decdatasystem

COS310 system reference manual

digital

COS-310
System Reference Manual

Order No. AA-D647A-TA

October 1978

This is a reference manual for the COS-310 system user who wants to use the DIBOL language in developing application programs.

SUPERSESSION/UPDATE INFORMATION:	This is a new manual.
OPERATING SYSTEM AND VERSION:	COS-310 V 8.00
SOFTWARE VERSION:	COS-310 V 8.00

To order additional copies of this document, contact the Software Distribution Center, Digital Equipment Corporation, Maynard, Massachusetts 01754.

digital equipment corporation · maynard. massachusetts

First Printing, October 1978

The postage-prepaid READER'S COMMENTS form on the last page of this document requests the user's critical evaluation to assist us in preparing future documentation.

The following are trademarks of Digital Equipment Corporation:

DIGITAL	DECsystem-10	MASSBUS
DEC	DECtape	OMNIBUS
PDP	DIBOL	OS/8
DECUS	EDUSYSTEM	PHA
UNIBUS	FLIP CHIP	RSTS
COMPUTER LABS	FOCAL	RSX
COMTEX	INDAC	TYPESET-8
DDT	LAB-8	TYPESET-11
DECCOMM	DECSYSTEM-20	TMS-11
ASSIST-11	RTS-8	ITPS-10

CONTENTS

		Page
PREFACE		xi
INTRODUCTION		xii
CHAPTER 1	DIBOL LANGUAGE	1-1
1.1	SOURCE PROGRAM	1-1
1.2	STATEMENTS	1-3
1.2.1	ACCEPT - Input/Output Statement	1-4
1.2.2	CALL - Control Statement	1-6
1.2.3	CHAIN - Control Statement	1-7
1.2.4	Data Manipulation Statements	1-9
1.2.4.1	Data Conversion	1-11
1.2.4.2	Arithmetic Expressions	1-11
1.2.4.3	Clearing Fields and Records	1-13
1.2.4.4	Moving Alphanumeric Data	1-14
1.2.4.5	Moving Numeric Data	1-14
1.2.4.6	Moving Records	1-15
1.2.4.7	Data Formatting	1-16
1.2.5	DISPLAY - Input/Output Statement	1-18
1.2.6	END - Compiler Statement	1-20
1.2.7	FINI - Input/Output Statement	1-21
1.2.8	FORMS - Input/Output Statement	1-22
1.2.9	GO TO - Control Statement	1-23
1.2.9.1	Unconditional GO TO	1-23
1.2.9.2	Computed GO TO	1-23
1.2.10	IF - Control Statement	1-24
1.2.11	INCR	1-24
1.2.12	INIT - Input/Output Statement	1-26
1.2.13	ON ERROR - Control Statement	1-28
1.2.14	PROC - Compiler Statement	1-29
1.2.15	READ - Input/Output Statement	1-30
1.2.16	RECORD - Data Definition Statement	1-31
1.2.17	RETURN - Control Statement	1-33
1.2.18	START - Compiler Statement	1-34
1.2.19	STOP - Control Statement	1-35
1.2.20	TRACE/NO TRACE - Debugging Statement	1-36
1.2.21	TRAP - Control Statement	1-37

			Page
	1.2.22	WRITE - Input/Output Statement	1-39
	1.2.23	XMIT - Input/Output Statement	1-40
CHAPTER	**2**	**THE MONITOR**	**2-1**
	2.1	MASTER CONTROL PROGRAM	2-1
	2.1.1	MOUNT Messages	2-3
	2.1.2	Operating Procedures	2-4
	2.2	MONITOR COMMANDS	2-4
	2.2.1	BATCH	2-5
	2.2.2	DATE	2-6
	2.2.3	DELETE	2-7
	2.2.4	DIRECTORY	2-8
	2.2.5	PLEASE	2-10
	2.2.6	RUN	2-11
	2.2.7	SAVE	2-13
	2.3	EDITOR COMMANDS	2-14
	2.3.1	ERASE	2-15
	2.3.2	FETCH	2-16
	2.3.3	LIST	2-17
	2.3.4	Line Number	2-18
	2.3.5	Number Commands	2-20
	2.3.6	RESEQUENCE	2-21
	2.3.7	WRITE	2-22
	2.4	MONITOR ERROR MESSAGES	2-23
	2.5	RUN-TIME ERROR MESSAGES	2-24
CHAPTER	**3**	**SYSTEM GENERATION PROGRAM (SYSGEN)**	**3-1**
	3.1	SYSGEN/B OPERATING PROCEDURES	3-1
	3.2	SYSGEN/C OPERATING PROCEDURES	3-3
	3.3	SYSGEN ERROR MESSAGES	3-5
CHAPTER	**4**	**DATA FILE UTILITY PROGRAM (DFU)**	**4-1**
	4.1	DFU OPERATING PROCEDURES	4-1
	4.1.1	DFU,filnam Operating Procedures	4-2
	4.1.2	DFU/B Operating Procedures	4-2
	4.1.3	DFU/K Operating Procedures	4-3
	4.1.4	DFU/D Operating Procedures	4-3
	4.1.5	DFU/DL Operating Procedures	4-4
	4.1.6	DFU/E Operating Procedures	4-5
	4.1.7	DFU/EL Operating Procedures	4-6
	4.2	LOGICAL UNIT ASSIGNMENTS ON THE COS-310 SYSTEM	4-7
	4.2.1	Determining Logical Unit Size	4-7
	4.2.2	How Logical Units are Assigned by DFU	4-8
	4.3	DISK USERS	4-9
	4.4	DFU ERROR MESSAGES	4-10

			Page
CHAPTER	**5**	**DIBOL COMPILER (COMP)**	**5-1**
	5.1	COMP OPERATING PROCEDURES	5-1
	5.1.1	Source Program Compilation Listing	5-2
	5.1.2	Storage Map Listing	5-3
	5.2	CONDITIONAL COMPILATION PROCEDURE (CCP)	5-5
	5.3	SIZE OF THE BINARY PROGRAM	5-6
	5.4	COMPILER ERROR MESSAGES	5-8
CHAPTER	**6**	**DIBOL DEBUGGING TECHNIQUE (DDT)**	**6-1**
	6.1	DDT OPERATING PROCEDURES	6-1
	6.2	DDT COMMANDS	6-2
	6.3	DDT ERROR MESSAGES	6-3
CHAPTER	**7**	**CROSS REFERENCE PROGRAM (CREF)**	**7-1**
	7.1	CREF OPERATING PROCEDURES	7-1
	7.2	CREF ERROR MESSAGES	7-2
CHAPTER	**8**	**PERIPHERAL INTERCHANGE PROGRAM (PIP)**	**8-1**
	8.1	PIP OPERATING PROCEDURES	8-1
	8.1.1	Transfer Binary File (OPT- B)	8-3
	8.1.2	Copy Device (OPT- C)	8-3
	8.1.3	Transfer Data Files (OPT- D)	8-4
	8.1.4	Consolidate Space in Directory (OPT- E)	8-5
	8.1.5	Allocate Space to Binary Scratch Area (OPT- E)	8-6
	8.1.6	Copy and Verify (OPT- I)	8-7
	8.1.7	Perform a Read/Check (OPT- R)	8-7
	8.1.8	Transfer Source Files (OPT- S)	8-8
	8.1.9	Transfer System Program (OPT- V)	8-8
	8.1.10	Return to Monitor (OPT- X)	8-9
	8.2	PIP ERROR MESSAGES	8-9
CHAPTER	**9**	**SORT PROGRAM (SORT)**	**9-1**
	9.1	SORT OPERATING PROCEDURES	9-1
	9.2	SORT COMMAND FILE	9-2
	9.2.1	Record Descriptor Division	9-2
	9.2.2	INPUT/OUTPUT Division	9-2
	9.3	MERGE OPERATING PROCEDURE	9-4
	9.3.1	Merge Using SORT and the /A Option	9-5
	9.3.2	Merge Using SORT and the /M Option	9-5
	9.3.3	Merge Using SORT and the /n Option	9-6
	9.4	SORT ERROR MESSAGES	9-7

			Page
CHAPTER	**10**	**FILE EXCHANGE PROGRAM (FILEX)**	**10-1**
	10.1	UNIVERSAL DISKETTE	10-1
	10.2	FILEX OPERATING PROCEDURES	10-4
	10.3	COPY (OPT:C)	10-5
	10.3.1	OS/8 ASCII Input (Mode A)	10-5
	10.3.2	COS-310 Data Input (Mode D)	10-6
	10.3.3	Universal Input (Mode U)	10-6
	10.3.4	Output Modes (A, D, S, U)	10-7
	10.3.4.1	COS/8 ASCII Output (Mode A)	10-7
	10.3.4.2	COS-310 Data File Output (Mode D)	10-8
	10.3.4.3	COS-310 Source File Output (Mode S)	10-8
	10.3.4.4	Universal Diskette Output (Mode U)	10-9
	10.4	DELETE (OPT:D)	10-11
	10.5	LIST (OPT:L)	10-11
	10.6	EXIT (OPT:X)	10-12
	10.7	ZERO (OPT:Z)	10-12
	10.8	FILEX ERROR MESSAGES	10-13
CHAPTER	**11**	**PATCH PROGRAM (PATCH)**	**11-1**
	11.1	PATCH OPERATING PROCEDURES	11-1
	11.2	ERROR CORRECTION	11-3
	11.2.1	CTRL/U or R (Restart)	11-4
	11.2.2	Wrong Old Value	11-4
	11.2.3	Bad Checksum	11-4
	11.3	PATCH ERROR MESSAGES	11-5
CHAPTER	**12**	**BOOT PROGRAM (BOOT)**	**12-1**
	12.1	BOOT OPERATING PROCEDURES	12-1
	12.2	BOOT ERROR MESSAGES	12-1
CHAPTER	**13**	**LINE CHANGE PROGRAM (LINCHG)**	**13-1**
	13.1	LINCHG OPERATING PROCEDURES	13-1
	13.2	LINCHG ERROR MESSAGES	13-2
CHAPTER	**14**	**FORMAT PROGRAMS (DKFMT, DYFMT)**	**14-1**
	14.1	FORMATTING RK05 DISKS	14-1
	14.2	FORMATTING RX02 DISKETTES	14-2
CHAPTER	**15**	**DUMP AND FIX TECHNIQUE (DAFT)**	**15-1**
	15.1	DAFT COMPILING PROCEDURE	15-1
	15.2	DAFT OPERATING PROCEDURES	15-1
	15.3	DAFT COMMAND FILE	15-2
	15.4	DAFT COMMANDS	15-2

			Page
	15.4.1	Symbols Used in DAFT Commands	15-2
	15.4.2	DAFT Command Summary	15-3
	15.5	DAFT OUTPUT	15-5
	15.6	DAFT ERROR MESSAGES	15-7
CHAPTER	**16**	**REPORT PROGRAM GENERATOR (PRINT)**	**16-1**
	16.1	PRINT COMPILING PROCEDURE	16-1
	16.2	PRINT OPERATING PROCEDURES	16-2
	16.2.1	FILEX - Creation of Source File	16-2
	16.2.2	Compilation	16-3
	16.2.3	Program Execution	16-3
	16.3	PRINT COMMAND FILE	16-3
	16.3.1	IDENT Section	16-3
	16.3.2	HEAD1 and HEAD2 Section	16-4
	16.3.3	INPUT Section	16-4
	16.3.4	COMPUTE Section	16-5
	16.3.5	PRINT Section	16-6
	16.3.6	END Section	16-8
	16.4	PRINT ERROR MESSAGES	16-8
CHAPTER	**17**	**FLOWCHART GENERATOR PROGRAM (FLOW)**	**17-1**
	17.1	FLOW COMPILING PROCEDURE	17-1
	17.2	FLOW OPERATING PROCEDURES	17-1
	17.3	FLOW COMMANDS	17-2
	17.3.1	PROC Command	17-3
	17.3.2	DISK Command	17-3
	17.3.3	IF Command	17-4
	17.3.4	CALL Command	17-4
	17.3.5	START Command	17-5
	17.3.6	STOP Command	17-5
	17.3.7	GOTO Command	17-5
	17.3.8	CGOTO Command	17-6
	17.3.9	I/O Command	17-6
	17.3.10	TITLE Command	17-6
	17.3.11	SBTTL Command	17-7
	17.3.12	PAGE Command	17-7
	17.4	FLOW EXAMPLE	17-7
	17.5	FLOW ERROR MESSAGES	17-7
CHAPTER	**18**	**MENU PROGRAM (MENU)**	**18-1**
	18.1	MENU OPERATING PROCEDURES	18-1
	18.2	MENU COMMAND FILE	18-1
	18.2.1	Display Section	18-2
	18.2.2	Command Section	18-3
	18.2.3	Accept Section	18-3
	18.3	MENU ERROR MESSAGES	18-4

		Page
APPENDIX A	**COS-310 CHARACTER SET**	**A-1**
APPENDIX B	**COS-310 FILES**	**B-1**
B.1	COS-310 SOURCE FILES	B-1
B.2	COS-310 DATA FILES	B-1
B.3	COS-310 BINARY FILES	B-2
B.4	COS-310 SYSTEM FILES	B-2
B.5	SYSTEM DEVICE FORMAT	B-2
APPENDIX C	**ERROR MESSAGE INDEX**	**C-1**
APPENDIX D	**ADVANCED PROGRAMMING TECHNIQUES**	**D-1**
D.1	ACCEPT AND DISPLAY	D-1
D.1.1	Background Information	D-1
D.1.2	Interaction of ACCEPT and DISPLAY	D-1
D.1.3	Example Using ACCEPT and DISPLAY	D-2
D.1.4	Generalized ACCEPT Subroutines	D-2
D.1.4.1	Hardware Display Clear Feature	D-2
D.1.4.2	Clear Incorrect Data by Displaying Spaces	D-4
D.1.4.3	Other Desired Features	D-6
D.1.4.4	Escape Code Sequences as Terminators	D-8
D.2	DIRECT ACCESS TECHNIQUES	D-8
D.2.1	Background Information	D-8
D.2.2	The Reason for Direct Access	D-9
D.2.3	How the Direct Access Technique Works in DIBOL	D-9
D.2.4	Unsorted File	D-10
D.2.5	Sorted File	D-11
D.2.6	Rough Table, No Index File	D-12
D.2.7	Rough Table Plus Index File	D-13
D.2.8	Summary	D-14
D.2.9	Record Count	D-14
D.3	DIRECT ACCESS NOTES	D-15
D.3.1	XMIT Statements (Extending a File)	D-15
D.3.1.1	Truncating a File	D-15
D.3.1.2	Appending to a File	D-15
D.3.1.3	Rewriting a File	D-16
D.4	NUMERIC FIELD VERIFICATION	D-16
D.5	CHAIN STATEMENT NOTES	D-17
D.5.1	Interaction of CHAIN and INIT (channel, SYS)	D-17
D.5.2	Transferring Variable Values	D-17
D.5.3	Multiple CHAIN Entry Points	D-18
D.6	DIBOL PROGRAMMING OF SOURCE FILES	D-19
D.6.1	OPERATING PROCEDURES	D-19
D.6.2	Data Division	D-19
D.6.3	Procedure Division	D-20

		Page
D.7	CHECKDIGIT FORMULA	D-21
D.8	VT50/VT52 ESCAPE SEQUENCES	D-22
GLOSSARY		Glossary-1
INDEX		Index-1

FIGURES

1-1	Sample Source Program	1-1
10-1	Universal Diskette	10-3
10-2	Flowchart of FILEX OPT:C	10-10
B-1	Monitor Organization	B-3

TABLES

1-1	Source Program Limitations	1-3
1-2	Terminating Characters	1-5
1-3	Special Characters	1-17
2-1	Monitor Keyboard Commands	2-2
5-1	DIBOL Statement Words of Code Requirements	5-7
A-1	Characters Representing Negative Numbers	A-1
A-2	COS-310 Character Set	A-2

PREFACE

This is a reference document for those interested in applying the DIBOL language in a COS-310 system environment. Readers of this manual are assumed to possess a basic knowledge of programming and of the DIBOL language. Additional background may be obtained, however, by consulting the COS-310 New User's Guide (AA-D758A-TA).

This manual is constructed in a usable order, yet it is not intended that it be read sequentially. Each chapter and major section is constructed to be as informationally independent from other sections as possible. This method was followed to allow reference to specific information without the need for frequent cross-referencing.

In addition to the information in the chapters, additional reference material and summaries are provided in Appendices A through D and in the Glossary.

INTRODUCTION

THE DIBOL LANGUAGE

DIGITAL's Business Oriented Language (DIBOL) is a COBOL-like language used to write business application programs. The DIBOL language consists of data definition and procedure statements.

OVERVIEW OF THE COS-310 SYSTEM

The COS-310 system is designed for the small system user. It is a disk-based data processing system that is adaptable to a wide variety of business-related processing tasks.

The COS-310 system is enhanced with use of diskettes, video display terminals, and other DIGITAL hardware. Specially developed software has been included to act as tools for both application programmers and system operators.

Programs provided as part of COS-310 include but are not limited to the following:

- MONITOR/EDITOR controls the calling and operating of all other programs in the COS-310 system; provides the I/O control for the peripheral devices; and contains editing capabilities for correcting user programs. This program is referred to throughout the manual as the Monitor.
- SYSGEN builds and changes the system hardware configuration.
- DFU assigns logical units and displays or prints a logical unit table.
- COMP translates DIBOL language source statements into a binary program which can be run on the Datasystem 308 or 310.

- PIP transfers data, source, or binary files between two devices.
- SORT sequences records according to key characters or fields. Records may be sorted into ascending (0-9 or A-Z) or descending (Z-A or 9-0) sequence.

THE COS-310 FILE STRUCTURE

Four types of files are used in COS-310: data, source, binary, and system. Source files, compiled binary files, and system files can be saved in COS-310 directories. Data files cannot be saved in COS-310 directories.

- Data files are completely devoted to the storage of data to be processed by DIBOL or system programs.
- Source files contain control programs or DIBOL programs.
- Binary files are the output of the compiler and contain DIBOL programs translated into COS-310 interpretive code.
- System files include programs (MONITOR, SYSGEN, DFU, PIP, COMP, SORT, etc.) supplied as part of the COS-310 package.

MANUAL NOTATION CONVENTIONS

The following symbols, characters, and terms are used throughout this manual.

Symbols	Example	Explanation
Lowercase characters	PROC n	User determined information to be supplied.
Uppercase characters	NO TRACE	Words or characters to be entered exactly as shown.
...	RU CHAIN0+...CHAIN7	Optional continuation of arguments.
Characters in red	.RU SYSGEN/C	Your input to the system.

Symbols	Example	Explanation
␣	RU␣PIP	A space or blank.
{ }	START { /T /N /L }	Braces indicate that you must make a choice of one of the items enclosed.
[ ]	PROC [n] [/x]	Brackets indicate that you must decide whether or not to use an optional feature.
	RETURN	Information input via the keyboard must be followed by a RETURN. The RETURN code indicates that a line of information (input) is complete. No symbol for RETURN will be used in this text. It will be assumed that its use is understood.

COS-310 CHARACTERS

COS-310 characters include letters A-Z; numbers 0-9; and the special characters:

```
!   (   /   @   "   )   &
:   [   #   *   ;   \
$   +   <   ]   %   '
>   ,   .   ?   ␣   ↑
```

TERMS

Alphanumeric
Refers to the entire COS-310 character set. Initialized values in alphanumeric fields must be enclosed by single quotes.

Decimal, Octal
Refer to the numeric values associated specifically with base ten (decimal) and base eight (octal).

dev

Refers to a three-character designation for the device upon which data is located. The first two characters indicate the type of device; the third character indicates the drive the device is mounted on. The types of devices are listed below.

RX indicates RX01 diskettes.
DY indicates RX02 diskettes.
DK indicates RK05 disks.

Expressions

Refer to variables, constants, or arithmetic expressions made up of variables, constants, and the operators #, +, -, *, /.

Filnam,pronam,label,cmndfl

Identify names assigned to files, programs, statements, and input lines. These names may be of any length but only the first six characters are significant.

Label can only contain alphabetic and numeric characters, but must begin with an alphabetic character.

Pronam and cmndfl can contain any COS-310 character except /, +, ., -,␣, @, and ,.

It is advisable not to use -, /, @,␣, tab, and , in a data filnam. These characters may complicate or inhibit some program execution.

Logical Units

Refer to data file storage areas. These units are accessed via a logical unit table which is referenced by logical unit numbers.

Numeric

Refers to the characters 0-9 and combinations thereof.

CHAPTER 1

DIBOL LANGUAGE

This chapter provides reference information on the DIBOL language as used in the COS-310 system. If other basic information is desired, refer to the COS-310 New User's Guide.

1.1 SOURCE PROGRAM

A DIBOL source program (see Figure 1-1) consists of statements arranged in two divisions, Data and Procedure. Comments following a semicolon document the contents and purpose of each statement. Statement lines that begin with a semicolon contain only comments. Tab settings are used in the program to improve clarity and ease of reading.

The Data Division contains statements which define the type and size of the information to be used in the program, and optionally contain labels which identify the memory location where information can be referenced. Initial values for these statements can be included in this division.

The Procedure Division consists of statements which control program execution. An alphanumeric label can be assigned to a statement within the Procedure Division. A label must begin with a letter and can have a maximum of six significant characters. Labels precede and are separated from the statement by a comma. These labels are referenced in other statements.

Each statement within a DIBOL program begins on a numbered line. Line numbers are assigned manually or automatically when the program is being typed. DIBOL does not use these numbers, but error messages, editing functions, and debugging aids refer to them.

```
0100 START                                ;Optional compiler statement.
0110 RECORD INBUF                         ;Record named INBUF.
0120         STOCKN, D4                   ;Numeric field named STOCKN.
0130         DESC, A25                    ;Alphanumeric field named DESC.
0140         UCOST, D5                    ;Five-character numeric field.
0150         QORDER, D4                   ;Four-character numeric field.
0160         , D9                         ;Unreferenceable unnamed field.
0170 RECORD OUTBUF                        ;Record named OUTBUF.
0180         , D4                         ;Unnamed numeric field.
0190         , A25                        ;Twenty-five character field.
0200         , D5                         ;Unnamed field.
0210         , D4                         ;Temporary storage field.
0220         ECOST, D9                    ;Numeric field named ECOST.
0230 RECORD                               ;Unnamed record-temporary storage
0240                                      ;cannot be directly referenced.
0250         TITLE, A6, 'OVRHED'          ;Field initialized to 'OVRHED'.
0260 PROC                                 ;Beginning of Procedure Division.
0270         INIT(1,I,TITLE)              ;Input 'OVRHED'  on channel 1.
0280         INIT(2,O,'OUTPUT')           ;'OUTPUT' on channel 2-output.
0290 LOOP,   XMIT(1,INBUF,EOF)            ;Transfers INBUF to EOF.
0300         OUTBUF=INBUF                 ;INBUF moved to OUTBUF.
0310         IF(STOCKN.LT.1000) GO TO LOOP   ;Conditional statement.
0320         ECOST=UCOST*QORDER           ;UCOST times QORDER moved to
0330                                      ;ECOST.
0340         XMIT(2,OUTBUF)               ;Transfer OUTBUF onto channel 2.
0350                                      ;Blank line.
0360         GO TO LOOP                   ;Branch control to LOOP.
0370 EOF,    FINI (2)                     ;Identifies end of logical unit.
0380         FINI (1)                     ;Writes record and closes file.
0390         STOP                         ;Stops program execution.
0400 END                                  ;Marks the end of the program.
```

Figure 1-1 Sample Source Program

Table 1-1
Source Program Limitations

Maximum characters	about 8,000 per file
Maximum number of source files per program	7
Maximum number of symbols 16K byte system/24K byte system or larger	365/511
Line numbers available	0-4095
Maximum characters per line	120

1.2 STATEMENTS

There are six kinds of statements in DIBOL:

1. Compiler statements (START, PROC, and END) label the beginning, division, and end of the program. These three statements are non-executable. START and END are optional and PROC marks the end of the Data Division and the beginning of the Procedure Division.

2. Data definition statements (RECORD) describe the type and size of data to be stored. Must include field information.

3. Data manipulation statements and INCR control the movement of data within memory.

4. Control statements (CALL, CHAIN, GO TO, IF, ON ERROR, RETURN, STOP, TRAP) effect the order of program statement execution.

5. Input/Output statements (ACCEPT, DISPLAY, READ, WRITE, and XMIT) control data movement within memory or between memory and peripheral devices. INIT and FINI associate and disassociate channel numbers used by the program. The FORMS statement controls line spacing and paging on the printer.

6. Debugging statements (TRACE, NO TRACE) permit statement-by-statement following of program execution.

These statements are discussed in alphabetical order rather than by kind or by order of use.

1.2.1 ACCEPT - Input/Output Statement

The ACCEPT statement takes input from the keyboard, stores it in a specified alphanumeric field, and stores the decimal value of the terminating character (the last key typed) in a decimal field (see Table 1-2 Terminating Characters). The terminating value can be used in statements later in the DIBOL program.

ACCEPT is primarily used with the DISPLAY statement and has the form:

```
ACCEPT (dfield, afield)
```

where:

dfield — is the name of a numeric field where the decimal value of the terminating character is to be stored.

afield — is the name of an alphanumeric field where the keyboard input is to be stored.

When an ACCEPT statement is encountered, the system waits for input from the keyboard. Program execution continues either after afield is full or after a terminating character is typed. When ACCEPT is terminated before afield is full, the remaining character positions in the afield remain unchanged. It is desirable to clear the afield before an ACCEPT statement is executed.

Example:

```
ACCEPT (A,B)            ;Stores input from the keyboard in B and
                        ;stores the value of terminating the char-
                        ;acter in A.

ACCEPT (A(3),B(4,5))    ;Stores input from the keyboard in the 4th
                        ;and 5th characters of B, and stores the
                        ;terminating value in the 3rd element of
                        ;array A.
```

Table 1-2
Terminating Characters

Decimal Value	Last Key Typed
00	Null or end-of-field
01	CTRL/A
02	CTRL/B
04	CTRL/D
05	CTRL/E
06	CTRL/F
07	CTRL/G
08	BACKSPACE or CTRL/H
09	TAB
10	LINE FEED
11	CTRL/K
12	CTRL/L
13	RETURN or ENTER
14	CTRL/N
16	CTRL/P
18	CTRL/R
20	CTRL/T
21	CTRL/U
22	CTRL/V
23	CTRL/W
24	CTRL/X
25	CTRL/Y
26	CTRL/Z
27	ESC
63	DELETE

1.2.2 CALL - Control Statement

The CALL statement branches program control to an internal subroutine. CALL has the form:

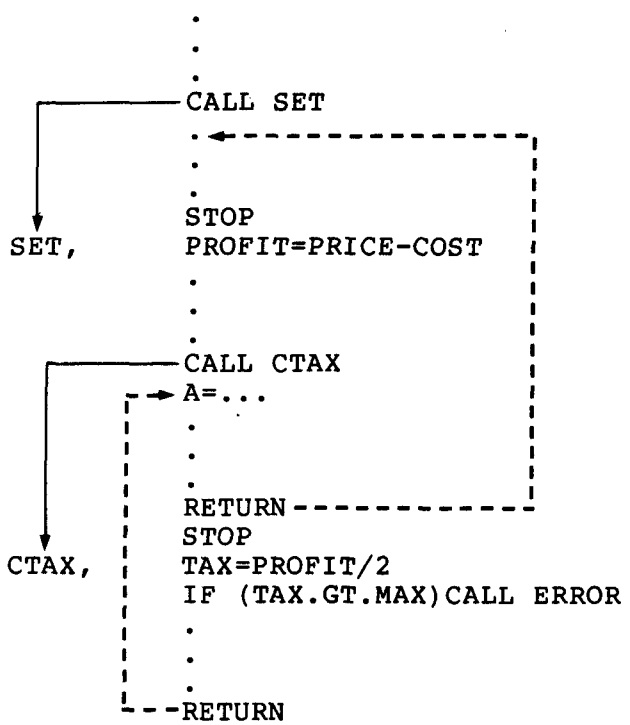
```
CALL label
```

where:

label — is the label of the first statement of a subroutine in the Procedure Division of the program.

The CALL statement saves the return location in a pushdown stack so that program control will return to that location when the subroutine has been executed. Additional subroutine CALL statements may be nested within a subroutine to a depth of 50.

Example:



This example shows how execution control branches to and returns from one subroutine to the next. The solid lines show the result of CALL statements while the broken lines show the result of RETURN statements.

1.2.3 CHAIN - Control Statement

The CHAIN statement allows a DIBOL program which has exceeded the available user memory to be separated into two or more smaller DIBOL programs which are executed sequentially. Each program is written and compiled separately. CHAIN programs are executed beginning with the statement immediately following the PROC statement.

The form of the CHAIN statement is:

```
CHAIN n
```

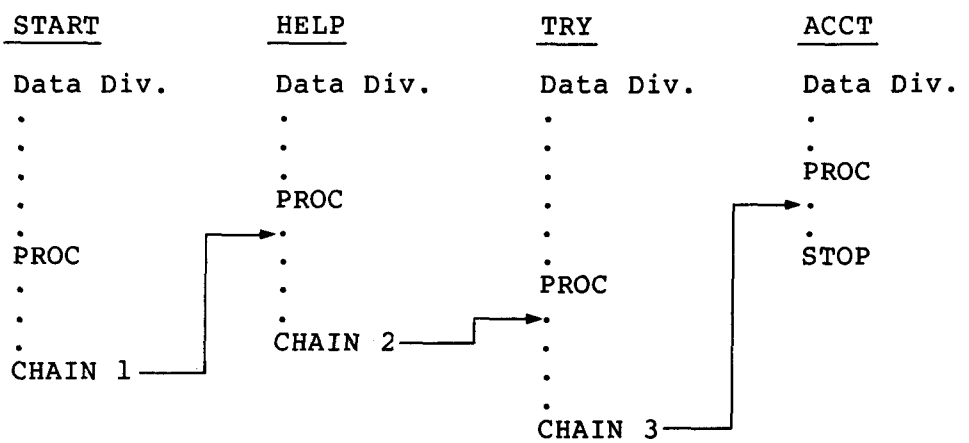
where:

n — is a numeric variable (0-7) representing the sequence number of DIBOL binary programs as specified in the RUN command. A maximum of seven programs can be chained together.

When the CHAIN statement is encountered in a DIBOL program, execution of the current program is stopped, and the indicated CHAIN program is loaded. All CHAIN programs must be properly declared in a RUN command. The program loaded by CHAIN will not return to the place where the initial program was stopped. A return to a program requires a CHAIN statement to that program.

Example:

```
.RUN START+HELP+TRY+ACCT
```



In this example START is chain 0, HELP is chain 1, TRY is chain 2, and ACCT is chain 3. Although this example executes the program sequentially, the programs could be chained in any order. A program could chain back into itself.

All file status information is destroyed between chained programs. Therefore, all output and update files should be closed with a FINI statement before executing a CHAIN to prevent the loss of information.

Both the TRACE and TRAP features are turned off when a CHAIN statement is executed. They may be turned on again in the CHAIN program by using an appropriate TRACE or TRAP statement. If DDT is being used when a CHAIN statement is executed, control returns to DDT.

Any DIBOL record in a program loaded by the RUN command is automatically cleared. However, if the record is in a program loaded by the CHAIN statement, the record retains whatever contents it had in the previous program unless the clear option (,C) is specified for the record.

Executing a CHAIN statement with an argument which does not correspond to a valid DIBOL binary program in the RUN command results in the error message ILLEGAL CHAIN.

1.2.4 Data Manipulation Statements

Data manipulation statements convert data from numeric to alphanumeric and vice versa, calculate arithmetic expressions, clear fields and records, move data between fields or records, and format data. The contents of a source area are stored in a destination area. Record names are used in data manipulation statements only when data is moved or cleared. The form of the statement is:

```
destination = source
```

where:

source — is a variable, literal, or an expression identifying data which is to be manipulated and then copied into a destination area.

destination — is the area where the manipulated data is stored.

The destination area must be defined in the Data Division. The source is always converted and justified to the data type defined for the destination area. Data in the source remains unchanged; the destination area is always altered.

Variables:

A quantity that can be assigned any of a given set of values. The following three formats are used with variables:

```
name
name (subscript)
name (position m, position n)
```

where:

name — is a label (maximum of six characters) used to identify a record or field.

name(subscript) — represents a subscripted array; the subscript must be a numeric or an arithmetic expression whose value is between 1 and the dimension specified for the array in the Data Division. If name does not identify an array, or the value of the subscript exceeds the dimension of that array, other locations in memory are referenced. No error message is generated unless locations are referenced outside of the Data Division.

name (position m, position n)

This form of subscripting references those characters from position m to position n inclusive. Position n must be greater than or equal to position m; position n should be less than or equal to the dimension of the array associated with name. Positions m and n must be numeric characters with a value of 1 or greater. If the variable name is subscripted, the successive array elements should be considered strung out left to right.

Example:

```
Data Division defines A as 4D4

A(3,9)     ;Wants 7 digits, 2 from A(1), all of
           ;A(2), and 1 from A(3).
```

Literals:

Numeric literals consist of a sequence of from 1 to 15 digits.

Alphanumeric literals consist of a sequence of COS-310 characters (except single quotes) enclosed in single quotes.

A RECORD literal is used anywhere in the Procedure Division where a record is allowed for XMIT (on channels opened for O, T, or L by an INIT statement), for WRITE, or as the source in a data manipulation statement. It is similar to an alphanumeric literal except it begins with a double quote (") and ends with a single quote (').

Example:

```
        PROC
XMIT (8,"HELLO')        ;HELLO is a RECORD literal.
```

Example:

```
RECORD REC
        FLD1, A5
            , A1
        FLD2, A5
PROC
        FLD1 = 'HELLO'      ;HELLO is an alphanumeric literal.
        FLD2 = 'THERE'      ;THERE is an alphanumeric literal.
        XMIT (8,REC)
```

Expressions:

An ordered set of characters treated in its totality as a symbol for one idea or value.

1.2.4.1 Data Conversion

Numeric values can be converted to alphanumeric and vice versa for the purposes of input, output, calculations, and verification of numeric data. This is done with the data manipulation statement:

```
destination = source
```

Alphanumeric to numeric conversions are right-justified, and spaces are replaced by zeros. Numeric to alphanumeric conversions are right-justified, and zeros are replaced by spaces. If the numeric expression equals zero, the alphanumeric field will contain only a zero.

```
alphanumeric destination = alphanumeric source   ;left-justified
alphanumeric destination = numeric source        ;right-justified
numeric destination      = numeric source        ;right-justified
numeric destination      = alphanumeric source   ;right-justified
```

Alphanumeric values to be converted to numeric must be 15 or fewer characters in length. If the alphanumeric field contains characters other than digits, spaces, and the signs + or -, the message BAD DIGIT results at run time. The data conversion statement should be preceded by an ON ERROR statement if the contents of the alphanumeric source may contain bad digits.

Data conversion is used to edit and verify numeric data following an ACCEPT statement. Data is typed into an alphanumeric field and is then converted to a numeric field by a data manipulation statement preceded by an ON ERROR statement. Spaces and signs are not counted as characters that are moved. Spaces in the alphanumeric field are ignored. Signs may be imbedded anywhere in the alphanumeric field. If the ON ERROR statement is executed, the data entered was not numeric.

1.2.4.2 Arithmetic Expressions

Arithmetic expressions are allowed only as the source in a data manipulation statement. The expression can contain numeric elements, subscripted data elements, constants, variables, and arithmetic operators.

There are five binary arithmetic operators which are executed in order of priority. Operators with the same priority are executed left to right.

```
# (rounding)
/ (division)
* (multiplication)
+ (addition)
- (subtraction)
```

Order of priority:

1. rounding
2. multiplication and division
3. addition and subtraction

These operators are used with numeric values.

The signs -, +, # can be used in COS-310 as binary or as unary operators. As binary operators they indicate the mathematical operation to be performed in an expression. As unary operators, the + has no effect; the unary - negates the numeric value to which it is affixed.

The operator # can be used to convert an alphanumeric character to its equivalent decimal code. This code can then be used by the program. In this application, # appears before the character. For example:

```
A = #B          ;A is a numeric field and B is an alphanumeric
                ;field.  If B contains the characters XYZ, X is
                ;converted to its internal code, and the decimal
                ;equivalent, 57, is stored in A.

A = 3 + #B      ;A is a numeric field and B is an alphanumeric field
                ;containing XYZ.  After conversion, the decimal
                ;equivalent of the first character of B, 57, is
                ;added to 3 and stored in A.
```

Since expressions in parentheses are executed first, the order of priority can be altered with the use of parentheses. In the following expressions, A=10.

```
F1= 100*A/2+3-1     = 502

F1= 100* (A/2+3-1) = 700
```

Rounding:

Rounding sets variables to specified character formats and increments by 1 the least significant digit if the digit to the right before formatting was 5 or more (sign is unchanged). The sign # appears after the character being rounded. The format for rounding is:

```
destination = A#B
```

A is rounded by B places; B is not greater than 7 (becomes modulo 8) and is treated as a positive integer. The sign # appears after the character that is being rounded.

```
TEMP=MONEY#2

     If MONEY was 123456, TEMP becomes 1235.

     If MONEY was 123446, TEMP becomes 1234.

     If MONEY was -1473, TEMP becomes -15.
```

Typically, this feature is used for rounding to the dollar.

Division:

The result of a division operation is expressed in unrounded whole numbers.

The result of 5/3 is 1 and -14/5 is -2. An error message occurs when division by zero is attempted.

Multiplication, Addition, Subtraction:

These are basic arithmetic operations that will execute as requested. If the resulting value (or an intermediate result) exceeds the size of the destination field, the leftmost digits are dropped without an error message being displayed.

1.2.4.3 Clearing Fields and Records

Data manipulation statements clear fields and records when used in the form:

```
destination =
```

The destination can be a name designating a single field, a record, or an array. Alphanumeric destinations are cleared to all spaces; numeric destinations are cleared to all zeros. Any part of a field can be accessed in a program statement by subscripting the beginning and ending positions of the character string. An array name without any subscripts clears only the first element in the array.

When a record is cleared, all numeric and alphanumeric fields are set to spaces.

Examples:

```
F1(5,7)=   ;Clears characters 5, 6, and 7 in field F1.

A1(5)=     ;Clears the fifth element in an array.

A1(A)=     ;Clears element A in array A1.

F1(1,1)=   ;Clears the first character in F1.

A1=        ;Clears the first element in array A1.

RECNAM=    ;Clears RECNAM to all spaces.
```

Record names can be subscripted to allow reference to record areas as though they were in an array. All records to be so referenced must follow one another and be of the same length.

Example:

```
RECORD CUSNAM
   ,A3
RECORD BYRNAM
   ,A3
PROC
    .
    .
    .
RECORD(2)=      ;Clears RECORD BYRNAM.
```

1.2.4.4 Moving Alphanumeric Data

Use the data manipulation statement to move the contents of one alphanumeric field (source) to another alphanumeric field (destination).

```
destination = source
```

If source is shorter than destination, data from source is left-justified, and the rightmost characters of destination are undisturbed. If source is longer than destination, the rightmost characters in source are not moved into destination. The source remains unchanged.

