8	110
I	11	I	12-9	71	I	12-9	111
J	12	J	11-1	41	J	11-1	112
K	13	K	11-2	42	K	11-2	113
L	14	L	11-3	43	L	11-3	114
M	15	M	11-4	44	M	11-4	115
N	16	N	11-5	45	N	11-5	116
O	17	O	11-6	46	O	11-6	117
P	20	P	11-7	47	P	11-7	120
Q	21	Q	11-8	50	Q	11-8	121
R	22	R	11-9	51	R	11-9	122
S	23	S	0-2	22	S	0-2	123
T	24	T	0-3	23	T	0-3	124
U	25	U	0-4	24	U	0-4	125
V	26	V	0-5	25	V	0-5	126
W	27	W	0-6	26	W	0-6	127
X	30	X	0-7	27	X	0-7	130
Y	31	Y	0-8	30	Y	0-8	131
Z	32	Z	0-9	31	Z	0-9	132
0	33	0	0	12	0	0	060
1	34	1	1	01	1	1	061
2	35	2	2	02	2	2	062
3	36	3	3	03	3	3	063
4	37	4	4	04	4	4	064
5	40	5	5	05	5	5	065
6	41	6	6	06	6	6	066
7	42	7	7	07	7	7	067
8	43	8	8	10	8	8	070
9	44	9	9	11	9	9	071
+	45	+	12	60	+	12-8-6	053
-	46	-	11	40	-	11	055
*	47	*	11-8-4	54	*	11-8-4	052
/	50	/	0-1	21	/	0-1	057
(	51	(	0-8-4	34	(	12-8-5	050
)	52	)	12-8-4	74	)	11-8-5	051
$	53	$	11-8-3	53	$	11-8-3	044
=	54	=	8-3	13	=	8-6	075
blank	55	blank	no punch	20	blank	no punch	040
, (comma)	56	, (comma)	0-8-3	33	, (comma)	0-8-3	054
. (period)	57	. (period)	12-8-3	73	. (period)	12-8-3	056
" (quote)	60	≡	0-8-6	36	#	8-3	043
	61	[	8-7	17	[	12-8-2	133
	62	]	0-8-2	32	]	11-8-2	135
	63	% ††	8-6	16	% ††	0-8-4	045
	64	≠	8-4	14	" (quote)	8-7	042
	65	←	0-8-5	35	_ (underline)	0-8-5	137
	66	∨	11-0	52	!	12-8-7	041
	67	∧	0-8-7	37	&	12	046
	70	↑	11-8-5	55	' (apostrophe)	8-5	047
	71	↓	11-8-6	56	?	0-8-7	077
<	72	<	12-0	72	<	12-8-4	074
>	73	>	11-8-7	57	>	0-8-6	076
	74	≤	8-5	15	@	8-4	100
	75	≥	12-8-5	75	\	0-8-2	134
	76	¬	12-8-6	76	~ (circumflex)	11-8-7	136
; (semicolon)	77	; (semicolon)	12-8-7	77	; (semicolon)	11-8-6	073

†Twelve zero bits at the end of a 60-bit word in a zero byte record are an end of record mark rather than two colons.

††In installations using a 63-graphic set, display code 00 has no associated graphic or card code; display code 63 is the colon (8-2 punch). The % graphic and related card codes do not exist and translations yield a blank (55_8).

TABLE A-2. CDC CHARACTER SET COLLATING SEQUENCE

Collating Sequence Decimal	Collating Sequence Octal	CDC Graphic	Display Code	External BCD	Collating Sequence Decimal	Collating Sequence Octal	CDC Graphic	Display Code	External BCD
00	00	blank	55	20	32	40	H	10	70
01	01	≤	74	15	33	41	I	11	71
02	02	%	63 †	16 †	34	42	∨	66	52
03	03	[	61	17	35	43	J	12	41
04	04	→	65	35	36	44	K	13	42
05	05	≡	60	36	37	45	L	14	43
06	06	∧	67	37	38	46	M	15	44
07	07	↑	70	55	39	47	N	16	45
08	10	↓	71	56	40	50	O	17	46
09	11	>	73	57	41	51	P	20	47
10	12	≥	75	75	42	52	Q	21	50
11	13	¬	76	76	43	53	R	22	51
12	14	.	57	73	44	54	]	62	32
13	15	)	52	74	45	55	S	23	22
14	16	;	77	77	46	56	T	24	23
15	17	+	45	60	47	57	U	25	24
16	20	$	53	53	48	60	V	26	25
17	21	*	47	54	49	61	W	27	26
18	22	-	46	40	50	62	X	30	27
19	23	/	50	21	51	63	Y	31	30
20	24	,	56	33	52	64	Z	32	31
21	25	(	51	34	53	65	:	00 †	none†
22	26	=	54	13	54	66	0	33	12
23	27	≠	64	14	55	67	1	34	01
24	30	<	72	72	56	70	2	35	02
25	31	A	01	61	57	71	3	36	03
26	32	B	02	62	58	72	4	37	04
27	33	C	03	63	59	73	5	40	05
28	34	D	04	64	60	74	6	41	06
29	35	E	05	65	61	75	7	42	07
30	36	F	06	66	62	76	8	43	10
31	37	G	07	67	63	77	9	44	11

†In installations using the 63-graphic set, the % graphic does not exist. The : graphic is display code 63, External BCD code 16.

TABLE A-3. ASCII CHARACTER SET COLLATING SEQUENCE

Collating Sequence Decimal	Collating Sequence Octal	ASCII Graphic Subset	Display Code	ASCII Code	Collating Sequence Decimal	Collating Sequence Octal	ASCII Graphic Subset	Display Code	ASCII Code
00	00	blank	55	20	32	40	@	74	40
01	01	!	66	21	33	41	A	01	41
02	02	"	64	22	34	42	B	02	42
03	03	#	60	23	35	43	C	03	43
04	04	$	53	24	36	44	D	04	44
05	05	%	63†	25	37	45	E	05	45
06	06	&	67	26	38	46	F	06	46
07	07	'	70	27	39	47	G	07	47
08	10	(	51	28	40	50	H	10	48
09	11	)	52	29	41	51	I	11	49
10	12	*	47	2A	42	52	J	12	4A
11	13	+	45	2B	43	53	K	13	4B
12	14	,	56	2C	44	54	L	14	4C
13	15	-	46	2D	45	55	M	15	4D
14	16	.	57	2E	46	56	N	16	4E
15	17	/	50	2F	47	57	O	17	4F
16	20	0	33	30	48	60	P	20	50
17	21	1	34	31	49	61	Q	21	51
18	22	2	35	32	50	62	R	22	52
19	23	3	36	33	51	63	S	23	53
20	24	4	37	34	52	64	T	24	54
21	25	5	40	35	53	65	U	25	55
22	26	6	41	36	54	66	V	26	56
23	27	7	42	37	55	67	W	27	57
24	30	8	43	38	56	70	X	30	58
25	31	9	44	39	57	71	Y	31	59
26	32	:	00†	3A	58	72	Z	32	5A
27	33	;	77	3B	59	73	[	61	5B
28	34	<	72	3C	60	74	\	75	5C
29	35	=	54	3D	61	75	]	62	5D
30	36	>	73	3E	62	76	^	76	5E
31	37	?	71	3F	63	77	_	65	5F

†In installations using a 63-graphic set, the % graphic does not exist. The : graphic is display code 63.

TABLE A-4. EBCDIC 64-CHARACTER SUBSET COLLATING SEQUENCE

Collating Sequence Decimal	Collating Sequence Octal	Graphic	EBCDIC Punch	Display Code	EBCDIC Code
00	00	blank	no punch	55	40
01	01	.	12-8-3	57	4B
02	02	<	12-8-4	72	4C
03	03	(	12-8-5	51	4D
04	04	+	12-8-6	45	4E
05	05	|	12-8-7	66	4F
06	06	&	12	67	50
07	07	$	11-8-3	53	5B
08	10	*	11-8-4	47	5C
09	11	)	11-8-5	52	5D
10	12	;	11-8-6	77	5E
11	13	¬	11-8-7	76	5F
12	14	-	11	46	60
13	15	/	0-1	50	61
14	16	,	0-8-3	56	6B
15	17	%	0-8-4	63	6C
16	20	_	0-8-5	65	6D
17	21	>	0-8-6	73	6E
18	22	?	0-8-7	71	6F
19	23	:	8-2	00	7A
20	24	#	8-3	60	7B
21	25	@	8-4	74	7C
22	26	'	8-5	70	7D
23	27	=	8-6	54	7E
24	30	"	8-7	64	7F
25	31	¢	12-8-2/12-0	61	4A
26	32	A	12-1	01	C1
27	33	B	12-2	02	C2
28	34	C	12-3	03	C3
29	35	D	12-4	04	C4
30	36	E	12-5	05	C5
31	37	F	12-6	06	C6

TABLE A-4. EBCDIC 64-CHARACTER SUBSET COLLATING SEQUENCE (Contd)

Collating Sequence Decimal	Collating Sequence Octal	Graphic	EBCDIC Punch	Display Code	EBCDIC Code
32	40	G	12-7	07	C7
33	41	H	12-8	10	C8
34	42	I	12-9	11	C9
35	43	!	11-8-2/11-0	62	5A
36	44	J	11-1	12	D1
37	45	K	11-2	13	D2
38	46	L	11-3	14	D3
39	47	M	11-4	15	D4
40	50	N	11-5	16	D5
41	51	O	11-6	17	D6
42	52	P	11-7	20	D7
43	53	Q	11-8	21	D8
44	54	R	11-9	22	D9
45	55	none	0-8-2	75	E0
46	56	S	0-2	23	E2
47	57	T	0-3	24	E3
48	60	U	0-4	25	E4
49	61	V	0-5	26	E5
50	62	W	0-6	27	E6
51	63	X	0-7	30	E7
52	64	Y	0-8	31	E8
53	65	Z	0-9	32	E9
54	66	0	0	33	F0
55	67	1	1	34	F1
56	70	2	2	35	F2
57	71	3	3	36	F3
58	72	4	4	37	F4
59	73	5	5	40	F5
60	74	6	6	41	F6
61	75	7	7	42	F7
62	76	8	8	43	F8
63	77	9	9	44	F9

TABLE A-5. UNIVAC 1100 SERIES COLLATING SEQUENCE (UNI)

Collating Sequence Decimal/Octal		1108 Graphic	Card Punch	Display Code	CYBER Graphic
00	00	@	8-7	61	[
01	01	[	12-8-5	75	≥
02	02	]	11-8-5	70	↑
03	03	∨	12-8-7	77	;
04	04	Δ	11-8-7	73	>
05	05	blank	no punch	55	blank
06	06	A	12-1	01	A
07	07	B	12-1	02	B
08	10	C	12-3	03	C
09	11	D	12-4	04	D
10	12	E	12-5	05	E
11	13	F	12-6	06	F
12	14	G	12-7	07	G
13	15	H	12-8	10	H
14	16	I	12-9	11	I
15	17	J	11-1	12	J
16	20	K	11-2	13	K
17	21	L	11-3	14	L
18	22	M	11-4	15	M
19	23	N	11-5	16	N
20	24	O	11-6	17	O
21	25	P	11-7	20	P
22	26	Q	11-8	21	Q
23	27	R	11-9	22	R
24	30	S	0-2	23	S
25	31	T	0-3	24	T
26	32	U	0-4	25	U
27	33	V	0-5	26	V
28	34	W	0-6	27	W
29	35	X	0-7	30	X
30	36	Y	0-8	31	Y
31	37	Z	0-9	32	Z

TABLE A-5. UNIVAC 1100 SERIES COLLATING SEQUENCE (UNI) (Contd)

Collating Sequence Decimal/Octal	1108 Graphic	Card Punch	Display Code	CYBER Graphic
32 40	)	12-8-4	52	)
33 41	–	11	46	–
34 42	+	12	45	+
35 43	<	12-8-6	76	¬
36 44	=	8-3	54	=
37 45	>	8-6	63	%
38 46	&	8-2	00	:
39 47	$	11-8-3	53	$
40 50	*	11-8-4	47	*
41 51	(	0-8-4	51	(
42 52	%	0-8-5	65	→
43 53	:	8-5	74	⩽
44 54	?	12-0	72	<
45 55	!	11-0	66	∨
46 56	,	0-8-3	56	,
47 58	\	0-8-6	60	≡
48 60	0	0	33	0
49 61	1	1	34	1
50 62	2	2	35	2
51 63	3	3	36	3
52 64	4	4	37	4
53 65	5	5	40	5
54 66	6	6	41	6
55 67	7	7	42	7
56 70	8	8	43	8
57 71	9	9	44	9
58 72	'	8-4	64	≠
59 73	;	11-8-6	71	↓
60 74	/	0-1	50	/
61 75	.	12-8-3	57	.
62 76	□	0-8-7	67	∧
63 77	≠	0-8-2	62	]

GLOSSARY B

Access Control -
Protection of data from unauthorized access or modification. Access control is provided by the CDCS privacy module.

Access Control Key -
The value an applications program must supply to CDCS in order to gain access to a particular data base area.

Access Mode -
Manner in which records can be inserted into or retrieved from a file. Can be sequential, random, or dynamic, depending on the ACCESS MODE clause. Access mode, open mode, and file organization affect subsequent operations permitted.

Actual-Key File -
File described by ORGANIZATION IS ACTUAL-KEY clause. For initial actual-key files, the primary key specifies the block and record slot number in which the record is stored. For extended actual-key files, the primary key is a record number that AAM converts to the storage location of the record. Program must perform any key indexing.

Advanced Access Methods (AAM) -
File manager that processes indexed sequential, direct access, and actual-key file organizations and supports the Multiple-Index Processor. See CYBER Record Manager.

Alphabetic Character -
Character belonging to set of letters A through Z, and the space.

Alphanumeric Character -
Any character in a computer character set defined in appendix A. Most formats allow only characters from the COBOL character set.

Alternate Record Key -
Record key defined by ALTERNATE RECORD KEY clause. Can be used to read, but not write or update, a record in an indexed, direct, or actual-key file. System creates an index that relates alternate record keys to primary record keys using the file defined by the second implementor-name of an ASSIGN clause. Job must preserve index file.

ANSI Standard Language -
Language defined in American National Standard X3.23-1974, COBOL, on which this COBOL 5 compiler is based. Control Data extensions to the language are not shaded in this guide.

Application -
In the context of MCS, the COBOL programs, sources, destinations, queues, and journals that are defined as an application through the Application Definition Language (ADL). Typical examples are order entry, inventory control, and accounting.

Application Definition Language (ADL) -
A language that defines and describes application components to MCS.

Area -
Uniquely named data base subdivision that contains data records; a file.

Arithmetic Expression -
Any combination of numeric elementary items, numeric literals, and the figurative constant ZERO connected by arithmetic operators to form an expression that reduces to a single value when it is evaluated during program execution.

Arithmetic Operator -
+ to indicate addition; - to indicate subtraction; * to indicate multiplication; / to indicate division; ** to indicate exponentiation.

Assumed Decimal Point -
Decimal point position that does not involve the existence of an actual character in a data item. Has logical meaning but no physical representation.

At End Condition -
Condition that exists when no further records or data are available for processing during execution of: READ statement, RETURN statement, SEARCH statement when no conditions are satisfied. Sets FILE STATUS data item.

Basic Access Methods (BAM) -
File manager that processes sequential and word addressable file organizations. See CYBER Record Manager.

Beginning-of-Information (BOI) -
CYBER Record Manager defines beginning-of-information as the start of the first user record in a file. System-supplied information, such as an index block or control word, does not affect beginning-of-information. Any label on a tape exists prior to beginning-of-information.

Blocks -
The term block has several meanings depending on context. On tape, a block is information between interrecord gaps. CYBER Record Manager defines blocks according to organization. See table B-1.

TABLE B-1. BLOCK MEANINGS

Organization	Blocks
Indexed sequential	Data Block; index block
Direct access	Home block; overflow block
Actual Key	Data block
Sequential	Block type I, C, K, E

Body Group -
In Report Writer, generic name for a report group defined by a TYPE clause with a DETAIL, CONTROL HEADING, or CONTROL FOOTING phrase.

Boolean Data Item -
A data item consisting entirely of the boolean characters zero and one.

Boolean Expression -
An identifier of a boolean data item, a boolean literal, such identifiers and/or literals separated by a boolean operator, two boolean expressions separated by a boolean operator, or a boolean expression enclosed in parentheses.

Boolean Literal -
A literal composed of one or more boolean characters delimited on the left by the separator B" and on the right by the quotation mark separator.

Break Control Item -
In Report Writer, item defined by CONTROL clause. Synonymous with control data item defined by ANSI standard.

Called Program -
Program that is the object of a CALL statement.

Calling Program -
Program that executes a CALL statement.

Capsule -
A relocatable collection of one or more programs bound together in a special format that allows the programs to be loaded and unloaded dynamically from an executing program by the Fast Dynamic Loader facility.