Example:

```
RECORD ALPHA
    A,A5,'ABCDE'
    B,A3,'FGH'
RECORD NAMES
    NAME,A4,'FRED'
    NAME1,A7,'JOHNSON'
PROC
    A=B                 ;Field A would contain FGHDE.
    NAME=NAME1          ;NAME would contain JOHN.
```

1.2.4.5 Moving Numeric Data

Use the data manipulation statement to move the contents of one numeric field to another numeric field.

```
destination = source
```

If source is shorter than destination, zeros are inserted on the left. If source is longer, the most significant digits are not moved to destination.

Example:

```
RECORD A
     FIGR, D3, 123
     FIGR1,D5, 45678
RECORD B
     NUMB1,D5, 45678
     NUMB2,D3, 123
PROC
     FIGR1 = FIGR   ;FIGR1 would contain 00123.
     NUMB2 = NUMB1  ;NUMB2 would contain 678.
```

1.2.4.6 Moving Records

Movement of entire records can be accomplished with data manipulation statements. All fields within a record are treated collectively as alphanumeric fields during record manipulation. The manipulation statement has the form:

```
destination = source
```

where:

destination
: is a record label or a subscripted record label.

source
: is a record label, a subscripted record label, or a record literal.

This data manipulation statement moves the contents of the source into the space reserved in the destination. If the source is shorter than the destination, the rightmost characters of destination are undisturbed. If source is longer, the rightmost characters of source will not be moved into the destination.

Example:

```
RECORD PRTREC
         ,A92
RECORD DATA
    NAME , A25
         , A5
    ADDR , A20
         , A5
    CITY , A20
         , A5
    STATE, A2
         , A5
    ZIP  , A5
PROC
    PRTREC=DATA     ;Move DATA into PRTREC.
    XMIT (8,PRTREC)
```

1.2.4.7 Data Formatting

Any numeric data field can be formatted into an alphanumeric field to contain spaces and punctuation marks which are not stored with the records on disk, and which cannot be present during arithmetic calculations. This is done with the following data manipulation statement:

```
Alphanumeric variable = numeric expression, format
```

Format specifies special characters to be inserted with the numeric expression.

Example:

```
A = D, '-XXX,XXX.ZZ'
```

The eight-digit numeric at D is converted to alphanumeric code, reformatted with specified punctuation, and stored in alphanumeric field A.

The format string must be an alphanumeric expression. Most characters on the printer or the keyboard can be used in a format string, but use the following special characters with care: X, Z, *, -, ., '. Table 1-3 explains the special use of these characters.

Examples:

AMT is an alphanumeric field the same size as the associated format string:

```
AMT=123,'XXXXXX'          ;AMT contains       123
AMT=123,'ZZZZZZ'          ;AMT contains       123
AMT=123,'*XXXXX'          ;AMT contains    ***123
AMT=-1123,'-XXX,XXX'      ;AMT contains  -  1,123
AMT=123,'$*XXX.XX-'       ;AMT contains $***1.23-
AMT=123456,'-XX.XX'       ;AMT contains   -12.34
```

Each comma, period, slash, minus sign, or any other special notation, must be counted as a character position. In the AMT=123,'$*XXX.XX-' example, AMT must be defined in a RECORD statement as a nine-character alphanumeric field.

Table 1-3
Special Characters

Character	Explanation
X	Used in a format to arrange a numeric field for printout. Each X represents a digit and leading zeros are automatically suppressed.
Z	Used to suppress leading zeros when formatting output.
*	Used in a format string to replace leading zeros and eliminate trailing spaces on printout. If the * is anyplace except the first character in the string, digits may also be replaced.
-	Inserts an arithmetic sign in a number to be printed. The sign can be placed before or after the number. If the number is positive, a space is substituted for the minus sign. If the minus sign is placed in a position following the first significant digit but previous to the last position of a format string, it is printed like any other insertion character.
.	Inserts a decimal point in a format string and forces zeros to the right of the decimal point to be significant.
,	Used to insert a comma in a format string if there are significant digits to the left.

All other COS-310 characters are treated as unconditional insertion characters.

1.2.5 DISPLAY - Input/Output Statement

DISPLAY is used to show messages on the screen and to move the screen cursor to a specified line and character position. Numeric fields are used only for special effects.

The form of the DISPLAY statement is:

```
DISPLAY (y,x, { literal } )
              { afield  }
              { dfield  }
```

where:

y — is a numeric expression representing the screen line number. If the specified line number is greater than the number of lines on a screen, the cursor is moved to the last line on the screen.

A statement with y equal to 0 outputs a message beginning at the present location of the cursor. If y is zero, no positioning is done and x is ignored.

x — is a numeric expression representing the character position. If the specified character position is greater than the width of the screen, the results are unpredictable.

literal — is an alphanumeric string or a numeric character string. An alphanumeric string must be enclosed in single quotes (') and is displayed at the character position specified. There is no carriage return/line feed after the message; the cursor remains at the character position at the end of the message.

afield — is an alphanumeric field containing a message to be displayed.

dfield — is a special numeric code that causes a particular operation to occur.

The following numerics are recognized as special codes by the COS-310 system.

0	Position cursor.
1	Clear to end-of-screen.
2	Clear to end-of-line.
7	Sound terminal alarm.

If a number is to be displayed, the numeric field must be converted to an alphanumeric field before it can be displayed on the screen.

Examples:

```
DISPLAY(0,0,7)          ;Sound terminal alarm.

DISPLAY(10,1,1)         ;Clear from line 10, character position 1
                        ;to end-of-screen.

DISPLAY(2,20,DAY)       ;Beginning on line 2, character position 2
                        ;display the contents of DAY.

DISPLAY(1,10,0)         ;Position cursor at first line,
                        ;10th character position.

DISPLAY(11,37,2)        ;Clears line 11 from character 37 to the end
                        ;of the line.

DISPLAY(0,0,'DATE')     ;Starting at the current cursor position,
                        ;display the word DATE.

DISPLAY(11,12,7)        ;Position cursor at line 11, character
                        ;position 12 and sound terminal alarm.

DISPLAY(Y,X,ALARM)      ;If the Data Division contains:
                             RECORD
                             Y,D2,20
                             X,D2,36
                             ALARM, D1,7
                        ;the cursor is set at line 20,
                        ;character position 36 and an
                        ;audible alarm is sounded.

DISPLAY(1,1,'HELLO')    ;Display HELLO in the upper left-hand corner
                        ;(line 1, character 1) of the screen.
```

1.2.6 END - Compiler Statement

This optional statement is the last statement of a program. The statement has the form:

```
END[/x]
```

where:

/x is one of the following option switches.

- /N suppresses the printing of a compiler source program storage map listing.
- /T displays a compiler source program storage map listing on the screen.

If no options are specified, a compiler source program storage map listing is printed as usual.

An END statement option switch or lack of it can be overridden with an option switch (/N, /G, /T) in the compiler RUN command. If no storage map listing is printed, the label count and number of free locations are not printed.

1.2.7 FINI - Input/Output Statement

A FINI statement disassociates the channel number from the mode as specified in an INIT statement. FINI is only necessary for mass storage output and update files, but it is good programming practice to close each data file opened. If an output or an update file is not closed, records may be lost.

The FINI statement has the form:

```
FINI  (channel)
```

where:

channel — is a numeric expression (1-15) which specifies a channel number which was associated with a mode by an INIT statement.

The following information is useful in determining the effect of a FINI statement on files associated for various uses.

INIT mode	Effect of FINI Statement
I	Reading of the file stops. File may be reopened and read from the beginning. Disassociates channel number.
O	An end-of-file (EOF) mark is written, the file is closed, and the length of the file is written in the directory. Disassociates channel number.
U	Reading/writing of the file stops. Disassociates channel number.
K,T,L	Disassociates channel number.
S	Reading of the file stops. Cannot be reopened. Disassociates channel number.

Examples:

```
FINI (1)     ;Disassociates channel 1 from a device and file.

FINI (A+B)   ;Uses the sum of A+B as a channel number, and
             ;disassociates that channel from a device and file.
```

1.2.8 FORMS - Input/Output Statement

The FORMS statement is used to format printer output and has the form:

```
FORMS  (channel,skip-code)
```

where:

channel is a numeric expression (1-15) associated with the printer in a previous INIT statement. If the channel specified is not associated with a printer, the statement is ignored.

skip-code is a numeric expression which causes the printer to go to the top of the page (0) or to skip the number of lines specified (1 to 4095).

Negative numbers cause unpredictable results.

If the code exceeds 4095, then 4096 is subtracted from the code and the remainder is used as the skip-code.

Example:

```
INIT(1,L)
.
.
.
FORMS(1,3)      ;1 is the channel number specified in a previous
                ;INIT statement and 3 is the number of lines to be
                ;left blank.
```

Example:

```
INIT(5,L)
.
.
.
FORMS(5,0)      ;5 is the channel number and 0 sends the listing
                ;to the top of the printer page.
```

1.2.9 GO TO - Control Statement

The GO TO statement branches program control to a line in the program identified by the label. The GO TO statement has two forms: unconditional and computed.

1.2.9.1 Unconditional GO TO

The form of the unconditional GO TO is:

```
GO TO label
```

where:

label — is the label assigned to a statement line in the Procedure Division where control is to be transferred.

Example:

```
GO TO SET      ;Transfers control to SET.
```

1.2.9.2 Computed GO TO

The computed GO TO has the form:

```
GO TO (label1...,labeln),variable
```

where:

label1...,labeln — are statement labels. There can be any number of labels up to the limit that can be stored on one line (120 characters).

variable — is a decimal variable or expression representing a value. This value identifies the label where control is to branch.

Control is branched to the label corresponding to the sequence number indicated by the variable. If the variable is negative, zero, or greater than the number of labels, control passes to the next statement in the program.

Example:

```
GO TO (LOOP,LIST,TOT),KEY  ;Transfer control to LOOP if KEY=1,
                           ;LIST if KEY=2, or TOT if KEY=3.
```

1.2.10 IF - Control Statement

An IF statement conditionally executes certain statements on the basis of the result of a relational comparison between expressions. The form of the statement is:

```
IF (expression1.rel.expression2)statement
```

where:

expression1 and expression2
: are literals, variables, or arithmetic expressions of the same type.

.rel.
: is one of the following relational operators:

.EQ.	Equal
.NE.	Not equal
.LT.	Less than
.LE.	Less than or equal
.GT.	Greater than
.GE.	Greater than or equal

statement
: is one of the following control statements which is executed if the relationship is true.

GO TO label	STOP
CALL label	TRACE
RETURN	NO TRACE
ON ERROR label	

Expression1 and expression2 must be of the same data type: both numeric or both alphanumeric. In a numeric comparison, the shorter field is internally filled to the length of the longer field, then the comparison is made between the longer field and the zero-filled field. In an alphanumeric comparison, the comparison is made on the number of characters in the shorter field.

If the result of the comparison is not true, the next statement in program sequence is executed.

Examples:

```
IF (A.EQ.B) GO TO LABEL3          ;If A is equal to B, control
                                  ;is transferred to LABEL3.

IF (SLOT.NE.2) CALL BAD           ;If SLOT is not equal to 2,
                                  ;control is transferred to
                                  ;BAD.

IF(SALES.LT.PROFIT+TAX-RENT) NO TRACE
                                  ;If SALES is less than PROFIT
                                  ;plus TAX minus RENT, the
                                  ;TRACE command will terminate.
```

1.2.11 INCR

The INCR (increment) statement adds 1 to the specified numeric variable and has the form:

```
INCR variable
```

where:

variable is a numeric variable to be incremented by 1.

INCR should only be used with positive numbers and is typically used to add one to a counter. Its use is faster than a data manipulation statement.

Example:

```
INCR A2     ;Add 1 to A2.  This is identical in meaning to
            ;the data manipulation statement A2=A2+1.
```

1.2.12 INIT - Input/Output Statement

The INIT statement associates a channel number with a logical unit on a mass storage device or a character-oriented input/output device and initializes the device. The form of the INIT statement is:

```
INIT (channel,mode[,filnam][,logical unit #])
```

where:

channel is a numeric expression which specifies a channel number (1-15) to be associated with a logical unit or character-oriented device. This channel number is used in the program to refer to the associated device.

If the number specified exceeds 15, it is interpreted modulo 16; 16 is subtracted from the number specified and the remainder is used by the system.

The following channels are initially associated with the specified devices at program startup. These assignments can be changed with an INIT statement.

6 is the channel number for the printer.
7 is the channel number for the keyboard.
8 is the channel number for the screen.

A channel that is associated with a logical unit on a mass storage device must be closed by a FINI statement prior to another INIT. Opening of an output file causes the previous contents of the file to be deleted.

mode is the one-character mode designation of a device to be associated with the channel number. The mode designations are:

Mode	Meaning	Explanation
I	IN	Mass storage device (logical unit) to be used for input.
O	OUT	Mass storage device (logical unit) to be used for output.
U	UPDATE	Mass storage device (logical unit) to be used for direct access both input and output.

(Continued on next page)

Mode	Meaning	Explanation
K	KBD	Input from keyboard.
T	TTY	Output to screen.
L	LPT	Output to printer.
S	SYS	Input from a source file located on the system device. This file name must have been specified in a RUN command.

filnam is an alphanumeric literal or variable that identifies the data file on the logical unit. A data file name is necessary with the I, O, and U modes, and is illegal with other modes.

If the file name is not present for an input, a MOUNT message is displayed. On output, if another file is already located on the designated logical unit, REPLACE? is displayed.

A temporary file name such as a file name beginning with $ may be used. When this file name is used, no REPLACE? is generated because the program recognizes this file name as temporary and replaces it.

logical unit #
is an optional numeric expression which specifies the logical unit (1-15) where the data file is stored or is to be stored.

If the number specified exceeds 15, then 16 is subtracted from it and the remainder is used by the system.

It is good programming practice to specify the logical unit number and the data file name. Data file names should be unique so that REPLACE? messages are avoided.

Example:

```
INIT (15,I,'RENEW',6)  ;Initializes channel number 15 for input.
                       ;RENEW is the name of the input file found
                       ;on logical unit 6.
```

1.2.13 ON ERROR - Control Statement

The ON ERROR statement branches program control to a specified statement when a nonfatal executed error occurs in the statement following the ON ERROR statement. ON ERROR can be written into the source program immediately prior to a statement where a possible error might occur. The form of this statement is:

```
ON ERROR label
```

where:

label — is a label assigned to a statement in the Procedure Division where control is to be transferred.

Example:

```
ON ERROR TRAP          ;Statement which would branch program
                       ;control to TRAP if DEC =ALPHA creates a
                       ;nonfatal error.
DEC = ALPHA
```

The ON ERROR statement prevents a return to the Monitor for the following run-time errors.

Message	Explanation
BAD DIGIT	A non-numeric digit is used in an alphanumeric-to-numeric conversion.
END OF FILE	An end-of-file label was not specified in an XMIT statement.
ILLEGAL RECORD	Record number is too large or 0, or length specified in the record header word does not match the length of the XMIT record.
LINE TOO LONG	Input line overflowed the record into which it was read.
NO FILE	No file specified in RUN command to satisfy INIT (SYS) statement.
NUMBER TOO LONG	A field of more than 15 digits is used in a calculation.
ZERO DIVISOR	Division by zero attempted.

1.2.14 PROC - Compiler Statement

The PROC statement separates the Data and Procedure Divisions of a DIBOL program. It is of the form:

```
PROC [n][/x][;comment]
```

where:

n — is a single digit, 0-7 (not an expression), indicating the maximum number of logical units which the program will have open simultaneously. If no number is specified, the compiler assumes 7. The available memory is divided into buffers to handle the number of logical units specified. The more buffers specified, the smaller they must be. Smaller buffers generally result in slower sequential data file processing and faster random data file processing.

If the program opens more logical units than were specified in the PROC statement, a run-time error occurs.

/x — is one of the following option switches:

/N suppresses compiler listing of source program.

/L lists source program and errors on the printer.

/T displays source program and errors on the screen.

The PROC option switch is active until disabled by a START or END statement option switch. If no option switch is specified, a compiler listing is printed.

A /N in the RUN COMP command overrides /L's or /T's in the program; all printout except errors is suppressed.

comment — is an optional string of text preceded by a semicolon which is stored for output as a heading for the Procedure Division of the compiler listing. When the compiler encounters the PROC statement, the printer moves to the top of the next page and outputs the comment as a header line.

Examples:

```
PROC                        ;Printer will go to top-of-the-page.

PROC 4/L;  TEST PROG.       ;Four mass storage logical units will
                            ;be open, source program listing and
                            ;errors will be listed on the printer,
                            ;the page heading will be TEST PROG.
```

1.2.15 READ - Input/Output Statement

The direct access READ statement allows a specified data record to be moved from a named file to a specified area in memory. It has the form:

```
READ (channel,record,rec#)
```

where:

channel — is a numeric expression with a value of 1-15 specifying a channel number which links the READ statement to the related INIT statement. The INIT statement must specify Input or Update as the mode.

record — is the name of the record into which data is to be read.

rec# — is a numeric or arithmetic expression specifying the number of the record to be read.

If the program reads past the end-of-file mark, the results are unpredictable (see Section 1.2.22 for restrictions on READ and WRITE usage.)

Examples:

```
READ (5,REX,88)      ;Reads the 88th record of the device linked
                     ;to the channel which was opened with the
                     ;INIT (5,...) statement and places it in the
                     ;memory area labeled REX.

READ (6,BLT,EXPR)    ;Reads the record specified by the
                     ;expression EXPR and stores it in the memory
                     ;area labeled BLT.
```

1.2.16 RECORD - Data Definition Statement

The RECORD statement reserves areas of memory where data is stored during processing. A RECORD statement without field statement information is of no use in the program. The total size of the data fields within a named record cannot exceed 510 characters. An unnamed record can contain no more than 4094 characters. Only named records can be used in an I/O operation. Without a name, the record can act only as a temporary storage area. The RECORD statement has the form:

RECORD [name][{ ,X / ,C }]

where:

name — names the record, begins with an alphabetic letter, contains a maximum of six significant characters, and is unique within the program. The name is optional unless the record is to be referenced for data transfer.

,X — allows one record to be overlayed into the same area as another. The X must be preceded by a comma. More than one overlay may define the same record, but the overlaying record must be equal to or smaller than the record being overlayed. A series of overlaying records must be preceded by a record without a ,X. Overlaying is useful in reformatting a previously defined data record area.

,C — clears the contents of a record loaded by a CHAIN command; all numeric fields are set to zeros and all alphanumeric fields are set to blanks if no initial value has been specified. Do not use ,X and ,C in the same record; no error will occur, but the program will ignore the ,C.

Accompanying field information is of the form:

[fldnam], [m]xn[,initial value][{ ,D / ,P / ,S }]

where:

fldnam, — is an optional name that identifies the file. A comma can be used without the fldnam if the program does not reference the individual field.

m is an optional repetition count character used to indicate an array of values that can be referenced with a single field name. The values must be of the same type and size and can be initially entered as a continuous string separated by commas following the type and size designation. Not all values of an array need to be inserted at the program's inception.

x indicates the field type as either alphanumeric A or numeric D.

n indicates the number of characters (maximum 15 numeric, 510 alphanumeric) in the field.

,initial value
an optional value initially inserted in the field. Must agree in type and size with the xn designation.

,D an optional switch which calls the current date from the Monitor and inserts it at the location designated by the data field information. Cannot be used in the same field as ,P.

,P an optional switch which, when the program is ready for execution, asks for the insertion of information via the keyboard. Cannot be used in the same field as a ,D. Enter all information as alphanumeric.

,S an optional character used to assign the value of a variable equal to the options used at run time.

Example:

```
RECORD INVENT              ;Record named INVENT.
     THINK0, D5            ;Field named THINK0 with five numeric
                           ;characters.
     THINK1, 4D7           ;Array named THINK1 with four
                           ;7-character values.
     STKOPT, A6, 'OPTION';Field named STKOPT initialized to
                           ;OPTION.
     BNKBAL, D7,1234567    ;Numeric field initialized to 1234567.
     TRNSDT, D6, D         ;Field to enter date from the Monitor.
     INPUT, A10, P         ;Field to allow entry to ten characters
                           ;from keyboard.
     RUNSW, A2, S          ;Field to allow entry of /xx values
                           ;from R PROG /xx.
RECORD, C                  ;Clears record loaded by CHAIN command.
     NAME, A20             ;Alphanumeric field with 20 characters.
RECORD, X                  ;Unnamed record to overlay preceding
                           ;record.
     LNAME, A10            ;Alphanumeric field with 10 characters.
     FNAME, A10            ;Alphanumeric field named FNAME.
```

1.2.17 RETURN - Control Statement

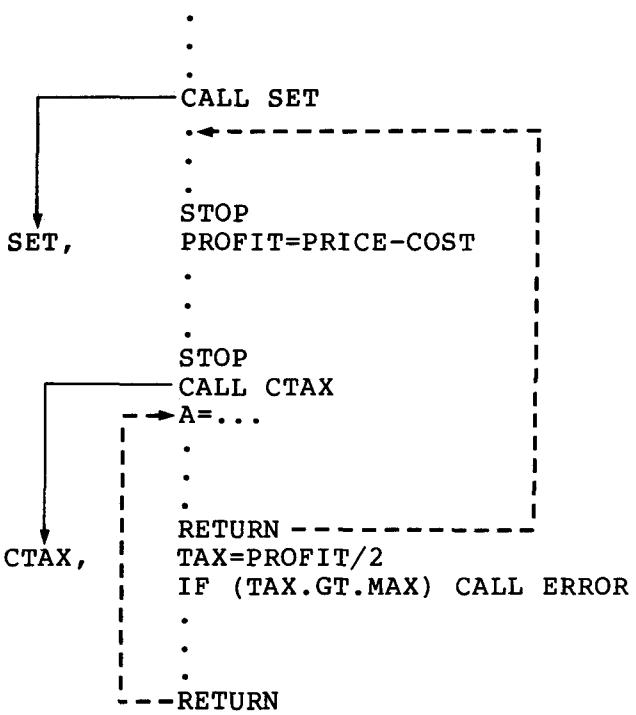
The RETURN statement is placed at the logical end of an internal subroutine. It has the form:

```
RETURN
```

The RETURN statement returns control to the statement immediately following the last CALL statement or to the location from which a TRAP statement transfered control.

A RETURN WITHOUT CALL error message results if a RETURN is attempted when no CALL or TRAP has been executed.

Example:



This example shows how execution control branches to and returns from one subroutine to the next. The solid lines show the result of CALL statements and the broken lines show the result of RETURN statements.

1.2.18 START - Compiler Statement

This optional statement can be used any number of times and can be inserted anywhere in the source program. Each time a START statement is encountered during compilation, a top-of-page command occurs and a new page heading is printed. START is frequently used to segment major sections of programs. It has the form:

```
START[/x][;comment]
```

where:

/x is one of the following option switches:

- /N suppresses compiler listing of source program.
- /L resumes listing source program and errors on printer.
- /T resumes display of source program and errors on the screen.

The START option switch is active until disabled by another option switch. If no option switch is specified, a compiler listing is produced on the printer. The switches can be overridden with an option switch in the RUN COMP command.

;comment is an optional method to stipulate a line of text to be output as a heading on the compiler listing.

If /N is specified in the RUN COMP command, it overrides /L or /T in the source program and stops output of everything except errors. The /L and /T will still determine to what device errors are output.

Example:

```
START;WAREHOUSE INFORMATION
          ;The printer will begin at the top of the page and
          ;will list WAREHOUSE INFORMATION as the title of the
          ;page.

START/T   ;Resumes display of source program and errors on the
          ;screen.
```

1.2.19 STOP - Control Statement

STOP terminates program execution and returns control to the Monitor. It can be inserted anywhere in the Procedure Division. The form of the statement is:

```
STOP
```

There can be more than one STOP statement in a program. STOP does not close files; a FINI statement must be inserted before the STOP to close files previously opened by an INIT statement.

The STOP statement has the same effect as END, but END can only be used as the last statement in a program.

1.2.20 TRACE/NO TRACE - Debugging Statement

These statements are debugging tools used to follow the order of statement execution. TRACE/NO TRACE can be written any place in the Procedure Division of a program. The appearance of TRACE statements in a program does not cause any TRACE output to be generated unless the /T option is specified when the RUN command is given. The form of the statement is:

```
TRACE
.
.
.
NO TRACE
```

Between the TRACE and NO TRACE statements are program statements which are being debugged.

When program tracing is enabled, TRACE lists the order in which statements are executed. TRACE causes the following message to be output on the printer (xxxx is the line number):

```
AT LINE xxxx
```

If the line number identifies a data manipulation statement, the value which is produced and stored by the statement is printed on the following line. Tracing continues until a NO TRACE statement is encountered.

Example:

```
AT LINE 0200
000006
```

Indiscriminate placement of TRACE statements causes excessive output to the printer. To use the TRACE statement to best advantage, use IF statements to isolate the problem to a certain part of the program and then use TRACE on that part of the program.

1.2.21 TRAP - Control Statement

The TRAP statement allows a DIBOL program to be executed at the same time that output is being printed. The format of the statement is:

```
TRAP label
```

where:

label — is the label of a printer subroutine in the Procedure Division of the program.

Because the printer is much slower than the central processing unit (CPU), the TRAP statement is implemented to allow the CPU to continue to execute rather than having to wait for material to be printed. A buffer holds the characters until the printer can use them.

Whenever the print buffer empties, DIBOL statement execution temporarily halts and a call is made to the label specified in the last TRAP statement executed. When a RETURN is made from this call, normal program execution resumes. The TRAP statement normally precedes a FORMS or XMIT statement.

The following information is necessary to effectively use the TRAP statement.

1. If the printer buffer empties during execution of an INIT, XMIT, READ, WRITE, DISPLAY, or FINI statement while I/O is in progress, the TRAP is delayed until execution of the I/O statement is complete.

2. If the printer buffer empties during execution of an ACCEPT statement, the ACCEPT statement is interrupted while the printer buffer is loaded. A noticeable delay occurs only if the keyboard buffer is filled during the time that the TRAP subroutine is being executed. Since the keyboard buffer is approximately 18 characters long, this filling usually takes several seconds.

3. Always construct a TRAP subroutine so that output to the printer is immediately followed by a RETURN statement.

4. Printers are limited to output lines of 126 characters in length when using the TRAP statement. Outputting longer lines results in the program spending all of its time servicing printer TRAPs, thus cancelling the advantage of the TRAP statement.

5. A DIBOL program is slowed down approximately 5 to 10% by the TRAP processor.

Example:

The following example program outputs numbers 1-500 on the printer while some other task is being performed:

```
          RECORD A
N,        D3
          PROC
          TRAP SUB
          FORMS(6,0)   ;Start LPT.
          .
          .
          .            ;Perform task.
          .
          .
LOOP,     IF (N.LT.500) GO TO LOOP
          STOP
SUB,      N=N+1
          IF (N.GT.500) RETURN
          XMIT(6,A)
          RETURN
```

1.2.22 WRITE - Input/Output Statement

A direct access WRITE statement moves a data record from an area in memory to a specified file. It has the form:

```
WRITE (channel,record,rec#)
```

where:

channel — is a numeric value of 1-15, specifying a channel which relates the WRITE statement to an INIT statement. The INIT statement must have specified Update as the device mode.

record — is the record from which data is output.

rec# — is an expression specifying the number of the record on which data is to be written.

Example:

```
WRITE (5,REX,88)     ;Returns the 88th record from the memory area
                     ;REX to the device associated with the
                     ;channel specified by INIT (5,...).
```

Several restrictions have been placed upon the use of READ and WRITE statements.

1. The channel involved must refer to a logical unit on a mass storage device.

2. The file to be accessed with READ or WRITE operations must contain records of uniform size.

3. Only one volume of a multivolume file (the one currently mounted) can be accessed by a READ/WRITE statement.

4. The record which is specified from the Data Division must be the same size as the records of the file being accessed.

5. Attempting to READ or WRITE over the end-of-file mark results in an error message and program termination.

6. Reading or Writing a record after end-of-file usually results in an error message. Certain unpredictable conditions will not crash the COS-310 system but will cause garbled data (on a READ) or the loss of the output record (on a WRITE).

7. Unless a FINI statement is used before terminating a DIBOL program which has Update files, the data from the last few WRITE statements will not be output properly.

XMIT

1.2.23 XMIT - Input/Output Statement

The XMIT statement transfers a data record and is of the form:

```
XMIT (channel,record[,eof label])
```

where:

channel — is a numeric expression (1-15) specifying a channel number which associates the XMIT statement with the related *INIT* statement.

record — is a name previously used in a RECORD statement which identifies the area in memory to which or from which data is to be transmitted. It may be a simple or subscripted variable, or a record literal.

Subscripted record names must be used with care. A single subscript, such as REC(3), should only be used if there are equal length consecutive *records.* The first record must have a name but the others may be unnamed.

Examples:

```
RECORD REC      ;REC(1)
     ,D6
     ,A10
RECORD          ;REC(2)
     ,D6
     ,A10
RECORD          ;REC(3)
     ,D6
     ,A10
```

If a double subscript form is used, e.g., REC (n,m), then n must be less than m-1, n must be odd and m must be even (or the last character in the record). This double subscript form refers to characters n-2 through m-2 inclusive in the record. If n=1, it refers to characters 1 through m-2. Whenever an XMIT occurs referring to the record, two characters before character n in the record are destroyed; this is the *COS-310 word count.* Do not be concerned about this if n=1.

eof label is the label of a statement to which the program branches if an end-of-file is read. It is used with input files only. The input file is closed automatically when an end-of-file is read. If a label is not specified, an error message is output when an end-of-file occurs. The same effect of an end-of-file label can be achieved by an ON ERROR statement preceding the XMIT without an end-of-file.

Examples:

```
XMIT (3,INV,EOF)        ;Transfers a record from the input file
                        ;associated with the statement INIT
                        ;(3,IN,..), to the record area in memory
                        ;labeled INV. If end-of-file is
                        ;reached, control branches to the
                        ;Procedure Division statement labeled
                        ;EOF. If the length of the record
                        ;being read is greater than the defined
                        ;size, an error message is output at run
                        ;time. If the size of the record being
                        ;read is smaller than the defined size,
                        ;the record is left-justified and padded
                        ;with spaces on the right. This format
                        ;is just the opposite of the data
                        ;manipulation statement which does an
                        ;alphanumeric-to-alphanumeric operation.

XMIT (1,CUST,NEXT)      ;Transfers a record from the input file
                        ;associated with the INIT(1,I,...)
                        ;statement to the RECORD area CUST. At
                        ;end-of-file, it branches to the
                        ;statement labeled NEXT.

XMIT (2,BUFF)           ;Takes a record from RECORD area BUFF
                        ;and puts it in the file associated
                        ;with the INIT(2,...) statement
                        ;(assuming channel 2 is initialized for
                        ;output, printer, etc.).

XMIT (8, "HI THERE')    ;Would output the message HI THERE on
                        ;the operator's terminal if channel 8
                        ;was INITed to the TTY.

XMIT (8,CUST(1,7),EOF)  ;Accesses the first five characters of
                        ;the record area CUST.
```

CHAPTER 2

THE MONITOR

2.1 MASTER CONTROL PROGRAM

The Monitor is the master control program for the COS-310 system. It contains all the system I/O handlers:

- Terminal
- Mass Storage
- Printer

The Monitor enables you to edit, compile, save, and execute programs. It also maintains a directory of all programs stored on the system device and lets you label, open, and close files as needed.

During program execution, the Monitor produces the messages which instruct the user to mount files. It also provides the means for batching commands for sequential execution.

The editing feature of the Monitor can be used to create DFU (Data File Utility) tables and source files on the system device. These tables and files may be stored for later use.

Table 2-1 lists the Monitor Keyboard Commands that are available in on-line operations.

Table 2-1
Monitor Keyboard Commands

Keyboard Command	Function
CTRL/C	Returns control to the Monitor. The Monitor displays a dot and awaits a command. If the Monitor is already in control, CTRL/C has the same effect as a CTRL/U.
CTRL/O	Suppresses terminal echo of typed output. If echo is already suppressed, CTRL/O restores the terminal echo. CTRL/O is also used to halt and resume output from an LI command or the compiler. The echo always resumes the next time the dot is printed. All suppressed output is lost.
CTRL/Q	Resumes output to the screen from the point at which it was halted by CTRL/S.
CTRL/S	Halts output to screen. No output data is lost. CTRL/Q will resume output.
CTRL/U	Deletes the current input line.
CTRL/Z	Signals the end of input and returns control to the Monitor. Halts output of line numbers from an LN command.
DELETE	Erases the last character typed and moves the cursor to that character's position.
RETURN	Indicates that a line of input is complete.

2.1.1 MOUNT Messages

The Monitor displays MOUNT messages on the screen whenever an input or output logical unit number must be specified. These messages have the form:

```
MOUNT filnam #nn FOR INPUT:

MOUNT filnam #nn FOR OUTPUT:
```

where:

filnam	is the name of the data file desired by the program currently executing.
#nn	is the volume number (1 to 63) of the data file.

Respond to this message with the logical unit number(1-15) which indicates the location of the data storage unit. If an error is made in the reply, type CTRL/U and the correct reply.

The MOUNT message is displayed and the volume number is incremented whenever the program reaches the end of the mass storage device yet more information remains to be read or written.

When logical unit numbers specified in control programs are not available, a question mark precedes the MOUNT message:

```
?MOUNT filnam #01 FOR INPUT:
```

Respond to this message with a different logical unit number.

The following message is displayed when a file is already stored on the logical unit specified in a MOUNT message:

```
REPLACE filnam #nn ?
```

Answer REPLACE with a Y to replace the old file; answer with any other character to keep the old file. File labels beginning with any character other than A-Z,], ^, or [, are considered to be temporary files, and no REPLACE message is displayed.

If output is to a previously unused logical unit, a garbled file name or volume number may be displayed in the REPLACE message. This is because random characters are on the storage unit where the label should be. Answer the REPLACE message with YES to replace the garbled file.

2.1.2 Operating Procedures

The Monitor is loaded via a bootstrap routine each time the system is started. The Monitor signals that it is loaded into memory by displaying the message:

```
COS MONITOR V 8.00 (or current version number)
DATE?
```

To proceed, type DATE (or DA), a space, the current date in the form DA dd-mmm-yy. The date must be entered before proceeding. This date is used during program execution to date reports, files, and newly created programs.

The Monitor indicates that it is ready to accept other commands by displaying either a dot (.) or by executing the batch file START. START is executed if you selected the batch file option in the SYSGEN operation (see Section 3.1 for SYSGEN operation).

2.2 MONITOR COMMANDS

The following commands apply to Monitor functions.

BATCH	sequentially executes a string of monitor commands
DATE	stores a date
DELETE	erases programs from a device directory
DIRECTORY	prints the names of stored files
PLEASE	displays text on screen during program execution
RUN	loads and executes programs
SAVE	stores binary programs on a storage device

Only the first two characters of the command must be typed (R is sufficient for the RUN command). Any additional characters up to the first blank are ignored. All commands must be followed by the RETURN key before execution will begin.

2.2.1 BATCH

A BATCH command sequentially executes a string of Monitor commands. As soon as a command file associated with a Monitor command is completed, another Monitor command is executed. Certain system programs started by the RUN command may either terminate BATCH or will not accept input from the batch stream; these require operator interaction.

The form of the BATCH command is:

```
BA cmndfl
```

where:

cmndfl is the name assigned to a previously stored file containing a list of Monitor commands.

Following is an example of a command file:

```
0090  RUN COMP,JOB1
0100  SAVE JOB1
0110  RUN DFU, SYSTAB
0120  RUN JOB1
0130  RUN JOB2
0140  RUN SORT, SRCL
0150  RUN DFU,OLDTAB
0160  DE JOB1/B
```

All batch command files must be on the system device. A batch command file should contain a BATCH command only as its last line.

A Monitor command read from a batch file is displayed on the screen and executed. Type CTRL/C to terminate a batch command file; the batch can then be restarted only at the beginning of the file.

All of the necessary programs and data files must be available during BATCH execution. If an error occurs, BATCH terminates, control returns to the Monitor, and a dot is displayed on the screen.

After the error is corrected, the entire batch command file can be restarted or each remaining command can be individually typed.

When the batch command file is finished, control returns to the Monitor and a dot is displayed on the screen.

2.2.2 DATE

The DATE command stores a date which is assigned to all programs that are created or to reports that are printed. This date remains the same until a new DATE command is issued or the system is rebooted.

The form of the DATE command is:

```
DA  dd-mmm-yy
```

where:

dd	is a two-digit decimal number representing the day.
mmm	is a three-character alphabetic string which must be the first three letters of the month.
yy	is a two-digit decimal number representing the last two digits of the year.

After the system is booted, the Monitor displays the message:

```
COS MONITOR V 8.00 (or current version number)
DATE?
.
```

If anything is typed before the date is entered, the Monitor repeats:

```
DATE?
```

If the date is entered incorrectly, the Monitor displays:

```
BAD DATE
```

Enter the date whenever the Monitor is booted. It is also used to change the system date.

Examples:

```
.DA  25-SEP-76
.DATE  5-JUL-77
```

2.2.3 DELETE

The DELETE command erases the named source, binary, or system program from the specified device directory.

The form of the DELETE command is:

```
DE pronam[,dev]/x
```

where:

pronam — is the name of the program to be removed from the directory.

dev — is the three-character designation for the physical device where the program is stored. If no device is specified, the system device is assumed.

/x — Is a one- or two-character code indicating that the program to be deleted is either a source (S), binary (B), or system (SV) program. This code is necessary to differentiate between three programs with the same name but of different types. The code SV is used rather than V to make it more difficult to mistakenly delete a system program.

Data files are not deleted, they can only be replaced.

Examples:

```
.DE JOB1, RX3/B      ;Delete binary program named JOB1 from device
                     ;RX3.
.DE PROGA,DK3/S      ;Delete source program named PROGA from
                     ;device DK3.
.DE INV/S            ;Delete source program named INV from system
                     ;device.
.DE FILEX/SV         ;Delete system program named FILEX from
                     ;system device.
```

2.2.4 DIRECTORY

The DIRECTORY command prints a list of programs stored on a physical device or the name of the data file stored on a logical unit. Be sure the printer is on-line before issuing the DI command.

The form of the DIRECTORY command is:

```
        ⎧[ dev][/T]      ⎫
DI      ⎨                ⎬
        ⎩/logical unit #  ⎭
```

where:

dev — is a three-character designation for the mass storage device on which the directory is stored, and must be preceded by a comma or space. If not specified, the system device is assumed.

/T — is an optional switch which causes the directory to be displayed on the screen. If /T is not specified, the directory will be listed on the printer.

/logical unit # — is the number (1-15) of the logical unit assigned with DFU (Data File Utility). A logical unit # must be preceded by /. Specifying a logical unit # causes a label to be listed on the printer. The DI command must be repeated each time a logical unit # is to be printed.

The directory contains the current date, names of programs, types of programs, length (LN) in 512-byte blocks, and the date each program was stored.

A logical unit label contains the file name, sequence number (if a multivolume file), the date the file was created, file length in segments, and the number of the logical unit where the file was stored when the label was requested. Segments are sixteen 512-byte blocks long.

The only directory entry dates that are valid are those for the current year and seven years preceding the current date. Any dates prior to this time will be printed incorrectly.

Examples:

The command:

```
.DI DK0    ;Directory from physical device DK0.
```

causes a directory similar to the following to be output on the printer:

```
DIRECTORY       15-FEB-72

NAME    TYPE LN     DATE

COMP      V  14   19-JAN-78
MORE      S  10   15-FEB-78
 <0006 FREE BLOCKS>
TST2      S  07   12-FEB-78
 <0007 FREE BLOCKS>
TST4      S  07   15-FEB-78
GLOP      S  10   15-FEB-78
 <0579 FREE BLOCKS>
```

The command:

```
.DI/3      ;Directory from logical unit 3.
```

outputs a file label similar to the one below.

```
*****************************
*                           *
*   NAME    SEQ.    DATE    *
*                           *
*   DEP     #01   18-NOV-75 *
*                           *
*   LENGTH: 0046  UNIT: 3   *
*                           *
*****************************
```

2.2.5 PLEASE

The PLEASE command displays text on the screen during execution of a batch command file.

The form of the PLEASE command is:

```
PLEASE text
```

The text is displayed exactly as entered and the terminal alarm is sounded. Do as requested by the PLEASE text and type any key (including CTRL/C) to continue the batch program.

To make a two-line PLEASE command, the first line is terminated with AND and the second line begun with another PLEASE. The AND lets the operator know more text is to follow.

Example:

```
0020  RUN JOB1
0030  PLEASE PUT INVOICES IN PRINTER AND
0040  PLEASE TYPE 3 TO THE NEXT MOUNT MESSAGE
0050  RUN JOB3
0060  PLEASE PUT REGULAR PAPER IN PRINTER
```

When this batch command file is executed, JOB1 will be run, the first PLEASE text will be displayed, and the terminal alarm will be sounded. The system waits for a key to be typed in reply to the PLEASE text, then it displays the next PLEASE command. When a key is typed in reply to the text, JOB3 is executed and the last PLEASE text is displayed. Control returns to the Monitor when a key is typed in reply to the last PLEASE text.

If a PLEASE command is given in a non-BATCH mode, the terminal alarm sounds and the system waits for a RETURN key to be typed.

2.2.6 RUN

The RUN command loads and executes a system program or a binary program using the named file. This command provides access to all other system programs, such as:

RUN SYSGEN	To build a new system or change system handlers.
RUN SORT	To sort data files.
RUN PIP	To move information between physical devices.
RUN COMP	To compile a user source program into a binary program.

The RUN command has the form:

```
RUN [ { pronam                          } ][,filnam1...,filnam7][/xx]
      { chain0+chain1...+chain7         }
```

where:

pronam is the name of the program to be run.

If the program name is omitted, the Monitor loads and executes the DIBOL program in the binary scratch area.

chain0+chain1...
are binary files which are part of one large program which has been divided into several chained programs. For example:

```
.RUN CHAIN0+CHAIN1+CHAIN2+CHAIN3
```

would execute program CHAIN0. CHAIN0 would then determine whether program CHAIN1, CHAIN2, or CHAIN3 would be run next.

filnam1...,filnam7
: are source files which must be on the system device. If one of the system programs is executed via the RUN command and no source files are specified as input, the file in the edit buffer is used as input (system programs only).

  The maximum number of binary and source files per program is eight (including pronam or chain0). Multiple files are concatenated and passed to system programs as one large file.

/xx
: is one or a combination of option switches associated with the program being run.

If the program file specified is not found, the following error message is displayed.

```
FILE NOT FOUND
```

When a program is loaded into memory by a RUN command, the Monitor temporarily stores the contents of the edit buffer in the editing scratch area on the system device. The contents of the edit buffer are returned to memory when program execution is complete.

Examples:

```
.RU                 ;Executes most recently compiled DIBOL
                    ;program.

.RU JOB1            ;Runs program called JOB1.

.RUN COMP, CHECK    ;Compiles the source program CHECK.

.RUN  ,BIN1,BIN2    ;Runs the program from the binary scratch
                    ;area using BIN1 and BIN2 as input files.
```

2.2.7 SAVE

This command copies the binary program from the binary scratch area and stores it on the named device. The saved binary will be of type B.

The form of the SAVE command is:

```
SA pronam[,dev][/Y]
```

where:

pronam — is the name to be assigned to the binary program being stored.

dev — is the three-character designation for any mass storage device which has a directory. If no device is given, the system device is assumed.

/Y — is used to bypass the REPLACE? message when a duplicate name is encountered. Normally used in a batch mode to bypass operator response.

If the program name specified is a name already in the directory, the Monitor displays:

```
REPLACE?
```

Type Y or YES to replace the old file with the new file. Type N or any other character to leave the old file and return to the Monitor.

2.3 EDITOR COMMANDS

The COS-310 Monitor contains a line number editor. Every line of text input to the Monitor is assigned a line number.

Example:

```
0100            START
0110            RECORD A
0120  A1,       A64
0130            PROC 2
0140            INIT (2,IN,'MINT')
0150  LOOP,     XMIT (2,A,EOF)
0160            XMIT (8,A)
0170            GO TO LOOP
0180  EOF,      FINI(2)
0190            STOP
0200            END
```

Insertions, changes, and deletions are done with these line numbers.

The following commands are functions of the editor.

ERASE	clears text from the edit buffer
FETCH	loads a source file into memory
LIST	outputs text to the screen or printer
Line Number	outputs incremented line numbers
Number Commands	edits text within the edit buffer
RESEQUENCE	renumbers program lines
WRITE	stores source files for later editing

These commands can be entered in response to the Monitor dot. Only the first two characters of the command are needed. The exceptions to this first-two-letter convention are in the line number and number commands. All commands must be followed by the RETURN key.

2.3.1 ERASE

The ERASE command erases (clears) text from the edit buffer.

The form of the ERASE command is:

```
ER [n1][,n2]
```

where:

n1	is the number of the line to be erased or the first of two line numbers which delimit the lines to be erased. If omitted, erasing starts at the beginning of the edit buffer.
,n2	is the second of the two delimiting line numbers; it indicates where erasing ends. If n2 is omitted but the comma is included, erasing continues to the end of the edit buffer.

If no line numbers are specified, the ERASE command clears the entire edit buffer. Use this command to erase the edit buffer before entering any material to be edited.

Examples:

```
.ER       ;Clears the entire edit buffer.

.ER 5     ;Clears line 5.

.ER ,5    ;Clears from the start of the buffer through line 5.

.ER 5,10  ;Clears from line 5 through line 10.

.ER 5,    ;Clears from line 5 through the end of the buffer.
```

FETCH

2.3.2 FETCH

The FETCH command erases the edit buffer and loads the named source file from the specified device into memory.

The form of the FETCH command is:

```
FE filnam[,dev]
```

where:

filnam — is the name of a previously stored source file which is to be brought into memory.

dev — is the three-character designation for the mass storage device where the file is stored. If the device designation is omitted, the system device is assumed.

If the source file is not found, the Monitor displays the message:

```
FILE NOT FOUND
```

Retype the command with the correct source file name or device designation. Consult the directory to find the proper name.

Examples:

```
.FE RICH        ;Moves file RICH from the system device to the
                ;edit buffer.

.FE PAYROL,DK2 ;Moves file PAYROL from an RK05 disk on drive 2 to
                ;the edit buffer.
```

2.3.3 LIST

The LIST command outputs the specified lines or the entire contents of the edit buffer to the printer or to the screen.

The form of the LIST command is:

```
LI [n1][,n2][/L]
```

where:

n1	is the number of the line to be output or the first of two line numbers which delimit the lines to be output. If omitted, output starts at the beginning of the edit buffer.
,n2	is the second of two line numbers; it indicates where output ends. If n2 is omitted but the comma is included, output continues to the end of the edit buffer.
/L	is the one-letter switch which will output the list to the printer. If /L is not indicated, the list is displayed on the screen.

If no line numbers are specified, the entire contents of the edit buffer is output. CTRL/O stops output from an LI command; CTRL/S halts display on the screen; CTRL/Q resumes display halted by CTRL/S.

Examples:

```
.LI/L      ;List entire edit buffer on printer.

.LI 5      ;Display line 5 on the screen.

.LI ,5     ;Display from beginning of buffer through line 5.

.LI 5,10   ;Display line 5-10, inclusive, on the screen.

.LI 5,     ;Display from line 5 through the end of the buffer.
```

2.3.4 Line Number

The Line Number command automatically outputs incremented line numbers so new lines can be entered without manually typing each line number.

The form of the Line Number command is:

```
LN [n][,inc]
```

where:

n	is the number of the starting line. If no starting line number is specified, 100 is assumed.
,	If the comma after the starting number and the increment number are omitted, the starting number and increment number are the same. If the command is LN 100, the start line number is 100 and the increment remains unchanged from the last LN command. Once in memory, the increment returns to 10.
inc	is the increment between line numbers. If no increment is specified, 10 is assumed.

If Line Number command is terminated and some editing has been done, type the Line Number command (LN) with no arguments to display the next number in sequence.

The LN command does not clear the edit buffer. Line Numbers 0 to 4095 are available. Under the default conditions (start at 100, increment by 10), the program can be approximately 400 lines long.

The maximum number of characters on an input line, including the line number and space, is 120. The line number and space are counted as two characters.

No terminal screens are 120 characters wide. When the screen is full, the Monitor executes a carriage return/line feed but does not display the next line number. If the 120-character input line length is exceeded, the Monitor gives the error message LINE TOO LONG and the entire input line is lost.

If RETURN is the first key typed after an automatic line number, the line number increments but a blank line is not generated. To obtain a blank line, type the SPACE bar and the RETURN key. To obtain a blank line after you manually enter a line number, type two spaces and the RETURN key.

Tabs can be used to increase the readability of a program. The TAB key on most terminals is set to produce up to 8 spaces. The first tab goes to column 13 because the line number and space take the first five spaces.

Type CTRL/Z to indicate the end of input and to halt the automatic line numbering.

Examples:

```
.LN       ;Requests line numbers starting at 100 with increments
          ;of 10.

.LN 10,5  ;Requests line numbers starting at 10 with increments
          ;of 5.

.LN ,100  ;Requests line numbers starting at 100 (default) with
          ;increments of 100.

.LN 50    ;Requests line numbers starting at 50 with increments
          ;of 50.
```

If an error is made when using automatic line numbers, use the DELETE key or CTRL/U prior to typing the RETURN key. The DELETE key erases the last character typed. CTRL/U erases the entire line; the Monitor redisplays the line number.

If the edit buffer is full, the error message EDIT BUFFER FULL appears, and the last line entered is lost.

The edit buffer can be separated into two or more source files. This is done with the following procedure:

WRITE the edit buffer as file B.

ERASE the last half of the edit buffer.

WRITE the edit buffer as file A.

FETCH file B.

ERASE the first half of the edit buffer.

WRITE the edit buffer as file B.

2.3.5 Number Commands

Any line beginning with a number can be edited in the edit buffer. Lines are edited using the following form:

```
nnnn [text]
```

where:

nnnn	is the number of a line you want to work on.
text	is data to be input on the line. The data must be separated from the line number by one SPACE or a TAB. A TAB becomes the first character of text.

If text is already at that line number, the new text replaces it. Lines are stored in increasing line number order. Type the line number and RETURN right after the line number to clear data from that line.

An input line is limited to 116 characters plus the four-character line number (a total of 120 characters).

Examples:

Text before editing:

```
.LI
0035  PROC
0040  INIT(I,V,IN)
  .
0047  XMIT(6,B)
  .
0060  END
```

Editing commands:

```
.35  PROC 1                  ;Inserts PROC1 at line 35.
.40  INIT(1,IN,'LABEL',2)    ;Inserts new text at line 40.
.47                          ;Erases text and line number.
```

Text after editing:

```
.LI
0035  PROC 1
0040  INIT(1,IN,'LABEL',2)
  .
0060  END
```

2.3.6 RESEQUENCE

The RESEQUENCE command renumbers the program lines to adjust for addition and deletion of lines.

The form of the RESEQUENCE command is:

```
RE [n][,inc]
```

where:

n	is the starting line number. If no starting line number is specified, 100 is assumed.
,	If the comma after the starting number and the increment number are omitted, the starting number and increment number are the same. If the comma is included, the starting number is as designated and the increment remains unchanged unless the Monitor is read back into memory; once in memory, the increment returns to 10.
inc	is the increment between line numbers. If no increment is specified, 10 is assumed.

If the line number exceeds 4095 following a RESEQUENCE command, the error message LINE # TOO LARGE results. Enter another RESEQUENCE command with smaller increments. If this is not done, the text will be only partially resequenced and duplicate line numbers may result.

Examples:

```
.RE         ;Resequences line numbers of a program in the edit
            ;buffer using 100 as the starting line number and 10
            ;as the increment.

.RE 10,5    ;Resequences line numbers of a program in the edit
            ;buffer using 10 as the starting line number and 5 as
            ;the increment.

.RE ,100    ;Resequences line numbers of the program in the edit
            ;buffer using 100 or the last specified line number as
            ;the starting line and 100 as the increment.

.RE 50,     ;Resequences line numbers of a program in edit buffer
            ;using 50 as the starting line number without changing
            ;the increment.
```

2.3.7 WRITE

The WRITE command stores a source file on the specified device so it can later be compiled or fetched for editing.

The form of the WRITE command is:

```
WR filnam[,dev][/Y]
```

where:

filnam	is the name (up to six characters) of the source file to be stored.
,dev	is the three-character designation for the mass storage device where the program is to be stored. If no device is specified, the system device is assumed.
/Y	bypasses the REPLACE? message when a duplicate name is encountered. Normally used in a batch file to bypass operator response.

If the filnam specified is a duplicate name, the Monitor displays:

```
REPLACE?
```

Type Y or YES to replace the old file with the new file. Type N or any other character to leave the old file and return to the Monitor.

2.4 MONITOR ERROR MESSAGES

Message	Explanation
BAD COMPILATION	Attempted to SAVE a compiled binary that had errors. Correct errors before compiling.
BAD DATE	Typed an unrecognizable date. Retype.
BAD DIRECTORY	Attempted to reference or store a file on a device with a damaged or nonexistent directory. Only devices with directories can be used. If the directory is damaged, call your Software Specialist.
BAD LABEL	No data file label, or label's form is incorrect. Check for correct label.
EDIT BUFFER FULL	Greater than 8,150 characters (see Section 2.3.4 Line Number).
ERROR IN COMMAND	Miscellaneous. Check previous command for form and accuracy.
ERROR ON dev, RETRY?	The wrong device was designated. Type N for no retry. Any other input causes the device handler to retry.
FILE NOT FOUND	The file specified was not found on the directory that was specified. Re-enter file name or review directory for file existence.
ILLEGAL PROGRAM	Attempted to run a system program that has a different version number than the Monitor. Version numbers must be the same.
ILLEGAL UNIT	Either the specified logical unit number is illegal (not 1-15) or the specified device is not currently part of the system. Replace the illegal number with a correct unit number. Stipulate a correct device designation.
IN USE	The specified logical unit is already open. Select another logical unit.

Message	Explanation
LINE TOO LONG	More than 120 characters entered on an input line. Line must be shortened.
LINE # TOO LARGE	Line number greater than 4095. Resequence line numbers or reduce the total number of lines.
?NO FILE TO SAVE	Nothing in the edit buffer when WRITE command is issued. New data must be entered.
NO INIT	Program attempted to read or write on a device that was not opened by the system program. Device must be opened with an INIT statement.
NO LP BUFFER	Not enough memory to support the selected printer (e.g., LQP printer requires 24K bytes of memory). Add more memory or select another type of printer.

2.5 RUN-TIME ERROR MESSAGES

All errors are fatal unless the error is trapable and the statement in which it occurs is immediately preceded by an ON ERROR statement with a valid label (see Section 1.2.13 ON ERROR).

The messages marked with an asterisk (*) cannot be checked with an ON ERROR statement.

AT LINE nnnn is displayed under all run-time error messages; nnnn is the DIBOL source program line number where the error occurred. If COMP/O was specified for a program, nnnn is meaningless.

Message	Explanation
*BAD CHAIN	CHAIN argument does not match .RUN command. All chained programs must be stipulated in the RUN command.
BAD DIGIT	A character other than +, -, space, or the digits 0-9 was encountered in an alphanumeric-to-numeric conversion. Check and delete bad digits.

Message	Explanation
*BAD PROGRAM	Attempted to run a binary program which contains a compilation error. Check compilation listing for errors. Correct errors and recompile.
*DIBOL FILE NUMBER IN USE	In INIT, the channel number is already associated with a device. Enter new channel number/device combination.
*DIBOL FILE NUMBER NOT INITED	An attempt was made to XMIT, READ, or WRITE with a channel number that was not associated with a device. Either INIT the channel number or use a channel number already opened.
END OF FILE	The last record of an input file has been read and the end-of-file mark encountered, but no EOF label was specified in the XMIT statement or in the ON ERROR statement preceding the XMIT statement. Rewrite XMIT statement.
*ILLEGAL DEVICE	Attempted to WRITE on a file that was not opened for UPDATE or attempted to READ from a file that was not opened for INPUT or UPDATE. INIT file properly.
ILLEGAL RECORD #	Either the record number is 0, past the end of the logical unit, or the records are not all the same length when you are using UPDATE mode.
*ILLEGAL SUBSTRING	A DIBOL Procedure Division statement attempted to access a subscripted data field, F1 (m,n), but m<1 or m>n. Redefine data field.
LINE TOO LONG	Attempted to XMIT a record that is too long for the area defined in the Data Division. Redefine the area in the Data Division.
*NO BUFFERS LEFT	Not enough memory available for I/O buffers (e.g., DIBOL program is too big). An I/O buffer of some multiple of 512 characters is set up for each active mass storage file. Another possibility: too few files were specified in the PROC statement. Specify more files.

Message	Explanation
NO FILE	No file specified in RUN command to satisfy INIT (SYS) statement Specify file.
NUMBER TOO LONG	A numeric field longer than 15 digits was used in a calculation. Reduce to within 15-digit limitation.
*PROGRAM TOO BIG	Binary program does not fit in available memory. Reduce program size, or chain program.
*PUSHDOWN OVERFLOW	Statement is too complex and/or subroutines are nested to a depth greater than 50. Simplify statement, reduce nesting, or both.
*RETURN WITHOUT CALL	The program tried to execute a RETURN, but there was no place to go. Implement CALL or TRAP statement or delete the RETURN statement.
*SUBSCRIPT TOO BIG	Program attempted to use a subscript larger than that defined in the Data Division. Note that the run-time system does not detect all illegal subscripts, only those which would cause the program or the system to be destroyed. Redefine subscript.
ZERO DIVISOR	The program attempted to divide by zero. Eliminate division by zero.

CHAPTER 3

SYSTEM GENERATION PROGRAM (SYSGEN)

The System Generation Program (SYSGEN) is a conversational program used to create a system on a new device or to change the system handlers in the current system. The SYSGEN statement has the following form:

```
RUN SYSGEN { /B }
           { /C }
```

where:

/B	builds a system in a new device.
/C	changes handlers on the current system.

3.1 SYSGEN/B OPERATING PROCEDURES

Use SYSGEN/B to build a system on a new device.

At least two drives must be running and loaded on the system in order to perform this operation. To execute SYSGEN/B, type:

```
RUN SYSGEN/B
```

After this command, SYSGEN displays the following question:

```
WHAT IS THE NEW SYSTEM DEVICE?
```

Respond by typing the three-character designation for the device that you want to build a new system on. A message similar to the following then appears on the screen.

```
ENTER NUMBER CORRESPONDING TO DESIRED CONFIGURATION

1    DK        RK05 CARTRIDGE DISK DRIVES

2    RX        RX01 FLOPPY DISK DRIVES

3    DY        RX02 FLOPPY DISK DRIVES

4    DK & RX   RK05 AND RX01 DISK DRIVES

5    DK & DY   RK05 AND RX02 DISK DRIVES
```

Type the number corresponding to the kind of drive(s) that you want. SYSGEN responds with:

```
PLEASE TYPE NUMBER OF PRINTER MODEL ON SYSTEM

1    LA8A      DECPRINTER I USING DKC8-AA INTERFACE

2    LA35      DECWRITER II

3    LA36RO    DECWRITER II

4    LQP       LETTER-QUALITY PRINTER

5    LP05      300 LPM PRINTER

6    LA8       DECPRINTER I

7    LA120     DECWRITER III

8    NONE      NO PRINTER
```

Type the number corresponding to the printer you want. If you select a printer that does not have forms hardware, SYSGEN asks:

```
HOW MANY LINES PER PAGE?
```

Type the number of lines you want on each page. The default value is 66 lines. After lines-per-page has been specified, the system asks:

```
DO YOU WANT START-UP BATCH FILE?
```

Answer YES if you want the Monitor to execute the batch file START every time you use the Monitor DATE command. Answer NO if you do not

want to automatically execute the batch file. This option does not require any additional memory for the COS-310 Monitor or space on the system device except for the space needed for the START file. You create START by writing a batch file and naming it START.

After you enter YES or NO, SYSGEN responds by asking:

DO YOU WANT TO TRANSFER YOUR FILES?

Answer YES to copy the Monitor and the system, source, and binary files onto the new system device. This transfer destroys anything previously stored on the new system device. Answer NO to empty the new device's directory and to copy the COS-310 Monitor onto the new device; no files are transferred.

SYSGEN then asks:

IS EVERYTHING CORRECT?

Type YES if your answers are correct. Type NO and SYSGEN repeats the questions starting at the request for the new system device.

The new system is built only after you give a YES response to SYSGEN's last question.

If you chose not to transfer files, the COS MONITOR message is immediately displayed. If you chose to transfer files, the time needed to make the transfer delays the COS MONITOR message.

SYSGEN/B does not reset the logical unit assignments to reflect the new area occupied by the system on a disk. Use DFU to make new logical unit assignments.

The SYSGEN/B operation will fail if you attempt to transfer your files to a device that does not have enough room for both the system and the files. The operation can be done by rebooting the system and running SYSGEN/B without transferring files onto one device and then running SYSGEN/B without transferring files a second time onto another device. Then use PIP OPT- E to put source files on one device and binary files on the other device.

3.2 SYSGEN/C OPERATING PROCEDURES

Use SYSGEN/C to change handlers within the current system. To execute SYSGEN/C, type:

RUN SYSGEN/C

SYSGEN displays the following statement:

```
ENTER NUMBER CORRESPONDING TO DESIRED CONFIGURATION

1    DK        RK05 CARTRIDGE DISK DRIVES

2    RX        RX01 FLOPPY DISK DRIVES

3    DY        RX02 FLOPPY DISK DRIVES

4    DK & RX   RK05 AND RX01 DISK DRIVES

5    DK & DY   RK05 AND RX02 DISK DRIVES
```

Type the number corresponding to the kind of drive(s) that you want. SYSGEN responds with:

```
PLEASE TYPE NUMBER OF PRINTER MODEL ON SYSTEM

1    LA8A      DECPRINTER I USING DKC8-AA INTERFACE

2    LA35      DECWRITER II

3    LA36RO    DECWRITER II

4    LQP       LETTER-QUALITY PRINTER

5    LP05      300 LPM PRINTER

6    LA8       DECPRINTER I

7    LA120     DECWRITER III

8    NONE      NO PRINTER
```

Type the number corresponding to the printer you want. If you select a printer that does not have forms hardware, SYSGEN asks:

```
HOW MANY LINES PER PAGE?
```

Type the number of lines you want on each page. The default value is 66 lines. After lines-per-page has been specified, the system asks:

```
DO YOU WANT START-UP BATCH FILE?
```

Answer YES if you want the Monitor to execute the batch file START every time you use the Monitor DATE command. Answer NO if you do not want to automatically execute the batch file. This option does not require any additional memory for the COS-310 Monitor or space on the system device except for the space needed for the START file. You create START by writing a batch file and naming it START.

After you enter YES or NO, SYSGEN asks:

```
IS EVERYTHING CORRECT?
```

Type NO and the entire sequence of questions begins again. Type YES and SYSGEN responds:

```
SYSGEN COMPLETE--PLEASE RE-INITIALIZE SYSTEM
```

The system automatically halts and must be rebooted. The new handlers are now in the system and SYSGEN/C is completed.

3.3 SYSGEN ERROR MESSAGES

Message	Explanation
BAD SWITCH	Attempted to use a switch other than B or C. Use /B or /C.
dev MUST BE INCLUDED IN CONFIGURATION	Attempted to operate SYSGEN/C without designating a device handler. Enter the device designation where needed.
FULL	Ran out of room for files on new device. Run SYSGEN/B twice without transferring files and use PIP OPT- E to put source files on one device and binary files on the other device.

The most common indication of error is the repeat of the question. Answer the question again.

CHAPTER 4

DATA FILE UTILITY PROGRAM (DFU)

Use the Data File Utility Program (DFU) to make logical unit assignments or to print a table of these assignments for reference. The COS-310 system is shipped with logical units unassigned.

4.1 DFU OPERATING PROCEDURES

To execute DFU, type:

```
RUN DFU { ,filnam }
        { /xx     }
```

where:

,filnam — is the name of a previously created file containing a table of logical unit assignments. This file is stored on the system device.

/xx — is an option switch which determines the specific function of DFU.

- /B — makes logical unit assignments from a table created in the edit buffer.
- /K — makes new logical unit assignments directly from the keyboard.
- /D — displays the table of current logical unit assignments on the screen.
- /DL — lists the table of current logical unit assignments on the printer.
- /E — displays an expanded table of current logical unit assignments. Similar to /D with the addition of

the file name, volume sequence number, creation date, and length of the data file.

/EL lists the expanded table of logical unit assignments on the printer.

If neither file name nor option switch is specified, DFU defaults to /B.

4.1.1 DFU,filnam Operating Procedures

DFU,filnam takes the logical unit assignments from a table stored as a file on the system device. To execute DFU,filnam type:

```
RUN DFU,filnam
```

The table is created with editor commands and contains the device designation and the number of segments in each logical unit. The sequence of the entries determines the number associated with the logical unit. The table is created in the edit buffer and stored as a named file. The table is similar to the following:

```
RXO, 41
RX1, 41
RK1, 41
RK3, 21
.
.
.
```

A maximum of 15 entries can be made in the table.

The DFU program makes the assignments on the appropriate devices but produces no output on the screen or printer.

4.1.2 DFU/B Operating Procedures

DFU/B makes logical unit assignments from a table in the edit buffer. To execute DFU/B, type:

```
RUN DFU/B
```

The table is the same as the one used by DFU,filnam. The table must be in the edit buffer. The processor makes the assignments on the appropriate devices but produces no output on the screen or printer.

4.1.3 DFU/K Operating Procedures

DFU/K makes new logical unit assignments from the keyboard. To execute DFU/K, type:

```
RUN DFU/K
```

DFU responds by prompting you to enter the logical unit assignments. For example:

```
.RUN DFU/K
DFU V 8.00
 1 = RX0,41
 2 = RX1,41
 3 = RX2,41
 4 = RX3,41
 5 = END
```

4.1.4 DFU/D Operating Procedures

DFU/D displays the table of current logical unit assignments on the screen. To execute DFU/D, type:

```
RUN DFU/D
```

A table similar to the following is displayed on the screen.

```
UNIT DEV.  SEGS.

  1  RX0   41
  2  RX1   41
  3  RX2   41
  4  RX3   41
  5  -UNDEFINED-
  6  -UNDEFINED-
  7  -UNDEFINED-
  8  -UNDEFINED-
  9  -UNDEFINED-
 10  -UNDEFINED-
 11  -UNDEFINED-
 12  -UNDEFINED-
 13  -UNDEFINED-
 14  -UNDEFINED-
 15  -UNDEFINED-
```

where:

UNIT is the logical unit number.

DEV. is the three-character designation of the mass storage device where the unit is located.

SEGS. is the number of segments assigned to each logical unit.

4.1.5 DFU/DL Operating Procedures

DFU/DL lists a table of current logical unit assignments. This is the same table as in DFU/D except the table is listed on the printer. To execute DFU/DL, type:

```
RUN DFU/DL
```

A table similar to the following is listed on the printer.

```
UNIT DEV.  SEGS.

  1  RX0   41
  2  RX1   41
  3  RX2   41
  4  RX3   41
  5  -UNDEFINED-
  6  -UNDEFINED-
  7  -UNDEFINED-
  8  -UNDEFINED-
  9  -UNDEFINED-
 10  -UNDEFINED-
 11  -UNDEFINED-
 12  -UNDEFINED-
 13  -UNDEFINED-
 14  -UNDEFINED-
 15  -UNDEFINED-
```

where:

UNIT is the logical unit number.

DEV. is the three-character designation of the mass storage device where the unit is located.

SEGS. is the number of segments assigned to each logical unit.

4.1.6 DFU/E Operating Procedures

DFU/E displays the expanded table of current logical unit assignments. This is similar to DFU/D with the addition of the file name, the volume sequence number (1-63), the creation date, and the data length. To execute DFU/E, type:

```
RUN DFU/E
```

A table similar to the following is displayed on the screen.

```
UNIT   DEV. SEGS.  NAME    SEQ.  DATE     LEN.

  1    RX1  0037   FILE1   1     2/ 4/76  0010
  2    RX1  0002   FILE2   1     1/28/76  0002
  3    RX1  0002   FILE3   1     3/ 1/76  0002
  4    - UNDEFINED -
  5    - UNDEFINED -
  6    - UNDEFINED -
  7    - UNDEFINED -
  8    - UNDEFINED -
  9    - UNDEFINED -
 10    - UNDEFINED -
 11    - UNDEFINED -
 12    - UNDEFINED -
 13    - UNDEFINED -
 14    - UNDEFINED -
 15    - UNDEFINED -
```

where:

UNIT — is the logical unit number.

DEV. — is the three-character designation of the mass storage device where the unit is located.

SEGS. — is the number of segments assigned to each logical unit.

NAME — is the name assigned to the data file.

SEQ. — is the sequence number (what volume in a multivolume file) of that specific data file.

DATE — is the creation date of the data file.

LEN. — is the number of segments used by the data file.

When using DFU/E or DFU/EL, failure to mount each mass storage device where logical units are assigned will either cause an error message or will cause the system to stop processing until the mass storage device is mounted.

4.1.7 DFU/EL Operating Procedures

DFU/EL lists the expanded table of current logical unit assignments. This is the same as DFU/E except the table is listed on the printer. To execute DFU/EL, type:

```
RUN DFU/EL
```

A table similar to the following is listed on the printer.

```
UNIT   DEV. SEGS.  NAME    SEQ.  DATE     LEN.

  1    RX1  0037   FILE1   1     2/ 4/76  0010
  2    RX1  0002   FILE2   1     1/28/76  0002
  3    RX1  0002   FILE3   1     3/ 1/76  0002
  4    - UNDEFINED -
  5    - UNDEFINED -
  6    - UNDEFINED -
  7    - UNDEFINED -
  8    - UNDEFINED -
  9    - UNDEFINED -
 10    - UNDEFINED -
 11    - UNDEFINED -
 12    - UNDEFINED -
 13    - UNDEFINED -
 14    - UNDEFINED -
 15    - UNDEFINED -
```

where:

UNIT — is the logical unit number.

DEV. — is the three-character designation of the mass storage device where the unit is located.

SEGS. — is the number of segments assigned to each logical unit.

NAME — is the name assigned to the data file.

SEQ. — is the sequence number (what volume in a multivolume file) of that specific data file.

DATE — is the creation date of the data file.

LEN. — is the number of segments used by the data file.

The number of segments in a logical unit is up to you. COS-310 allows one file in each logical unit.

4.2 LOGICAL UNIT ASSIGNMENTS ON THE COS-310 SYSTEM

The assignment of logical units to mass storage devices provides greater utilization of the storage area.

The COS-310 system handles storage using the following hierarchy:

```
2 characters = 1 word
256 words    = 1 block
16 blocks    = 1 segment
41 segments  = RX01 storage capacity
61 segments  = RX02 storage capacity
406 segments = RK05 storage capacity
```

Do not assign logical units to devices not currently part of the system configuration.

4.2.1 Determining Logical Unit Size

The following procedure works when all records within a file are the same size.

The size of a logical unit is dependent upon the record size and the number of records required in the data file. To determine the number of segments required for a logical unit, use the following steps:

Step 1 Determine the number of data characters in a record (maximum of 510 characters). The number must be even. If the number is odd, add one to make it even.

Step 2 COS-310 requires two characters to store the number determined in Step 1; add these two characters to the total from Step 1. This new total is the record size.

Step 3 Determine the total number of characters required in the file by multiplying the record size found in Step 2 by the number of records desired.

Step 4 Add 512 characters for a file header and 2 characters for a file trailer. This plus the total from Step 3 is the total number of characters to be used in the logical unit.

Step 5 Because the logical unit size is expressed in segments rather than in characters, the number from Step 4 must be divided by 8192. Round up any remainder.

The following algorithm can be used to perform this calculation by assigning values to fields DATA and RECS.

```
RECORD
      DATA,     D3   ;Number of data characters per record.
      RECS,     D5   ;Number of records in file.
      FSIZE,    D15  ;Number of characters in file.
RECORD RESULT
          ,     A18,  'TRUE RECORD SIZE: '
      RSIZE,    D3
          ,     A26, ' CHARACTERS   FILE SIZE: '
      SIZE,     D5
          ,     A9, ' SEGMENTS'
PROC
      RSIZE = 2 + ((DATA + 1)/2)*2
      FSIZE = 512 + RECS * RSIZE + 2
      SIZE = FSIZE/8192
      IF (FSIZE.EQ.SIZE*8192)GO TO OK
      SIZE = SIZE + 1
OK    XMIT (8, RESULT)
```

4.2.2 How Logical Units are Assigned by DFU

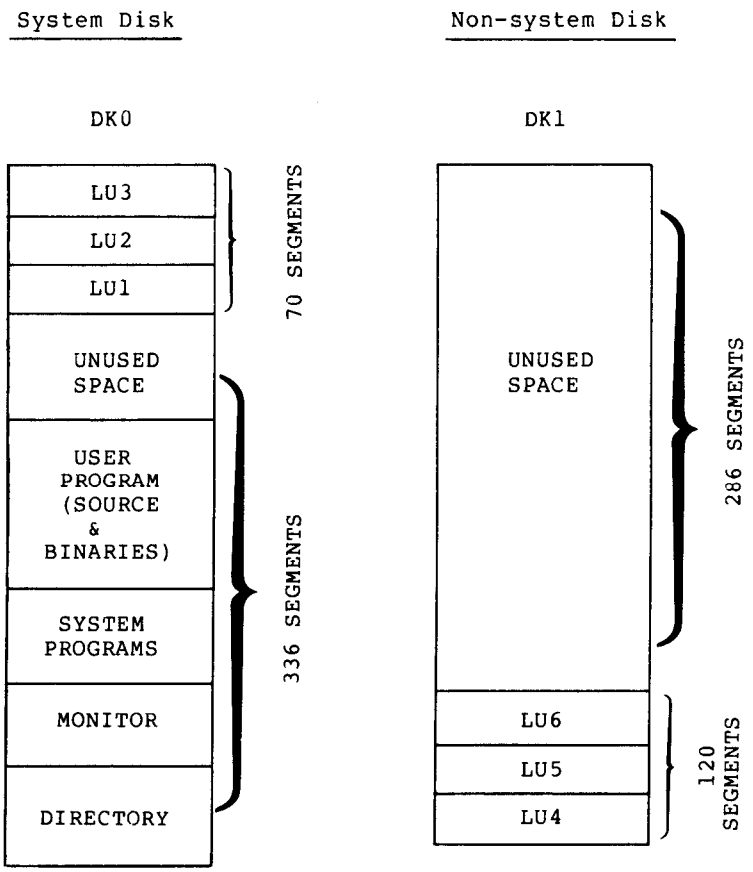
On the system device, logical units are assigned in a pushdown order beginning, at the end of the device. For example, a disk with three logical units would be arranged, starting from the beginning, as: Directory; Monitor; System programs; User programs; Unused space to which new user programs may be added; Logical Unit 1; Logical Unit 2; Logical Unit 3. As more logical units are assigned, Logical Unit 1, Logical Unit 2, etc., move closer to the beginning of the device.

On nonsystem devices, logical units are assigned in sequential order starting at the beginning of the device. For example, a disk with 4 logical units might be arranged: Logical Unit 1; Logical Unit 2; Logical Unit 3; Logical Unit 4; Unused space.

Example:

Logical Unit	Device	Size (Segments)
1	DK0	5
2	DK0	5
3	DK0	5
4	DK1	20
5	DK1	21
6	DK1	20

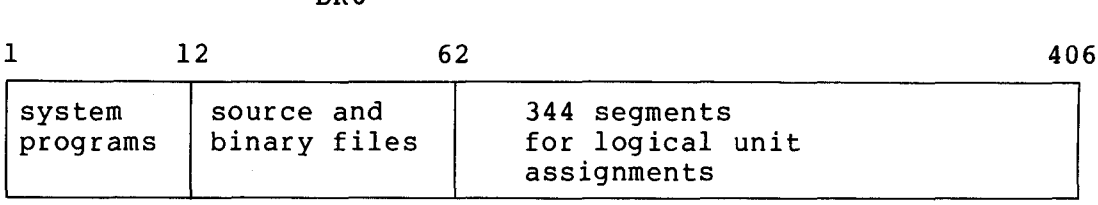
The preceding logical unit assignments would cause an RK05 system disk (DK0) and nonsystem disk (DK1) to be organized as follows:



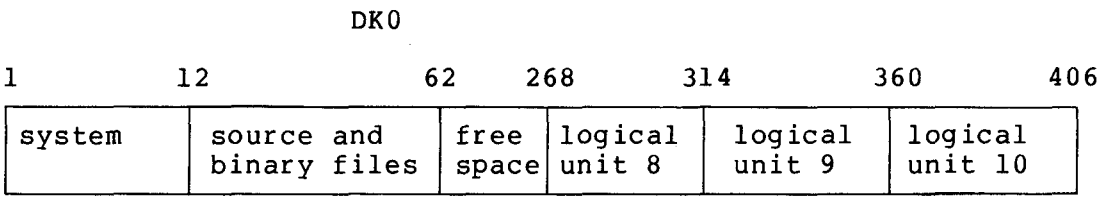
It is advisable to create logical units only slightly larger than the actual data file size since a short file in a large logical unit wastes storage.

4.3 DISK USERS

The RK05 disk cartridges each contain 406 segments. Up to four RK05 drives can be mounted on a system. Approximately 200 blocks (or 12 segments) must be left unassigned to hold the operating system and system programs. In addition, 50 segments should be left to store source programs, control programs, and binary programs. This leaves 344 segments for logical unit assignments.



A sample logical unit assignment might be:



The area between logical unit 8 and the system can be left unassigned for system program overflow.

4.4 DFU ERROR MESSAGES

Message	Explanation
BAD SWITCH	Attempted to use an option switch other than /B, /K, /D, /DL, /E, or /EL. Use only allowable option switches.
ILLEGAL DEVICE	A device other than one usable by the system was designated in the logical unit table. Enter correct device designation.
INSUFFICIENT SPACE ON DEVICE	Attempted to allocate more segments than are available on a device. Either allocate fewer segments, make more segments available, or use a larger device.
NOT ENOUGH ROOM FOR SYSTEM AND FILES	Device designated was too small to hold system program and files. Use PIP OPT- E to put system on one device and files on another.
# TOO LARGE	Number entered was larger than 4095. Enter number smaller than 4095.
SYNTAX ERROR	Missing commas, extra characters, etc. Correct error and reenter.