Character-String -
Sequence of contiguous characters that form a COBOL word, a literal, a PICTURE clause character-string, or a comment-entry.

Clause -
Ordered set of COBOL character-strings that make up an entry.

COBOL Character Set -
51-character set defined by ANSI standard. Contrast with computer character set.

COBOL Communication Facility (CCF) -
The part of the COBOL language that allows a user to send and receive messages from terminals. CCF allows the COBOL user to interface with the Message Control System (MCS). This interface is allowed only under the NOS operating system.

Collating Sequence -
Sequence in which the characters that are acceptable to a computer are ordered for purposes of sorting, merging, and comparing. Defined by default, COLLATING SEQUENCE clause, ALPHABET clause, or SET statement.

Comment-Entry -
Entry in Identification Division that can contain any combination of characters from computer character set.

Comment Line -
Source program line with asterisk or slash in column 7 and any characters from computer character set in area A and area B of that line. Documentary only. / in column 7 ejects page before comment is listed in source listing.

Compilation Time -
Time at which a COBOL source program is translated by the COBOL 5 compiler to an object program that can be loaded and executed. Defined by control statement in batch job or interactive command. Contrast with execution time.

Computer Character Set -
Character set listed in appendix A.

Control Break -
In Report Writer, change in value of data item defined by CONTROL clause that is used to control hierarchical structure of a report. Control totals are accumulated and formatted for presentation to the report file when a control break occurs.

Control Data Item -
In Report Writer, a synonym for break control item.

Control Word -
A system-supplied word that precedes each W type record in storage.

Currency Symbol -
Character in program character set equivalent to display code value 53; or character defined by CURRENCY SIGN clause.

CYBER Record Manager (CRM) -
A generic term relating to the common products BAM and AAM, which run under the NOS and NOS/BE operating systems, that allows a variety of record types, blocking types, and file organizations to be created and accessed. The execution time input/output of COBOL, Sort/Merge 5, ALGOL, and the DMS 170 products is implemented through CYBER Record Manager. Neither the input/output of the NOS and NOS/BE operating systems themselves nor any of the system utilities such as COPY or SKIPF is implemented through CYBER Record Manager. All CYBER Record Manager file processing requests ultimately pass through the operating system input/output routines.

Data Base -
Collection of information defined by a schema. Subschema defines portion of data base that is to be accessed by a COBOL program. Created through Data Description Language; accessed through CYBER Database Control System.

Data-Name -
User-defined word that names a data item described in a Data Description entry. Cannot be subscripted, indexed, qualified, or reference modified unless specifically permitted by a given format.

Deadlock -
A situation that arises in concurrent data base access when two or more applications programs are contending for a resource that is locked, and none of the programs can proceed without that resource.

Debugging Facility -
Capability within COBOL 5 compiler and execution routines that implements the DEBUGGING MODE clause, lines with D in column 7, and debugging declarative sections.

Debugging Line -
Any line with D in column 7 of the source program. Compiled as executable code if DEBUGGING MODE clause is used or DB=DL parameter appears on compiler call; otherwise, compiled as comment line.

Delimited Scope Statement -
Any statement that includes an explicit scope terminator.

Delimiter -
Character or sequence of contiguous characters that identify the end of a string of characters and separate that string of characters from the following string of characters. Not part of the string of characters that it delimits.

Dequeue -
The process of removing a message from a queue.

Destination -
A named terminal or collection of terminals that can receive messages under MCS. The recipient of a transmission from an output queue.

Direct Access File -
In the context of AAM, a direct access file is one of the three file organizations. It is characterized by the system hashing of the unique key within each file record to distribute records randomly in blocks, called home blocks, of the file. Synonymous with direct file.

In the context of NOS permanent files, a direct access file is a file that is accessed and modified directly, as contrasted with an indirect access permanent file.

Direct File -
File described by ORGANIZATION IS DIRECT clause. Characterized by preallocated home blocks. Location of record determined by hashing key of record to a home block number. A direct file is implemented according to AAM direct access (DA) files.

Disable -
The term that indicates deactivating the logical connection between MCS and one or more communications devices under the NOS operating system.

Dynamic Access -
Access mode that allows a nonsequential file on mass storage to be accessed randomly or sequentially depending on the format of the access statement.

Dynamic Program -
Program loaded by Fast Dynamic Loader facility at the time the first CALL statement referencing the program is executed.

Edited Item -
Item whose PICTURE clause contains an editing symbol B 0 + - CR DB Z * $, . or /.

Embedded Key -
A key that is an integral part of a record, as opposed to a key that is defined in the Working-Storage Section of a COBOL program.

Enable -
A term used to indicate activating or reactivating the logical connection between MCS and one or more given communication devices under the NOS operating system.

End-of-Group Indicator (EGI) -
A character defined in the application definition that logically separates a group of several messages from succeeding messages and signals the end of a group of messages to MCS or a COBOL program.

End-of-Information (EOI) -
CYBER Record Manager defines end-of-information in terms of the file organization and file residence. See table B-2.

TABLE B-2. END-OF-INFORMATION BOUNDARIES

File Organization	File Residence	Physical Position
Sequential	Mass Storage	After last user record.
	Labeled tape in SI, I, S, L format	After last user record and before any file trailer labels.
	Unlabeled tape in SI or X format	After last user record and before any file trailer labels.
	Unlabeled tape in S or L format	Undefined.
Word Address	Mass storage	After last word allocated to file, which might be beyond the last user record.
Indexed, Actual-Key	Mass storage	After record with highest key value.
Direct	Mass storage	After last record in most recently created overflow block or home block with the highest relative address.

End-of-Message Indicator (EMI) -
A conceptual indicator that delimits one message from the next message and notifies MCS or a COBOL program that the end-of-message condition exists.

End-of-Segment Indicator (ESI) -
A conceptual indicator that delimits one segment within a message from the next segment within a message and notifies MCS or a COBOL program that the end-of-segment condition exists.

Entry -
Descriptive set of consecutive clauses terminated by a period and written in Identification, Environment, or Data Division.

Execution Time -
Time at which a compiled source program is executed. Also known as object time. Defined by LGO or EXECUTE control statement or their equivalent in a batch job or interactive command.

Explicit Scope Terminator -
A reserved word that is included in a delimited scope statement and terminates the scope of a particular conditional statement. Examples are END-IF, END-SEARCH, and END-PERFORM.

Extended -
A term used in conjunction with indexed, direct, and actual-key files to denote a specific type of internal processing by AAM and MIP. Processing is indicated by setting the ORG FIT field to NEW. Contrast with Initial.

Extended Memory -
Core type storage which is physically located outside of the machine. Formerly referred to as Extended Core Storage (ECS) or Large Central Memory (LCM).

External File -
A file that contains the EXTERNAL clause in the FD entry for the file. The file can be described in any program in the run unit, but it exists externally to the program and is shared by all programs that describe it. All External files in a run unit must be described in the main program.

File -
Collection of records defined by SELECT clause and described by FD, SD, or RD entry.

File Information Table (FIT) -
A table through which a COBOL program communicates with CRM. All file processing executes on the basis of information in this table. COBOL sets the majority of the fields automatically.

File Organization -
Defined by ORGANIZATION clause. Can be sequential, indexed, relative, direct, actual-key, or word-address. Established at the time the file is created and cannot change as long as the file exists. Affects access mode, open mode, and formats of statements that can be used to manipulate file records.

High-Order End -
Leftmost character of string of characters; leftmost bit of a string of bits.

Home Block -
Mass storage allocated for a file with direct organization at the time the file is created.

Implementor-Name -
System-name that refers to a particular feature available in COBOL 5. Particular implementor-names such as CDC-64 and SWITCH-6 are unique to COBOL 5 but are within ANSI standard.

Index -
In the context of AAM, a series of keys and pointers to records associated with the keys. The system creates an index for AAM files which relates alternate record keys to primary keys, using the file defined by the second implementor-name of an ASSIGN clause.

In general context, a computer storage area or register, the content of which represents the identification of a particular element.

Index Data Item -
Data item defined by USAGE IS INDEX clause. Can hold index-names.

Index File -
In the context of AAM, a file that contains a series of keys and pointers to records associated with the keys. The index file, once created, must be preserved by a job. See Index.

Index-Name -
User-defined word that names an index associated with a specific table. Defined by INDEXED phrase of OCCURS clause.

Indexed File -
File described by ORGANIZATION IS INDEXED clause. Implemented according to AAM indexed sequential (IS) files. Characterized by records always being in physical and logical order by prime record key values. Synonymous with indexed sequential file.

Indirect Access File -
A mass storage permanent file. Indirect access files can be accessed only through a working copy of the file. If requested, the working copy replaces the permanent file.

Initial -
A term used in conjunction with indexed, direct, and actual-key files to denote a specific type of internal processing by AAM and MIP. Processing is indicated by setting the ORG FIT field to OLD. Contrast with Extended.

Input File -
File referenced in OPEN statement with INPUT phrase after OPEN statement execution and before CLOSE statement execution.

Input Procedure -
In Sort/Merge, set of statements defined by INPUT PROCEDURE phrase of SORT statement. Executes prior to physical sort. Alternative to USING phrase. Must include RELEASE statement for each record to be sorted.

Integer -
Numeric literal or numeric data item that does not include any character positions to the right of the assumed decimal point. Must not be signed or have a zero value unless explicitly allowed for a given format.

Interactive -
Job processing in which the user and the system communicate with each other, rather than the user submitting his job at a central site and receiving output. Interactive processing provides the user with control over his job during processing.

Invalid Key Condition -
Condition that exists during execution when a specific value of the key associated with an indexed, direct, actual-key, or relative file is not valid for the access being attempted. Sets FILE STATUS data item.

Journal -
A file that receives copied messages as they are transferred into or out of queues. The journal provides a record of message transfer activities for recovery purposes.

Key Item -
Data item that defines the location, size, and characteristics of a key for a relative, indexed, direct, actual-key, or word-address file. During execution, contents of item specifies information the system can use to locate record in file. Also, data item that serves to identify the ordering of data.

Key of Reference -
Either the primary key or alternate record key currently being used to access record in an indexed, direct, or actual-key file.

Keyword -
Reserved word whose presence is required when the format in which the word appears is used in a source program.

Language-Name -
System-name that specifies a particular programming language.

Level -
Value of a level-number that indicates either the hierarchical position of a data item or the special properties of a Data Description entry. Value can be 01 through 49, 66, 77, or 88.

For system-logical-records, an octal number 0 through 17 in the system-supplied 48-bit marker that terminates a short or zero-length PRU.

Library -
In COBOL, source of text to be used during COPY statement processing. In operating system context, file produced by EDITLIB utility containing executable code in format required by system loader.

Library-Name -
User-defined word that names a file containing a random UPDATE program library that is to be used by the compiler during compilation of a source program containing COPY statements. Also used to specify the library from which a dynamic subprogram is to be loaded.

Literal -
Character-string whose value is implied by the ordered set of characters that make up the string. See nonnumeric literal and numeric literal.

Local File -
A file type that refers to a temporary file other than the primary file. It often contains a copy of an indirect access file or data from a magnetic tape.

A file currently assigned to a job.

Logical Record -
In COBOL, equivalent to a record.

Under NOS, a data grouping that consists of one or more PRUs terminated by a short PRU or zero-length PRU. Equivalent to a system-logical-record under NOS/BE.

Low Order End -
Rightmost character of a string of characters; rightmost bit of a string of bits.

Mass Storage -
Disk pack or other rotating mass storage device. Not a magnetic tape.

Mass Storage File -
File assigned by control statements to a disk or disk pack. Can have any organization.

Message -
Data associated with an EMI or an EGI.

Message Control System (MCS) -
The software that provides a generalized method of queuing, routing, and journaling of messages that pass between COBOL programs and telecommunications equipment. It can only be used under the NOS operating system.

Mnemonic-Name -
User-defined word associated with an implementor-name in the SPECIAL-NAMES paragraph of the Environment Division.

Mode-Name -
System-name that refers to a particular method of data representation on a magnetic tape. DECIMAL and BINARY in the RECORDING MODE clause are the only two mode-names.

Multiple File Set -
Tape reel or reels with more than one individually labeled file.

Multiple-Index File -
An indexed sequential, direct access, or actual key file for which additional keys, called alternate keys, are defined.

Native Character Set -
Character set associated with system. Defined by installation when COBOL 5 is installed.

Native Collating Sequence -
Collating sequence associated with native character set. Defined by installation. Can be overridden by COLLATING SEQUENCE clause or SET statement.

Network Access Method (NAM) -
The software routines that allow MCS and COBOL programs access to a network of terminals.

Noise Record -
Number of characters the tape drivers discard as being extraneous noise rather than a valid record. Value depends on installation settings.

Nonnumeric Literal -
Literal bounded by quotation marks. Can include any character in computer character set. To represent a single quotation mark character within a nonnumeric literal, two contiguous quotation marks must be used.

Numeric Character -
Digit 0 through 9.

Numeric Literal -
Literal composed of one or more numeric characters. Can contain decimal point, algebraic sign, or both. Decimal point must not be the rightmost character. Algebraic sign must be leftmost character.

Object Time -
See execution time.

On-Line -
Interacting with the network; connected to the network.

Open Mode -
File state that allows records to be either written to or read from a file. Defined by OPEN statement phrase. Affects subsequent statements that can be used to access file.

Operand -
Data referenced by any lowercase word or words that appear in a statement or entry format.

Output Procedure -
In Sort/Merge, set of statements defined by OUTPUT PROCEDURE phrase of SORT or MERGE statement. Executes after physical sort or merge. Alternative to GIVING phrase. Must include RETURN statement for each record to be retrieved from sort or merge.

Overflow Block -
Mass storage the system adds to a file with direct organization when records cannot be accommodated in the home block.

Paragraph -
In Procedure Division, paragraph-name followed by separator period and zero, one, or more entries. In Identification Division and Environment Division, paragraph header followed by zero, one, or more entries.

Paragraph Header -
In Identification Division and Environment Division, specific predefined reserved words followed by separator period.

Partition -
Defined by BAM as a division within a file with sequential organization. Generally, a partition contains several records or sections. Implementation of a partition boundary is affected by file structure and residence. See table B-3.

Notice that in a file with W-type records, a short PRU of level 0 terminates both a section and a partition.

Permanent File -
Feature of operating system. When the job requests permanent file status, the file remains on mass storage when the job terminates so that it can be accessed by other jobs in the future. Requested by control statements outside source program.

Phrase -
Optional portion of a clause or statement.

Physical Record Unit (PRU) -
Under NOS and NOS/BE, the amount of information transmitted by a single physical operation of a specified device. The size of a PRU depends on the device (see table B-4). A PRU that is not full of user data is called a short PRU; a PRU that has a level terminator but no user data is called a zero-length PRU.

Primary Key -
Record key defined by RECORD KEY IS clause for indexed, direct, or actual-key file. Determines physical placement of a record. File creation and updating is only by key. Synonym for prime key. Contrast with alternate record key.

Printable Item -
In Report Writer, printable line is defined by a LINE NUMBER clause.

TABLE B-3. PARTITION BOUNDARIES

Device	RT	BT	Physical Boundary
PRU device	W	I	A short PRU of level 0 containing one-word deleted record pointing back to last I block boundary, followed by a control word with flag indicating partition boundary.
	W	C	A short PRU of level 0 containing a control word with a flag indicating partition boundary.
	D,F,R, T,U,Z	C	A short PRU of level 0 followed by a zero-length PRU of level 17_8.
	S		A zero-length PRU of level number 17_8.
S or L format tape	W	I	Separate tape block containing as many deleted records of record length 0 as required to exceed noise record size, followed by a deleted one-word record pointing back to the last I block boundary, followed by a control word with flag indicating a partition boundary.
	W	C	Separate tape block containing as many deleted records of record length 0 as required to exceed noise record size, followed by a control word with a flag indicating a partition boundary.
	D,F,T, R,U,Z	C,K,E	Tapemark.
	S	-	Zero-length PRU of level number 0.
Any other tape format	-	-	Undefined.

TABLE B-4. PRU SIZES

Device	Size in Number of 60-Bit Words
Mass storage	64
Tape in SI format with coded data	128
Tape in SI format with binary data	512
Tape in I format	512
Tape in other format	Undefined

Procedure -
In Procedure Division, paragraph or group of logically successive paragraphs or a section or group of logically successive sections.

Procedure-Name -
In Procedure Division, user-defined word that names a paragraph or section.

PRU Device -
Under NOS and NOS/BE, a mass storage device or a tape in SI or I format, so called because records on these devices are written in PRUs.

Pseudo-File-Name -
User-defined word that names a file residing on a tape referenced in a MULTIPLE TAPE FILE clause for which no FD entry is specified. File cannot be accessed in program.

Pseudo-Text -
Source program text consisting of character-strings, comment lines, and/or separators bounded by, but not including, pseudo-text delimiters ==.