CHAPTER 5

DIBOL COMPILER (COMP)

The Compiler converts a DIBOL source program into a binary program and reserves storage space for the constants, variables, and statements used by the program.

The Compiler outputs a source program compilation listing and a storage map listing of the records and fields used by the program. Turn on the printer before running the Compiler.

5.1 COMP OPERATING PROCEDURES

To execute the Compiler program, type:

```
RUN COMP[,filnaml...,filnam7][/xx]
```

where:

filnaml...,filnam7
: are file(s) to be compiled into one binary program. If no files are specified, the program in the edit buffer is compiled.

/xx
: is one or a combination of the following option switches:

  /N
  : stops output of the compilation listing and the storage map listing.

  /G
  : compiles the program and, if no errors are detected, executes the binary program; implies /N. The message LOADING is displayed when compiling is successfully completed. If INIT SYS is used in the program, the program must have an END statement to be compiled and executed with the /G option.

  /T
  : enables the TRACE function; implies /G.

/D transfers control to DDT; implies /G.

/O creates a binary program that requires less memory space by eliminating the TRACE feature and accurate error reporting. Execution speed of the compiled program is increased by as much as 20%. This option can be combined with /N or /G.

The /O option saves memory space as follows:

- Saves one location for each executable statement.
- Saves one location for each label.
- Uses one location for each ON ERROR statement.

Use the /O option on thoroughly debugged programs.

Unless the /N or /G option is specified in the RUN COMP command, the Compiler outputs a two-part compilation listing (Data Division and Procedure Division) of the source program and a storage map either on the printer or on the device specified in START, PROC, or END.

The Compiler underscores the number of the line where an error occurs and inserts a caret (^) pointing to the error. Other errors are listed on the storage map. Errors must be corrected before the program can be executed.

The Compiler displays the number of errors as nn ERRORS

5.1.1 Source Program Compilation Listing

```
COS DIBOL    12-JUL-78   WED     COMPILATION LISTING   V 8.00  PAGE 01
          DATA DIVISION   OPTIONAL COMPILATION STATEMENT

0100    START                          ;Optional compilation statement.
0110    RECORD INBUF                   ;Record named INBUF.
0120           STOCKN, D4              ;Numeric field named STOCKN.
0130           DESC, A25               ;Alphanumeric field named DESC.
0140           UCOST, D5               ;Five-character numeric field.
0150           QORDER, D4              ;Four-character numeric field.
0160           , D9                    ;Unreferencable unnamed field.
0170    RECORD OUTBUF                  ;Record named OUTBUF.
0180           , D4                    ;Unnamed numeric field.
0190           , A25                   ;Twenty-five character field.
0200           , D5                    ;Unnamed field.
0210           , D4                    ;Temporary storage field.
0220           ECOST, D9               ;Numeric field named ECOST.
0230    RECORD                         ;Unnamed record-temporary storage
0240                                   ;cannot be directly referenced.
0250           TITLE, A6, 'OVRHED'     ;Field initialized to 'OVRHED'.
```

```
COS DIBOL    12-JUL-78   WED     COMPILATION LISTING   V 8.00  PAGE 02
    PROCEDURE  DIVISION   END DATA DIVISION-BEGIN PROCEDURE DIVISION

0260    PROC                            ;Beginning of Procedure Division.
0270            INIT(1,I,TITLE)         ;Opens TITLE on channel 1-input.
0280            INIT(2,0, 'OUTPUT')     ;'OUTPUT' on channel 2-output.
0290    LOOP,   XMIT(1,INBUF,EOF)       ;Transfer INBUF to EOF.
0300            OUTBUF=INBUF            ;INBUF moved to OUTBUF.
0310            IF(STOCKN.LT.100) GO TO LOOP ;Conditional statement.
0320            ECOST=UCOST*QORDER      ;UCOST times QORDER moved to ECOST.
0340            XMIT(2,OUTBUF)          ;Transfer OUTBUF onto channel 2.
0350                                    ;
0360            GO TO LOOP              ;Branch control to LOOP.
0370    EOF,    FINI (2)                ;Identifies end of logical unit.
0380            FINI (1)                ;Writes record and closes file.
0390            STOP                    ;Stops program execution.
0400    END                             ;Marks the end of the program.
```

5.1.2 Storage Map Listing

```
COS DIBOL   12-JUL-78   WED    STORAGE MAP LISTING     V 8.00  PAGE 03

#         NAME       TYPE        DIM   SIZE     ORIGIN

0001      INBUF      RECORD       01     49      20000
0002      STOCKN     DECMAL       01     04      20002
0003      DESC       ALPHA        01     25      20006
0004      UCOST      DECMAL       01     05      20037
0005      QORDER     DECMAL       01     04      20044
0006      OUTBUF     RECORD       01     49      20062
0007      ECOST      DECMAL       01     09      20132
0010      TITLE      ALPHA        01     06      20146
0011      ..1        DECMAL       01     01      20154
0012      ..2        DECMAL       01     01      20155
0013      ..OUTP     ALPHA        01     06      20156
0014      LOOP       LABEL        00     01      10110
0015      EOR        LABEL        00     01      10144
0016      ..1000     DECMAL       01     04      20164

0014 labels

NO ERRORS DETECTED.   08 K CORE REQUIRED [3956 FREE LOCS -14 BUFFERS]
```

The storage map lists the record and field names and the labels as they were processed by the Compiler. The information is arranged in six columns with the following headings:

contains the internal number of the name in column 2. This number is only used in machine-level programming.

NAME — is the name (field name, record name, program label) or literal used in the compiled program. Literals are numeric or alphanumeric characters which appear in the Procedure Division of the source program. Only the first four characters of a numeric literal are used. Each numeric literal is preceded by two periods (..) to distinguish as an internal name. Numeric literals with four characters or less appear only once on the storage map even though they may occur more than once in the program. Numeric literals with more than four characters are listed each time they occur in the program. Record literals begin with a double quote and end with a single quote.

TYPE — describes the use of name in the program.

- ALPHA — used as the name of an alphanumeric field or as an alphanumeric literal.
- DECMAL — used as the name of a numeric field or as a numeric literal.
- RECORD — used as a record name or as record literal.
- LABEL — used as a program label.
- REDEF — is multiply defined (redefined). All attempts at definition after the first are flagged as errors in the compiler listing.
- UNDEF*** — is an undefined label referenced by the program. For example: GO TO TAG1 in a program where TAG1 does not appear as a label.

  This error is output to the printer even if the /N option is in effect. The line number where the label is used is displayed.

DIM — contains the array dimension (number of fields) of the alphanumeric or numeric labels. The column is meaningless for other types of labels.

SIZE — lists the size of the name. The size of a RECORD is the number of characters in all its labels plus 2.

ORIGIN — gives the octal byte memory address of the name.

The number of labels used, number of errors detected, memory required, and free locations are listed at the bottom of the storage map. You cannot get this information if you suppress listing of storage map.

Maximum number of labels allowed in a 16K-byte system is 365; in 24K-byte or larger systems, 511.

Use the SAVE command to store the binary program.

5.2 CONDITIONAL COMPILATION PROCEDURE (CCP)

The Conditional Compilation Procedure (CCP) is a feature which permits you to include statements in a source program which will be compiled only if you elect to have those statements compiled.

Statements included in a program for conditional compilation are enclosed within angle brackets as in the following example.

```
        RECORD A
B1,      D5
C1,      A4
PROMPT, D1
        RECORD N
NAME, A6
        PROC
<PROMPT
        XMIT(8,"ENTER NAME:')
>
        XMIT(7,N)
        STOP
        END
```

The left angle bracket (<) is followed by a control variable (in this case PROMPT). Unless the control variable is turned on before the left angle bracket is encountered, statements between the angle brackets will be ignored. A right angle bracket marks the end of a conditional area and is on a line by itself. The command to turn on a control variable is as follows:

```
=control variable
```

The above program requires the operator to type in a name on the keyboard. If this same program is recompiled with the control variable PROMPT on, it produces a DIBOL program which first displays a message to the operator.

```
        RECORD A
B1,      D5
C1,      A4
PROMPT, D1
        RECORD N
NAME, A6
        PROC
=PROMPT             ;Turn on prompt.
<PROMPT
        XMIT(8,"ENTER NAME:')
>
        XMIT(7,N)
        STOP
        END
```

Conditional compilation can also be used to debug statements in a source program. Once the program has been tested, the control variable can be removed by deleting the command to turn it on.

CCP also allows several similar (but not identical) programs to be combined into one source program.

If the control variable used in a CCP statement is undefined, the compiler will automatically set aside space for it; this is wasteful of space. For CCP, use variables that are already being used for some other purpose.

The CCP value of a variable (on or off) is independent of the variable's ordinary DIBOL value.

If a CCP variable is used in the middle of a record definition (in the Data Division of a DIBOL program) the variable must have been previously defined, otherwise the Compiler will allocate additional space for it in the middle of a record.

CCP sections can be nested to any depth. Any CCP section that is turned off will be ignored by the Compiler. To indicate that certain statements are not being used, the Compiler listing will not print the line number for that statement. There must be a matching > for each < used. If this condition is not met, the Compiler generates a CCP ERROR message. This error is fatal if angle brackets do not match by the end of the program.

5.3 SIZE OF THE BINARY PROGRAM

Each variable uses as many bytes of memory as specified in its data definition statement. For example: a variable defined as 6D3 requires 18 bytes of storage. This is 9 words since a computer word consists of 2 bytes.

Variables defined in an overlay record share memory with the variables in the record being overlaid.

Each RECORD statement requires one additional word of memory. This word is reserved for storing the COS-310 word count during I/O operations.

Each record begins on a word boundary (an even-numbered byte address). If the record length is odd (in bytes), one byte of memory is wasted.

Each literal used in the Procedure Division of a DIBOL program requires storage (in bytes) equal to the length of the literal. The length of a numeric literal is equal to the number of digits in the number, including leading zeros.

Each distinct literal with a length of four or fewer characters appears only once in the space reserved for literals. Thus, if the literal 32 appears three times in the program, it will appear only once in the reserved data area. However, literals larger than four characters require space each time they appear in the program.

Each statement requires an overhead of one word. Each statement label requires one word. Unlabeled statements with line numbers 1000 more than the previous line number require one additional word each.

The number of words of memory generated by an expression can be determined by the following formula:

Add together the number of variables and literals used.

Add in the number of binary operators which appear. The binary operators include +, -, /, *, #.

Add one for each subscript reference.

The following table shows how many words of code are required by various DIBOL statements.

Table 5-1
DIBOL Statement Words of Code Requirements

Statement	No. of Words of Code Generated
ACCEPT (y,x)	y+x+1
CALL label	1
CHAIN chnum	chnum+1
DISPLAY (line,column,expr)	line+column+expr+1
END [/list control]	1
FINI (channel)	channel+1
FORMS (channel,skipcode)	channel+skipcode+1
GOTO label	1
GOTO (label1,...,labeln),key	key+n+2
IF (expr1.rel.expr2)stmnt	expr1+expr2+3
INCR var	var+1
INIT (channel, dev)	channel+2
INIT (channel, dev,filnam[,unit])	channel+3+filnam+unit
ON ERROR label	1
PROC [n] [/list control]	0
READ (channel,record,number)	channel+number+record+1
RETURN	1
START [/list control]	0
STOP	1
[NO] TRACE	1
TRAP	2
var=	var+1
var=expr	var+expr+1*
var=expr1,expr2	var+expr1+expr2+1
WRITE (channel,record,number)	channel+number+record+1
XMIT (channel,record[,label])	channel+record+2

For the statement marked with an asterisk (*) in the previous table, subtract 1 if the principal operator of expr is binary + or -, and if both types are numeric.

Example:

```
D = 3+5       takes 4 words of storage, while
D = 3*5       takes 5 words. Similarly,
D = 3+4+5     takes 6 words while
D = 3*(4+5)   takes 7 words.
```

Additional space is also required by the internal symbol table. This table consists of two words for each distinct variable, statement label, or literal used.

5.4 COMPILER ERROR MESSAGES

Most Compiler error messages are printed on the source listing directly after the line on which the error occurs. A caret (^) in the error message points to the approximate location of the error. Other errors are listed in the storage map listing.

Message	**Explanation**
BAD ALPHA VALUE	Initial value in an alphanumeric data definition statement did not begin or end with a single quotation mark. Insert single quotes.
BAD NUMERIC VALUE	The initial value for a numeric field was incorrectly formed. Check and form correctly.
BAD PROC #	The number in a PROC statement was not a digit from 0 to 7. Enter 0 through 7.
BAD RELATIONAL	An illegal relational occurs in an IF statement. For example, a .GX. instead of a .GT. Retype correctly.
CCP ERROR	Matching angle bracket (< or >) missing. Insert brackets.
COMMA MISSING	No comma appeared where one was expected. Insert comma.
DATA INITIALIZATION MISSING	No data initialization followed a comma in a data definition statement. Remove comma or input initial value.

Message	Explanation
EXPECTED LABEL IS MISSING	A required label is missing. Enter label.
EXPRESSION NOT ALLOWED	A complex expression or bad character occurs to the left of an = or where only a variable is allowed. Find and correct.
EXTRA CHARS AT STMNT END	Extra characters occur at the end of a legal statement. Eliminate extra characters.
FIELD TOO LARGE OR 0	In a data description statement, the dimension was 0 or more than 3 digits long, or the field size was 0 or larger than 511. Bring size dimensions within limits.
ILLEGAL OPERATOR	A bad character was encountered in an expression where an operator would be expected. Check and replace with correct character.
ILLEGAL STMNT	The statement was not a data manipulation statement (it had no =) nor did it start with a recognizable keyword. Use appropriate keyword and use = sign.
INITIAL VALUE WRONG SIZE	The initial value in a data specification statement had a length different from the field size specified. Make initial value agree with defined size.
LABEL NOT ALLOWED	A label in an expression was the wrong type or a record or a label which had been redefined was used. Use unique label of the correct type.
MISSING CLOSE PAREN	No close parenthesis occurred where one was expected. Add parenthesis.
MISSING OPEN PAREN	No open parenthesis occurred where one was expected. Add parenthesis.
MISSING OPERAND	A binary operator occurs in an expression with no operand following it; or no expression at all occurs where one is expected. Insert operand and/or appropriate expression.

Message	Explanation
MISSING OR BAD MODE	The mode designation in an INIT statement was missing or began with an illegal character. Insert correct mode designation.
MISSING QUOTE	The statement contained an odd number of quotes ('). Delete or add quote when appropriate.
MISSING RELATIONAL	No relational appeared in an IF statement. Enter legal relational.
NAME PREVIOUSLY DEFINED	An attempt was made to redefine a previously defined name. Use unique name.
NOT A OR D	A character other than A or D occurred in a data specification statement where A or D was expected. Replace character with A or D.
NOT LABEL	A symbol which was not a 'label' occurred where a label was required. Enter proper label.
PROGRAM TOO BIG	Binary output too big for the binary scratch area. Enlarge scratch area with PIP OPT- E.
RECORD TOO BIG	A named record exceeded 510 characters in size. Either use unnamed record or reduce the size of record.
STMNT TOO COMPLEX	The statement is too complex or is nested too deep. Simplify the statement.
SUBSCRIPT ERROR	No comma or close parenthesis occurred after a subscript. Enter appropriate punctuation.
SUBSCRIPT NOT NUMERIC	The type of a subscript was not numeric. Use numeric subscript.
TOO MANY ITEMS	More elements were initialized in an array than are specified in the field dimension. Eliminate excess elements.

Message	Explanation
TOO MANY SYMBOLS!	A fatal error message. Only 365 symbols allowed in symbol table in 16K-byte system, and only 511 symbols allowed in larger systems. The compiler stops compiling; no storage map can be produced. Rewrite and shorten program.
TOO MUCH DATA	Data Division exceeds 24K bytes. Rewrite program.
UNDEFINED NAME	A name is used which was not defined in the Data Division. Define this name or use a name already defined.
WRONG DATA TYPE	Mixed data types occurred in an expression, an argument which was supposed to be numeric was not, or one of the arguments in a data manipulation statement was of the wrong type. Replace the data.

CHAPTER 6

DIBOL DEBUGGING TECHNIQUE (DDT)

The DIBOL Debugging Technique (DDT) is used to debug binary programs. If a program is compiled with the DDT option (/D), the compiled binary program automatically branches to DDT upon execution. The features of DDT include breakpoint, variable examination, subroutine call trace-back, and iteration.

6.1 DDT OPERATING PROCEDURES

To execute a binary program with DDT, type:

```
RUN [ { pronam                    } ][,filnaml...,filnam7]/D
    { chain0+chainl...+chain7 }
```

where:

pronam is the name of the binary program to be debugged.

If the program name is omitted, the Monitor loads and executes the DIBOL program in the binary scratch area.

chain0+chainl...
are binary programs which constitute one large program broken up into several chained programs. These are the programs to be debugged.

filnaml...,filnam7
are names of source files on the system device.

/D is the option switch that requests DDT.

An additional 768 words of memory plus 3 words for each label in the Data Division are required because of the /D option.

During execution of the program, control is passed to DDT. The DDT program displays an appropriate DDT version number followed by a hyphen (-) to indicate that it is ready to accept commands.

6.2 DDT COMMANDS

Command	Explanation
variable=	Display the contents of variable (a label from the Data Division). Variable can have single or double subscripts.
variable=v	Set variable equal to v (v is any legal alphanumeric string). If v has more characters than defined for variable, ERR IN CMD is displayed.
=	Display the contents of the last variable examined.
=v	Set the last variable examined equal to v.
$nnnn	Set a breakpoint at line nnnn. One breakpoint is active at a given time. A breakpoint set at line 0 is meaningless because the program never executes line 0.
>n	Execute the breakpoint at the nth occurrence of line nnnn. For example: -$300 ->4 When the program starts to execute line 300 for the fourth time, the breakpoint is executed and control is transferred to DDT.
CTRL/Z	Start execution of DIBOL program. If a breakpoint, $, is set at line number nnnn, control reverts to DDT when nnnn is reached and the following message is printed: BREAK! Type additional commands in response to the hyphen (-).

Display the lines from which calls (CALL or TRAP commands) were made (pushdown stack) during execution of the DIBOL program. This command is generally used to trace the execution of the program after a breakpoint or system error has occurred. The is usually the shift/six key.

While a DDT breakpoint is pending, if a DIBOL program error causes a message such as ILLEGAL SUBSCRIPT or NUMBER TOO LONG to appear, control is transferred to DDT; DDT commands can be used for program examination. If an error is fatal, the DIBOL program cannot be restarted by the CTRL/Z command.

Once a DIBOL program is running under DDT, DDT cannot be restarted unless a breakpoint occurs or an error occurs with a breakpoint pending. Therefore, if you do not require a breakpoint but want to return to DDT for program examination if an error occurs, set a breakpoint at a line number which will not be executed.

6.3 DDT ERROR MESSAGES

Message	Explanation
ERR IN CMD	Entered an invalid DDT command. Correct the command and retry.

CHAPTER 7

CROSS REFERENCE PROGRAM (CREF)

The Cross Reference Program (CREF) is primarily an aid to program development. It provides a table showing an alphabetical listing of all labels used in a DIBOL source program, the line number where each label is defined, and the line numbers where each label is used.

7.1 CREF OPERATING PROCEDURES

To execute CREF, type:

```
RUN CREF[,filnam1...,filnam7]
```

where:

filnam1...,filnam7
: are the parts of a DIBOL source program (maximum 7). If no files are specified, the program in the edit buffer is used.

The CREF program reads the DIBOL program, lists the cross-reference table on the line printer, and returns control to the Monitor.

CREF requires 16K bytes of memory and can handle any 16K-byte program that does not have an excessive number of symbols and labels. If 24K bytes or more is available, CREF expands its cross-reference table to make use of the available space.

A minimal amount of error checking is performed by CREF; no attempt should be made to cross reference programs having compilation errors. If CREF finds a line it cannot work with, it prints:

```
nnnn IS BEING IGNORED
```

where:

nnnn
: is the number of the line CREF cannot work with.

Following is the cross-reference table for the DIBOL program in Figure 1-1 of Chapter 1.

```
COS-310   CREF          V 8.00   24-MAY-78     WED          PAGE  1

LABEL     DEF       REFERENCES

DESC      130
ECOST     220       340
EOF       390       310
INBUF     110       310 320
LOOP      310       330 380
OUTBUF    170       320 360
QORDER    150       340
STOCKN    120       330
TITLE     260       280
UCOST     140       340

LABELS DEFINED BUT NEVER REFERENCED:  01
```

where:

LABEL is the name of the label used in the program.

DEF is the line number in the program where the label is defined.

REFERENCES are the line numbers where each label is referenced.

7.2 CREF ERROR MESSAGES

Message	Explanation
nnnn IS BEING IGNORED	CREF detected a line it cannot work with. Usually means an invalid DIBOL statement. Check line number for valid statement and retry.

CHAPTER 8

PERIPHERAL INTERCHANGE PROGRAM (PIP)

The Peripheral Interchange Program (PIP) moves files between two logical units, copies the contents of one device onto another, and consolidates files to remove free blocks. PIP is also used to allocate more space to the binary scratch area.

8.1 PIP OPERATING PROCEDURES

To execute PIP, type:

```
RUN PIP[,cmndfl][/n]
```

where:

,cmndfl is a previously stored file containing PIP commands. Each command is on a separate line; no blank lines or comments are used. When the command file is specified, PIP reads a line from the file each time one of the following prompts is displayed:

```
OPT-
IN-
OUT-
MORE?
TYPES OF FILES TO BE SKIPPED (S,B,V):
```

An end-of-file mark terminates the command file and requires all responses to come from the keyboard.

The command file is ignored on machines with less than 24K bytes of memory capabilities.

Example:

If the cmndfl EXAMP contains:

```
0100  C
0110  RX0
0120  RX1
0130  X
```

Then:

```
.RUN PIP,EXAMP     ;will copy RX0 to RX1.
```

/n indicates the number (0-9) of segments to allocate to the binary scratch area. The /n switch is used in conjunction with OPT- E, but is entered at the time that the RUN PIP command is typed.

PIP responds to the RUN PIP command with:

```
PIP V 8.00 (or current version number)
OPT-
```

Respond with one of the following options:

Option	Explanation
B	transfer a binary file
C	copy device
D	transfer a data file
E	consolidate directory space
I	copy and verify data
R	perform a read/check
S	transfer source file
V	transfer system file
X	return to Monitor

After you respond, PIP displays IN and OUT questions requesting option-dependent information.

Following is a summary of the PIP options and the information being requested by the IN and OUT questions.

OPT	IN	OUT
B	filnam[,dev]	filnam[,dev]
C	dev	dev
D	filnam[/logical unit #] /K	filnam[/logical unit #] /L /T
E	dev	dev
I	dev	dev
R	dev	
S	filnam[,dev]	filnam[,dev]
V	filnam[,dev]	filnam[,dev]

8.1.1 Transfer Binary File (OPT- B)

Type B in response to OPT- to move a binary program between two file-oriented devices.

Answer the IN question with the name of the binary program to be moved and, optionally, a comma and an input device designation. If no device is designated, the system device is assumed.

Answer the OUT question with the name to be assigned to the output file and, optionally, a comma and an output device designation. If no device is designated, the system device is assumed.

If you attempt to move data to or from a non-file-oriented device, the IN or OUT message is repeated.

Example:

```
.RUN PIP
PIP V8.00
OPT- B          ;Request move of binary file.
IN- TEST,DK0    ;File named Test from DK0.
OUT- TEST,DK1   ;File named Test to DK1.
OPT- X
```

8.1.2 Copy Device (OPT- C)

Type C in response to OPT- to copy the contents of one device onto a similar device.

Answer the IN question with a device designation.

Answer the OUT question with a device designation.

Example:

```
.RUN PIP
PIP V8.00
OPT- C          ;Request a copy between devices.
IN- DKO         ;Input device is DK0.
OUT- DK3        ;Output device is DK3.
OPT- X
```

8.1.3 Transfer Data Files (OPT- D)

Type D in response to OPT- to transfer data files between devices.

Answer the IN question with a file name and optionally, a logical unit number (1-15) preceded by a slash, or answer the IN question with /K.

Answer the OUT question with a file name and optionally, a logical unit number preceded by a slash, or answer the OUT question with a device switch. Output device switches are:

```
/L   printer
/T   screen
```

When the end of the input file is reached, PIP asks:

```
MORE?
```

Type N and the RETURN key if there is no more input or Y and the RETURN key to specify more input.

PIP transfers alphanumeric data only. A negative number is treated as the letter which has the equivalent code.

Examples:

```
.RUN PIP
PIP V8.00
OPT- D
IN- EMPNAM/1    ;Dump EMPNAM from logical unit 1 onto the printer.
OUT- /L
MORE? N
OPT- X

.RUN PIP
PIP V8.00
OPT- D
IN- HRPAY,2     ;Combines two data files into one output file.
OUT- PAYFIL/1
MORE? Y
IN- SALPAY,3
MORE? N
OPT- X
```

8.1.4 Consolidate Space in Directory (OPT- E)

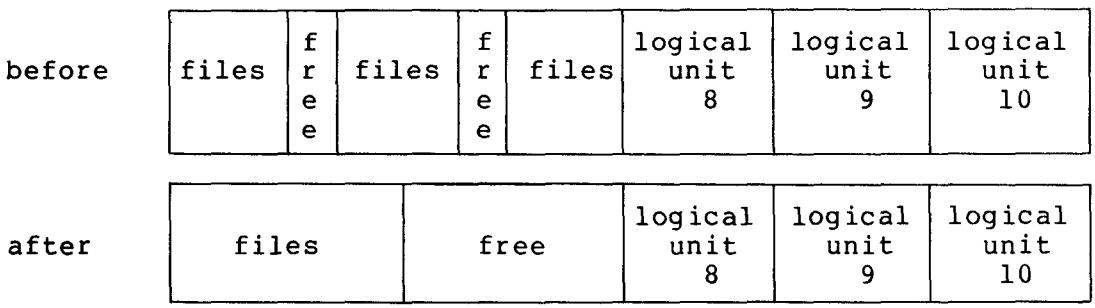
Type E in response to OPT- to consolidate the free blocks on the input device and store the files on the output device. It is possible to erase one or two of the kinds of files (source, binary, or system) during the consolidation. Free blocks are shown in the file directory and are created when a file is deleted from the directory. The bootstrap, Monitor, and logical units are not copied by OPT- E.

Answer the IN question with a device designation.

Answer the OUT question with a device designation.

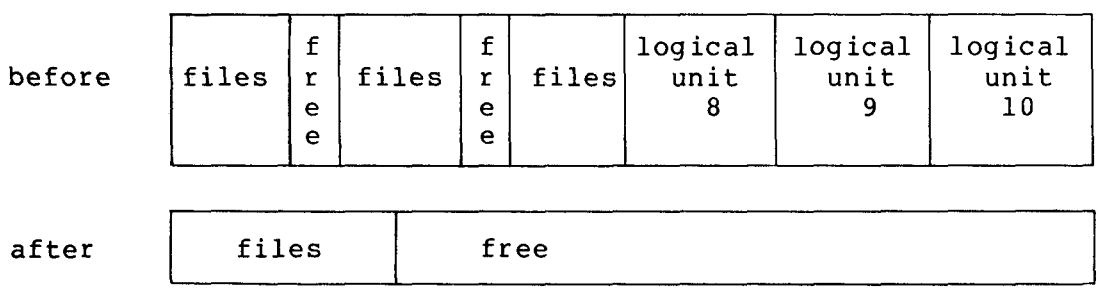
When consolidating the system device onto itself, PIP OPT- E eliminates the free space as shown below:

SYSTEM DEVICE (SYS)



When consolidating a device other than the system device onto another device, PIP OPT- E consolidates the free space but does not copy logical units.

NONSYSTEM DEVICE



The following message asks which files you do not want copied, consolidated, and stored. The files you choose to skip will be erased.

```
TYPES OF FILES TO BE SKIPPED (S,B,V):
```

If all files are to be copied, consolidated, and stored, type the RETURN key. If one or two of the types of files are to be skipped (erased), type one or a combination of two of the characters S, V, B separated by a comma and followed by the RETURN key.

The following example will consolidate free space but will not copy source files from DK0 to DK1.

Example:

```
.RUN PIP
PIP V 8.00
OPT- E
IN- DK0
OUT- DK1
TYPES OF FILES TO BE SKIPPED (S,V,B): S
```

All CTRL keys are ignored until the PIP OPT- E consolidation operation is completed.

8.1.5 Allocate Space to Binary Scratch Area (OPT- E)

The PIP OPT- E is used in conjunction with /n to change the size of the binary scratch area. The /n is typed along with the RUN PIP command.

Extremely large DIBOL programs may need more space than available in the two segments (32 blocks) usually allocated to the binary scratch area. Up to nine segments can be allocated with PIP OPT- E.

The following PIP operation will allocate two additional segments (32 blocks) to the binary scratch area on DK0. This particular operation uses /2 as the /n option typed in the RUN PIP command.

```
.RUN PIP/2
PIP V 8.00
OPT- E
IN- DK1
OUT- DK0
TYPES OF FILES TO BE SKIPPED (S,V,B):
```

The following information is helpful when using OPT- E to change the size of the binary scratch area.

- If the output device is not the input device, the size of the binary scratch area on the output device equals the sum of the two segments normally in the binary scratch area plus the number of segments stipulated by the /n in the RUN PIP command.

- If a device is consolidated onto itself, the binary scratch area is set either to the size of the current area or to 2+n, whichever is less. Compressing a device onto itself can shrink the binary scratch area. The binary scratch area cannot be expanded if a device is being compressed onto itself because that would require writing over existing files.

- If /n is not specified in the RUN PIP command, the binary scratch area is assumed to be the same size as the binary scratch area of the input device.

8.1.6 Copy and Verify (OPT- I)

Type I in response to OPT- to copy an entire device onto a similar device and verify the copy.

Answer the IN question with a device designation.

Answer the OUT question with a device designation.

Example:

```
.RUN PIP
PIP V8.00
OPT- I          ;Request copy and verify.
IN- DK1         ;Input device is DK1.
OUT- DK2        ;Output device is DK2.
OPT- X
```

If your machine configuration includes an LQP printer, PIP OPT- I will require 32K bytes of memory.

8.1.7 Perform a Read/Check (OPT- R)

Type R in reply to OPT- to verify the readability of a device.

Answer the IN question with the designation of the device to be read.

No OUT question is displayed.

Example:

```
.RUN PIP
PIP V8.00
OPT- R          ;Request Read/Check.
IN- DK1         ;Read contents of DK1.
OPT- X
```

8.1.8 Transfer Source Files (OPT- S)

Type S in response to OPT- to transfer source files between two file-oriented devices.

Answer the IN question with the name of the source file to be transferred and, optionally, a comma and the input device designation. If no device is specified, the system device is assumed.

Answer the OUT question with the name to be assigned to the output file and, optionally, a comma, and the output device designation. If no device is specified, the system device is assumed.

Example:

```
.RUN PIP
PIP V8.00
OPT- S          ;Request transfer of source file.
IN- TEST,DK0    ;Transfer TEST from DK0.
OUT- TEST,DK1   ;Receive TEST into DK1.
OPT- X
```

If you attempt to transfer to or from a non-file-oriented device, the IN or OUT question is repeated.

8.1.9 Transfer System Program (OPT- V)

Type V in response to OPT- to move a system program between two file-oriented devices.

Answer the IN question with the name of the system program to be transferred and, optionally, a comma and the designation for the input device. If no device is specified, the system device is assumed.

Answer the OUT question with the name to be assigned to the output file, and, optionally, a comma and the output device designation. If no device is specified, the system device is assumed.

Example:

```
.RUN PIP
PIP V8.00
OPT- V          ;Request transfer of system program.
IN- SORT,DK1    ;Transfer SORT from DK1.
OUT- SORT,DK3   ;Transfer SORT to DK3.
OPT- X
```

If you attempt to transfer to or from a non-file-oriented device, the IN or OUT question is repeated.

8.1.10 Return to Monitor (OPT- X)

Type X in response to OPT- to terminate PIP and return to the Monitor.

This OPT- X feature is useful when PIP is included in a string of Monitor commands in a BATCH program. The OPT- X signals the end of the PIP program and the next Monitor command in the BATCH program is executed.

8.2 PIP ERROR MESSAGES

Message	Explanation
BAD DIRECTORY	Attempted to reference or store a file on a device with a damaged or nonexistent directory. Only files with directories can be used. If the directory is damaged, call your Software Specialist.
COMPARISON ERROR	The verification part of OPT- I found a discrepancy of information between the original text and its copy. Retry the operation. If discrepancies continue, you have a media problem or a hardware problem.
ILLEGAL DEVICE SWITCH	A switch was specified for OPT- D that was not /K for input or /L or /T for output. Use one of the allowable switches.
NO ROOM	Attempted to store a file on a full device. Stipulate another device.

CHAPTER 9

SORT PROGRAM (SORT)

SORT is a utility program that arranges the records within COS-310 data files according to your needs. The data files must contain fixed-length records.

Before you can execute SORT you must write a SORT command file that defines the records to be sorted, specifies labels for input and output files, and designates the arrangement (key) to be used in the sort.

SORT uses a command file to rearrange one data file at a time. There must be a separate SORT command file for each file sorted.

The SORT command file sorts each volume of a multivolume file separately. A merge pass must then be done to combine the volumes into one file.

9.1 SORT OPERATING PROCEDURES

To execute SORT, type:

```
RUN SORT,cmndfl1...,cmndfl7[/L]
```

where:

cmndfl1...,cmndfl7
: is the SORT command file which can be stored as more than one file. This file defines the records to be sorted, specifies the labels for input and output files, and designates the arrangement (key) to be used. If no files are specified, the command file in the edit buffer is used.

/L
: lists the SORT command file on the printer.

9.2 SORT COMMAND FILE

The SORT command file is created with editor commands and written on a mass storage device. The command file consists of a Record Descriptor Division and an INPUT/OUTPUT Division.

9.2.1 Record Descriptor Division

The Record Descriptor Division defines the fields within the records to be stored. This division has the form:

```
DEFINE
Fs, xn
.
.
.
```

where:

DEFINE — is the division heading (must be DEFINE) and is the first statement in the file.

Fs — are the fields in the record (must be F). All fields must be defined in the Record Descriptor Division and numbered in the order they appear in the record. These numbers (s) begin with 1, are nonskipped sequential, and cannot exceed 511. Total record size cannot exceed 510 characters.

xn — represents the field type (alphanumeric or numeric), and the number of characters (1-510) in the field.

Each field descriptor statement (Fs) is entered on consecutive lines and is terminated with the RETURN key.

9.2.2 INPUT/OUTPUT Division

The INPUT/OUTPUT Division specifies the names of input and output files and how many logical units are to be used for work areas during the SORT operation. This division has the following format:

```
INPUT [filnam][/logical unit #][,filnam][/logical unit #]
[SORT n /logical unit #,...logical unit #]
KEY Fs[(m,n)][-],...
OUTPUT [filnam][/logical unit #]
END
```

where:

INPUT [filnam][/logical unit #][,filnam][/logical unit #]

is the name of the file containing the records to be sorted. If no name is specified, SORTIN is assumed. If the command file is used for a separate merge operation, the second file name is the name of a file to be merged with the first. The logical units identify the storage location of the file.

[SORT n /logical unit #...,logical unit #]

is the number (n) of logical units (3 to 7) to be used as work areas during the sort. These work areas are labeled $WORK1, $WORK2...,etc. If the SORT statement is not present, 4 units are assumed. The size of the work units should be as large as one volume of the input file. The logical unit numbers are default work units. Using default units bypasses the MOUNT message.

KEY Fs[(m,n)][-],...

Fs is the field name (F) and number(s) of the field to be used as the SORT key. The [(m,n)] delimit the part of the field to be used as a SORT key. If no characters are specified, the entire field is used as a SORT key. The - requests a SORT in descending order. Up to eight fields or parts of fields can be specified for the SORT key. The total size of the fields which comprise the key cannot be larger than 510 characters. The SORT is done left to right. The leftmost key is most significant, and the leftmost character in each key field is most significant for sorting purposes.

OUTPUT [filnam][/logical unit #]

is the file name to be given to the sorted records. If this statement is missing, SORT assigns the name SRTOUT to the output. For multivolume files, the names $TMPnn (nn can be any two-character numeric from 00 to 99) are used. The /logical unit number is the default unit for the output file.

END terminates control program.

Following is an example of a SORT command file:

```
0010   DEFINE
0020   F1,D6            ;Part number.
0030   F2,A30
0040   F3,D7
0050   F4,D6            ;Date.
0060   F5,D6
0070   INPUT PRTFIL/1   ;Data file name.
0080   SORT 4/2,3,4,5   ;Work units.
0090   KEY F1-,F4       ;Sort part numbers in descending order,
                        ;Sort date in ascending order.
0100   OUTPUT PRTFIL/1  ;Sorted data file name.
0110   END
```

9.3 MERGE OPERATING PROCEDURE

To execute SORT as a merge operation, type:

```
RUN SORT,cmndfl1...,cmndfl7/x[L]
```

where:

cmndfl1...,cmndfl7
: is the command file (possibly stored in two or more files).

/x
: is one of the following switches:

  /A
  : names of files to be merged are entered from the keyboard in answer to the message INPUT FILE LABELS:. The output data file and default unit name are specified in the OUTPUT line of the command file.

  /M
  : names of files to be merged are listed in the INPUT line of the SORT command file. This option bypasses the message INPUT FILE LABELS:.

  /n
  : name of files to be merged (all files must have the same name) is listed in the INPUT line of the SORT command file. This checks for the number of files with the same name on the number of default units as specified. If the number of units specified is more than the number of units shown on the SORT control INPUT line, a MOUNT message is displayed for those files not on the INPUT line.

/L
: can optionally be used with any of the above switches and lists the SORT control program on the printer.

9.3.1 Merge Using SORT and the /A Option

To use the SORT program with the /A option to merge data files, first write a SORT command file.

Following is a sample command file named PAYKEY:

```
0100   DEFINE
0110   F1,A6
0120   F2,D5
0130   F3,D11
0140   INPUT
0150   SORT 3/1,2,3
0160   KEY F2
0170   OUTPUT PAYROL/6
0180   END
```

To execute this sample command file with the SORT program and the /A option, type:

```
·RUN SORT,PAYKEY/A
```

The program displays the following message to request the names of the data files to be merged:

```
INPUT FILE LABELS:
```

Enter up to a maximum of six data file names and default units. Enter at least two names or the error message NO INPUT is displayed.

After you enter the file names and default units, the program is executed. There are three SORT work units: logical units 1, 2, and 3. The output data file is PAYROL; the sorted file is stored on unit 6.

9.3.2 Merge Using SORT and the /M Option

To use the SORT program with the /M option to merge data files, first write a command file.

Following is a sample command file named PAYKEY:

```
0100   DEFINE
0110   F1,A6
0120   F2,D5
0130   F3,D11
0140   INPUT PAYROL/4, PAY1/2
0150   SORT 3/1,2,3
0160   KEY F2
0170   OUTPUT PAYROL/6
0180   END
```

To execute this sample command file with the SORT program and the /M option, type:

```
.RUN SORT,PAYKEY/M
```

The input data file names to be merged are found in the INPUT line of the control file:

```
PAYROL on logical unit 4
PAY1 on logical unit 2
```

The output data file, PAYROL, is put on logical unit 6.