Queue -
A storage area for messages. A logical collection of messages stored in an area on disk or in central memory before being transmitted or delivered to a specified destination.

Queue (compound) -
A queue that has subqueues; a hierarchical structure.

Queue (input) -
A queue that accumulates messages acquired from external sources.

Queue (output) -
A queue that accumulates messages to be delivered to external destinations.

Queue (simple) -
A queue that does not have subqueues. Output queues and interprogram queues are simple queues. Input queues can be either simple or compound queues.

Random Access -
Access mode that allows a nonsequential file on mass storage to be accessed by key value.

Random File -
In the context of CYBER Record Manager, a file with word addressable, indexed sequential, direct access, or actual key organization in which individual records can be accessed by the values of their keys.

In the context of the NOS or NOS/BE operating systems, a file with the random bit set in the file information table in which individual records are accessed by their relative PRU numbers.

Rank -
The level of a record occurrence in the hierarchical tree structure of a relation.

Realm -
File that is described by the user in a subschema. A named collection of data base records subject to access by a COBOL program.

Record -
In COBOL, unit of a file. Defined by level 01 Data Description entry.

CYBER Record Manager defines a record as a group of related characters. A record or a portion thereof is the smallest collection of information passed between CYBER Record Manager and a user program. Eight different record types exist, as defined by the RT field of the file information table.

Other parts of the operating systems and their products have additional or different definitions of records.

Record Area -
Memory area used to process record. Named by record-name in level 01 Data Description entry.

Record Key -
For indexed, direct, and actual-key files, primary key or alternate record key. For relative files, the key item defined by the RELATIVE KEY clause. For word-address files, key item defined by WORD-ADDRESS KEY clause.

Record Type -
The term record type can have one of several meanings, depending on the context. CYBER Record Manager defines eight record types established by an RT field in the file information table. Tables output by the loader are classified as record types such as text, relocatable, or absolute, depending on the first few words of the tables.

Reference Modification -
A method of referencing a data item at a position other than the first character position by specifying its leftmost character position and length.

Relation -
The logical structure formed by the joining of records based on common identifiers.

Relative File -
File described by ORGANIZATION IS RELATIVE clause. Characterized by fixed-length records with key values equivalent to ordinal positions of records in file.

Repeating Group -
A group data item which is described with an OCCURS clause or a group data item subordinate to a data item which is described with an OCCURS clause.

Report File -
In Report Writer, file whose FD entry contains a REPORT clause. Details of report are specified in RD entry in Report Section.

Reserved Word -
COBOL word that can be used in a COBOL source program but must not appear in program as user-defined words or system-names.

Root File -
The file that ranks lowest in a relation; its record occurrences are pictured at the root of a tree in a hierarchical tree structure.

Run Unit -
Main program and any subprogram it calls plus execution routines needed to execute the main and subprograms.

Schema -
A detailed description of the internal structure of a data base. A schema is not specified in the COBOL environment.

Section -
In COBOL, group of one or more paragraphs introduced by section header; predefined in Environment and Data Divisions, user-defined in Procedure Division.

Defined by BAM as a division within a file with sequential organization. Generally, a section contains more than one record and is a division within a partition of a file. A section terminates with a physical representation of a section boundary. See table B-5.

The NOS and NOS/BE operating systems equate a section with a system-logical-record of level 0 through 16_8.

Sentence -
One or more consecutive statements terminated by a separator period.

Sequential Access -
Access mode that allows a sequential or nonsequential file to be accessed by record position.

Sequential File -
File described by ORGANIZATION IS SEQUENTIAL clause. Characterized by access to records only by position of record in file. Can reside on magnetic tape or mass storage.

Short PRU -
A PRU that does not contain as much user data as the PRU can hold and that is terminated by a system terminator with a level number.

Under NOS, a short PRU defines EOR.

Under NOS/BE, a short PRU defines the end of a system-logical-record. In the CYBER Record Manager context, a short PRU can have several interpretations depending on the record and blocking types.

Source -
A terminal from which messages can be received by an MCS application.

TABLE B-5. SECTION BOUNDARIES

Device	RT	BT	Physical Representation
PRU device	W	I	Deleted one-word record pointing back to last I block boundary followed by a control word with flags indicating a section boundary. At least the control word is in a short PRU of level 0.
	W	C	Control word with flags indicating a section boundary. The control word is in a short PRU of level 0.
	D,F,R,T,U,Z	C	Short PRU with level less than 17 octal.
	S	Any	Undefined.
S or L format tape	W	I	A separate tape block containing as many deleted records of record length 0 as required to exceed noise record size followed by a deleted one-word record pointing back to last I block boundary followed by a control word with flags indicating a section boundary.
	W	C	A separate tape block containing as many deleted records of record length 0 as required to exceed noise record size followed by a control word with flags indicating a section boundary.
	D,F,R,T,U,Z	C,K,E	Undefined.
	S	Any	Undefined.
Any other tape format	-	-	Undefined.

Source Program -
Syntactically correct set of COBOL statements beginning with an Identification Division and ending with the end of the Procedure Division. Also known as program.

Sparse Key -
An alternate key that is used infrequently. Only those alternate key values of interest are included in the index file. The ALTERNATE RECORD KEY WITH DUPLICATES clause is used to specify keys of interest.

Statement -
Syntactically valid combination of words and symbols written in Procedure Division.

Static Program -
Program that is loaded with the base module. Contrast with dynamic program.

Subschema -
Description of part of a data base that is to be accessed by COBOL program. Created through Data Description Language; accessed through CYBER Database Control System interfaces. Items can be used in a program, but the description of the items is in the subschema rather than the program itself.

Sum Counter -
In Report Writer, signed numeric data item established by SUM clause. Contains result of summing operations.

Switch -
Software convention that can have the status OFF or ON. SWITCH-1 through SWITCH-126 can be defined in SPECIAL-NAMES paragraph. First six switches are external and can be manipulated by SWITCH control statement, terminal user, operator, or SET statement. Switches 7 through 126 are internal and can be manipulated by the SET statement.

System-Logical-Record -
Under NOS/BE, a data grouping that consists of one or more PRUs terminated by a short PRU or zero-length PRU. These records can be transferred between devices without loss of structure.

Equivalent to a logical record under NOS.

Equivalent to a CYBER Record Manager S type record.

Table -
Set of repeated items of data that is defined by OCCURS clause.

Text Editor Facility -
A software routine that allows the user to make many changes to a job or to a file while at an interactive terminal.

Time-Sharing -
The simultaneous sharing of a computer's resources by many people at many terminals. Each terminal user is unaware of other users and is given the impression that the computer is servicing him only.

Update -
Product running under operating system that allows a program library to be created in a special format which can be used by COPY statement processing. Described in UPDATE reference manual.

User-Defined Word -
Word supplied by programmer to satisfy format of clause or statement. 1 through 30 characters A through Z, 0 through 9, or hyphen; hyphen cannot be first or last character. Different types of words might have futher restrictions for uniqueness, length, or characters.

Variable-Occurrence Data Item -
Data item described with OCCURS clause that has a DEPENDING ON phrase.

Word-Address File -
File described by ORGANIZATION IS WORD-ADDRESS clause. Characterized by records identified by a key that indicates the relative word within mass storage file at which the record begins.

W-Type Record -
One of the eight record types supported by CYBER Record Manager. Such records appear in storage preceded by a system supplied control word. The existence of the control word allows files with sequential organization to have both partition and section boundaries.

Zero-Byte Terminator -
12 bits of zero in the low order position of a word that marks the end of the line to be displayed at a terminal or printed on a line printer. The image of cards input through the card reader or terminal also has such a terminator.

Zero-Length PRU -
A PRU that contains system information, but no user data. Under CYBER Record Manager, a zero-length PRU of level 17 is a partition boundary. Under NOS, a zero-length PRU defines EOF.

ADDITIONAL SOFTWARE FOR DATA BASE PROGRAMS C

Section 14, CDCS Interface, includes a COBOL subschema listing and a COBOL 5 program that accesses data base files. This appendix contains listings of the schema, the master directory, and the program that creates the data base files.

Figure C-1 shows the source listing for the schema DBFILES, which is used in the execution of the COBOL 5 program shown in section 14. Figure C-2 shows the creation of the master directory. The COBOL 5 program shown in figure C-3 creates the data base files; the input data for the program is shown in figure C-4.

```
SCHEMA NAME IS DBFILES.
  AREA NAME IS ADDRESSES
   ACCESS-CONTROL LOCK FOR UPDATE IS "UPDATE ADDRESSES"
   ACCESS-CONTROL LOCK FOR RETRIEVAL IS "READ ADDRESSES".
    RECORD IS VENDOR-CUSTOMER WITHIN ADDRESSES.
     02  VEN-CUST-NO          PICTURE "9(6)".
     02  REC-TYPE             PICTURE "X".
     02  STREET-NO            PICTURE "X(5)".
     02  STREET               PICTURE "X(15)".
     02  CITY                 PICTURE "X(15)".
     02  STATE                PICTURE "XX".
     02  ZIP-CODE             PICTURE "9(5)".
     02  VEN-CUST-NAME        PICTURE "X(30)".
  AREA NAME IS ORDERS
   ACCESS-CONTROL LOCK FOR UPDATE IS "UPDATE ORDERS".
    RECORD IS ORDERED WITHIN ORDERS.
     02  ORDER-NO             PICTURE "X(6)".
     02  CUST-NO              PICTURE "9(6)".
     02  ORDER-DATE           PICTURE "9(6)".
     02  BILL-DATE            PICTURE "9(6)".
     02  BILL-AMOUNT          PICTURE "9(6)V99".
     02  AUTOKEY              TYPE FIXED 10.
  AREA NAME IS LINEITEMS.
    RECORD IS LINE-ITEM WITHIN LINEITEMS.
     02  PART-NUM             PICTURE "X(6)".
     02  PART-DESC            PICTURE "X(20)".
     02  ORD-NUMBER           PICTURE "X(6)".
     02  QTY-ORDER            PICTURE "9(4)".
     02  QTY-SHIP             PICTURE "9(4)".
     02  UNIT-PRICE           PICTURE "9(4)V99".
     02  AUTO-KEY             TYPE FIXED 10.
DATA CONTROL.
    AREA IS ADDRESSES
        KEY IS VEN-CUST-NO.
    AREA IS ORDERS
        KEY IS AUTOKEY.
    AREA IS LINEITEMS
        KEY IS AUTO-KEY.
    RELATION NAME IS CUST-ORDERS
        JOIN WHERE VEN-CUST-NO EQ CUST-NO
                   ORDER-NO EQ ORD-NUMBER.
```

Figure C-1. Source Listing for Schema DBFILES

```
SCHEMA NAME IS DBFILES
    FILE NAME IS DBFILES.
    AREA NAME IS ADDRESSES
      PFN IS "CSTFLE"
      UN IS "USER123".
    AREA NAME IS ORDERS
      PFN IS "ORDFLE"
      UN IS "USER123".
    AREA NAME IS LINEITEMS
      PFN IS "ITMFLE"
      UN IS "USER123".
SUBSCHEMA NAME IS BILLING
    FILE NAME IS SUBLIB.
```

Figure C-2. Source Listing for Master Directory

```
1     IDENTIFICATION DIVISION.
2     PROGRAM-ID.  CREATDB.
3     ENVIRONMENT DIVISION.
4     CONFIGURATION SECTION.
5     SOURCE-COMPUTER.  CYBER-170.
6     OBJECT-COMPUTER.  CYBER-170.
7     SPECIAL-NAMES.
8         SUB-SCHEMA IS BILLING.
9     INPUT-OUTPUT SECTION.
10    FILE-CONTROL.
11        SELECT CARD-IN ASSIGN TO INPUT.
12    DATA DIVISION.
13    FILE SECTION.
14    FD  CARD-IN
15            LABEL RECORDS ARE OMITTED
16            DATA RECORDS ARE CUST-REC, ORD-REC, LINE-REC.
17    01  CUST-REC.
18        03  CUST-NUMBER          PICTURE 9(6).
19        03  REC-TYPE             PICTURE X.
20        03  STREET-NO            PICTURE X(5).
21        03  STREET               PICTURE X(15).
22        03  CITY                 PICTURE X(15).
23        03  STATE                PICTURE XX.
24        03  ZIP-CODE             PICTURE 9(5).
25        03  CUST-NAME            PICTURE X(30).
26        03  CARD-CODE-1          PICTURE 9.
27    01  ORD-REC.
28        03  ORDER-NO             PICTURE X(6).
29        03  CUST-NO              PICTURE 9(6).
30        03  ORDER-DATE           PICTURE 9(6).
31        03  BILL-DATE            PICTURE 9(6).
32        03  BILL-AMOUNT          PICTURE 9(6)V99.
33        03  FILLER               PICTURE X(47).
34        03  CARD-CODE-2          PICTURE 9.
35    01  LINE-REC.
36        03  PART-NUM             PICTURE X(6).
37        03  PART-DESC            PICTURE X(20).
38        03  ORD-NUMBER           PICTURE X(6).
39        03  QTY-ORDER            PICTURE 9(4).
40        03  QTY-SHIP             PICTURE 9(4).
41        03  UNIT-PRICE           PICTURE 9(4)V99.
42        03  FILLER               PICTURE X(33).
43        03  CARD-CODE-3          PICTURE 9.
44    WORKING-STORAGE SECTION.
45    01 UPDATE-PASSWORD-ORD PIC X(30).
46    01 UPDATE-PASSWORD-ADDR PIC X(30).
47    PROCEDURE DIVISION.
48    DECLARATIVES.
49    CUST-ACC-CONTROL SECTION.
50        USE FOR ACCESS CONTROL ON I-O
```

Figure C-3. Program for Creating Data Base Files (Sheet 1 of 2)

```
51          KEY IS UPDATE-PASSWORD-ADDR FOR ADDRESSES.
52      ACC-1.
53          MOVE "UPDATE ADDRESSES" TO UPDATE-PASSWORD-ADDR.
54      ORDERS-ACC-CONTROL SECTION.
55          USE FOR ACCESS CONTROL ON I-O
56          KEY IS UPDATE-PASSWORD-ORD FOR ORDERS.
57      ACC-2.
58          MOVE "UPDATE ORDERS" TO UPDATE-PASSWORD-ORD.
59      END DECLARATIVES.
60      OPENING.
61          OPEN INPUT CARD-IN.
62          OPEN OUTPUT ADDRESSES, ORDERS, LINEITEMS.
63      WRITING.
64          READ CARD-IN RECORD AT END GO TO CLOSING.
65          IF CARD-CODE-1 EQUALS 1 PERFORM CUST-WRITE.
66          IF CARD-CODE-2 EQUALS 2 PERFORM ORD-WRITE.
67          IF CARD-CODE-3 EQUALS 3 PERFORM LINE-WRITE.
68          GO TO WRITING.
69      CUST-WRITE.
70          MOVE CUST-NUMBER TO CUST-NUM.
71          MOVE CORRESPONDING CUST-REC TO CUSTOMER.
72          WRITE CUSTOMER INVALID KEY GO TO KEY-ERROR.
73      ORD-WRITE.
74          MOVE ZEROS TO AUTOKEY.
75          MOVE CORRESPONDING ORD-REC TO ORDERED.
76          WRITE ORDERED INVALID KEY GO TO KEY-ERROR.
77      LINE-WRITE.
78          MOVE ZEROS TO AUTO-KEY.
79          MOVE CORRESPONDING LINE-REC TO LINE-ITEM.
80          WRITE LINE-ITEM INVALID KEY GO TO KEY-ERROR.
81      KEY-ERROR.
82          DISPLAY "BAD KEY FOR RECORD  " CUST-REC.
83          GO TO WRITING.
84      CLOSING.
85          CLOSE CARD-IN, ADDRESSES, ORDERS, LINEITEMS.
86          STOP RUN.
```

Figure C-3. Program for Creating Data Base Files (Sheet 2 of 2)

```
100150C2331 WINNETKA AVE    MINNEAPOLIS     MN55411POLAR GEAR AND SUPPLY CO.     1
101333V11347BROADWAY        HOLLYWOOD       CA90025CHEAP PART COMPANY            1
216800C1236 INDUSTRIAL DR   SAN JOSE        CA95151INTERNATIONAL TRIKE PARTS     1
649025C11185MAIN STREET     FAIRFIELD       CA94533ALL AMERICAN BIKES            1
030667216800041176000000000144500                                                2
149020216800042576000000000154000                                                2
095188649025040776000000000067250                                                2
206193649025042176000000000406375                                                2
TRK030BLACK EXTENSION TUBE0306670000010000045                                    3
TRK2108 INCH TIRE           0306670000002500095O                                 3
FRM009HANDLE BAR            0306670000005000207 5                                3
DRV001CHAIN LINKS           0306670000010000012 5                                3
BRK891BRAKE ASSEMBLY        1490200000003000225 0                                3
TRK200REAR WHEEL            1490200000005000173 0                                3
BRP001BRAKE PAD LEFT        0951880000012500009 5                                3
BRP002BRAKE PAD RIGHT       0951880000012500009 5                                3
FRM005REAR WHEEL BRACE      0951880000007500058 0                                3
WHL011REAR WHEEL HUB ASSY   2061930000005000125 0                                3
WHL000BACK WHEEL ASSEMBLY   2061930000005000259 5                                3
BKS020FLAT SEAT COVER       2061930000025000019 5                                3
DRV002MASTER CHAIN LINK     2061930000012500082 5                                3
FRM009HANDLE BAR            2061930000003000207 5                                3
```

Figure C-4. Input Data for Creating Data Base Files

Figure D-1 illustrates a Message Control System (MCS) application, named FINANCE, for a financial institution. There are three major functional subsystems within the organization: savings, loans, and general ledger. The three subsystems represent one MCS application.