9.3.3 Merge Using SORT and the /n Option

To use the SORT program with the /n option to merge data files with the same name, first write a command file. The INPUT line contains the name common to the files and the default units where the files are found.

Following is a sample command file named PAYKEY:

```
0100   DEFINE
0110   F1,A6
0120   F2,D5
0130   F3,D11
0140   INPUT PAYROL/4,2
0150   SORT 3/1,2,3
0160   KEY F2
0170   OUTPUT PAYROL/6
0180   END
```

To execute this sample command file with the SORT program and the /n option, type:

```
.RUN SORT, PAYKEY/2
```

The input data files to be merged are:

```
PAYROL on logical unit 4
PAYROL on logical unit 2
```

9.4 SORT ERROR MESSAGES

Message	Explanation
BAD DIGIT IN NUMERIC INITIAL VALUE	Alphanumeric character in a numeric initial value. Use only numeric characters in initial numeric values.
BAD RECORD SIZE	File contains records of variable length. All records to be stored must be the same length. Redefine record length.
BAD WORK UNIT COUNT	Number of work units not in range 3-7. Specify work units within allowable range.
EXTRA CHARS AT STMNT END	Characters not relating to statement appear on the statement line. Delete any nonessential characters from statement lines.
FIELD NUMBER MISSING OR 0	Field number is missing, is 0, or is greater than or equal to 512. Enter the missing number or enter a number between 1 and 511.
ILLEGAL SORT KEY	Bad syntax on KEY statement, KEY too complex, or KEY statement missing. Review key information and correct command file.
ILLEGAL UNIT	Default unit is 0 or greater than 15. Correct command file.
INITIAL ALPHA VALUE DOESN'T BEGIN WITH QUOTE	Beginning quotation mark missing for initial alphanumeric value. Put single quotation at the beginning of the initial value.
INITIAL VALUE TOO BIG	The initial value specified is larger than the field size. Either define a larger field size or reduce the size of the initial value.
INITIAL VALUE TOO SMALL	The initial value specified is smaller than the field size. Either redefine the field size or increase the size of the initial value.

Message	Explanation
MISSING CLOSE QUOTE ON ALPHA INITIAL VALUE	Quotation mark not specified at the end of an alphanumeric initial value. Add the missing close quotation marks.
MISSING INITIAL VALUE	Comma was inserted after type and size but initial value was not specified. Either delete comma or insert initial value.
NO COMMA AFTER FIELD NAME	No comma or a character other than comma was specified after the field name. Enter the missing comma or delete incorrect character and then enter the comma.
NO INPUT	Input file is null or not enough input files specified for a merge. Two files are needed to execute a merge. Use two nonempty files.
NOT A OR D	A character other than A or D occured in a data specification statement where A or D was expected. Correct the command file.
NOTHING AFTER FIELD NAME	Field type and size are not specified after field name and comma. Correct the command file.
NUMBER REPEATED OR OUT OF ORDER	A field sequence number is used more than once or is out of ascending order sequence. Correct the command file.
OUTPUT ERROR	Indicates an I/O error. Start SORT over. If it continues, check for media or hardware problem.
TOO MANY FILES	Merge only, more than 6 input files specified. Specify no more than six files for merge command file.
UNIT xx IS FREE	This is not an error. It is an informative message showing the logical units that are free. xx is a COS logical unit number.
UNRECOGNIZABLE LINE	Parameter line did not start with a good keyword. Correct the command file.

CHAPTER 10

FILE EXCHANGE PROGRAM (FILEX)

The File Exchange Program (FILEX) transfers files between diskettes in universal format and any COS-310 file storage device or any OS/8 file on an RK05 disk (OS/8 files cannot be transferred to or from diskettes).

Files are transferred in one of three formats: ASCII, IMAGE, or EBCDIC (universal format). The EBCDIC format is compatible with diskettes produced by an IBM 3741 except when using multivolume universal interchange files or when mapping bad sectors.

10.1 UNIVERSAL DISKETTE

A universal diskette contains 77 tracks (some of which cannot be used for data). Each track has 26 sectors numbered from 1 to 26. A sector contains one record of 128 characters or less. (See Figure 10-1 for a visual representation of a universal diskette.)

A record in a COS-310 file is assumed to be a string of characters preceded by a word count and independent of sector boundaries. A record on a universal diskette in EBCDIC format must begin on a sector boundary and only one record is allowed per sector. If the record does not fill a sector, the remainder of the sector is filled with blanks. Since these restrictions make EBCDIC format inefficient and wasteful of space, only use EBCDIC when you must read or write diskettes compatible with IBM 3741 format.

Track 0 of the universal diskette contains the information which describes the files on the diskette. Each of the 26 sectors on a diskette has a specific function.

Sector	Function
1-6	Reserved.
7	Identifies the diskette format. If bytes 0-3 contain VOL1 in EBCDIC characters, the diskette is assumed to have a universal interchange format directory. The remainder of sector 7 contains other information which FILEX does not use.
8-26	Contain the labels or the directory entries. These sectors contain information such as record length (up to 128 characters) and creation date. For further details on these sectors refer to the IBM manual, form number GA21-9128-0. Each byte in sectors 8-26 has a special function.

Bytes	Function
0-3	Are for label identification and contain HDR1 (DDR1 if the file has been deleted).
6-13	Contain the file name.
23-27	Specify the record length.
29-30	Contain two EBCDIC characters which identify the track number at the beginning of the data.
31	Must be EBCDIC 0 (360 octal).
32-33	Contain two EBCDIC characters which identify the sector number at the beginning of data.
35-36	Contain the number of the last track reserved for this file. Byte 37 must be EBCDIC 0.
38-39	Contain two EBCDIC characters which identify the number of the last sector reserved for this file.
48-53	Contain creation year, month, and day.

Bytes	Function
75-76	Contain the track number.
77	Must be EBCDIC 0.
78-79	Contain the number of the next unused sector.

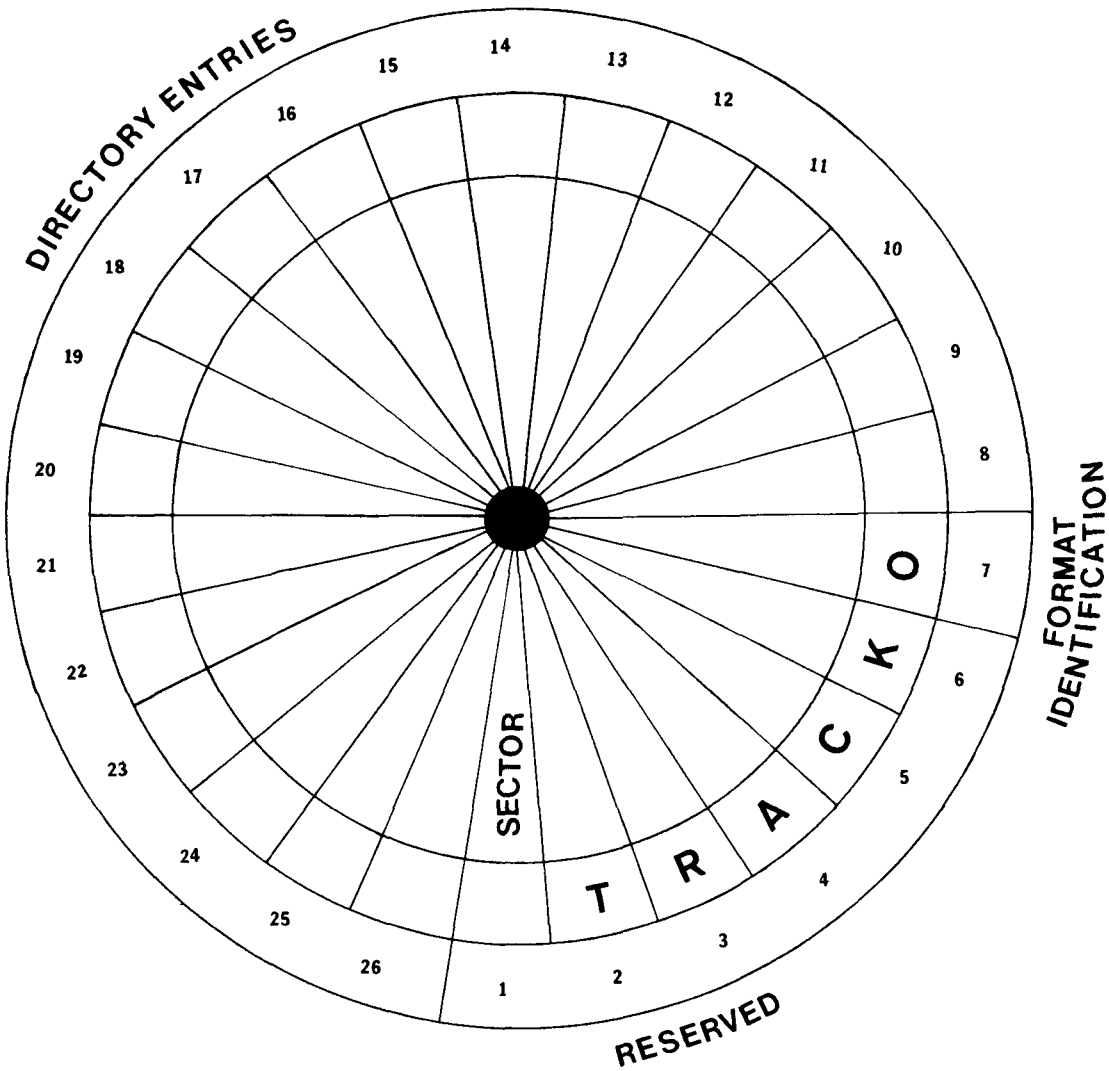


Figure 10-1 Universal Diskette

COS-310 character codes are never used in a universal interchange file. All data on a universal diskette is stored in either 7-bit ASCII or 8-bit EBCDIC. ASCII format is equivalent to data written as a continuous string of bytes ignoring sector boundaries; records are terminated by a carriage return/line feed. The first record of a file must begin on a sector boundary. All character code translation and record blocking is done implicitly by FILEX and need not be explicitly specified.

File name extensions are not normally recognized in universal interchange format; instead, a single eight-character file name is used. In order to provide some degree of compatibility with OS/8, FILEX has been designed to accept a six-character file name with a two-character extension. If a file name on a universal diskette has more than six characters, it must be entered in the format of filnam.ex. File names must not include spaces anywhere within the file name or between it and the extension.

10.2 FILEX OPERATING PROCEDURES

To execute FILEX, type:

```
RUN FILEX [,cmndfl]
```

where:

,cmndfl — is the name of a previously created file containing a table of desired logical unit assignments.

If the command file option is not used, FILEX uses the logical unit assignments already in the system.

If the command file is used, FILEX uses a special RX02 handler that reads and writes RX01 compatible diskettes and assigns logical unit numbers on RX01 and RX02 diskettes in the same system, provided the system is configured for RX02s. Assignments of this kind usually cannot exist on the same system.

These RX02 assignments remain in effect. When FILEX is completed, however, all logical units assigned to RX01s become undefined.

Following the RUN FILEX command, the program displays:

```
FILEX V8.00 (or current version number)
OPT (C, D, L, X, Z):
```

Enter one of the options; C, Copy; D, Delete; L, List; X, eXit; or Z, Zero (clear).

10.3 COPY (OPT:C)

OPT:C copies the contents of one file onto another. If you select option C, the system requests the input mode (the directory structure and the file format) of the file to be copied.

```
INPUT MODE (A, D, U):
```

Type the letter corresponding to the input mode to be used: OS/8 ASCII (A), COS-310 Data (D), or Universal (U).

10.3.1 OS/8 ASCII Input (Mode A)

ASCII format is that used by OS/8. If you select input mode A, the program displays:

```
FILE NAME:
```

Type the file name and the device designation in the following form:

```
filnam[.ex][,dev]
```

where:

filnam[.ex]
: is a six-character or less file name plus an optional two-character extension identifying the file to be input.

,dev
: is a three-character device designation. OS/8 RX01 and COS-310 RX01 diskettes are incompatible so do not specify an RXn device designation.

  If the device is not specified, the system device is assumed.

If the file name given already exists, FILEX displays:

```
REPLACE?
```

Type Y for YES, N or any other character for NO.

Having established the input file name, the program displays:

```
OUTPUT MODE (A, D, S, U):
```

Sections 10.3.4 through 10.3.4.4 explain the OUTPUT MODE.

10.3.2 COS-310 Data Input (Mode D)

If you select input mode D, the program displays:

```
FILE NAME:
```

Type the file name and the logical unit # in the following form:

```
filnam [/logical unit #]
```

where:

filnam is a six-character or less name identifying the file to be input.

[/logical unit #] identifies the logical unit where the file is found.

Having been given the input file name, the program displays:

```
OUTPUT MODE (A, D, S, U)
```

Sections 10.3.4 through 10.3.4.4 explain the OUTPUT MODE.

10.3.3 Universal Input (Mode U)

If you select input mode U, the program displays:

```
DISKETTE DATA MODE (A, I, U):
```

Type the letter corresponding to the diskette data mode to be used: A, ASCII; I, Image; U, Universal.

If you select any one of the diskette data modes A, I, or U, the program displays:

```
FILE NAME:
```

Type the input file name in the following form:

```
filnam[.ex][,RXn]
```

where:

filnam[.ex] is a six-character or less name plus an optional two-character extension. This name identifies the input file.

,RXn is a three-character device designation. Must be RX; n represents the drive on which it is mounted.

After you type the file name, FILEX displays:

```
OUTPUT MODE (A, D, S, U):
```

Sections 10.3.4 through 10.3.4.4 explain the OUTPUT MODE.

10.3.4 Output Modes (A, D, S, U)

The four output modes are: A, OS/8 ASCII; D, COS-310 data file; S, COS-310 source file; and U, Universal diskette.

10.3.4.1 OS/8 ASCII Output (Mode A)

If you select output mode A, the program displays:

```
FILE NAME:
```

Type the file name and the device designation in the following form:

```
filnam[.ex][,dev]
```

where:

filnam[.ex]
: is a six-character or less file name plus an optional two-character extension. This file name identifies the file where output is to go.

,dev
: is a three-character designation of the device where the output is to go. OS/8 RX01 and COS-310 RX01 diskettes are incompatible so do not specify an RXn device designation.

  If the device is not specified, the system device is assumed.

Type the file name; FILEX executes the transfer and returns to:

```
OPT (C, D, L, X, Z):
```

OS/8 files are always multiples of 16 blocks long. For this and other reasons, the resulting OS/8 output files may be longer than necessary. Use the OS/8 PIP program /A to recover the unnecessary space.

10.3.4.2 COS-310 Data File Output (Mode D)

If you select output mode D, the program displays:

```
FILE NAME:
```

Type the file name and logical unit # in the following form:

```
filnam[/logical unit #]
```

where:

filnam — is a six-character or less name identifying the file where output is to go.

/logical unit # — identifies the logical unit where the output file is found.

Type the file name; FILEX executes the transfer and returns to:

```
OPT (C, D, L, X, Z)):
```

10.3.4.3 COS-310 Source File Output (Mode S)

If you select output mode S, the program displays:

```
FILE NAME:
```

Type the file name in the following form:

```
filnam
```

where:

filnam — is a six-character or less name to be assigned to the COS-310 output file.

The output file is generated 16 blocks long.

To correct the directory entry to reflect the actual length of the file, do a FETCH and a WRITE as follows:

```
FE filnam       ;Fetch the file you have just created.

WR filnam/Y     ;The WRITE command enters the correct file
                ;length into the directory.  The /Y switch
                ;bypasses the REPLACE?  message response when
                ;a duplicate file name is encountered.
```

Type the file name; FILEX executes the transfer and returns to:

```
OPT (C, D, L, X, Z):
```

10.3.4.4 Universal Diskette Output (Mode U)

If you select output mode U, the program displays:

```
DISKETTE DATA MODE (A, I, U):
```

The three diskette data modes are: A, ASCII; I, Image; U, Universal.

Select diskette data mode A (ASCII), I (Image), or U (Universal), and the program displays:

```
FILE NAME:
```

Type the file name in the following form:

```
filnam[.ex][RXn]
```

where:

filnam[.ex]
: is a six-character or less name plus an optional two-character extension to be assigned to the output file.

,RXn
: is a three-character device designation. Must be RX; n represents the drive on which it is mounted.

If you selected diskette data mode A or I, FILEX performs the transfer and returns to:

```
OPT (C, D, L, X, Z):
```

In diskette data mode A or I, sector boundaries are ignored. An ASCII transfer (A) deletes nulls and rubouts, removes parity, and terminates each record with a RETURN. An Image transfer (I) reads and writes each byte exactly. The net effect of an Image transfer is similar to, and, in most cases, indistinguishable from an ASCII transfer.

If you selected diskette data mode U, the program displays:

```
OUTPUT RECORD SIZE (DEFAULT=80):
```

Type a number (1-128) representing the size of the output record. If you respond with RETURN, the record size defaults to 80. In this universal diskette data mode, one sector is equal to one record which is equal to one line.

Type the output record size; FILEX performs the transfer and returns to:

```
OPT (C, D, L, X, Z):
```

Figure 10-2 is a visual representation of the execution of the C option in FILEX.

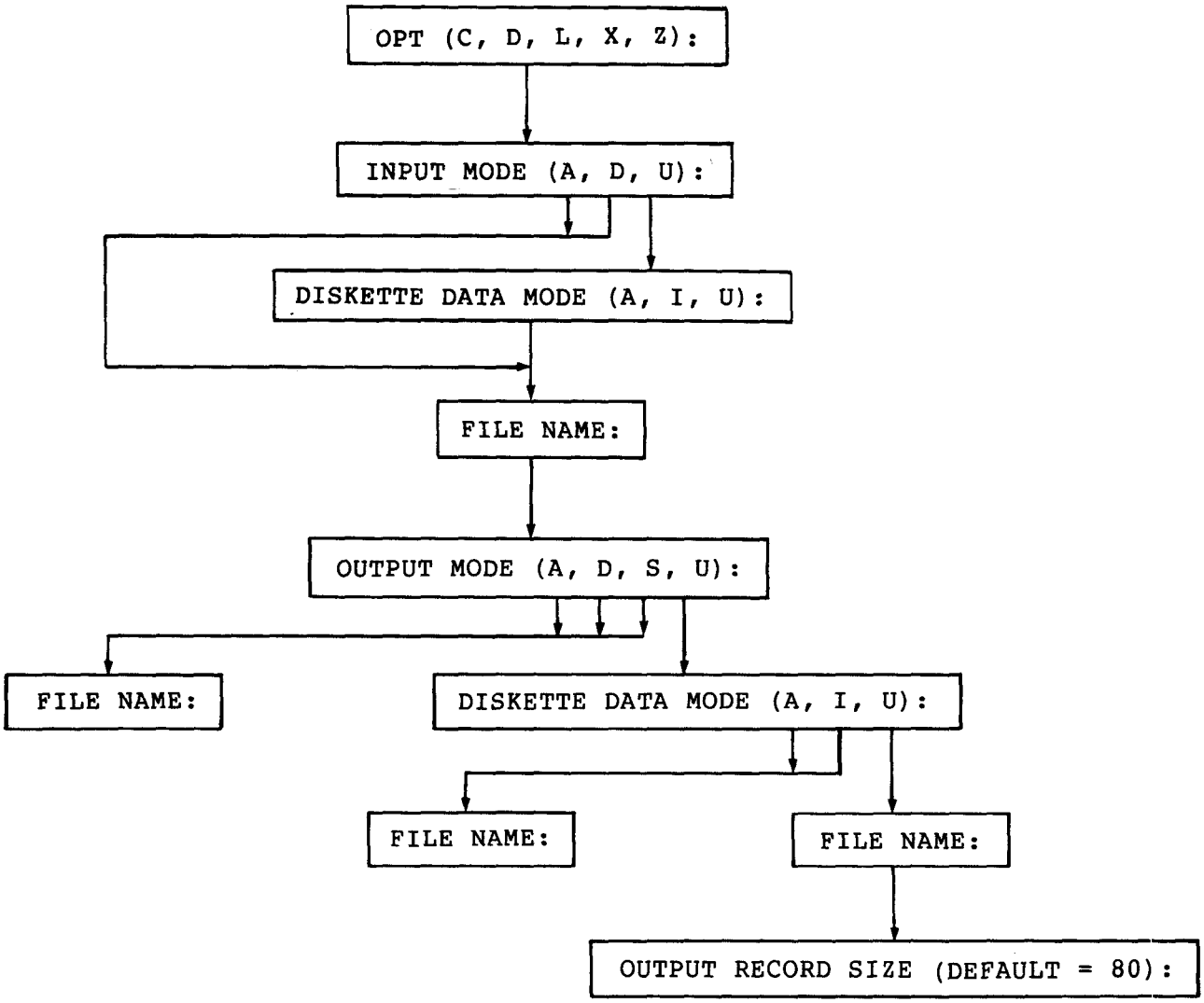


Figure 10-2 Flowchart of FILEX OPT:C

10.4 DELETE (OPT:D)

OPT:D deletes a single file from the universal diskette directory. If you select option D, the program displays:

```
FILE NAME:
```

Type the file name in the following form:

```
filnam[.ex],RXn
```

where:

filnam[.ex]
: is a six-character or less file name plus an optional two-character extension which identifies the file to be deleted.

,RXn
: is the three-character designation of the device on which the file is found. Must be RX; n identifies the drive on which the device is mounted.

Type the file name; FILEX deletes the file and returns to:

```
OPT (C, D, L, X, Z):
```

10.5 LIST (OPT:L)

OPT:L displays a listing of all the files in the universal diskette directory. If you select option L, the program displays:

```
DISKETTE DRIVE NUMBER:
```

Type the number corresponding to the drive on which the diskette is mounted. After you type the number, FILEX displays a table similar to the following:

```
NAME       RESERVED   USED     DATE
IMAGE            73     73   03-JUN-76
ASCII            73     73   03-JUN-76
IBM             576    576   03-JUN-76
ASC2             73     73   03-JUN-76
IBM2            193    193   03-JUN-76
<EMPTY>         910      0
<EMPTY>           0      0
<EMPTY>           0      0
<EMPTY>           0      0
<EMPTY>           0      0
<EMPTY>           0      0
<EMPTY>           0      0
```

```
<EMPTY>          0       0
<EMPTY>          0       0
<EMPTY>          0       0
<EMPTY>          0       0
<EMPTY>          0       0
<EMPTY>          0       0
<EMPTY>          0       0
```

where:

NAME — is either the file name or a designation for an empty area.

RESERVED — is either the number of sectors reserved for a file or the number of sectors available for additional files. There is space for a total of 19 files using a total of 1898 sectors.

USED — is the number of sectors actually used by the file.

DATE — is the creation date of the file.

FILEX then returns to:

```
OPT (C, D, L, X, Z):
```

10.6 EXIT (OPT:X)

OPT:X returns control to the COS-310 Monitor.

10.7 ZERO (OPT:Z)

OPT:Z zeros (clears) an entire universal diskette and makes it ready for new files. If you select option Z, the program displays:

```
DISKETTE DRIVE NUMBER:
```

Type the number corresponding to the drive on which the diskette to be zeroed is mounted. When zeroing is completed, FILEX returns to:

```
OPT (C, D, L, X, Z):
```

A zeroed universal diskette has one file name entry (DATA) in the directory. This reserves 1898 sectors (the entire diskette). Before any files can be transferred to this diskette, DATA must be deleted using OPT:D of FILEX.

10.8 FILEX ERROR MESSAGES

The most common error message is a return to the options. This is caused by inputting an answer, usually a file name, in the wrong format. Check the format of your answer and retry.

Message	Explanation
BAD DIRECTORY	Attempted to reference or store a file on a device without a directory or on a device where the directory has been destroyed. Only devices with directories can be used. If the directory is damaged, call your Software Specialist.
DEVICE ERROR	The system failed in an attempt to read from or write to a device. Retry the operation. Check for media problem (use PIP OPT- R), or check for hardware problem.
FILE ALREADY EXISTS	A file name already on the universal diskette was entered in response to the output FILE NAME: message. The system returns to OPT (C, D, L, X, Z): Start the option sequence again and use a unique file name.
FULL	COS-310 source output file exceeds 16 blocks; IBM output file too large for device. The system outputs as much as it can and then displays the error message. Use larger device, reduce the size of output, or determine if the loss is worth the change.
ILLEGAL DEVICE	The file name information contains a device that is not in agreement with the logical unit information. Stipulate new device information.
INSUFFICIENT SPACE ON DEVICE	Attempted to allocate more segments than are available on a device. Either allocate fewer segments, make more segments available, or use a larger device.
NO END	No end-of-file mark in the OS/8 input file. Correct the input file.

Message	Explanation
NO ROOM	No room is available for the file in the output device directory (OS/8, IBM). Delete to make space or use a device with directory space.
NOT ENOUGH ROOM FOR SYSTEM AND FILES	Designated a device that is too small to accomodate both the system program and the files. Use PIP OPT- E to put system program on one device and files on another.
NOT FOUND	The input file name was not found. The system displays FILE NAME: Check the directory for the name of a file on the system. Enter a name found in the directory.
NOT UNIVERSAL DISKETTE?	Requested a device that does not contain universal floppy. Request a device with universal floppy.
# TOO LARGE	The number of segments in a logical unit exceeded 4095.
SYNTAX ERROR	The file containing the logical unit assignments is formatted incorrectly. Reformat the file.
TOO BIG	The record is too large. It exceeds 120 characters for source file output or 510 characters for data file output. Reduce the size of the record.

CHAPTER 11

PATCH PROGRAM (PATCH)

PATCH is used to fix (patch) either a system program or the Monitor on a COS-310 system. All input information for the PATCH operation is distributed as official patches from Digital Equipment Corporation. The patch information is a line-by-line dialogue. No other information should be used.

System programs (files) and the Monitor consist of blocks of numerical information coded into machine language instructions and stored on the system device. These machine language instructions are numbered from 0 to 377 octal. PATCH reads one of these blocks, allows you to examine and/or change individual words within the block, and writes the block back out to the system device.

11.1 PATCH OPERATING PROCEDURES

To execute PATCH, type:

```
RUN PATCH[,cmndfl][/C]
```

where:

,cmndfl is a previously stored file of PATCH commands. Each command is on a separate line; there can be no blank lines or comments. When the command file option is specified, PATCH reads a line from the command file each time one of the following prompts is displayed:

```
FILE NAME:
BLOCK:
LOCATION:
NEW VALUE:
RELATIVE CHECKSUM:
```

After the last line of the command file has been used or an error is encountered, all responses must come from the keyboard.

/C changes the blocks on the system device. Without /C, PATCH simulates the patching operation but does not change the file on the system device. When run without the /C option, PATCH displays:

CHECKSUM CORRECT--USE OPTION C TO UPDATE

After the RUN PATCH command, the program displays:

```
PATCH V8.00 (or current version number)
FILE NAME:
```

Respond with the following information as provided by DIGITAL:

- The name of the file to be patched.
- /N to indicate a patch for the Monitor. The system responds PATCHING MONITOR.
- /X to indicate the end of the PATCH operation. The message EXIT is printed and control returns to the Monitor.

If you enter a file name or /N, PATCH displays:

```
BLOCK:
```

Answer with either the number corresponding to a block within a file, or END to indicate that no more blocks are to be patched.

If the block number is typed, the program displays:

```
LOCATION:
```

Respond with either the number corresponding to a location to be patched, or END to indicate that no more locations are to be patched.

If a location number is typed, the program displays:

```
OLD VALUE: nnnn
```

where:

nnnn is the old (current) value at the location.

This display requires no input and is followed by:

```
NEW VALUE:
```

Enter the new value as indicated in the information supplied by DIGITAL. The program displays:

```
LOCATION:
```

Answer with either the number corresponding to a location to be patched, or END to indicate that no more locations are to be patched.

If END is typed, the program displays:

```
RELATIVE CHECKSUM:
```

Enter the checksum from the information supplied by DIGITAL. If this checksum is correct, the program displays either:

```
NEW BLOCK PATCHED OK
        or
CHECKSUM CORRECT--USE OPTION C TO UPDATE
```

The patch has been accurately entered. Use option C to update the program. Once the program is updated it cannot be changed except by another PATCH routine. Following the OK statement, the program asks for another block number where patching is to be done:

```
BLOCK:
```

If further patching information is available, enter it. If no more patching information is provided, type END. Following END the program displays the number (nn) of blocks patched within the file:

```
nn BLOCK(S) PATCHED IN THIS FILE
```

With this statement the program requests the name of another file to be patched.

```
FILE NAME:
```

Enter the file name as supplied by DIGITAL. If patching is complete, type /X.

Following /X, the program displays:

```
EXIT

COS MONITOR V 8.00 (or current version number)
```

11.2 ERROR CORRECTION

Much of the seriousness of errors while patching can be eliminated with the use of the command file option.

11.2.1 CTRL/U or R (Restart)

If at any time prior to the end of the checksum statement an error is discovered, type R (for Restart) and PATCH will return to the FILE NAME question. During the PATCH operation, the DELETE key is inoperable. If you make an error on a line, type CTRL/U and the correct information.

11.2.2 Wrong Old Value

The old value displayed by the program must be the same as the old value supplied in the PATCH information. If it is not, go through the following procedure.

Step 1 Be sure that everything previously typed is letter perfect. If the wrong BLOCK number was typed, type R and restart at FILE NAME. If the wrong LOCATION was typed, type RETURN in answer to NEW VALUE. This makes no change to the location specified. Type the correct location number in answer to LOCATION.

Step 2 If everything typed was correct, check the version number of the Monitor or the system program in question.

Step 3 If everything seems in order but the dialogue doesn't agree, save all output and consult your Software Specialist.

11.2.3 Bad Checksum

If an error in the checksum is detected, the following message is displayed:

```
BAD CHECKSUM
LOCATION:
```

The faulty block is not written to the system device.

The newly changed block is still in memory. Review the numbers and the locations to see if they are correct. If the error is found, fix it and then type END to the LOCATION: message. If an error is not found, type R to restart the program and patch the entire block again.

11.3 PATCH ERROR MESSAGES

Most error messages result from incorrect entries. Check each entry for accuracy. All entries must be exactly as supplied by DIGITAL.

Message	Explanation
BAD CHECKSUM	An attempt was made to write a block which was incorrectly patched. Type R and restart the program.
BAD DIRECTORY	Attempted to reference or store a file on a device with a damaged or nonexistent directory. Only files with directories can be used. If the directory is damaged, call your Software Specialist.
BAD NUMBER	A number with either more than 4 digits, a nondigit, or the digits 8 or 9 was typed. Enter number correctly.
BLOCK TOO BIG	An incorrect block number was typed. It cannot be larger than the length of the file being patched. Enter the correct block number.
FILE NOT FOUND	The file was not found on the system device. Check the directory for the file name. If the file name is not found, check for correct version number.
LOCATION TOO BIG	A location greater than 377 was typed. Retype location number.
NO CHANGE IN BLOCK	An attempt was made to write a block with no changes in it. Make proper changes using patch information.

CHAPTER 12

BOOT PROGRAM (BOOT)

BOOT is used to bootstrap the system from one device to another, i.e., if the system has been moved from one type of device to another, boot is run to start the system on the new device.

12.1 BOOT OPERATING PROCEDURES

To execute BOOT, type:

```
RUN BOOT/xx
```

where:

/xx is the two-character designation for the device which you want to get into operation.

/DK is the RK05 disk unit 0.

/RX is the RX01 diskette unit 0.

/DY is the RX02 diskette unit 0.

An attempt to bootstrap a device which is not ready or does not exist will produce unpredictable results.

12.2 BOOT ERROR MESSAGES

Message	Explanation
NO	No device or an illegal device designation was specified. Control returns to the Monitor. Specify a legal device designation.

CHAPTER 13

LINE CHANGE PROGRAM (LINCHG)

The Line Change Program (LINCHG) is a utility program which temporarily changes the lines-per-page configuration of printed programs without affecting the SYSGEN lines-per-page default value of 66. Its use is limited to printers without forms hardware.

13.1 LINCHG OPERATING PROCEDURES

To execute LINCHG, type:

```
RUN LINCHG[/n]
```

where:

/n is the number of lines you want on a page.

If you do not use the /n option, the program displays:

```
HOW MANY LINES PER PAGE?
```

Type the number (1-99) of lines you want. LINCHG installs this specified number as the number of lines-per-page. The system defaults to 66 lines-per-page.

The LINCHG number will remain in effect unless a further call to LINCHG is made, the system is rebooted, or the system is closed down.

Example:

Following is a batch program with a SYSGEN default of 66 lines-per-page. Line change commands change the lines-per-page of various programs.

```
.RUN JOB1       ;66 lines-per-page.
.RUN LINCHG/10
.RUN JOB2       ;10 lines-per-page.
.RUN LINCHG/33
.RUN JOB3       ;33 lines-per-page.
.RUN LINCHG/99
.RUN JOB4       ;99 lines-per-page.
.RUN LINCHG
HOW MANY LINES PER PAGE?
50
.RUN JOB5       ;50 lines-per-page.
.RUN LINCHG/66
```

The above example shows how a series of programs may be run starting with the normal default for the first program, then incorporating a variety of changes for the programs which follow, and finishing with the default number. The 66 lines-per-page can also be reestablished by rebooting the system.

13.2 LINCHG ERROR MESSAGES

Message	Explanation
INVALID OPERATION	Attempted to change lines-per-page on a printer with forms hardware. Such a change cannot be made.

CHAPTER 14

FORMAT PROGRAMS (DKFMT, DYFMT)

Before an RK05 disk or an RX02 diskette can be used on COS-310, it must be initialized. Initialization consists of formatting the disks. Do not initialize a disk or diskette containing any important information such as the Monitor or other such files. Initialization destroys the data on the disk.

Formatting the RK05 and RX02 means writing the necessary timing and sense marks onto the disk or diskette and erasing any other information.

An RX01 diskette can be formatted to become an RX02 diskette. This procedure cannot be reversed. The RX01 diskette does not need to be formatted to be used on an RX01 drive.

14.1 FORMATTING RK05 DISKS

To format an RK05 disk, type:

```
RUN DKFMT
```

The program displays:

```
DKMFT  V 8.00
DRIVE?
```

Respond with the number (0-3) of the drive where the disk is mounted. After you type this number the following message is displayed:

```
ARE YOU SURE?
```

Any response other than Y (Yes) brings back the DRIVE? question. A Y (Yes) response causes the program to display:

```
WRITE PASS
READ PASS
```

These two phrases indicate that the program is in operation; they require no response from the keyboard. Some time (a matter of seconds) elapses after each phrase appears while the program completes that particular phase of operation.

When the formatting operation is complete, the program displays:

```
DRIVE?
```

This is a cue to begin formatting another disk. Time is allowed for the physical changing of disks. If the formatting is complete, type CTRL/C to return to the Monitor.

14.2 FORMATTING RX02 DISKETTES

To format an RX02 diskette, type:

```
RUN DYFMT
```

The program displays:

```
DYFMT  V 8.00
DRIVE?
```

Respond with the number (0-1) of the drive where the disk is mounted. The program displays:

```
ARE YOU SURE?
```

Any response other than Y (Yes) brings back the DRIVE question. A Y (Yes) response causes the program to display:

```
FORMATTING DRIVE n
```

where:

n is the number (0-1) of the drive previously indicated.

This statement remains on the screen until formatting is completed. When the formatting is completed, the program displays:

```
DRIVE?
```

This is a cue to begin formatting another diskette. Time is allowed for the physical changing of the diskettes. If the formatting is completed, type CTRL/C to return to the Monitor.

CHAPTER 15

DUMP AND FIX TECHNIQUE (DAFT)

The Dump and Fix Technique (DAFT) program is similar in function to an editor, but it is used for data records. DAFT allows you to search for, examine, and change records, and to list records or parts of records on the printer or on the screen.

DAFT allows one input and one output file to be open at the same time. These two files can be the same file when in UPDATE mode. Memory always contains a record from the input file known as the current record. The current record can be modified by the CHANGE command before being written on the output file. An output file is not needed if records from the input file are only being examined.

15.1 DAFT COMPILING PROCEDURE

Because the DAFT program is distributed as two DIBOL source files, these two source files must be compiled into one binary file before DAFT can be executed. To compile DAFT, type:

```
.RUN COMP,DAFTA,DAFTB
.SAVE DAFT
```

15.2 DAFT OPERATING PROCEDURES

To execute DAFT, type:

```
RUN DAFT[,cmndfl1...,cmndfl7]
```

where:

cmndfl1...,cmndfl7
: are previously stored files which contain DAFT commands to be used to dump or fix a data file. If the optional command files are not present, commands are entered via the keyboard. After the last command in the last file

is executed, additional commands can be entered through the keyboard. An asterisk (*) is displayed to indicate that the DAFT program is ready for a command.

15.3 DAFT COMMAND FILE

The command file is created with the COS editor and contains DAFT commands. The entries in the command file are ordered according to a sequence of needs within individual records. To create an effective command file you must know the contents of the record and the possible areas needing correction.

15.4 DAFT COMMANDS

The first word in a DAFT command is a keyword consisting of any number of nonblank characters, only the first of which is significant. Some commands involve both a keyword and arguments. These arguments are separated from each other and from the keyword by one or more spaces.

15.4.1 Symbols Used in DAFT Commands

n — represents an unsigned nonzero positive integer. If it is optional in a command and is omitted, n=1 is assumed.

<a,b> — represents a key field of character positions a through b inclusive. Both a and b are unsigned nonzero positive integers and b must not be smaller than a. If this is optional in a command and is omitted, the subscripted area specified in the KEY command is used.

\+ — indicates that before a record is read from the input file, the current record in memory (if there is one) is written on the output file. The + sign does not have a space before it unless it is the only argument.

data — represents a piece of data. Alphanumeric data has the form:

```
'characters...'
```

Numeric data has the form:

```
[-] digits...
```

Before being used in executing a command, data is adjusted to the same length as the key field <a,b>. If the data is smaller and alphanumeric, it is left-justified in the field and filled with spaces on the right. If data is smaller and numeric, it is right-justified and filled with zeros on the left. If data is larger and alphanumeric, excess characters on the right are ignored. If it is larger and numeric, excess characters on the left are ignored.

15.4.2 DAFT Command Summary

Commands are entered after DAFT displays an asterisk (*).

Command	Function
Advance [n][+]	Advances the input file n records.
Backspace [n]	Backspaces n records if the input file was opened with the UPDATE command.
Change [<a,b>]data	Replaces the data in the current key field of the record currently in memory with the data specified. If <a,b> is used, it temporarily overrides the key field specified in the key statement.
Display [n]	Sets the width of the line of the listing device (screen or printer) to n characters (maximum 130). If n is omitted, this command turns the grid on if it is off and turns it off if it is on.
Exit	Returns control to the COS-310 Monitor if no output file is open.
Fini [+]	Closes the output file. If + is specified, the current record and the remainder of the input file are first copied to the output file. To write data to a file opened for UPDATE, the + must be specified.
Goto n[+]	Makes record n the current record.
Help	Displays a summary of DAFT commands.