Within the savings subsystem, there are six terminals (SAVINGS001, SAVINGS002,...,SAVINGS006) from which savings-type transactions are entered. Two COBOL programs (SVMONEY and SVCHANG) exist to handle the two different types of transactions.

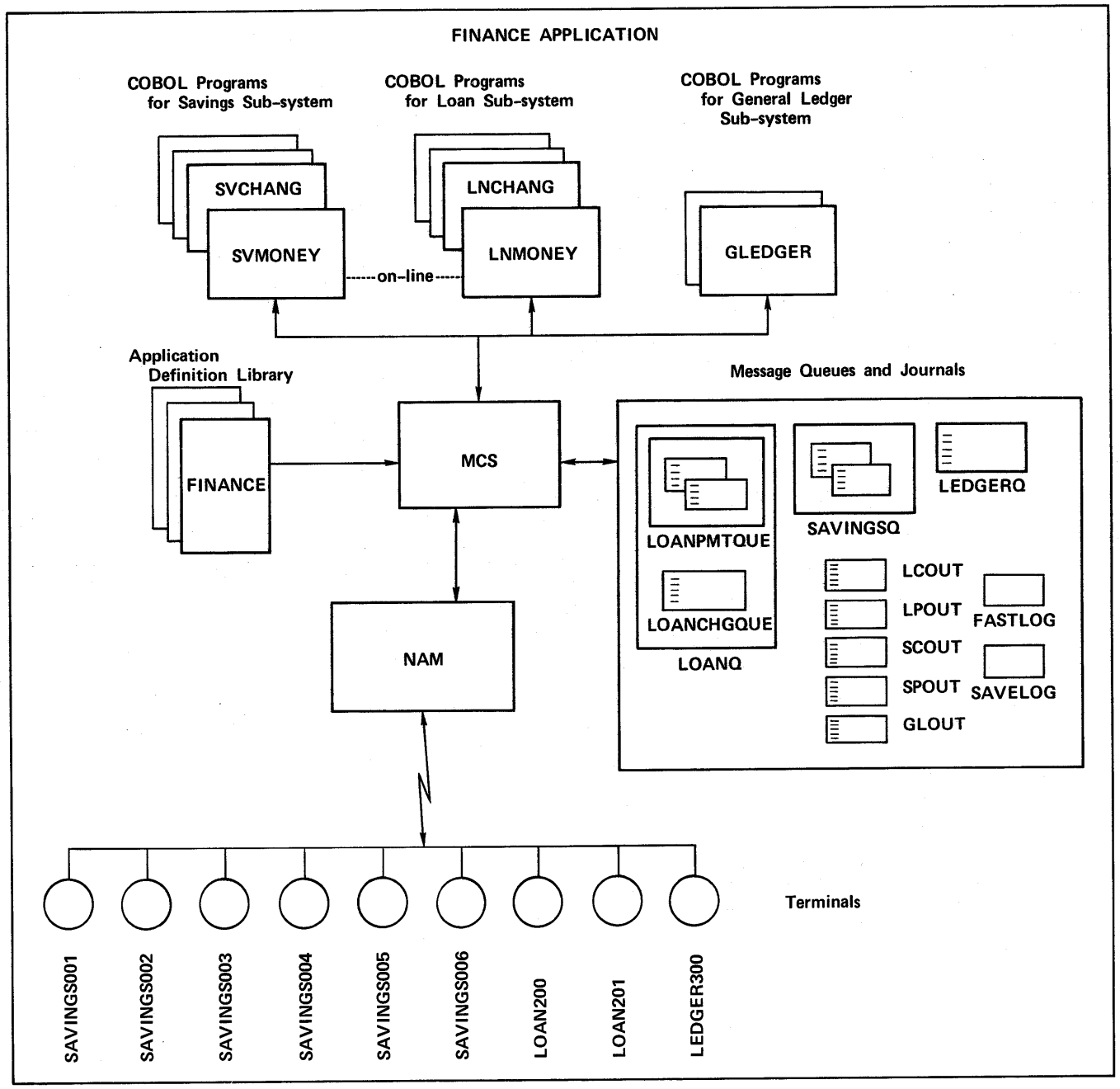


Figure D-1. An Overview of the FINANCE Application

Within the loan subsystem, there are two terminals (LOANS200 and LOANS201) from which loan-type transactions are entered. Two COBOL programs (LNMONEY and LNCHANG) exist to handle the two different types of transactions. A simple queue is used for one type of transaction; a compound queue is used for the other type.

Within the general ledger subsystem, there is one terminal (LEDGER300) from which general-ledger-type transactions are entered. One COBOL program (GLEDGER) exists to handle the credit or debit posting transactions.

Two of the COBOL programs (LNMONEY and SVMONEY) are on-line programs that are automatically invoked when the FINANCE application is initiated. The programs execute until 5 o'clock for the processing of all loan monetary transactions and savings monetary transactions. At 5 o'clock, all savings and loan terminals are sent a five-minute warning message. LNMONEY is terminated by MCS at 5:15. SVMONEY is terminated after 5:15 and after the appropriate queue is empty (as determined within the COBOL program by the NO DATA clause of the RECEIVE statement).

Three of the COBOL programs (SVCHANG, LNCHANG and GLEDGER) are off-line programs and are invoked automatically by MCS at 5 o'clock for the processing of statistical record changes (such as customer name or address change) and general ledger posting.

The Application Definition Language (ADL) shown in figure D-2 defines the following:

- The COBOL programs within the FINANCE application
- The message queues to be used by the COBOL programs
- The terminals to be used by the FINANCE application
- The journal files to be used by the FINANCE application

```
APPLICATION GLOBAL DIVISION

APPLICATION-NAME IS FINANCE
OPERATOR IS LOANLIST  PASSWORD IS "LNAOP"
INITIATION IS AUTOMATIC

APPLICATION PROGRAM DIVISION

PROGRAM IS LNMONEY
    INVOCATION-FILE IS LPJOB  OWNER IS "[redacted]"
PROGRAM IS LNCHANG
    INVOCATION-FILE IS LCJOB  OWNER IS "[redacted]"
PROGRAM IS SVMONEY
    INVOCATION-FILE IS SPJOB  OWNER IS "[redacted]"
PROGRAM IS SVCHANG
    INVOCATION-FILE IS SCJOB  OWNER IS "[redacted]"
PROGRAM IS GLEDGER
    INVOCATION-FILE IS GLJOB OWNER IS "[redacted]"

APPLICATION DATA DIVISION

MESSAGE IS LOANMSG
    SERIAL-NUMBER IS GENERATED WITH ECHO
SEGMENT IS LOANSEG
    FIELD IS FIELD1
    STARTS AT CHARACTER 1 EXTENDS TO INSTANCE 1 OF ","
    CONDITION IS LOANCH VALUE "CHANGE"
    CONDITION IS LOANPY VALUE "PAYMENT"
MESSAGE IS SAVINGSMSG
    SERIAL-NUMBER IS GENERATED WITH ECHO
SEGMENT IS SAVINGSSEG
    FIELD IS FIELD2
    STARTS AT CHARACTER 1 EXTENDS TO INSTANCE 1 OF ","
    CONDITION IS SAVECH VALUE "CHANGE"
    CONDITION IS SAVEPY VALUE "PAYMENT"
```

Defines five COBOL programs in FINANCE application.

Defines message types and conditions used for routing.