Command	Function
Input filnam[/logical unit #]	Opens the specified file for input. The first record is read and becomes the current record.
Key a,b	Sets the key to character positions a through b inclusive.
List [n][<a,b>][+]	Prints n consecutive records beginning with the current record. The subscript <a,b> represents the consecutive characters that are to be considered.
Output filnam[/logical unit #]	Opens the specified file for output.
Put [n]	Writes n copies of the record currently in memory onto the output file.
Query	Displays the names of the input and output files, the units where the files are located, the record currently in memory, and the version number of the DAFT program.
Rewind	Reopens the input file. The first record becomes the current record.
Search [<a,b>]data[+]	Searches the current record and then succeeding records for an occurrence of the specified data appearing in the key field.
Type [n][<a,b>][+]	Same as List except output is displayed on the screen.
Update filnam[/logical unit #]	Opens the specified file for updating. This command can only be specified for a file with fixed-length records since direct access I/O is used to move records.
Version	Displays the version number of DAFT.
Write [n]	Performs the same function as Advance [n][+]. The nth record after the current record becomes the new current record.
X	Outputs the record number and size of the current record on the output device (either screen or printer depending on whether the last record was output by a Type or List DAFT command). The printer is the initial output device.

15.5 DAFT OUTPUT

Records can be listed with a grid above them. The grid has two lines of numbers which show the character positions. The lower of the two lines represents the ones digits of the column counts. The upper line represents the tens digits. The tens digits are printed for the first and last column in the record (or part of the record) or whenever the tens digit increments. If there is a hundreds digit, it is printed in column 1 or whenever it increments.

Following is an example of a DAFT program in operation.

```
.R DAFT
*HELP
ADVANCE N+
BACKSPACE N
CHANGE <A,B> DATA
DISPLAY N
EXIT
FINI +
GOTO N+
HELP
INPUT LABEL/UNIT
KEY A,B
LIST N KEY+
OUTPUT LABEL/UNIT
PUT N
QUERY
REWIND
SEARCH <A,B> DATA +
TYPE N <A,B>+
UPDATE LABEL/UNIT
VERSION
WRITE N
X
*VERSION
DAFT VERSION 8.00
*INPUT MAILING/1
*DISPLAY 70
*T 1

RECORD 000001 OF FILE MAILNG,   RECORD LENGTH=140 CHARACTERS

 DIGITAL EQUIPMENT CORP.  D. F. PAVLOCK            12-3

      146 MAIN ST.             MAYNARD       MA017540S/8-1  0012345A11

*D
*T 2

RECORD 000001 OF FILE MAILNG,   RECORD LENGTH=140 CHARACTERS
```

```
 0        1         2         3         4         5         6         7
 12345678901234567890123456789012345678901234567890123456789012345678901234567890
 DIGITAL EQUIPMENT CORP.  D. F. PAVLOCK            12-3

 7        8         9        10         1         2         3         4
 12345678901234567890123456789012345678901234567890123456789012345678901234567890
      146 MAIN
 ST.              MAYNARD         MA017540S/8-1  0012345A11


RECORD 000002 OF FILE MAILNG,   RECORD LENGTH=140 CHARACTERS

 0        1         2         3         4         5         6         7

 12345678901234567890123456789012345678901234567890123456789012345678901234567890
 DIGITAL EQUIPMENT CORP.  K. RICHER                12-3

 7        8         9        10         1         2         3         4

 12345678901234567890123456789012345678901234567890123456789012345678901234567890
      146 MAIN ST.              MAYNARD
 MA01754COS 300 0001972T 3

*T 2<25,50>

RECORD 000002 OF FILE MAILNG,   RECORD LENGTH=140 CHARACTERS

 2    3         4         5
 56789012345678901234567890
  K. RICHER


RECORD 000003 OF FILE MAILNG,   RECORD LENGTH=140 CHARACTERS

 2    3         4         5
 56789012345678901234567890
  S. RABINOWITZ

*A 1
*KEY 1,50

*T 2

RECORD 000004 OF FILE MAILNG,   RECORD LENGTH=140 CHARACTERS

 0        1         2         3         4         5
 12345678901234567890123456789012345678901234567890
 DIGITAL                    R. LARY


RECORD 000005 OF FILE MAILNG,   RECORD LENGTH=140 CHARACTERS

 0        1         2         3         4         5
 12345678901234567890123456789012345678901234567890
 DEC                        S. G. WELCOME
```

```
*Q
INPUT FILE: MAILNG    OPEN
UNIT: 01
OUTPUT FILE:  /NONE/
UNIT: 00
KEY=<001,050>

RECORD 000005 OF FILE MAILNG,   RECORD LENGTH=140 CHARACTERS

DAFT VERSION 8.00
```

15.6 DAFT ERROR MESSAGES

Message	Explanation
BAD DIGIT IN DATA	In a Change or Search command, a character other than digits or a minus sign is contained in a numeric data field. Remove bad characters.
CANT BACKSPACE PAST BEGIN OF FILE	Attempted to backspace past the beginning of file. The first record in the file becomes the current record.
CANT BACKSPACE WITH SEQUENTIAL INPUT	Attempted to backspace with sequential input. Backspace only possible when file is in update mode.
END OF INPUT FILE AT RECORD nnnn	Attempted to read past the end-of-file mark on the input file. This is not necessarily an error. nnnn was the last record read. The input file is closed. Reopen the file.
EXCESSIVE GRID SIZE	The grid (printer width) may not be greater than 130 characters. Reduce the grid size.
EXTRA CHARS	Extra characters were found after the end of a command. Remove extra characters.
ILLEGAL RECORD - CLOSING FILE	The file being updated contains a bad record (one not the same size as record 1). Only fixed-length records are permitted on such files. The file is closed. Reopen the file.

Message	Explanation
KEY ENTIRELY PAST END OF RECORD	The key specified in a List or Type DAFT command began with a character greater than the record size. Reduce the size of the key.
KEY EXTENDS PAST RECORD END	Attempted a change with a key that extends past the end of a record. However, a list with such a key is possible. In such a case, the list is terminated at the end of the record.
KEY TOO BIG	The key exceeded 100 characters. Reduce the size of key.
NO DATA	Data was not specified in a CHANGE or SEARCH command. Specify required data.
NO INPUT FILE	The command does not have an input file. Open an input file.
NO LABEL NAME	The file name was omitted in an INPUT, OUTPUT, or UPDATE command. Implement a name.
NO OUTPUT FILE	The command requires an output file but one is not open. The command is terminated at the point just prior to writing the current record on the output file. Open an output file.
OUTPUT FILE ALREADY OPEN	A request was made to open an output file while one was already open. Only one output file can be open at a time. The request is ignored. Close the current output file before opening a new file.
OUTPUT FILE STILL OPEN	An EXIT cannot be made when the output file is open. The output file can be closed with the FINI command or CTRL/C.
PUSHDOWN OVERFLOW	The program will abort with this message when too many errors are made. Restart DAFT.
0 NOT ALLOWED	The 0 (zero) is not a permissible argument. Don't use 0.

CHAPTER 16

REPORT PROGRAM GENERATOR (PRINT)

PRINT eases the creation of report programs. Using a command file which describes the report, PRINT generates a DIBOL program which produces the report.

PRINT is two programs chained together. The first program reads and validates the command file while creating memory table entries. If no command file errors are detected, the second program produces the DIBOL report program. The two programs which generate the report program require a total of 16K bytes of memory.

16.1 PRINT COMPILING PROCEDURE

PRINT is distributed as several source files. The files must be compiled into two programs before PRINT can be executed.

The PRINT source files contain the following information:

- PRINT1, PRINT2 are the data sections.
- PRINT3, PRINT4, PRINT5, PRINT6 are the procedure sections of the parsing (reading and validating) phase.
- PRINT7, PRINT8, PRINT9, PRINT0 are the procedure sections of the generation phase.

Use the following procedure to compile the distributed PRINT source files into binary programs.

```
.RUN COMP,PRINT1,PRINT2,PRINT3,PRINT4,PRINT5,PRINT6
.SAVE PRINTA
.RUN COMP,PRINT1,PRINT2,PRINT7,PRINT8,PRINT9,PRINT0
.SAVE PRINTB
```

16.2 PRINT OPERATING PROCEDURES

To execute PRINT, type:

```
RUN PRINTA+PRINTB,cmndfl[/xy]
```

where:

cmndfl is the name of a previously stored command file.

PRINTA+PRINTB
are the compiled PRINT DIBOL programs.

x is a switch which determines whether the command file is to be listed on the printer; N means no list, L means list.

y is a switch which determines whether the DIBOL program (data file) created by executing PRINT is to be listed on the printer; N means no list, L means list.

If x and y are both N, the switches and their preceding slash can be omitted.

The output from the RUN PRINT command is a data file which must be converted to a source file with the use of FILEX.

16.2.1 FILEX - Creation of Source File

Use the following FILEX command sequence to create a source file from the data file created by PRINT. (See Chapter 10 for FILEX information.)

```
.RUN FILEX
FILEX V 8.00
OPT (C, D, L, X, Z): C
INPUT MODE (A, D, U): D
FILE NAME: $RPG/logical unit #
OUTPUT MODE (A, D, S, U): S
FILE NAME:  pronam
OPT (C, D, L, X, Z): X
```

Pronam is any name desired, but it is usually the same name as is used in the IDENT line of the command file.

16.2.2 Compilation

To compile the DIBOL source program created by running the PRINT output through FILEX, type:

```
.RUN COMP,pronam
.SAVE pronam
```

16.2.3 Program Execution

To execute the compiled DIBOL program, type:

```
.RUN pronam
```

where:

pronam — is the name of the source program created by FILEX and compiled in Section 16.2.2

The execution of this program produces the report.

16.3 PRINT COMMAND FILE

The Print Command File has six sections:

IDENT	identifies the program and author
HEAD1,HEAD2	provides page headings for the report
INPUT	describes the input file
COMPUTE	describes any computation to be done
PRINT	describes the report column headings
END	is an optional directive at the end of the Print Command File

16.3.1 IDENT Section

The form of the IDENT section is:

```
IDENT pronam[/logical unit #][,author]
```

where:

pronam — is the name of the DIBOL source program to be generated.

/logical unit# — is the number referencing the storage location of the program.

,author any text from 1 to 24 characters in length.

Example:

```
IDENT TEST41/14, JOHN DOE ;Program named TEST41, on logical  unit
                          ;14, written by John Doe.
```

16.3.2 HEAD1 and HEAD2 Section

HEAD1 is the first heading line on each page of the report. HEAD2 is the second line. HEAD1 and HEAD2 are both optional. The only difference between HEAD1 and HEAD2 is that HEAD1 information will be expanded (if space permits) by inserting a space between each character. HEAD2 has no such expansion capability.

The form is:

```
[HEAD1 'text']

[HEAD2 'text']
```

where:

text is a string of up to 132 characters from the COS-310 character set, exclusive of single quotes.

There can be more than one HEAD1 or HEAD2 line. If this is the case, the individual texts are linked together.

Example:

```
HEAD1 'COMPUTATION AND SUMMARY RESULTS'

HEAD1 'FOR AUGUST, 1973'
```

16.3.3 INPUT Section

The INPUT section consists of the INPUT statement on one line followed by field description lines describing the fields of the input record.

The form of the input statement is:

```
INPUT [filnam[/logical unit #]][,S]
```

where:

filnam is the name of the input file.

/logical unit #
: is the logical unit on which the input file resides.

,S
: summarizes rather than describes the report.

If the file name is omitted, the generated program will request it when the report is run.

The form of each field description line is:

$$[fldnam], \left\{ \begin{matrix} A \\ \\ D \end{matrix} \right\} n[.m][,Lr[P]]$$

where:

fldnam
: is the name of the field.

A
: is an alphanumeric field.

D
: is a numeric field.

n
: is the size of the field, expressed in characters (510 maximum for A field, 15 maximum for D field).

.m
: is the number of decimal places in the field and is valid only for numeric fields.

,Lr
: is used only for the break fields (a break field is used in conjunction with ACCUMULATE to print totals). The r is a single digit expressing the relative importance of the field; 1 indicates least important and 9 indicates most important. When totals are printed for a more important break, totals for all lesser breaks are also printed.

P
: starts a new report page after the totals for this break are printed.

The maximum number of fields is 20.

16.3.4 COMPUTE Section

This optional section starts with a line containing only one word:

```
COMPUTE
```

DIBOL statements appear on succeeding lines in the following form:

```
fldnam = expression
```

where:

fldnam — is the name of a destination field and must not duplicate any input field name nor any previous computation result name.

expression — is any valid DIBOL expression. It may be a single alphanumeric field or literal, or any numeric expression.

Unlike DIBOL, PRINT uses decimal places. It will not allow addition and subtraction of expressions with different numbers of decimal places. The result of such an expression will have the same number of decimal places as the elements of the expression.

The number of decimal places in the result of a multiplication expression is the sum of the decimal places in the two expressions.

The number of decimal places in the result of a division expression is the difference in decimal places between the number being divided and the divisor.

To adjust the number of decimal places for any expression, multiply by the constant 1.00 with the number of decimal places equal to the longest number of decimal places in the other values. For example, expr 1 has 2 places, and expr 2 no places, then

```
expr 1 * expr 2 * 1.00
```

has four decimal places.

PRINT allows decimal places in numeric constants.

16.3.5 PRINT Section

The PRINT section begins with a line containing only one word:

```
PRINT
```

The next 12 lines describe the fields to be printed (not all 12 lines must be used). The form of these field descriptions is:

```
fldnam, 'text'[,A][,format]
```

where:

fldnam — is the name of the field and must be either an input field or the result of a computation.

'text' is an alphanumeric string of COS-310 characters delimited by single quotes. This text is used as the heading of the report columns and on the total lines for break fields.

,A is present only if the field is to be accumulated and the sum is to be printed on total lines. The field must be numeric.

format is a string of text showing the format for the numeric fields. If the format is not included, PRINT will create one using the description of the field (if this is an accumulated field, two extra places will be assumed). That created format will use an appropriate number of decimal and integer places in the following form:

```
XX,XXX.XX-
```

The text used for titles may be several words separated by asterisks. This centers each word over a column in separate lines. For example:

```
GPAY,  'GROSS*PAY'
```

will cause

```
GROSS
PAY
```

to be placed over the GPAY column. If GPAY is also a break field, then:

```
GROSS PAY TOTAL
```

will be used on the total line instead of GROSS*PAY TOTAL.

The field descriptor lines may also be of the form:

```
,An
```

Where n is the size of the field. This will produce n blank columns in the report.

If any two field descriptors are not separated by a filler descriptor, then two blank columns will separate the fields in the report.

Example:

```
PRINT
NAME, 'EMPLOYEE NAME'
DEPT, 'DEPARTMENT'
GPAY, 'GROSS*PAY',A,XX,XXX.XX
```

16.3.6 END Section

The END section is optional and consists of a line containing only the word:

```
END
```

16.4 PRINT ERROR MESSAGES

Most errors in PRINT result from incorrect information in the command file. These are correctable with the editor and Monitor commands.

PRINT evaluates each statement in the command file for correctness. Whenever errors occur, the entire line where the error occured and a message are printed. The number printed near the error message indicates the character position where the error occurred. The following error messages are used by PRINT.

Message	Explanation
ALPHA LITERAL REQUIRED	Expected alphanumeric literal is missing. Insert where appropriate.
ALREADY DEFINED	Attempt to name a field in the INPUT or COMPUTE section with a name that was previously used. Correct the command file.
HEADER IS TOO LONG	The header line exceeds 132 characters. Correct the command file.
IMPROPER DEFINITION	Filler item in the PRINT section is used incorrectly. Correct the command file.
IMPROPER LITERAL	Literal too long. Shorten the literal.
IMPROPER USE OF DECIMAL PLACES	The number of decimal places exceeds the size of the field being defined. Reduce the number of decimal places.
INTEGER FROM 1-15 REQUIRED	The size of a numeric field specified in the INPUT section must be between 1-15. Correct the command file.

Message	Explanation
INTEGER FROM 1-132 REQUIRED	The expected numeric fields out of range. Reduce the field size to fewer than 132 characters.
INTEGER REQUIRED	Integer missing where expected. Check and insert as needed.
LITERAL TOO LONG	Field description exceeds 30 characters. Reduce to fewer than 30 characters.
MUST BE IDENT	The first section in the command file must be IDENT. Correct the command file.
MUST BE NUMERIC ITEM	An item expected to be numeric is defined incorrectly. Redefine.
MUST BE S	S is the only legal option in the INPUT statement that follows the comma. Insert S.
NEED FILE NAME	File name missing from IDENT statement. Correct the command file.
NO ENDING QUOTE	No closing quote for a HEAD1, HEAD2, or PRINT statement. Insert closing quote.
NO INPUT DIRECTIVE	The INPUT statement is missing. Correct the command file.
NO PRINT ITEMS	No fields are specified following the PRINT statement. Correct the command file.
NOT DEFINED	Attempted to print a field that has not been defined. Correct the command file.
NOT ENOUGH RIGHT PARENTHESES	A statement in the COMPUTE section has too few right parentheses. Correct the command file.
PICTURE TOO LONG	The picture or edit mask for printing exceeds 22 characters. Reduce to fewer than 22 characters.

Message	Explanation
SYNTAX ERROR	Statement contains illegal characters or options. Correct the command file.
TOO MANY COLUMNS IN REPORT	More than 132 columns under HEAD1, HEAD2, or PRINT sections. Correct the command file.
TOO MANY COMPUTE STATEMENTS	More than eight COMPUTE statements were specified. Correct the command file.
TOO MANY DATA ITEMS	More than 20 data items in the INPUT section. Correct the command file.
TOO MANY LEFT PARENTHESES	A statement in the COMPUTE section is too complicated to be deciphered by PRINT. Simplify the command file.
TOO MANY LIST ITEMS	More than 20 list items in the INPUT section. Correct the command file.
TOO MANY RIGHT PARENTHESES	A statement in the COMPUTE section has too many right parentheses. Correct the command file.
UNKNOWN DIRECTIVE	An invalid section statement. Only IDENT, HEAD1, HEAD2, INPUT, COMPUTE, PRINT, and END are legal. Check your statements for incorrect statements; correct the command file.

CHAPTER 17

FLOWCHART GENERATOR PROGRAM (FLOW)

The flowchart generator program (FLOW) produces a flowchart from a set of input commands.

The flowchart is always written to a file named $PASS1 located on logical unit 1. A printed flowchart can also be produced by using the appropriate option switches. The flowchart generator programs are distributed as source programs and must be compiled before use.

17.1 FLOW COMPILING PROCEDURE

FLOW consists of several DIBOL programs.

To compile the FLOW programs, type:

```
.RUN COMP,FLOW1,FLOW2,FLOW3,FLOW4
.SAVE FLOW
.RUN COMP,KREF
.SAVE KREF
```

17.2 FLOW OPERATING PROCEDURES

The commands to execute FLOW have the form:

```
RUN FLOW,cmndfl1...,cmndfl7[/xx]
RUN SORT,KRFSRT
RUN KREF
```

where:

cmndfl1...,cmndfl7
: are previously stored source files containing the FLOW commands to be used in generating a flowchart.

/xx
: is one of the following option switches:

/L lists the flowchart on the printer. If /L is omitted, the flowchart will only be placed in file $PASS1 on logical unit 1.

/P indicates that the input files are DIBOL programs with the FLOW commands imbedded in the program. When the /P option is used, only input lines beginning with a semicolon followed by any number of periods followed by a space or a tab are treated as FLOW commands. No semicolon - period - space configuration is needed if the FLOW commands are not part of a DIBOL program.

KRFSRT is a special sort command file that will sort the cross-reference scratch file. KRFSRT is distributed as part of the COS-310 software.

KREF is a cross-reference DIBOL program that works specifically with FLOW. This produces a cross-reference table containing an alphabetical listing of all labels used in the flowchart, the page number where each label is defined, and the page numbers where each label is used. KREF is distributed as part of the COS-310 software.

FLOW uses logical units 1, 2, 3, 4, and 5. Logical unit 1 must be large enough to contain the flowchart print image (usually 10 segments is sufficient). Logical units 2, 3, 4, and 5 must each be large enough to contain the KREF scratch file (usually 5 segments in each logical unit).

17.3 FLOW COMMANDS

Although some FLOW commands look like DIBOL statements, they are defined and used differently. FLOW commands have the following general format:

```
[;.. ][label][,] command
```

where:

;.. is the special indicator used with the /P option to distinguish between FLOW commands and DIBOL statements.

label is the FLOW statement label.

command is one of the following FLOW commands:

```
PROC [;][text]
DISK [;][text]
IF {YES|NO} :label [;][text]
```

```
CALL label [;][text]
START [;][text]
STOP [;][text]
GOTO label
CGOTO label1,label2,...
I/O [;][text]
TITLE [;][text]
SBTTL [;][text]
PAGE
```

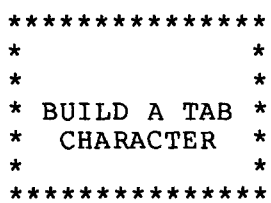
text is the information to be placed in the flowchart block. This text is usually centered within the blocks. Some commands require the text in a specific format.

Spaces and/or tabs may be inserted for legibility.

17.3.1 PROC Command

The PROC (process) command allows you to put up to 65 characters inside a process block. The following process block will be generated by the command line:

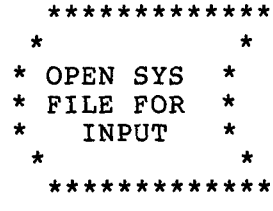
```
PROC   ;BUILD A TAB CHARACTER
```



17.3.2 DISK Command

The DISK command allows you to put up to 55 characters inside a disk block. The following disk block will be generated by the command line:

```
DISK   ;OPEN SYS FILE FOR INPUT
```



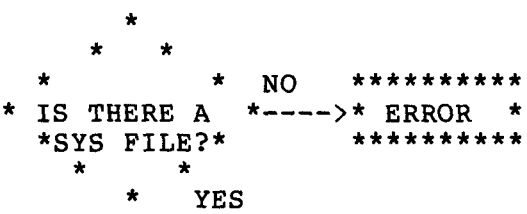
17.3.3 IF Command

The IF command allows you to put up to 37 characters inside a decision block. The IF command requires that the text field be preceded by the following field:

```
YES
     :label to branch to
NO
```

The following decision block will be generated by the command line:

```
IF NO:ERROR ;IS THERE A SYS FILE?
```



17.3.4 CALL Command

The CALL command allows you to put up to 33 characters inside a subroutine block. The CALL command requires that the text field be preceded by the subroutine name.

The following subroutine block will be generated by the command line:

```
CALL    HOF    ;OUTPUT PAGE MARKER
```

17.3.5 START Command

The START command allows you to put up to 13 characters inside a start block. The following start block will be generated by the command line:

```
START     ;HOF ROUTINE

       *************
      * HOF ROUTINE *
       *************
```

17.3.6 STOP Command

The STOP command allows you to put up to 13 characters inside a stop block. The following stop block will be generated by the command line:

```
STOP      ;RETURN

       ************
      *   RETURN   *
       ************
```

17.3.7 GOTO Command

The GOTO command allows you to put up to six characters inside a GOTO block. The following GOTO block will be generated by the command line:

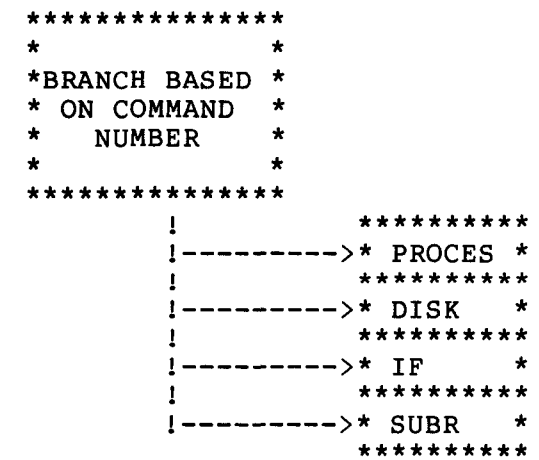
```
GOTO      NEXT

                 **********
          ------>* NEXT   *
                 **********
```

17.3.8 CGOTO Command

The CGOTO (computed GOTO) command allows you to flowchart multiway branches. The text field consists of labels separated by commas without imbedded spaces. The following blocks will be generated by the command lines:

```
PROC     ;BRANCH BASED ON COMMAND NUMBER
CGOTO     PROCES,DISK,IF,SUBR
```



17.3.9 I/O Command

The I/O command allows you to put up to 47 characters inside an I/O block. The following I/O block will be generated by the command line:

```
I/O      ;DISPLAY 'ERROR'
```

17.3.10 TITLE Command

The TITLE command allows you to specify up to 40 characters as a flowchart title. The title will appear at the top of all subsequent pages.

17.3.11 SBTTL Command

The SBTTL command allows you to specify up to 40 characters as a subtitle. The subtitle is printed on the line below the title. The SBTTL command implies a top-of-page command.

17.3.12 PAGE Command

The PAGE command advances the listing to the top of the next page. FLOW automatically generates new pages when necessary, making the PAGE command unnecessary in most instances.

17.4 FLOW EXAMPLE

The best example of the use of the flowchart generator is FLOW itself. FLOW commands have been inserted into the FLOW source files (FLOW1, FLOW2, FLOW3, FLOW4). To produce a flowchart of FLOW, use the following procedure:

- Assign the necessary logical units using DFU.
- Compile the flowchart programs.
- Enter the following commands:

```
.RUN FLOW,FLOW1,FLOW2,FLOW3,FLOW4/PL
.RUN SORT, KRFSRT
.RUN KREF
```

17.5 FLOW ERROR MESSAGES

Message	Explanation
NO INPUT	FLOW has no information to build a flowchart. Build a command file.
ERROR	An error has occurred. The line of text where the error occurred will be displayed on the next line. Check the line and correct the error.

CHAPTER 18

MENU PROGRAM (MENU)

The MENU program allows you to select and execute commands from a previously created command file. MENU permits more orderly execution of commands.

18.1 MENU OPERATING PROCEDURES

To run MENU, type:

```
RUN MENU,cmndfl
```

where:

cmndfl — is the name of the MENU command file stored on the system device. If none is specified, the file in the edit buffer is used.

MENU displays the text found in the Display Section of the command file and will accept a six-character operator response at the screen location specified in the Accept Section. The operator response is compared to the list of valid responses specified in the Command Section. If a match is found, the corresponding Monitor or editor commands are then executed by COS-310.

MENU can be included in a batch command file only as the last command in the file.

18.2 MENU COMMAND FILE

The MENU command file is created using COS-310 editor commands and is stored on the system device. It consists of three sections: Display, Command, and Accept. The order of these sections within the file is vital.

Example:

```
     DISPLAY
COPY COPY RX0 TO RX1
DIR  PRINT DIRECTORY OF RX0
     COMMAND
COPY =ER
     =1 C
     =2 RX0
     =3 RX1
     =4 X
     =WR $COPY/Y
     =PLEASE MOUNT DISKETTE ON DRIVE 1
     =R PIP,$COPY
     =DE $COPY/S
     =R MENU,cmndf1
DIR  =DI,RX0
     =R MENU,cmndf1
     ACCEPT (24,10)
```

18.2.1 Display Section

The Display Section has the form:

```
DISPLAY [/N]
text
.
.
.
```

where:

DISPLAY — is the first statement in the command file (must be DISPLAY).

/N — is an optional switch to suppress clearing the screen prior to displaying the text. Without /N the screen is cleared.

text — is text to display on the screen beginning on line one. Each line of text begins with a new line number. If a line of text contains more characters than can be displayed on a screen line, the extra characters are lost. Any line beginning with a semicolon is assumed to be a comment and is not displayed.

18.2.2 Command Section

The Command Section has the form:

```
COMMAND
code=command
.
.
.
```

where:

COMMAND — is the first line in the section (must be COMMAND).

code — is an operator response that contains a maximum of six characters. The Command Section may contain up to sixty codes.

command — is a COS-310 Monitor or editor command that is executed when its corresponding code is entered as the operator response. There can be no spaces or tabs between the equal sign and the command.

A series of commands may be executed by listing the commands on subsequent lines with no code to the left of the equal sign. The series of commands is combined to produce a batch command file. This batch command file cannot be longer than one block.

There may be as many as 3995 characters in the Command Section.

18.2.3 Accept Section

The Accept Section has the form:

```
ACCEPT (y,x) [/N]
```

where:

ACCEPT — is the first line in the section (must be ACCEPT).

y — is a decimal number (cannot be an expression) designating the screen line number where the operator response is to be entered. If y is greater than the number of lines on the screen the results are unpredictable.

x is a decimal number (cannot be an expression) representing the screen column number where the operator response is to be entered. If x plus the operator response (maximum of six) is greater than the screen width, the results are unpredictable.

/N is an optional switch to suppress clearing the screen prior to executing the selected command from the Command Section.

The location (y,x) may fall within the text displayed by the Display Section.

18.3 MENU ERROR MESSAGES

Message	Explanation
ACCEPT SECTION NOT FOUND	No Accept Section in the command file. Correct command file.
COMMAND SECTION NOT FOUND	No Command Section in the command file. Correct command file.
DISPLAY SECTION NOT FOUND	No Display Section in the command file. Correct command file.
ILLEGAL CURSOR POSITION	An illegal cursor position (or none) was requested in the Accept Section. Correct command file.
ILLEGAL STATEMENT	Command file contains a meaningless statement. Correct command file.
TOO MANY COMMANDS	The Command Section is too large. Reduce size of Command Section.
TOO MANY COMMANDS FOR 1 CODE	The series of commands under one code exceeds 1 block in length. Correct command file.

APPENDIX A

COS-310 CHARACTER SET

In both source and data files, characters (alphanumeric and numeric) are stored two characters per word in six-bit binary. Negative numbers are stored with the high-order bit of the low-order digit set to 1. For example, the number 1234- is stored as two words in the following form:

22 1	23 2	WORD 1
24 3	65 4	WORD 2 (with high-order bit on)

This number is recognized as 123T. This means that any program in which the numeric-to-alphanumeric conversion is not made might produce negative numbers with letters. Refer to Table A-1 for a list of characters representing negative numbers.

Table A-1
Characters Representing Negative Numbers

Negative Number	Equivalent Character	Decimal Code	Octal Code
-0	P	49	61
-1	Q	50	62
-2	R	51	63
-3	S	52	64
-4	T	53	65
-5	U	54	66
-6	V	55	67
-7	W	56	70
-8	X	57	71
-9	Y	58	72

Table A-2
COS-310 Character Set

Decimal Code	Octal Code	Character	Decimal Code	Octal Code	Character
00	00	Null	32	40	?
01	01	Space	33	41	@
02	02	!	34	42	A
03	03	"	35	43	B
04	04	#	36	44	C
05	05	$	37	45	D
06	06	%	38	46	E
07	07	&	39	47	F
08	10	'	40	50	G
09	11	(	41	51	H
10	12	)	42	52	I
11	13	*	43	53	J
12	14	+	44	54	K
13	15	,	45	55	L
14	16	-	46	56	M
15	17	.	47	57	N
16	20	/	48	60	O
17	21	0	49	61	P
18	22	1	50	62	Q
19	23	2	51	63	R
20	24	3	52	64	S
21	25	4	53	65	T
22	26	5	54	66	U
23	27	6	55	67	V
24	30	7	56	70	W
25	31	8	57	71	X
26	32	9	58	72	Y
27	33	:	59	73	Z
28	34	;	60	74	[
29	35	<	61	75	Tab
30	36	=	62	76	]
31	37	>	63	77	↑

APPENDIX B

COS-310 FILES

There are four types of files in the COS-310 system: source, binary, data, and system. Source, binary, and data files have similar structure. System files use standard OS/8 SAVE format.

B.1 COS-310 SOURCE FILES

Each line in a source command file or DIBOL source file must be input with a line number. This makes all source files look the same and makes them compatible with COS-310. Each input line has the following format:

word count (n)	line number	n-1 words, two COS-310 characters per word

The first word contains the word count for that line. It is computed with the following expression.

n = ((number of characters on line +1)/2)+1

The second word is the statement line number, 0000-7777 octal (0000-4095 decimal).

The third and successive words contain the text of the line packed two COS-310 characters per word. The total characters of data per line does not include the two-character (1 word) word count number.

B.2 COS-310 DATA FILES

Every block in a data file is completely devoted to the storage of data. Each logical unit holds only one data file. Labels on data files are associated with logical units by the Monitor in conjunction with DIBOL or system programs.

The format of a line in a data file is similar to the format for a line in a source file except there is no line number on a data file.

A line of text in a data file has the following format:

word count (n)	n words, two characters per word

The first word contains the word count for that line. It is computed with the following expression:

n = (number of characters in record +1)/2

The second and successive words contain the text of the line, two COS-310 characters per word.

B.3 COS-310 BINARY FILES

Although the contents of a binary file are interpreted differently than the contents of a data file, externally the two files are structured exactly alike. That is, the binary code for each line of a DIBOL source program is stored as a word count followed by the interpretive code to be used by the run-time system.

B.4 COS-310 SYSTEM FILES

All system files are stored in OS/8 SAVE format. The first block of the file is a memory control block indicating where in memory the rest of the blocks of the file are to be loaded. Each successive block is a 256-word memory image. See the OS/8 Software Support Manual for details.

B.5 SYSTEM DEVICE FORMAT

COS-310 puts a label on all devices. This label occupies the first 256 words of each device; four words are the actual label, one word is the date, and the other words may be a bootstrap.

Figure B-1 illustrates the layout of the Monitor portion of the system device. As noted in the figure, COMP should be the first file in the file area. The location of COMP is particularly important when the binary scratch area is to be expanded.

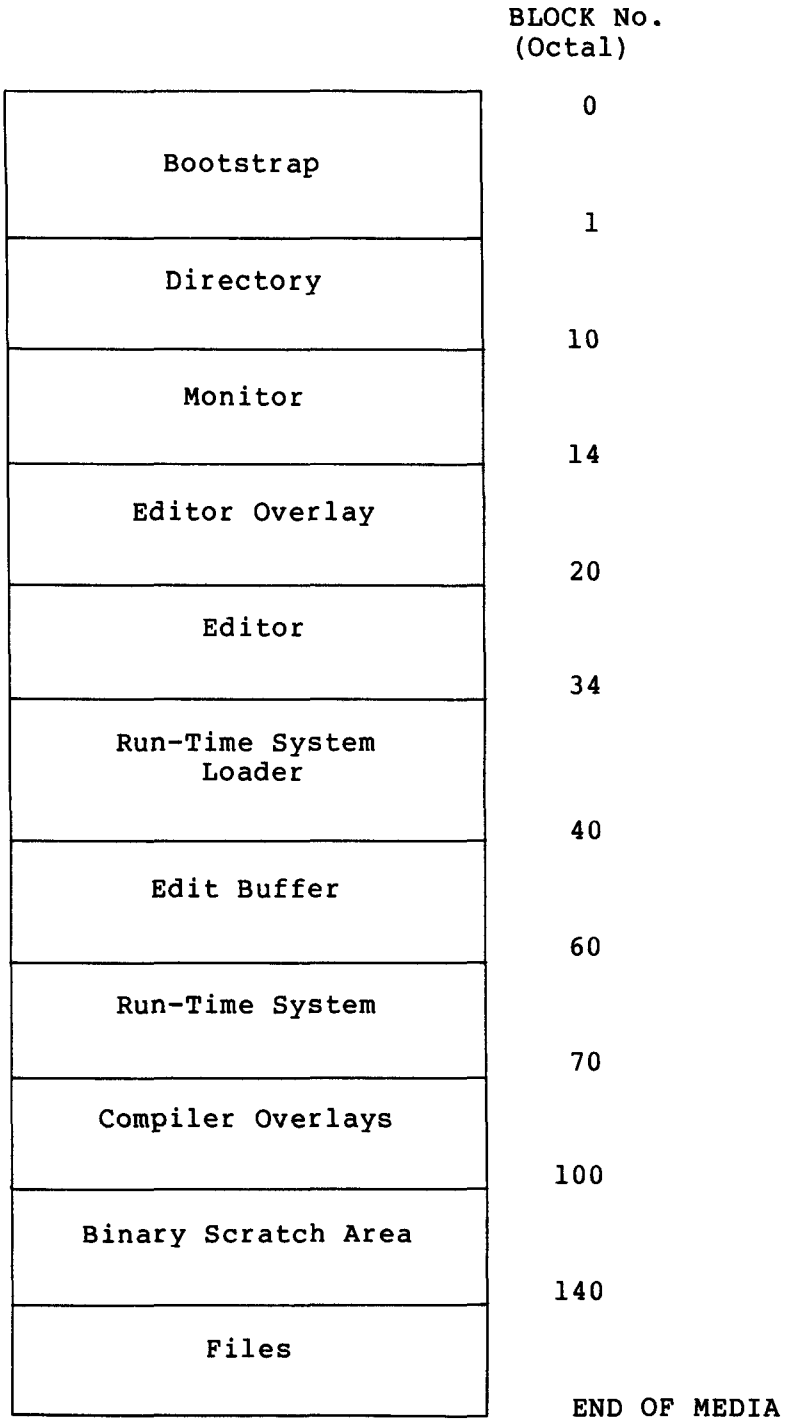


Figure B-1 Monitor Organization

A label is automatically put on a system device. The directory of a system device is organized as follows:

Word	Contents
0	The negative number of directory entries in this block.
1	The starting block number for file storage.
2	The link word to the next directory block or empty if end. There are seven directory blocks on all multifile devices.
3	Empty (unused).
4	The negative number of auxiliary words per entry (always equals -1).
5	The first two characters of name.
6	The next two characters of name.
7	The last two characters of name.
8	A two-character extension.
9	The date.
10	Length of the file (negative).
11-255	Repeat of 5-10 for each file.

Space for other kinds of files is allocated on the disk beginning at the first free block following the COS-310 system files. On an RK05 disk, the system directory knows that the available space for file storage only extends to block 4095.

Access to Data Files

Data Files are referenced by their logical unit numbers as assigned by DFU. DFU actually sets up an internal table containing the following information for each logical unit:

- Handler address
- Drive number
- Starting segment
- Length in segments

The handler address is a pointer to the specific device handler to use for a particular logical unit. The drive number indicates which disk drive to reference. The starting segment is indicated by a 12-bit number which points to the physical device space allocated for the logical unit. The length is the number of segments reserved for this logical unit.

Example:

If logical unit 14 is assigned to a 32-block area on DK1, the fourteenth entry in the table might contain the following information:

DK handler address	drive 1	starting segment 212 (octal)	length -40 (octal)

Any references to logical unit 14 would refer to segments 212-251 (octal) of DK1. The first block in segment 212 would have a label for that logical unit.

APPENDIX C

ERROR MESSAGE INDEX

This index will refer you to the chapters where corrective and background information is located. Locate the error message you have encountered and go to the chapter referenced. If the message is listed more than once, check the message listed with the program that you are running.

Message	Program	Refer to
ACCEPT SECTION NOT FOUND	MENU	Chapter 18
ALPHA LITERAL REQUIRED	PRINT	Chapter 16
ALREADY DEFINED	PRINT	Chapter 16
BAD ALPHA VALUE	COMP	Chapter 5
BAD CHAIN	Run-Time	Chapter 2
BAD CHECKSUM	PATCH	Chapter 11
BAD COMPILATION	Monitor	Chapter 2
BAD DATE	Monitor	Chapter 2
BAD DIGIT	Run-Time	Chapter 2
BAD DIGIT IN DATA	DAFT	Chapter 15
BAD DIGIT IN NUMERIC INITIAL VALUE	SORT	Chapter 9
BAD DIRECTORY	FILEX	Chapter 10
BAD DIRECTORY	Monitor	Chapter 2
BAD DIRECTORY	PATCH	Chapter 11
BAD DIRECTORY	PIP	Chapter 8

Message	Program	Refer to
BAD LABEL	Monitor	Chapter 2
BAD NUMBER	PATCH	Chapter 11
BAD NUMERIC VALUE	COMP	Chapter 5
BAD PROC #	COMP	Chapter 5
BAD PROGRAM	Run-Time	Chapter 2
BAD RECORD SIZE	SORT	Chapter 9
BAD RELATIONAL	COMP	Chapter 5
BAD SWITCH	SYSGEN	Chapter 3
BAD SWITCH	DFU	Chapter 4
BAD WORK UNIT COUNT	SORT	Chapter 9
BLOCK TOO BIG	PATCH	Chapter 11
CANT BACKSPACE PAST BEGIN OF FILE	DAFT	Chapter 15
CANT BACKSPACE WITH SEQUENTIAL INPUT	DAFT	Chapter 15
CCP ERROR	COMP	Chapter 5
COMMA MISSING	COMP	Chapter 5
COMMAND SECTION NOT FOUND	MENU	Chapter 18
COMPARISON ERROR	PIP	Chapter 8
DATA INITIALIZATION MISSING	COMP	Chapter 5
DEVICE ERROR	FILEX	Chapter 10
dev MUST BE INCLUDED IN CONFIGURATION	SYSGEN	Chapter 3
DIBOL FILE NUMBER IN USE	Run-Time	Chapter 2
DIBOL FILE NUMBER NOT INITED	Run-Time	Chapter 2
DISPLAY SECTION NOT FOUND	MENU	Chapter 18
EDIT BUFFER FULL	Monitor	Chapter 2

Message	Program	Refer to
END OF FILE	Run-Time	Chapter 2
END OF INPUT FILE AT RECORD nnnn	DAFT	Chapter 15
ERR IN CMD	DDT	Chapter 6
ERROR	FLOW	Chapter 17
ERROR IN COMMAND	Monitor	Chapter 2
ERROR ON dev, RETRY?	Monitor	Chapter 2
EXCESSIVE GRID SIZE	DAFT	Chapter 15
EXPECTED LABEL IS MISSING	COMP	Chapter 5
EXPRESSION NOT ALLOWED	COMP	Chapter 5
EXTRA CHARS	DAFT	Chapter 15
EXTRA CHARS AT STMNT END	COMP	Chapter 5
EXTRA CHARS AT STMNT END	SORT	Chapter 9
FIELD NUMBER MISSING OR 0	SORT	Chapter 9
FIELD TOO LARGE OR 0	COMP	Chapter 5
FILE ALREADY EXISTS	FILEX	Chapter 10
FILE NOT FOUND	Monitor	Chapter 2
FILE NOT FOUND	PATCH	Chapter 11
FULL	FILEX	Chapter 10
FULL	SYSGEN	Chapter 3
HEADER IS TOO LONG	PRINT	Chapter 16
ILLEGAL CURSOR POSITION	MENU	Chapter 18
ILLEGAL DEVICE	DFU	Chapter 4
ILLEGAL DEVICE	FILEX	Chapter 10
ILLEGAL DEVICE	Run-Time	Chapter 2
ILLEGAL DEVICE SWITCH	PIP	Chapter 8

Message	Program	Refer to
ILLEGAL OPERATOR	COMP	Chapter 5
ILLEGAL PROGRAM	Monitor	Chapter 2
ILLEGAL RECORD - CLOSING FILE	DAFT	Chapter 15
ILLEGAL RECORD #	Run-Time	Chapter 2
ILLEGAL SORT KEY	SORT	Chapter 9
ILLEGAL STATEMENT	MENU	Chapter 18
ILLEGAL STMNT	COMP	Chapter 5
ILLEGAL SUBSTRING	Run-Time	Chapter 2
ILLEGAL UNIT	Monitor	Chapter 2
ILLEGAL UNIT	SORT	Chapter 9
IMPROPER DEFINITION	PRINT	Chapter 16
IMPROPER LITERAL	PRINT	Chapter 16
IMPROPER USE OF DECIMAL PLACES	PRINT	Chapter 16
IN USE	Monitor	Chapter 2
INSUFFICIENT SPACE ON DEVICE	DFU	Chapter 4
INSUFFICIENT SPACE ON DEVICE	FILEX	Chapter 10
INITIAL ALPHA VALUE DOESN'T BEGIN WITH QUOTE	SORT	Chapter 9
INITIAL VALUE TOO BIG	SORT	Chapter 9
INITIAL VALUE TOO SMALL	SORT	Chapter 9
INITIAL VALUE WRONG SIZE	COMP	Chapter 5
INTEGER FROM 1-15 REQUIRED	PRINT	Chapter 16
INTEGER FROM 1-132 REQUIRED	PRINT	Chapter 16
INTEGER REQUIRED	PRINT	Chapter 16
INVALID OPERATION	PATCH	Chapter 13
KEY ENTIRELY PAST END OF RECORD	DAFT	Chapter 15

Message	Program	Refer to
KEY EXTENDS PAST END OF RECORD	DAFT	Chapter 15
KEY TOO BIG	DAFT	Chapter 15
LABEL NOT ALLOWED	COMP	Chapter 5
LINE TOO LONG	Monitor	Chapter 2
LINE # TOO LARGE	Monitor	Chapter 2
LINE TOO LONG	Run-Time	Chapter 2
LITERAL TOO LONG	PRINT	Chapter 16
LOCATION TOO BIG	PATCH	Chapter 11
MISSING CLOSE PAREN	COMP	Chapter 5
MISSING CLOSE QUOTE ON ALPHA INITIAL VALUE	SORT	Chapter 9
MISSING INITIAL VALUE	SORT	Chapter 9
MISSING OPEN PAREN	COMP	Chapter 5
MISSING OPERAND	COMP	Chapter 5
MISSING OR BAD MODE	COMP	Chapter 5
MISSING QUOTE	COMP	Chapter 5
MISSING RELATIONAL	COMP	Chapter 5
MOUNT filnam #nn FOR INPUT:	Monitor	Chapter 2
MOUNT filnam #01 FOR INPUT:	Monitor	Chapter 2
MOUNT filnam #nn FOR OUTPUT:	Monitor	Chapter 2
MUST BE IDENT	PRINT	Chapter 16
MUST BE NUMERIC ITEM	PRINT	Chapter 16
MUST BE S	PRINT	Chapter 16
NAME PREVIOUSLY DEFINED	COMP	Chapter 5
NEED FILE NAME	PRINT	Chapter 16
nnn IS BEING IGNORED	CREF	Chapter 7

Message	Program	Refer to
NO	BOOT	Chapter 12
NO BUFFERS LEFT	Run-Time	Chapter 2
NO CHANGE IN BLOCK	PATCH	Chapter 11
NO COMMA AFTER FIELD NAME	SORT	Chapter 9
NO DATA	DAFT	Chapter 15
NO END	FILEX	Chapter 10
NO ENDING QUOTE	PRINT	Chapter 16
NO FILE	Run-Time	Chapter 2
?NO FILE TO SAVE	Monitor	Chapter 2
NO INIT	Monitor	Chapter 2
NO INPUT	FLOW	Chapter 17
NO INPUT	SORT	Chapter 9
NO INPUT DIRECTIVE	PRINT	Chapter 16
NO INPUT FILE	DAFT	Chapter 15
NO LABEL NAME	DAFT	Chapter 15
NO LP BUFFER	Monitor	Chapter 2
NO OUTPUT FILE	DAFT	Chapter 15
NO PRINT ITEMS	PRINT	Chapter 16
NO ROOM	FILEX	Chapter 10
NO ROOM	PIP	Chapter 8
NOT A OR D	COMP	Chapter 5
NOT A OR D	SORT	Chapter 9
NOT DEFINED	PRINT	Chapter 16
NOT ENOUGH RIGHT PARENTHESES	PRINT	Chapter 16
NOT ENOUGH ROOM FOR SYSTEM AND FILES	DFU	Chapter 4

Message	Program	Refer to
NOT ENOUGH ROOM FOR SYSTEM AND FILES	FILEX	Chapter 10
NOT FOUND	FILEX	Chapter 10
NOT LABEL	COMP	Chapter 5
NOTHING AFTER FIELD NAME	SORT	Chapter 9
NUMBER REPEATED OR OUT OF ORDER	SORT	Chapter 9
# TOO LARGE	DFU	Chapter 4
NUMBER TOO LONG	Run-Time	Chapter 2
OUTPUT ERROR	SORT	Chapter 9
OUTPUT FILE ALREADY OPEN	DAFT	Chapter 15
OUTPUT FILE STILL OPEN	DAFT	Chapter 15
PICTURE TOO LONG	PRINT	Chapter 16
PROGRAM TOO BIG	COMP	Chapter 5
PROGRAM TOO BIG	Run-Time	Chapter 2
PUSHDOWN OVERFLOW	DAFT	Chapter 15
PUSHDOWN OVERFLOW	Run-Time	Chapter 2
RECORD TOO BIG	COMP	Chapter 5
REPLACE?	Monitor	Chapter 2
REPLACE filnam #nn ?	Monitor	Chapter 2
RETURN WITHOUT CALL	Run-Time	Chapter 2
STMNT TOO COMPLEX	COMP	Chapter 5
SUBSCRIPT ERROR	COMP	Chapter 5
SUBSCRIPT NOT NUMERIC	COMP	Chapter 5
SUBSCRIPT TOO BIG	Run-Time	Chapter 2
SYNTAX ERROR	DFU	Chapter 4
SYNTAX ERROR	PRINT	Chapter 16

Message	Program	Refer to
TOO MANY COLUMNS IN REPORT	PRINT	Chapter 16
TOO MANY COMMANDS	MENU	Chapter 18
TOO MANY COMMANDS FOR 1 CODE	MENU	Chapter 18
TOO MANY COMPUTE STATEMENTS	PRINT	Chapter 16
TOO MANY DATA ITEMS	PRINT	Chapter 16
TOO MANY FILES	SORT	Chapter 9
TOO MANY ITEMS	COMP	Chapter 5
TOO MANY LEFT PARENTHESES	PRINT	Chapter 16
TOO MANY LIST ITEMS	PRINT	Chapter 16
TOO MANY RIGHT PARENTHESES	PRINT	Chapter 16
TOO MANY SYMBOLS!	COMP	Chapter 5
TOO MUCH DATA	COMP	Chapter 5
UNDEFINED NAME	COMP	Chapter 5
UNKNOWN DIRECTIVE	PRINT	Chapter 16
UNRECOGNIZABLE LINE	SORT	Chapter 9
WRONG DATA TYPE	COMP	Chapter 5
ZERO DIVISOR	Run-Time	Chapter 2
0 NOT ALLOWED	DAFT	Chapter 15

APPENDIX D

ADVANCED PROGRAMMING TECHNIQUES

D.1 ACCEPT AND DISPLAY

D.1.1 Background Information

XMIT statements were originally used when the terminal was a Teletype[1]. The VT52 display terminal uses newer concepts -- programmable cursor control and hardware display clear. ACCEPT and DISPLAY statments were added to the DIBOL language to use these features. The terminal can now be used in two ways:

1. As a Teletype by using XMIT statements.

2. As a powerful data entry tool by using ACCEPT and DISPLAY statements.

(Refer to the ACCEPT and DISPLAY statements in Chapter 1 before proceeding further.)

D.1.2 Interaction of ACCEPT and DISPLAY

ACCEPT and DISPLAY statements are used extensively in data entry programs. These data entry programs typically work one of two ways. The first asks (DISPLAY) questions and interprets (ACCEPT) answers. This method of operation closely simulates a Teletype. The second method displays a format or heading on the screen and moves the cursor either to the right or to a position below the question to be answered.

With the second method, the format is never cleared but data is entered and cleared continuously from the screen. This method is used in repetitive data entry and updating. Quite often the four keys up arrow, down arrow, left arrow, and right arrow have special meanings. For example, assume ten headings are displayed on the screen, indicating ten fields are to be entered or updated. The up arrow might be used to re-enter information in the first field, no matter which field

is currently being entered; the down arrow might mean no more information for any of the fields; the left arrow might restart entering data into the current field; the right arrow might mean go on to the next field without changing the current field.

D.1.3 Example Using ACCEPT and DISPLAY

To enter a six-digit customer number and a 15-character customer name, the following program might be used:

```
                RECORD
TCHAR,D2
ALPHA,A15
CNO,D6
CNAME,A15
                PROC 1
                DISPLAY(1,1,1)  ;Clear screen and position cursor.
                DISPLAY(0,0,'CUSTOMER NO.   CUSTOMER NAME')
LOOP,           DISPLAY(2,1,2)  ;Clear line 2 and position cursor.
                ALPHA=          ;Clear this field.
                ACCEPT(TCHAR,ALPHA)
                ON ERROR LOOP   ;Re-enter if not numeric.
                CNO=ALPHA
                ALPHA=          ;Clear this field again.
                DISPLAY(2,16,0) ;Position cursor.
                ACCEPT(TCHAR,ALPHA)
                CNAME=ALPHA
                .
                .               ;Save data.
                .
                GO TO LOOP
```

D.1.4 Generalized ACCEPT Subroutines

D.1.4.1 Hardware Display Clear Feature - Although the previous example works properly, it lacks features which would be useful:

1. Type RUBOUT to clear the previously entered character from both the program and the display.

2. Type CTRL/U (a DIGITAL convention) to clear the entire current line from both the program and the display.

Since data acceptance is getting more sophisticated, it can best be performed by calls to a subroutine. The following two subroutines and test programs will accept data from the keyboard and use the RUBOUT

key and the CTRL/U key as previously specified. The first program uses the clearing feature built into the hardware of the VT52. Unfortunately, this feature destroys data if it is on the same line and to the right of what is being accepted.

```
        START   ;Erases remainder of line for errors.
        RECORD
KBDBUF, A80     ;Storage for keyboard input.
        RECORD ,X
KBDIN,  80A1

        RECORD  ;Work area.
ROW,    D2      ;Cursor Y-coordinate on entry to subroutine VT52
                ;(needed for correction only).
COL,    D2      ;Cursor X-coordinate on entry to subroutine VT52
                ;(needed for correction only).
TCHAR,  D2      ;Terminating character in an accept statement.
CHAR,   A1      ;Input character from an accept statement.
VT52IN, D2      ;Number of characters accepted by subroutine VT52.
VTLIM,  D2      ;Number of characters to be accepted by
                ;subroutine VT52.
        PROC
BEGIN,  DISPLAY(1,1,1)  ;Clear screen.          ***********
        DISPLAY(1,40,'ERASED IN CORRECTION')    *         *
        DISPLAY(1,1,'NAME:')                    *         *
        ROW=1                                   * SAMPLE  *
        COL=6                                   *  TEST   *
        VTLIM=20        ;20 characters maximum. * PROGRAM *
        CALL VT52                               *         *
        IF (KBDBUF.EQ.'END') STOP               *         *
        GO TO BEGIN                             ***********



;       Calling sequence
;               ROW= Y coordinate
;               COL= X coordinate
;               VTLIM= Maximum number of characters to accept
;               CALL VT52

;Accept a maximum of VTLIM characters at location specified by
;ROW and COL. Return when either the maximum number of characters
;(VTLIM) has been entered, a termination character is entered,
;or a space is entered. Rubout deletes last character entered and
;CTRL/U eliminates the entire entry. RUBOUT and CTRL/U clear the
;remainder of the line faster than displaying spaces.

VT52,   VT52IN=
        KBDBUF=
VT522,  ACCEPT(TCHAR,CHAR)
        IF (TCHAR.EQ.0) GO TO VT523     ;Nonterminating character.
        IF (TCHAR.EQ.21) GO TO VT524    ;CTRL/U.
```

```
        IF (TCHAR.EQ.32) GO TO VT525    ;RUBOUT.
        RETURN                          ;Terminating character other
                                        ;than rubout or CTRL/U.
VT523,  IF(CHAR.EQ.' ') RETURN          ;Space is a terminating
                                        ;character.
                                        ;To eliminate this feature,
                                        ;remove this statement and put
                                        ;label on next statement.
        INCR VT52IN                     ;VT52IN=# of input characters.
        KBDIN(VT52IN)=CHAR
        IF (VT52IN.EQ.VTLIM) RETURN     ;The specified number of
                                        ;characters were input.
        GO TO VT522
VT524,  IF(VT52IN.EQ.0) GO TO VT52
        DISPLAY(ROW,COL,2)              ;Clear characters entered
                                        ;to end-of-line.
        GO TO VT52
VT525,  IF(VT52IN.EQ.0) GO TO VT522
        KBDIN(VT52IN)=
        VT52IN=VT52IN-1
        DISPLAY(ROW,COL+VT52IN,2)       ;RUBOUT previous character
                                        ;to end-of-line.
        GO TO VT522
```

D.1.4.2 Clear Incorrect Data by Displaying Spaces - The following program clears incorrectly entered data by displaying spaces. This is slower than using the hardware display clear feature, but data on the same line and to the right is not cleared.

```
        START   ;Corrects only characters in error.
        RECORD
KBDBUF, A80     ;Storage for keyboard input.
        RECORD ,X
KBDIN,  80A1

        RECORD  ;Work area.
BLNK80, A80     ;80 blank characters.
ROW,    D2      ;Cursor Y coordinate on entry to subroutine VT52
                ;(needed for correction only).
COL,    D2      ;Cursor X coordinate on entry to subroutine VT52
                ;(needed for correction only).
TCHAR,  D2      ;Terminating character in an ACCEPT statement.
CHAR,   A1      ;Input character from an ACCEPT statement.
VT52IN, D2      ;Number of characters accepted by subroutine VT52.
VTLIM,  D2      ;Number of characters to be accepted by
                ;subroutine VT52.
VT52XX, D2      ;Temporary storage for subroutine VT52.
        PROC
```

```
BEGIN,  DISPLAY(1,1,1)  ;Clear screen.          ***********
        DISPLAY(1,40,'NEVER ERASED')            *         *
        DISPLAY(1,1,'NAME:')                    *         *
        ROW=1                                   * SAMPLE  *
        COL=6                                   *  TEST   *
        VTLIM=20        ;20 characters maximum. * PROGRAM *
        CALL VT52                               *         *
        IF (KBDBUF.EQ.'END') STOP               *         *
        GO TO BEGIN                             ***********


;       Calling sequence
;               ROW= Y-coordinate
;               COL= X-coordinate
;               VTLIM= Maximum number of characters to accept
;               CALL VT52
;Accept a maximum of VTLIM characters at location specified by
;ROW and COL. Return when either the maximum number of characters
;(VTLIM) has been entered, a termination character is entered,
;or a space is entered. RUBOUT deletes last character entered and
;CTRL/U eliminates the entire entry. RUBOUT and CTRL/U display
;space(s) to delete only the necessary characters (not the
;remainder of the line).

VT52,   VT52IN=
        KBDBUF=
VT522,  ACCEPT(TCHAR,CHAR)
        IF (TCHAR.EQ.0) GO TO VT523     ;Nonterminating character.
        IF (TCHAR.EQ.21) GO TO VT524    ;CTRL/U.
        IF (TCHAR.EQ.32) GO TO VT525    ;RUBOUT.
        RETURN                          ;Terminating character other
                                        ;than RUBOUT or CTRL/U.
VT523,  IF(CHAR.EQ.' ') RETURN          ;Space is a terminating
                                        ;character.
                                        ;To eliminate this feature,
                                        ;remove this statement.
        INCR VT52IN                     ;VT52IN=# of input characters.
        KBDIN(VT52IN)=CHAR
        IF (VT52IN.EQ.VTLIM) RETURN     ;The specified number of
                                        ;characters were input.
        GO TO VT522
VT524,  IF(VT52IN.EQ.0) GO TO VT52
        DISPLAY(ROW,COL,BLNK80(1,VT52IN))   ;Clear characters entered.
        DISPLAY(ROW,COL,0)              ;Reposition cursor.
        GO TO VT52
VT525,  IF(VT52IN.EQ.0) GO TO VT522
        KBDIN(VT52IN)=
        VT52IN=VT52IN-1
        VT52XX=VT52IN+COL
        DISPLAY(ROW,VT52XX,' ')         ;Rubout previous character.
        DISPLAY(ROW,VT52XX,0)           ;Reposition cursor.
        GO TO VT522
```

D.1.4.3 Other Desired Features - In addition to the features found in the previous program, the following features might also be desired:

1. Right justification of numeric fields.

2. Automatic cursor positioning.

These features are used in the following subroutine and test program:

```
        START   ;Subroutine VT52A and VT52N.
        RECORD
KBDBUF, A80     ;Storage for keyboard input.
        RECORD ,X
KBDIN,  80A1

        RECORD  ;Work area.
BLNK80, A80     ;80 blank characters.
ROW,    D2      ;Cursor Y coordinate.
COL,    D2      ;Cursor X coordinate.
TCHAR,  D2      ;Terminating character in an ACCEPT statement.
CHAR,   A1      ;Input character from an ACCEPT statement.
VT52IN, D2      ;Number of characters accepted.
VTLIM,  D2      ;Number of characters to be accepted.
VT52SW, D1      ;Cleared for alpha input, set to 1 for numeric input.
VT5215, D15     ;Contains numeric input for VT52N entry. Not changed
                ;or used in VT52A entry.
VT52XX, A16     ;Temporary storage for redisplay of numeric input.
        PROC 0
        DISPLAY(1,1,1)  ;Clear screen.          **********
BEGIN,  INCR ROW                                *        *
        IF (ROW .GT. 24) STOP                   *        *
        DISPLAY (ROW,53,'NOT ERASED')           *        *
        DISPLAY (ROW,1,'NAME:')                 *  SAMPLE *
        COL=7                                   *        *
        VTLIM=20        ;20 characters maximum. *   TEST  *
        CALL VT52A                              *        *
        IF (KBDBUF.EQ.'END') STOP               * PROGRAM *
        DISPLAY(ROW,30,'NO:')                   *        *
        COL=34                                  *        *
        VTLIM=15                                *        *
        CALL VT52N                              *        *
        GO TO BEGIN                             **********


;       Calling sequence
;               ROW= Y coordinate
;               COL= X coordinate
;               VTLIM= Maximum number of characters to accept
;               CALL VT52A for alphanumeric input
;               CALL VT52N for numeric input
```

```
;Accept a maximum of VTLIM characters at location specified by ROW
;and COL. Return when VTLIM characters or a termination character
;is entered.  For numeric input, a space is a terminator.
;RUBOUT deletes last character entered and CTRL/U eliminates the
;entire entry. RUBOUT and CTRL/U display space(s) to delete only
;the necessary characters (not the remainder of the line).
;For numeric input, the entire entry is redisplayed right-justified
;with leading zeros suppressed. VT5215 contains the number
;on return to the calling program.

VT52A,  VT52SW=          ;Entry for alphanumeric input.
        GO TO VT52
VT52N,  VT52SW=1         ;Entry for numeric input.
VT52,   VT52IN=
        KBDBUF=
        DISPLAY(ROW,COL,0)      ;Position cursor.
VT522,  ACCEPT(TCHAR,CHAR)
        IF(TCHAR.EQ.0) GO TO VT523      ;Nonterminating character.
        IF(TCHAR.EQ.21) GO TO VT524     ;CTRL/U.
        IF (TCHAR.EQ.32) GO TO VT525    ;RUBOUT.
VT522X, IF (VT52IN.EQ.0) RETURN         ;No input except terminating
                                        ;character.
        IF (VT52SW.EQ.0) RETURN         ;Alphanumeric input.
VT522Y, VT5215=KBDBUF(1,VT52IN)         ;Numeric input (can't exceed
                                        ;15 digits).
        VT52XX(1,VTLIM+1)=VT5215,'XXXXXXXXXXXXXXX-' ;Allows negative
                                        ;numbers.
        DISPLAY(ROW,COL,VT52XX(1,VTLIM+1))      ;Display numeric input
                                        ;right-justified and zero
                                        ;suppressed.
        RETURN
VT523,  IF (VT52SW.NE.1) GO TO VT523X   ;Save alphanumeric input.
        IF (CHAR.EQ.' ') GO TO VT522X   ;Space as a terminating
                                        ;character for numeric input.
        IF (CHAR.EQ.'-') GO TO VT523X   ;Minus sign is acceptable.
        IF (CHAR.LT.'0') GO TO VT523B   ;Check for numeric input.
        IF (CHAR.LE.'9') GO TO VT523X
VT523B, DISPLAY (0,0,7)                 ;Sound alarm--bad input.
        GO TO VT52                      ;Start over (don't clear
                                        ;the error).
VT523X, INCR VT52IN                     ;VT52IN=# of input characters.
        KBDIN(VT52IN)=CHAR
        IF (VT52IN.EQ.VTLIM) GO TO VT526    ;The specified number of
                                        ;characters were input.
        GO TO VT522
VT524,  IF (VT52IN.EQ.0) GO TO VT52
        DISPLAY(ROW,COL,BLNK80(1,VT52IN))   ;Clear characters entered.
        GO TO VT52
VT525,  IF (VT52IN.EQ.0) GO TO VT522
        KBDIN(VT52IN)=
        VT52IN=VT52IN-1
        VT52XX=VT52IN+COL
        DISPLAY(ROW,COL+VT52IN,' ')     ;Rubout previous character.
```

```
        DISPLAY(ROW,COL+VT52IN,0)       ;Reposition cursor.
        GO TO VT522
VT526,  IF(VT52SW.EQ.1) GO TO VT522Y
        RETURN
```

D.1.4.4 Escape Code Sequences as Terminators - A command protocol is built around the Escape code (27 decimal) to implement commands needed by the VT50 and VT52, but not found in 7-bit ASCII. Upon receiving the Escape code 27, the terminal is set to Escape mode and treats the next character received as a command. Commands created in this manner are called Escape Sequences.

In order to use the VT50/VT52 cursor positioning keys as terminators for an ACCEPT statement, the DIBOL program must check for the Escape code (decimal 27) and then execute another ACCEPT statement into a one character alphanumeric field. The contents of this variable can be checked to determine which key was typed. The program then will erase the alpha character entered in this manner and go to the routine associated to the key that was typed.

SPECIAL ESCAPE SEQUENCES

27-A↑	Cursor Positioning Functions
27-B↓	
27-C←	
27-D→	
27-P	Special function keys at top of numeric keypad (Unlabeled at present)
27-Q	
27-R	

D.2 DIRECT ACCESS TECHNIQUES

D.2.1 Background Information

A file contains records of fixed or variable length.

Regardless of the record size, the operating system automatically writes the records into 512-character blocks. The size of a record (in characters) is two plus the number of characters in all the fields in the record. (The two added characters represent the record size in characters divided by two.) If the resulting record size is odd, add one character since only an even number of characters may be written.

Example:

> If the two fields in a record are defined as a D9 field and an A88 field, the record size is 100 (2+9+88+1).

Assuming that all of the records in this file are the same length, the operating system will pack 5 records and the first 12 characters of the sixth record into the first block; the last 88 characters from the sixth record, 4 records, and the first 24 characters from the eleventh record into the second block; and so on to the end.

When this file is later processed, either sequentially (defined as input in an INIT statement) or through direct access (defined as UPDATE in an INIT statement), the operating system will completely restore the record, even if it overlaps two blocks, before passing it to the DIBOL program.

D.2.2 The Reason for Direct Access

Many applications involve the sequential processing of data. For example, a transaction file is entered in random order, sorted and then used to update a master file sorted in the same sequence. Errors in the transaction file cannot be found until the UPDATE program is run. The errors are corrected and a new transaction file is made for the corrected items, which is then sorted and run against the master file. This process continues until no more errors exist. This type of processing evolved 20 years ago with the age of electronic data processing. Systems specialists have desired a better method of operation.

The best method is to verify that data is entered correctly. The operator keying the data file should be able to interact with the master file. For example, a program can be written in which an operator entering payroll information could type an employee number and know within a second or two whether this employee exists on the master file. This would be impossible with sequential processing because of the time involved in sequentially accessing every record. Direct access permits retrieval of any desired record without processing any other records.

D.2.3 How the Direct Access Technique Works in DIBOL

DIBOL uses a record number to access any record in a file. The program has to convert operator input into a record number recognizable by the operating system. This section on direct access will explain several methods to make this conversion.

D.2.4 Unsorted File

Assume that you have an unsorted file containing 1 to 99 records. Each record contains a KEY field as well as other fields. This key will be used for direct access. The first thing done in the following program is to fill up a table. There is a one-to-one correspondence between each element in the table and each record in the file. No I/O is necessary to determine if a specified code is in the master file since this code would not have a match in the table lookup.

```
                RECORD MASTER
KEY,            D5                      ;Could be any size field.
,               A90                     ;Remainder of file.
                RECORD                  ;Working storage.
TABLE,          100D5                   ;Table containing keys.
I,              D3                      ;Index.
LOOKUP,         D5
                PROC 1
                INIT(1,INPUT,'FILNAM')
LOAD,           XMIT(1,MASTER,EOF)
                INCR I
                TABLE(I)=KEY
                GOTO LOAD
EOF,            FINI(1)
                INCR I
                TABLE(I)=99999          ;Indicates end of table.
                INIT(1,UPDATE,'FILNAM')
                .
                .                       ;LOOKUP contains code for master
                                        ;file lookup.
                .
                I=
FINDIT,         INCR I
                IF(TABLE(I).EQ.LOOKUP) GO TO FOUND ;Match.
                IF(TABLE(I).EQ.99999) GO TO NONE ;No match.
                GOTO FINDIT
NONE            XMIT (8,'RECORD NOT FOUND')
                STOP
FOUND,          READ(1,MASTER,I)        ;Read record I.
```

D.2.5 Sorted File

Use the same circumstances as in Section D.2.4 except sort the file by key. Filling the table is the same, but table lookup is faster since the code is not compared to every element in the table. A "no match" condition is known as soon as the table element exceeds the code.

It is possible to cut down the number of comparisons in the table lookup by comparing the middle of the table to the code, checking which half of the table might contain the code, determining the middle of that half of the table, and so on until the element is found. This technique allows faster access, but programming it is much more complicated.

```
                RECORD MASTER
KEY,            D5
,               A90
                RECORD              ;Working storage.
TABLE,          100D5
I,              D3
LOOKUP,         D5
                PROC 1
                INIT(1,INPUT,'FILNAM')
LOAD,           XMIT(1,MASTER,EOF)
                INCR I
                TABLE(I)=KEY
                GOTO LOAD
EOF,            FINI(1)
                INCR I
                TABLE(I)=99999      ;Indicates end of table.
                INIT(1,UPDATE,'FILNAM')
                .
                .                   ;Lookup contains code for master
 file.
                .
                I=
FINDIT,         INCR I
                IF(TABLE(I).EQ.LOOKUP) GO TO FOUND ;Match.
                IF(TABLE(I).GT.LOOKUP) GO TO NONE  ;No match.
                GOTO FINDIT
NONE,           XMIT (8,'RECORD NOT FOUND')
                STOP
FOUND,          READ(1,MASTER,I)    ;Read record I.
                .
                .
                .
```

It is impractical to use direct access with DIBOL on an unsorted file containing many records since an exceedingly large lookup table would be needed.

D.2.6 Rough Table, No Index File

At some point, a file will contain too many records for every key to be saved in a table. When this point is reached, two solutions are available.

The first is to create a "rough" index table containing every 10th or 20th key. For lookup, the rough index will specify within 10 or 20 records on the master file which one is desired. These 10 or 20 records are then sequentially examined to find the desired record (see the following example program).

The second solution is to create a "rough" index table and a "fine" index file. In this method, the rough index table specifies to within 10 or 20 records of the file desired. The index file is then sequentially examined to find the desired key. If a match occurs, the master file is then read.

The proper use of an index file technique can cut down on the number of I/O reads. For example, a master file of 98 characters per record would take up to four I/O reads to find the desired record if the rough index could narrow within 20 records. An index file technique would take one I/O read to find the master record. This technique becomes faster as the size of the master file record increases.

```
                RECORD MASTER
KEY,            D5
,               A90
                RECORD          ;Working storage.
TABLE,          100D5           ;1st,21st,41st key,etc.
I,              D4
J,              D4
LOOKUP,         D5
                PROC 1
                INIT (1,INPUT,'FILNAM')
LOAD,           XMIT(1,MASTER,EOF)
                INCR I
                IF (I.NE.I/20*20+1) GO TO LOAD
                INCR J
                TABLE(J)=KEY    ;Save only 1st,21st,41st key, etc.
                GO TO LOAD
EOF,            FINI(1)
                INCR J
                TABLE(J)=99999  ;Indicates end of table.
                INIT(1,UPDATE,'FILNAM')
```

```
                .
                .                ;LOOKUP contains code for master file.
                .
                I=1
ROUGH,          INCR I
                IF(TABLE(I).LE.LOOKUP) GO TO ROUGH ;No rough match yet.
                I=(I-2)*20       ;Set I to beginning of rough index-1.
FINE,           INCR I
                READ(1,MASTER,I)
                IF(KEY.LT.LOOKUP) GO TO FINE ;No match yet.
                IF(KEY.EQ.LOOKUP) GO TO FOUND ;No match.
                XMIT (8,'RECORD NOT FOUND')
                STOP
FOUND,          .
                .
                .
```

D.2.7 Rough Table Plus Index File

```
                RECORD MASTER
KEY,            D5
,               A90
                RECORD                      ;Working storage.
TABLE,          100D5                       ;1st,21st,41st key, etc.
I,              D4
J,              D4
LOOKUP,         D5
                RECORD INDEX                ;Index file.
XKEY,           D5
                PROC 2
                INIT(1,INPUT,'FILNAM')
                INIT(2,OUTPUT,'XFILE')
LOAD,           XMIT(1,MASTER,EOF)
                INCR I
                XKEY=KEY
                XMIT(2,INDEX)               ;Create fine index file.
                IF(I.NE.I/20*20+1) GO TO LOAD
                INCR J
                TABLE(J)=KEY                ;Save only 1st,21st,41st key.
                GO TO LOAD
EOF,            FINI(1)
                FINI(2)
                INCR J
                TABLE(J)=99999              ;Indicates end of table.
                INIT(1,UPDATE,'FILNAM')
                INIT(2,UPDATE,'XFILE')
                .
                .                ;LOOKUP contains code for master file.
                .
                I=1
```

```
ROUGH,          INCR I
                IF(TABLE(I).LE.LOOKUP) GO TO ROUGH ;No rough match yet.
                I=(I-2)*20      ;Set to beginning of rough index-1.
FINE,           INCR I
                READ(2,INDEX,I)           ;Read index record.
                IF(XKEY.LT.LOOKUP) GO TO FINE ;No match yet.
                IF(XKEY.EQ.LOOKUP) GO TO FOUND ;No match.
                XMIT (8,'RECORD NOT FOUND')
                STOP
                READ(1,MASTER,I)          ;Match.
FOUND,          .
                .
                .
```

D.2.8 Summary

This discussion on direct access does not include information about all possible situations. In cases where the master file is between 2,000 and 40,000 records, the approach might be to have a very rough table, a rough index file, a fine index file, and a master file.

It is possible to work with a large unsorted master file by creating an index file containing two fields: the key field and the record number of the master file. Sort the index file by key. When a match is found on the key field of the index file, the program uses the record number field to read the proper record of the unsorted master file.

Creation of an index table or an index file can be done in a separate program. This separate program can save from several seconds to several minutes each time the program is run. The index file would only need to be changed when a master file is updated (perhaps on a weekly or monthly basis).

D.2.9 Record Count

To keep track of the number of records in a master file, reserve one field in the first record to contain the record count. The record count is the number of records in the file. When a record is added to this file, the record count in the first record is incremented by one and written out. This technique will work fine with a master file that is out of order.

D.3 DIRECT ACCESS NOTES

D.3.1 XMIT Statements (Extending a File)

XMIT statements can be interspersed with direct access operations on a file. An XMIT following a READ with record n is equivalent to a READ of record n+1. Successive XMIT's read records n+2, n+3, etc.

An XMIT following a WRITE of record n transmits data to record n+1.

Records n+2 to the end of the file may be changed by successive XMIT's after a WRITE. However, to change a series of records in the middle of the file, do not use a WRITE followed by several XMIT's.

The XMIT statement used after a WRITE statement has the following useful applications.

D.3.1.1 Truncating a File - To truncate a file after record N, use the following sequence:

```
          READ(channel,record,n)
          WRITE(channel,record,n)
          XMIT(channel,NULL,EOF)
          .
          .
     EOF, FINI(channel)
```

where NULL is a record with no contents defined by:

```
          RECORD NULL
          RECORD
```

D.3.1.2 Appending to a File - To append records to the end of a file with n records, use the following sequence:

```
          READ(channel,record,n)
          WRITE(channel,record,n)
          XMIT(channel,record)        ;Append records to file.
          XMIT(channel,record)
               .
               .
               .
          XMIT(channel,NULL,EOF)
     EOF, FINI(channel)
```

D.3.1.3 Rewriting A File - To rewrite a file from record n to the end of the file, use the following sequence:

```
        WRITE(channel,record,N)
        XMIT(channel,record)
        XMIT(channel,record)
             .
             .                          ;Rewrite records to end-of-file.
             .
        XMIT(channel,NULL,EOF)
   EOF, FINI(channel)
```

D.4 NUMERIC FIELD VERIFICATION

Any numeric field that is entered in a DIBOL program should be checked to determine if it contains only numeric data. The numeric field should be read as an alphanumeric field through an XMIT or ACCEPT statement. Then it is moved to a numeric field. This move is preceded by an ON ERROR statement to check for non-numeric data. For example:

```
        RECORD
TCHAR,  D2
DECMAL, D5
ALPHA,  A5
PROC
.
.
.
    ALPHA=
    ACCEPT(TCHAR,ALPHA)
    ON ERROR FIX
    DECMAL=ALPHA
```

With an alphanumeric-to-numeric move, many checks are done. The following examples illustrate most cases:

ALPHANUMERIC	NUMERIC
' 123'	00123
'123 '	00123
'00123'	00123
' -123'	0012S
' 123-'	0012S
'-123-'	00123
' 12-3'	0012S
'1-2-3'	00123
'1+2+3'	00123
'1+2-3'	0012S
'1 23 '	00123
'0012S'	illegal

The only legal characters in an alphanumeric-to-numeric move are 0 to 9, ␣, +, and -.

If a data file contains numeric fields, these fields must be read as numeric. If they contain a negative number, the least significant character contains a minus sign and is listed with its equivalent character. For example, -37 would look like 3W. If 3W were read as alphanumeric, and then converted to numeric, a run-time error would occur since any letter of the alphabet is illegal in an alphanumeric-to-numeric conversion.

D.5 CHAIN STATEMENT NOTES

D.5.1 Interaction of CHAIN and INIT (channel,SYS)

Source input files can be specified in a RUN command containing chained programs. Accessing such files must be done according to the following rules.

1. All CHAIN files must be listed in the RUN command before the source files.

   ```
   .RUN PRONAM+CHAIN1+CHAIN2,INP1,INP2
   ```

2. Any CHAIN statement which is to open the first source input file must first "skip over" the remaining chained files by issuing dummy INIT(channel,SYS) statements. In the above RUN command, in order to read file INP1, PRONAM would have to issue two dummy INIT(channel,SYS) statements. Because CHAIN2 is the last chain program, it would not have to issue any dummy statements.