Figure D-2. ADL Source Listing for FINANCE (Sheet 1 of 3)

```
SOURCE-DESTINATION DIVISION

INVITATION-LIST IS LOANLIST
    SOURCES ARE LOAN200 AND LOAN201
    MESSAGES ARE LOANMSG
    MODE IS DATA
    STATUS IS ENABLED
BROADCAST-LIST IS LOANDISPLAY
    DESTINATIONS ARE LOAN200 AND LOAN201
SYMBOLIC-NAME IS LOAN200
    TYPE IS INTERACTIVE
SYMBOLIC-NAME IS LOAN201
    TYPE IS INTERACTIVE
INVITATION-LIST IS SAVINGSLIST
    SOURCES ARE SAVINGS001 AND SAVINGS002 AND SAVINGS003
          AND SAVINGS004 AND SAVINGS005 AND SAVINGS006
    MESSAGES ARE SAVINGSMSG
    MODE IS DATA
    STATUS IS ENABLED
BROADCAST-LIST IS SAVDISPLAY
    DESTINATIONS ARE SAVINGS001 AND SAVINGS002 AND SAVINGS003
          AND SAVINGS004 AND SAVINGS005 AND SAVINGS006
SYMBOLIC-NAME IS SAVINGS001
    TYPE IS INTERACTIVE
SYMBOLIC-NAME IS SAVINGS002
    TYPE IS INTERACTIVE
SYMBOLIC-NAME IS SAVINGS003
    TYPE IS INTERACTIVE
SYMBOLIC-NAME IS SAVINGS004
    TYPE IS INTERACTIVE
SYMBOLIC-NAME IS SAVINGS005
    TYPE IS INTERACTIVE
SYMBOLIC-NAME IS SAVINGS006
    TYPE IS INTERACTIVE
SYMBOLIC-NAME IS LEDGER300
    TYPE IS INTERACTIVE
SYMBOLIC-NAME IS FASTLOG
    TYPE IS JOURNAL
    ALIAS IS FLLOG
SYMBOLIC-NAME IS SAVELOG
    TYPE IS JOURNAL
    ALIAS IS SPLOG
```

Defines symbolic names of terminals; specifies which terminals can receive and which terminals can send messages.

```
QUEUE DIVISION

INPUT SECTION

QUEUE IS LOANQ
  SUB-QUEUE-1 IS LOANPMTQUE
   SUB-QUEUE-2 IS FASTLOANPMT
          JOURNAL IS FASTLOG
   SUB-QUEUE-2 IS SLOWLOANPMT
          MEDIUM IS DISK
          RESIDENCY IS SLPFILE
  SUB-QUEUE-1 IS LOANCHGQUE
        MEDIUM IS DISK
        RESIDENCY IS LCQFILE

QUEUE IS LEDGERQ
    MEDIUM IS DISK
    RESIDENCY IS GLFILE

QUEUE IS SAVINGSQ
  SUB-QUEUE-1 IS SAVEPMTQUE
        JOURNAL IS SAVELOG
  SUB-QUEUE-1 IS SAVECHGQUE
        MEDIUM IS DISK
        RESIDENCY IS SCQFILE
```

Defines the simple input queue LOANCHGQUE (on disk) and the compound queue LOANPMTQUE (one subqueue on disk; one subqueue in central memory).

Defines three simple input queues: LEDGERQ and SAVECHGQUE (on disk) and SAVEPMTQUE (in central memory).

Figure D-2. ADL Source Listing for FINANCE (Sheet 2 of 3)

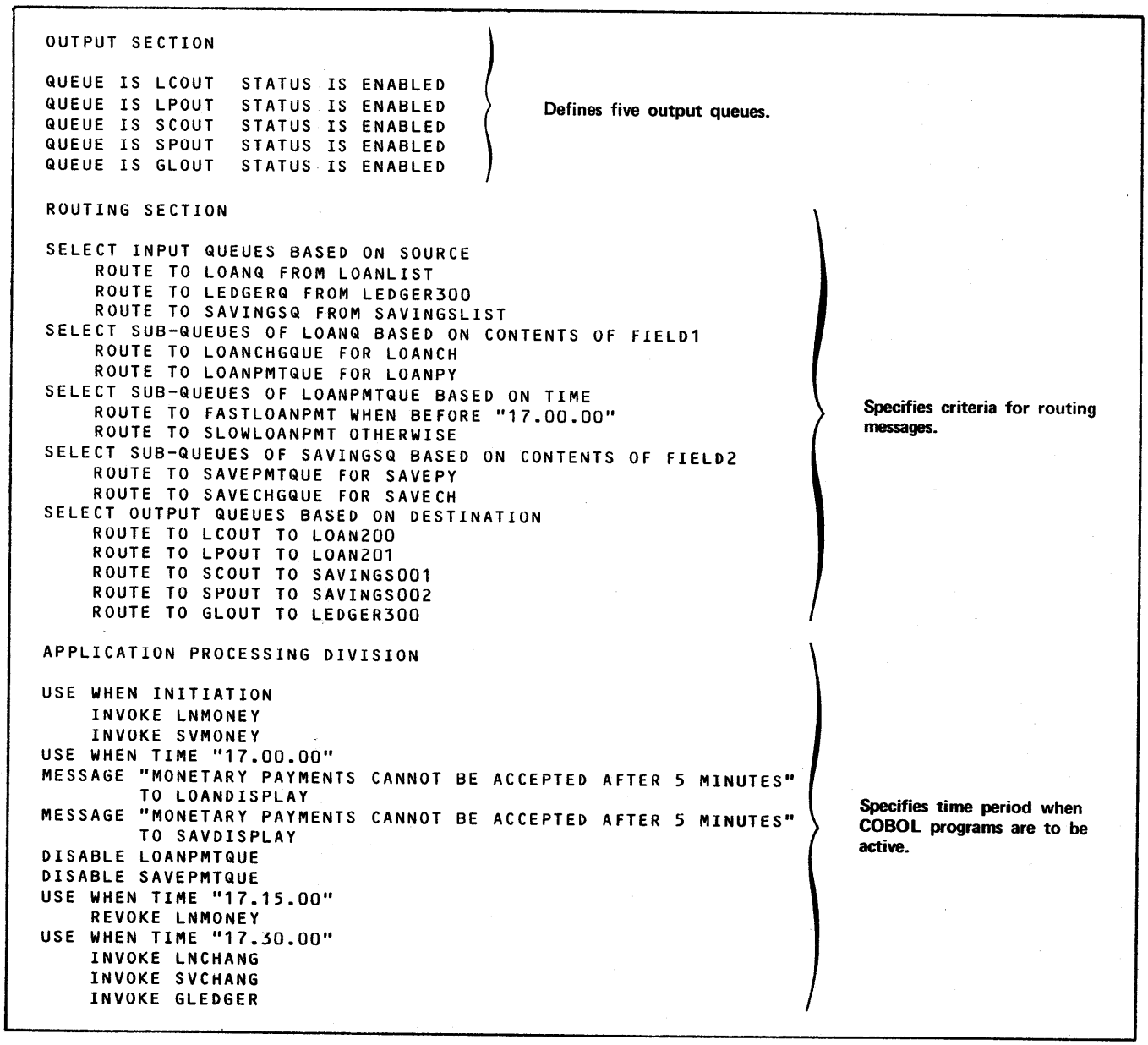


Figure D-2. ADL Source Listing for FINANCE (Sheet 3 of 3)

INDEX

COMMENT SHEET

MANUAL TITLE: COBOL Version 5 User's Guide

PUBLICATION NO.: 60497200

REVISION: E

This form is not intended to be used as an order blank. Control Data Corporation welcomes your evaluation of this manual. Please indicate any errors, suggested additions or deletions, or general comments on the back (please include page number references).

_____ Please reply _____ No reply necessary

FOLD FOLD

NO POSTAGE NECESSARY IF MAILED IN THE UNITED STATES

BUSINESS REPLY MAIL

FIRST CLASS PERMIT NO. 8241 MINNEAPOLIS, MN.

POSTAGE WILL BE PAID BY ADDRESSEE

CONTROL DATA

Publications and Graphics Division
Mail Stop: SVL104
P.O. Box 3492
Sunnyvale, California 94088-3492

FOLD FOLD

NO POSTAGE STAMP NECESSARY IF MAILED IN U.S.A.
FOLD ON DOTTED LINES AND TAPE

NAME:

COMPANY:

STREET ADDRESS:

CITY/STATE/ZIP:

TAPE TAPE

CORPORATE HEADQUARTERS, P.O. BOX 0, MINNEAPOLIS, MINN. 55440
SALES OFFICES AND SERVICE CENTERS IN MAJOR CITIES THROUGHOUT THE WORLD

LITHO IN U.S.

CONTROL DATA CORPORATION

10268015 9