If the RUN command were:

```
RUN pronam, filnam1...,filnam7
```

the source files could be processed more than once by executing a CHAIN 0 statement in pronam.

D.5.2 Transferring Variable Values

For the value of a variable to be successfully transmitted from one chained program to another, the variable in which the value appears must occupy the same location in both CHAIN programs. This may be accomplished by either of the two following methods.

1. Define all records which are to be passed between chained programs first, and make the definitions identical (except for variable names which may be different).

   Example:

```
Chain1                          Chain2

RECORD                          RECORD CPINFO
CUST, A30                       CUST, A30
PROD, D2                        PCODE, D2
RECORD INVENT                   RECORD INVENT
STOCK, D4                       QUANTY, D4
.                               .
.                               .
.                               .
```

2. Use the compiler storage maps listing for the two CHAIN programs to verify that the desired variables occupy the same storage location.

D.5.3 Multiple CHAIN Entry Points

Sometimes it is desirable to have several entry points into a CHAIN program. However, the CHAIN statement always starts execution of the chained program at the first statement following the PROC statement. Using the technique of transferring variable values between chained programs, multiple entry points can be programmed as indicated in the following example.

```
      Chain1                          Chain2

RECORD                          RECORD
WHERE,D2,01                     WHERE,D2
RETURN,D2                       RETURN,D2
.                               .
.                               .
.                               .
PROC                            PROC
GO TO(L1,L2,L3,L4),WHERE        GO TO(E1,E2,E3),WHERE
L1,RETURN=2                     .
.                               .
.                               .
.                               E1,...
CHAIN 2                         .
L2,...                          .
.                               .
.                               WHERE=RETURN
.                               CHAIN 1
```

D.6 DIBOL PROGRAMMING OF SOURCE FILES

D.6.1 Operating Procedures

Up to seven source files can be used in a DIBOL program. They are specified at run time by:

```
RUN pronam, filnaml...,filnam7
```

The edit buffer is not available to a DIBOL program.

D.6.2 Data Division

The RECORD description would be as follows:

```
RECORD recnam
LINENO, A2
CHAR, A120
```

LINENO contains a two-character line number in binary. Most programs ignore the line number. However, it can be converted to decimal by the statement:

```
varnam = #LINENO*64+#LINENO(2,2)
```

Varnam must be a four-digit field.

CHAR contains the characters of one line created by the editor. The DIBOL program may want to determine the number of characters in the record. This can be done by preceding the RECORD statement with the lines:

```
RECORD
TRICK, 2A1
```

and adding the following line in the Procedure Division:

```
varnam = (4096-64*#TRICK(3)-#TRICK(4)*2
```

Varnam must be a three-digit field.

There is no tabbing within CHAR. The tabbing seen by output from the Monitor command LIST or LIST/L is done by the operating system. Tabs are internally stored as characters with a decimal equivalent of 61. Any character may be checked for a tab by the statement:

```
IF(#CHAR(n,n).EQ.61) GO TO TAB
```

D.6.3 Procedure Division

The first source file specified in the RUN command is opened by the following statement:

```
INIT(channel,SYS)
```

Each record is accessed by the following statement:

```
XMIT(channel,record,eof label)
```

When an end-of-file condition occurs, the program transfers to the EOF label of the XMIT statement. At that EOF label, a FINI channel statement must be executed prior to an INIT(channel,SYS) to open a second source file. To handle a variable number of source files, precede the INIT(channel,SYS) statement by an ON ERROR statement. The program will transfer to the ON ERROR statement when an INIT(channel,SYS) statement is executed and there are no more source files.

The only way to process a source file more than once is to execute a CHAIN n statement which resets the operating system pointers to the source files.

This example combines up to seven source files into a single data file. The resulting data file can be converted to a source file using FILEX.

Example:

```
          RECORD
SIZE,     D3
          RECORD
TRICK,    2A1
          RECORD LINE          ;Line from source file.
,         A122
          PROC 2
          ON ERROR NOFILE
          INIT(1,SYS)          ;Open system file.
          INIT(2,0,'$FILE',1) ;Open output file on logical unit 1.
NEXT,     XMIT(1,LINE,EOF)     ;Read a line from source file.
          SIZE=(4096-64*#TRICK(3)-#TRICK(4))*2 ;Get size of line.
          XMIT(2,LINE(3,SIZE+2)) ;Output line without line number.
          GO TO NEXT
EOF,      FINI(1)              ;Close system file.
          ON ERROR DONE
          INIT(1,SYS)          ;Open next system file.
          GO TO NEXT
DONE,     FINI(2)              ;Close output file.
NOFILE,   STOP
```

D.7 CHECKDIGIT FORMULA

In most applications involving identification numbers, each number may be verified for accuracy by a checkdigit, a redundant digit added to the normal number. The checkdigit is determined by performing an arithmetic operation on the number in such a way that the usual errors encountered in transcribing a number are detected. The checkdigit is determined as follows:

Step 1 Start with a number....5764.

Step 2 Multiply the first digit and every other digit by 2 (left to right). 5 * 2 = 10 6 * 2 = 12

Step 3 Add the digits in the resulting numbers and the digits not multiplied. 1+0+7+1+2+4 =15

Step 4 Subtract the sum from Step 3 from the next higher number ending in zero. 20-15=5

Step 5 Add the checkdigit to the end of the original number. 57645 (This is the correct checkdigit if the number is entered in a D4 field.)

Note that a checkdigit procedure is not completely error proof. In the example given above, 5764 or 5673 give the same checkdigit. It is unlikely, however, that transpositions of this sort will occur. The checkdigit does not guard against the possible assignment of an incorrect but valid code, such as the assignment of a wrong valid identification code to a customer.

If the number entered for a checkdigit calculation is shorter than the field, the rightmost digit is used as the checkdigit and the remainder of the number is right-justified and padded with zeros on the left. The zeros are considered when the checkdigit formula is calculated.

D.8 VT50/VT52 ESCAPE SEQUENCES

A command protocol is built around the escape code (027) to implement those commands needed by the VT50/VT52 but not found in 7-bit ASCII. Upon receiving the escape code 027, the terminal is set to escape mode and treats the next character received as a command. Commands created in this manner are called Escape Sequences. The VT50/VT52 recognizes the following Escape Sequences:

Code	Character	Action Taken
27	ESC	The first 027 changes the mode, the second 027 changes it back.
65	A	Moves cursor up one line.
66	B	Moves cursor down one line.
67	C	Moves cursor right one position.
68	D	Moves cursor left one position.
72	H	Moves cursor to the home position.
74	J	Erases from cursor position to the end-of-screen.
75	K	Erases line from cursor to right margin.
90	Z	Requests the terminal to identify itself. The VT50 terminal will respond with 027 047 072; VT52 terminal will respond with 027 047 075. Other terminals will respond in different ways.

[1]Teletype is a registered trademark of the Teletype Corporation.

GLOSSARY

alphanumeric
A character set that contains letters, digits, and other characters such as punctuation marks. The COS-310 alphanumeric character set includes the uppercase letters A-Z, the digits 0-9, and most of the special characters on the terminal keyboard. Two of these characters, back slash (\) and back arrow (←) (shown on some terminals as an underscore), are illegal.

array
A DIBOL technique for specifying more than one field of the same length and type. The array 5D3 reserves space for five numeric fields, each to be three digits long. The array 2A10 describes two alphanumeric fields, each to be ten characters long.

ASCII
American Standard Code for Information Interchange. This is one method of coding alphanumeric characters.

batch file
A file containing a sequence of commands. A command to execute the file will cause the commands within the file to be executed sequentially.

batch processing
The technique of automatically executing a group of previously stored Monitor commands.

binary operator
An operator, such as + or -, which acts upon two or more constants or variables (e.g., A=B-C).

binary program
The kind of program which is output by the compiler.

binary scratch area
The area in memory where the binary program is stored during execution.

bit
A binary digit (0 or 1).

block
The basic COS-310 unit of mass storage capacity. A block consists of up to 512 characters.

bootstrap
A short routine loaded at system start-up time which enables the system software to be read into machine memory.

branch
A change in the sequence of execution of COS-310 program statements.

buffer
A temporary storage area usually used for input or output data transfers.

bug
An error or malfunction in a program or machine.

byte
A group of bits considered as a unit. A byte is the smallest unit of information that can be addressed in a DIBOL program.

channel
A number between 1 and 15 used to associate an input/output statement with a specified device.

character
A letter, digit, or other symbol used to control or to represent data.

character string
A connected linear sequence of characters.

clear
Setting an alphanumeric field to spaces or a numeric field to zeros.

command
An operator request for Monitor services; usually to be executed following a RETURN key.

comments
Notes for people to read; they are ignored by the compiler. Comments are optional and follow a semicolon on a statement line.

concatenated
Strung together without intervening space.

conversational program
A program that prompts responses from an operator and reacts depending upon the response from the operator.

cursor
The flashing light indicator which appears at the point on the screen where the next character will be displayed.

data
A representation of information in a manner suitable for communication, interpretation, or processing by either people or machines. In COS-310 systems, data is represented by characters.

data entry
The process of collecting and inputting data into the computer data files. Data entry is key to disk.

data management
The planning, development, and operation of a system like COS-310 by an organization to mechanize its information flow and make available the data needed by the organization.

debug
To detect, locate, and remove errors or malfunctions from a program or machine.

DEC
Acronym for Digital Equipment Corporation.

decimal
Refers to a base ten number.

delimiting
The bounds (beginning and end) of a series or string.

device designation
A three-character designation for a mass storage device. The first two characters designate the type of device; the third character designates the number of the drive on which the device is mounted.

device independence
COS-310 system design permits data files and programs to be stored on either diskettes or disks. At run time, the operator chooses the most suitable or most available input and output devices.

device designations
A three-character abbreviation used to name the COS-310 I/O devices.

TTY	= Screen
KBD	= Keyboard
LPT	= Printer
DK0-DK3	= Disk drives
RX0-RX3	= Disk drives
DY0-DY3	= Disk drives

DIBOL
Digital's Business Oriented Language is a COBOL-like language used to write business application programs. The source language of the COS-310 system.

direct access
The process of obtaining data from, or placing data into, a storage device where the availability of the data requested is independent of the location of the data most recently obtained or placed in storage. Direct access is available to users of COS-310 systems by writing the position number of any record in a data file. For example, you can request the 5th, 35th, and 711th records in a file.

directory
A place for listing information for reference. Displayed or printed with the DI command.

dump
To copy the contents of all or part of storage, usually from memory to external storage.

edit buffer
The work area in memory where source files are created and edited.

end-of-file mark
A control character which marks the physical end of a multivolume file. For both input and output files, the Monitor detects this EOF mark and types a message for the operator asking that the next volume in the file be mounted.

fatal error
An error which terminates program execution.

field
A specified area in a data record used for alphanumeric or numeric data; cannot exceed the specified character length.

file
A collection of records, treated as a logical unit.

fixed-length records
Each record in a data file is the same length. Fixed-length records are the only type handled by COS-310 utility programs and the only type on which direct access to data files is allowed.

flowchart
A pictorial technique for analysis and solution of data flow and data processing problems. Symbols represent operations, and connecting flowlines show the direction of data flow.

handlers
A specialized software function which interfaces between the system and peripheral devices.

illegal character
A character that is not valid according to the COS-310 design rules. DIBOL will not accept back slash (\) and back arrow (←) (back arrow is replaced on some terminals with underscore) in alphanumeric strings.

initialization
Putting a device into the correct format or position where it can successfully function in a configuration.

input
Data flowing into the computer.

input/output
Either input or output, or both. I/O.

jump
A departure from the normal sequence of executing instructions in a computer.

justify
The process of positioning data in a field whose size is larger than the data. In alphanumeric fields, the data is left-justified and any remaining positions are space-filled; in numeric fields the digits are right-justified and any remaining positions to the left are zero-filled.

key
One or more fields within a record used to match or sort a file. If a file is to be arranged by customer name, then the field that contains the customers' names is the key field. In a sort operation, the key fields of two records are compared and the records are resequenced when necessary.

load
To enter data or programs into main memory.

load-and-go
An operating technique in which there are no stops between the loading and executing phases of a program.

location
Any place where data may be stored.

logical unit number
A number (1-15) which identifies an entry in a logical unit table. The table references the number to a location on a mass storage device.

logical units
An area of storage on a mass storage device. Up to 15 logical units may be assigned at system start-up by the data file utility program (DFU). These areas and their assigned sizes are listed in the logical unit table printed by DFU.

loop
A sequence of instructions that is executed repeatedly until a terminal condition prevails. A commonly used programming technique in processing data records.

machine-level programming
Programming using a sequence of binary instructions in a form executable by the computer.

mass storage device
A device having large storage capacity.

master file
A data file that is either relatively permanent or that is treated as an authority in a particular job.

memory
The computer's primary internal storage.

merge
To combine records from two or more similarly ordered strings into another string that is arranged in the same order. The latter phases of a sort operation.

mnemonic
Brief identifiers which are easy to remember. Examples are KBD, LPT, and TTY.

mode
A designation used in INIT statements to indicate the purpose for which a file was opened or to indicate the input/output device being used.

modulo
A condition where the specified number exceeds the maximum condition in a variable. The maximum allowable number is then subtracted from the specified number and the remainder is used by the processor. In modulo 16, if 17 were specified (maximum is 15), 16 would be subtracted from 17 and the processor would use 1 as the variable.

Monitor
A COS-310 system program that loads and runs programs and performs other useful tasks.

nest
To embed subroutines, loops, or data in other subroutines or programs.

nonfatal error
An error which will not completely terminate program execution.

nonsystem device
A device that does not contain the operating system and the Monitor. A device used exclusively for data storage.

option switch
A one- or two-character designation indicating a special function in conjunction with a command. Usually preceded by a slash (/) in COS-310.

output
Data flowing out of the computer.

overlay
The technique of specifying several different record formats for the same data. Special rules apply.

parameter
A variable that is given a constant value for a specific purpose or process.

peripheral equipment
Data processing equipment which is distinct from the computer.

pushdown stack
A list of items where the last item entered becomes the first item in the list and where the relative position of the other items is pushed back one.

random access
Similar to direct access.

RECORD
A statement that reserves memory for DIBOL data language programs.

segment
Sixteen blocks of storage. A block is 512 bytes long.

sequential operation
Operations performed, one after the other.

serial access
The process of getting data from, or putting data into, storage where the access time is dependent upon the location of the data most recently obtained or placed in storage.

screen line number
The number which indicates the order of the horizontal lines on the screen.

sign
Indicates whether a number is negative or positive. Positive numbers do not require a sign, but negative numbers are prefixed with the minus sign (-).

significant digit
A digit that is needed or recognized for a specified purpose.

source program
A program written in COS-310 DIBOL language.

statement
An instruction in a source program.

string
A connected linear sequence of characters.

subscript
A designation which clarifies the particular parts (characters, values, records) within a larger grouping or array.

switch character
A single letter specified in a command following a slash (/).

syntax
The rules governing the structure of a language.

system configuration
The combination of hardware and software that make up a usable computer system.

system device
A mass storage device reserved for Monitor, Run-Time System, and other system and source programs.

systems directory
A list of programs on the systems device with lengths, dates of creation, and other useful information.

system handlers
The specialized software which interfaces between the system and peripheral devices.

terminal alarm
A signal emitted from the terminal.

unary operator
An operator, such as + or -, which acts upon only one variable or constant (e.g., A=-C).

utility program
A system program which performs common services and requires format programs. Examples are SORT and PRINT.

variable
A quantity that can assume any one of a set of values.

variable-length record
A file in which the data records are not uniform in length. Direct access to such records is not possible.

verify
To determine if a transcription of data has been accomplished accurately.

word
A string of 12 binary bits representing two COS-310 characters.

zero fill
To fill the remaining character positions in a numeric field with zeros.

A

/A, Sort, 9-4
ACCEPT,
  clear afield before, 1-4
  generalized subroutines, D-2
ACCEPT - Input/Output statement, 1-3, 1-4, 1-37
  subroutines, D-2
  used with DISPLAY, 1-4
ACCEPT and DISPLAY, interaction, D-1
Access to Data files, B-4
Add one to counter, 1-25
Addition (+), 1-11
Addition of lines, RESEQUENCE, 2-21
Advanced Programming Techniques, D-1
Afield,
  clear before ACCEPT, 1-4
  defined, 1-4
  stores keyboard entry, 1-4
Aid to program development, 7-1
Alarm,
  terminal, 1-18, 1-19
  terminal is sounded (PLEASE), 2-10
Algorithm for calculating segments of logical units, 4-7
Allocate space to binary scratch area, 8-1, 8-6
Alphanumeric,
  data, moving, 1-14
  definition, xv
  destination cleared to spaces, 1-13
  fields formatted to numeric fields, 1-16
  fields with embedded signs, 1-11
  label, 1-1
  literal, 1-9, 1-10
  string, 1-18
Alphanumeric values to numeric values, 1-11, 1-15
Angle brackets (CCP), 5-5
Appending a file, D-15
Arithmetic expressions, 1-11
Arithmetic expressions,
  addition, 1-13
  calculate, 1-9
  division, 1-13
  multiplication, 1-13
  rounding, 1-12
  subtraction, 1-13
Arithmetic operations, basic, 1-13
Arithmetic operators, binary, 1-11
Arrange records, 9-1
Array,
  dimensions specified, 1-9
  names without subscripts, 1-13
  subscripted, 1-9
Arrays, 1-32
Arrow key,
  down, D-1
  left, D-1
  right, D-1
  up, D-1
ASCII,
  files transferred in format, 10-1
  FILEX OS/8 input, 10-5
  FILEX OS/8 output, 10-7
Assignment of logical units, 4-1, 4-7
Assignments,
  logical unit on COS-310, 4-7
  table of logical unit, 4-2, 4-3, 4-5, 4-6
Automatic cursor positioning, D-6
Automatic line numbers, 2-19

B

BATCH,
  certain programs terminate, 2-5
  Monitor command, 2-5
  restart after error, 2-5
BATCH command file, 2-4, 2-5
Batch file, terminate with CTRL/C, 2-5
Batch file START (SYSGEN), 3-2, 3-4
Batch file START, space required, 3-2
Batch stream, not accept input from, 2-5
Batching commands, 2-1
Binary arithmetic operators, 1-11
Binary file, transfer, 8-3
Binary files, COS-310, xiv, B-2
Binary operators,
  addition (+), 1-11
  COS-310, 1-12
  division (/), 1-11
  multiplication (*), 1-11
  priority of execution, 1-11
  rounding (#), 1-11
  subtraction (-), 1-11

Binary program, 2-13
  compiler converts source program to, 5-1
  copied and stored on device, 2-13
  debug with DDT, 6-1
  erase with DELETE, 2-7
  RUN executes, 2-11
  size of, 5-6
  use SAVE to store, 5-4
Binary scratch area, 2-12
Binary scratch area, allocate space to, 8-6
Blank line, to obtain, 2-19
BOOT,
  error messages, 12-1
  operating procedures, 12-1
  program, 12-1
Bootstrap, BOOT is, 12-1
Bootstrap, Monitor loaded via, 2-4
Braces notation conventions, xv
Brackets, angle (CCP), 5-5
Brackets notation conventions, xv
Branch program control, 1-6, 1-23
Breakpoint, DDT, 6-1

C

,C option (clear record), 1-31
Calculate arithmetic expressions, 1-9
Calculating the segments for a logical unit, 4-7
CALL - Control statement, 1-3, 1-6
CALL to subroutine, 1-6
CALL traceback, subroutine, (DDT), 6-1
Caret points to error, 5-8
CCP (Conditional Compilation Procedure), 5-5
CCP sections nested to any depth, 5-6
CCP value independent of DIBOL value, 5-6
CHAIN and INIT, interaction, D-17
CHAIN - Control statement, 1-3, 1-7
CHAIN,
  control returns to DDT, 1-8
  DIBOL program, 1-7
  multiple entry points, D-18
  programs declared in RUN command, 1-7
  program example, 1-7
  TRACE and Trap turned off, 1-8
  with valid binary program, 1-8
Character Set, COS-310, xv, A-11
Characters,
  line number limitations, 2-20
  lowercase, xiv
  maximum in source program, 1-3
  maximum number on line, 2-18
  maximum per line, 1-3
  red, xiv
  special COS-310, xv
  special in formatting, 1-17
  terminating, 1-5
  uppercase, xiv
Channel (definition), 1-21
Channel in INIT, 1-26
Channel number associates mode, 1-26
Channel number disassociates mode, 1-21
Change system handlers, 3-1, 3-3
Change system date, 2-6
Changes to lines-per-page configuration, 13-1
Changes using line numbers, 2-14
Check, perform a Read/, 8-2, 8-7
Checkdigit formula, D-21
Clear data from line, 2-20
Clear fields and records, 1-9, 1-13, 1-19
Clear function, hardware display, D-2
Clear incorrect data, D-4
Clear text from edit buffer, 2-15
Clearing feature of VT52, D-3
Close data files, 1-21
Cmndfl (definition), xvi
  BATCH, 2-5
  DAFT, 15-1
  FILEX, 10-4
  FLOW, 17-1
  MENU, 18-1
  PATCH, 11-1
  PIP, 8-1
  PRINT, 16-2
  SORT, 9-1, 9-2
Code,
  COS-310 interpretive, xiv
  requirements, 5-7
  skip-code, 1-22
  words of required, 5-7
Command file,
  BATCH, 2-5
  DAFT, 15-2
  FILEX, 10-4
  FLOW, 17-1
  MENU, 18-1
  PIP, 8-1

PRINT, 16-2, 16-3
SORT, 9-1, 9-2
Commands, DDT, 6-2
Commands entered in response to Monitor, 2-4, 2-14
Commands, Monitor Keyboard, 2-2
Commands, orderly execution of (MENU), 18-1
Comments,
following semicolon, 1-1
on statement line, 1-1
with PROC, 1-29
with START, 1-34
COMP,
/D, 5-2
/G, 5-1
/N, 5-1
/O, 5-2
/T, 5-1
COMP (compiler),
accessed by RUN, 2-11
defined, xiii, 5-1
DIBOL, 5-1
operating procedures, 5-1
Comparison between expressions, 1-24
Comparison between relational expressions, 1-24
Compatibility with OS/8, 10-4
Compilation procedure, conditional (CCP), 5-5
Compilation, source program listing, 5-2
Compiler,
converts to binary program, 5-1
DIBOL, 5-1
error messages, 5-8
operating procedures, 5-1
statement, END, PROC, START, 1-3, 1-20, 1-29
statements, 1-3, 1-20
storage map listing, 5-3
Compiling procedure,
DAFT, 15-1
FLOW, 17-1
PRINT, 16-1
Computed GO TO, 1-23
Conditional compilation (CCP), 5-5
Consolidate space in directory, 8-2, 8-5
Consolidate files, 8-1
Control,
branches by CALL, 1-6
branches to RETURN, 1-6
branches with GO TO, 1-23
master program, 2-1
Control statements (DIBOL), 1-3
CALL, 1-6
CHAIN, 1-7
GO TO, 1-23
IF, 1-24
ON ERROR, 1-28
RETURN, 1-33
STOP, 1-35
TRAP, 1-37
Conventions,
braces, xv
brackets, xv
manual notation, xiv
Conversational program, 3-1
Conversion, data, 1-11
Conversions, justification of, 1-11
Convert to equivalent decimal code, 1-12
Converting data, 1-9
Converts and justify source data, 1-9
Copy and verify, 8-2, 8-7
Copy binary program onto device, 2-13
Copy device, 8-3
Copy Monitor and/or files, 3-3
COS-310,
arranges records in files, 9-1
binary files, B-2
binary operators, 1-12
character set, A-1
characters, xv
data files, B-1
data input, FILEX, 10-6
data output, FILEX, 10-8
file structure, xiv
files, B-1
interpretive code, xiv, 2-14
line number editor, 2-14
logical unit assignments, 4-7
records in, 10-1
source file output, FILEX, 10-8
source files, B-1
storage hierarchy, 4-7
system files, B-2
unary operators, 1-12
COS MONITOR, 2-4
Create system on new device, 3-1
Creation of report programs, 16-1
Creation of source file (PRINT), FILEX, 16-2
CREF (Cross Reference) program, 7-1
error messages, 7-2

operating procedures, 7-1
table, 7-1
CTRL/C
Monitor Keyboard command, 2-2,
to terminate batch file, 2-5
CTRL/O, 2-2, 2-17
CTRL/Q, 2-2, 2-17
CTRL/S, 2-2, 2-17
CTRL/U, 2-2, 2-19, 11-4, D-1
CTRL/Z, 2-2, 2-19, 6-3
Current date specification (,D), 1-32
Cursor positioning, 1-18, D-6

D

,D RECORD (data specification), 1-32
DAFT (Dump and Fix), 15-1
command file, 15-2
compiling procedure, 15-1
error messages, 15-7
keyword, 15-2
operating procedure, 15-1
output, 15-5
Data,
clear from line, 2-20
clear incorrect, D-4
copy and verify, 8-2, 8-7
Data conversion, 1-9, 1-11
Data definition statement (RECORD), 1-3, 1-31
Data division, 1-1, D-19
Data division, define destination area in, 1-9
Data entry programs, D-1
Data file output, COS-310 (FILEX), 10-8
Data file, transfer a, 8-2, 8-4
Data File Utility program (DFU), 4-1
Data files, xiv
access to, B-4
COS-310, B-1
COS-310 arranges records in, 9-1
replace, 2-7
transfer, 8-4
Data,
format, 1-9
formatting, 1-16
input, COS-310 (FILEX), 10-6
move between fields, 1-9
move from memory with WRITE, 1-39
moving alphanumeric, 1-14
moving numeric, 1-14
transfer with XMIT, 1-40
Data manipulation,
expressions, 1-10
literals, 1-10
statements (DIBOL), 1-3, 1-9
variable name, 1-9
variables, 1-9
Datasystem 308 or 310, xiii
DATE command, 2-4, 2-6
DATE, Monitor command, 2-6
Date, system stores, 2-6
Dates, valid, 2-8
DDT
(DIBOL Debugging Technique), 6-1
commands, 6-2
error messages, 6-3
in CHAIN, 1-8
operating procedures, 6-1
Debug binary programs, 6-1
Debug statements with CCP, 5-5
Debugging aids use numbers, 1-1
Debugging (DIBOL) statements 1-3
Debugging statements, TRACE/NO TRACE, 1-36
Debugging technique, DIBOL, 6-1
Debugging tools, 1-36
Decimal (definition), xv
Decimal value, stores in dfield, 1-4
Decimal value, terminating characters, 1-5
Decimal code, equivalent, 1-12
Default conditions, line number, 2-18
Default, DFU/B, 4-2
Default, lines-per-page (SYSGEN), 13-1
Default value (SYSGEN), 3-2
DELETE command, 2-2, 2-4, 2-7
Delete, FILEX, 10-11
DELETE key, 2-2, 2-19
Deletion of lines (RESEQUENCE), 2-21
Deletions with line numbers, 2-14
Destination,
area defined, 1-9
alphanumeric cleared to spaces, 1-13
defined in Data Division, 1-9
numeric cleared to zeros, 1-13
stores source data, 1-9
Determining logical unit size, 4-7
Dev (definition), xv

Development, aid to program, 7-1
Device,
  copy, 8-3
  create system on new, 3-1
  designation, xv
  order of logical units on, 4-8
  store binary program on, 2-3
  system format, B-2
Dfield (defined), 1-4
Dfield, stores decimal values, 1-4
DFU (Data File Utility), xiii, 4-1
DFU,
  error messages, 4-10
  logical unit assignments, 4-8
  operating procedures, 4-1
DIBOL compiler (COMP), 5-1
DIBOL debugging technique (DDT), 6-1
DIBOL (DIGITAL's Business Oriented Language), xiii
DIBOL,
  direct access, D-9
  kinds of statements in, 1-3
  language, 1-1
  programs in CHAIN, 1-7
  slowed by TRAP, 1-38
  table of symbols, 7-1
  programming of source files, D-19
  source program, 1-1
  statement, words of code required, 5-7
  statement, use terminating value, 1-4
DIBOL statements,
  compiler, 1-3
  control, 1-3
  data definition, 1-3
  data manipulation, 1-3
  debugging, 1-3
  Input/Output, 1-3
DIGITAL's Business Oriented Language (DIBOL), xiii
Dimensions of an array, 1-10
Direct address in DIBOL, D-9
Direct access,
  KEY field for, D-9
  access, reason for, D-9
  access, READ statement, 1-30
  access techniques, 8-5
  access, WRITE statement, 1-39
DIRECTORY command, 2-4, 2-8
Directory entry dates, valid, 2-8
DIRECTORY, Monitor command, 2-4
Directory space, consolidate, 8-5
Directory, system device, B-4
Disk, formatting an RK05 (DKFMT), 14-1
Disks, logical unit assignments on RK05, 4-9
Diskette,
  data mode, 10-6
  compatible with IBM 3741 format, 10-1
  formatting an RX02 (DYFMT), 14-2
  functions of sectors on universal, 10-1
  in universal format, 10-1
DISPLAY, ACCEPT used with, 1-4
Display clear feature, hardware, D-2
DISPLAY - Input/Output statement, 1-18
DISPLAY, interaction with ACCEPT, D-1
DISPLAY, numeric fields for special effects, 1-18
DISPLAY statement, 1-3, 1-8, 1-37
Division
  (/), 1-1, 1-13
  Data, 1-1, D-19
  Procedure, 1-1, D-20
  results of, 1-12
DKMFT (format RK05 disk), 14-1
Dollar, rounding to the, 1-12
Down arrow key, D-1
Dump and Fix Technique (DAFT), 15-1
Duplicate line numbers, 2-21
DYFMT (format RX02 diskette), 14-2

E

EBCDIC format, files transferred in, 10-1
Edit buffer,
  contents returned to memory, 2-12
  contents stored in editing scratch area, 2-12
  erase (clear) text from, 2-15
  lines edited in, 2-20
  list from, 2-17
  output to screen or printer, 2-17
  separated into files, 2-19
  source files loaded into, 2-16
Editing,
  features of the Monitor, 2-1
  functions refer to line numbers, 1-1

scratch area, 2-12
Editor commands,
ERASE, 2-14, 2-15
FETCH, 2-14, 2-16
LIST, 2-14, 2-17
Line Number, 2-14, 2-18
Number Commands, 2-14, 2-20
RESEQUENCE, 2-14, 2-21
WRITE, 2-14, 2-22
Editor, COS-310 line numbers, 2-14
Effective use of TRAP, 1-37
Eight-bit EBCDIC, 10-4
End of subroutine, RETURN, 1-33
END, same effect as STOP, 1-35
Erase edit buffer, load source file, 2-16
Eliminate free space, 8-5
End-of-file, 1-28, 1-40, 2-25
END compiler statement, 1-3, 1-20
Equivalent decimal code, 1-12
ERASE command, 2-14, 2-15
Erase program, DELETE, 2-7
Erase text from edit buffer, 2-15
Error, caret points to, 5-8
Error checking, minimal by CREF, 7-1
Error correction (PATCH,) 11-3
Error during automatic line numbering, 2-19
Error, line with underscored, 5-2
Error messages, C-1
Error messages, Appendix C
BOOT, 12-1
COMP, 5-8
CREF, 7-2
DAFT, 15-7
DDT, 6-3
DFU, 4-10
FILEX, 10-13
FLOW, 17-7
LINCHG, 13-2
MENU, 18-4
Monitor, 2-23
PATCH, 11-5
PIP, 8-9
PRINT, 16-8
Run-Time, 2-24
SORT, 9-7
SYSGEN, 3-5
Error terminates BATCH, 2-5
Errors, fatal, 2-24
Errors, trappable (nonfatal), 2-24
Escape code sequences as terminators, D-8
Escape sequences, VT50/VT52, D-22
Examination of variables with SORT, 6-1
Exit, FILEX, 10-12
Expressions,
arithmetic, 1-11
calculate arithmetic, 1-9
data manipulation, 1-10
(definitions), xvi, 1-10
parentheses in, 1-12
relational comparisons, 1-24
Extending a file, D-15

F

Fatal error, 2-24
Fatal error, DIBOL program under DDT, 6-3
FETCH command, 2-14, 2-16
Field descriptor statement, SORT, 9-1
Field, numeric verification, D-16
Field, part accessed by subscripting, 1-13
Field statement information, 1-31
Fields, clear, 1-9
Fields, clearing, 1-13
File,
appending, D-15
batch START (SYSGEN), 3-4
batch START, space required, 3-2
exchange program (FILEX), 10-1
extending a, D-15
index technique, use of, D-12
name extension, OS/8 compatibility, 10-4
output, COS-310 data (FILEX), 10-8
output, COS-310 source (FILEX), 10-8
replace an old, 2-13
rewriting a, D-16
source loaded into edit buffer, 2-16
source stored on specified device, 2-22
START batch, to execute, 3-2
status information destroyed (CHAIN), 1-8
structure, xiv
transfer a binary, 8-2, 8-3
transfer a data, 8-2, 8-3
transfer a source, 8-2, 8-3
transfer a system, 8-2, 8-3
truncating, D-15
File name, garbled, 2-3

File name extensions, 10-4
Files,
  binary, xiv, B-2
  consolidate, 8-1
  COS-310, B-1
  data, xiv, B-4
  source, xiv, B-1
  system, xiv, B-2
Files,
  DIBOL programming of source, D-19
  maximum number in program, 2-12
  multiples passed as one file, 2-12
  skipped (erased), 8-5
  sorted, D-11
  edit buffer separated into two, 2-19
  source per program, 1-3
  transferred in ASCII format, 10-1
  transferred in EBCDIC format, 10-1
  transferred in IMAGE format, 10-1
  unsorted, D-10
FILEX (File Exchange program), 10-1
  command file, 10-4
  creation of source file (PRINT), 16-2
  error messages, 10-13
  Input Mode, 10-5
  operating procedures, 10-4
  Option C flowchart, 10-10
FILEX options,
  C, Copy, 10-4, 10-5
  D, Delete, 10-4, 10-11
  L, List, 10-4, 10-11
  X, exit, 10-4, 10-12
  Z, Zero (clear), 10-4, 10-12
FILEX Output Modes, 10-7
Filnam, xvi
FINI statement, 1-3, 1-21, 1-37
Fix technique, Dump and, (DAFT), 15-1
Fixed-length records, 9-1
FLOW (Flowchart Generator),
  command file, 17-1
  commands, 17-2
  compiling procedures, 17-1
  error messages, 17-7
  example of, 17-7
Flowchart Generator program (FLOW), 17-1
Format data, 1-9
Format, diskettes compatible with IBM 3741, 10-1
Format for rounding, 1-12
Format printer output, 1-22
Format system device, B-2
Formats,
  files transferred in, 10-1
  programs, 14-1
Formatting data, 1-16
  RK05 disks (DKFMT), 14-1
  RX02 diskettes (DYFMT), 14-2
  numeric fields to alphanumeric fields, 1-16
  special characters, 1-17
Forms hardware, printers without, 3-4
FORMS statement, 1-3, 1-22

G

Garbled file name, 2-3
Generalized ACCEPT subroutines, D-2
GO TO - Control statement, 1-3, 1-23

H

Handler address, B-5
Handlers,
  change in system, 3-3
  contained in Monitor, 2-1
Hardware display clear feature, D-2
Hardware, printers without forms (SYSGEN), 3-4
Hierarchy, COS-310 storage, 4-7

I

IBM 3741 format, diskettes compatible with, 10-1
IF - Control statement, 1-3, 1-24
IF, to make best use of TRACE, 1-36
IMAGE format, files transferred, 10-1
Incorrect data, clear, D-4
Increment, rounding, 1-12
INCR (increment) statement, 1-3, 1-25
Index, error messages, C-1
Index file technique, D-12
INIT - Input/Output statement, 1-3, 1-26, 1-37

INIT, interaction of CHAIN and, D-17
INIT, logical unit No. in, 1-27
Initial values for statements, 1-1, 1-32
Initialization, 14-1
Input,
  COS-310 data (FILEX), 10-6
  line limitations, 2-20
  maximum characters on line, 2-18
  OS/8 ASCII (FILEX), 10-5
  universal diskette (FILEX), 10-6
Input/Output statements (DIBOL),
  ACCEPT, 1-4
  DISPLAY, 1-18
  INIT, 1-26
  READ, 1-30
  WRITE, 1-39
  XMIT, 1-40
I/O handlers, 2-1
  I/O statements, 1-3
Input/Output division (SORT), 9-2
Insertions with line numbers, 2-14
Interaction of ACCEPT and DISPLAY, D-1
Interaction of CHAIN and INIT, D-17
Internal subroutine, 1-6
  symbol table, 5-8
Iteration, DDT, 6-1

J

Justification of numeric fields, right, D-6
Justified conversions, 1-11
Justify source data, 1-9

K

KEY field for direct access, D-10
Key, last typed, 1-5
Keyboard commands, Monitor, 2-2
Keyboard input, stores in afield, 1-4
Keyword, DAFT, 15-2
KREF, FLOW, 17-2
KRFSRT, FLOW, 17-2

L

Label (definition), xvi, 1-1
Labels,
  maximum allowed in 16K byte system, 5-4
  maximum allowed in 24K byte system, 5-4
  referenced in statements, 1-1
  separated from statements, 1-1
  table of, used in DIBOL program, 7-1
Language,
  DIBOL, xviii, 1-1
Last key typed, 1-5
Left arrow key, D-1
Limitations,
  input line, 2-20
  source program, 1-3
LINCHG (Line Change program), 11-1
Line, characters per, 1-3
LINCHG,
  error messages, 13-2
  operating procedures, 13-1
Line Change program (LINCHG), 13-1
Line Number (LN) command, 2-14, 2-18
Line Number Editor, 2-13
Line Number Editor commands, 2-18
Line number exceeds 4095, 2-21
Line number default conditions, 2-18
Line numbers automatically output, 2-18
Line numbers,
  changes with, 2-14
  deletions with, 2-14
  insertions with, 2-14
Lines-per-page configuration, change, 13-1
List programs for review, 2-8
LIST command, 2-14, 2-17, 10-11
Listing, source program compilation, 5-2
Literals
  alphanumeric, 1-10
  data manipulation, 1-10
  (defined), 1-10, 5-6
  numeric, 1-10
  RECORD, 1-10
LN (Line Number) command, 2-14, 2-18
Logical units, xvi
Logical unit assignments, 4-1
  from the edit buffer, 4-2
  from a stored file, 4-2
  from the keyboard, 4-3
  displayed on screen, 4-3
  listed on printer, 4-4
  on RK05 disks, 4-9, 4-10

Logical unit,
  assigned by DFU, 4-8
  calculating segments for, 4-7
  defined, xvi
  pushdown, 4-8
  size, 4-7
  table, maximum entries, 4-2
Logical unit size, 4-7
Logical units,
  maximum open, 1-29
  order on a nonsystem device, 4-8
  order on a system device, 4-8
Lowercase characters, xiv

M

Machine language instructions, 11-1
Mapping bad sectors, 10-1
Master control program, 2-1
Master file, records in, D-14
Memory, contents of edit buffer returned to, 2-12
Memory, move data with WRITE, 1-39
Memory, record moved to, 1-30
Memory requirements because of DDT option (/D), 6-1
Memory saved by COMP/O, 5-2
MENU,
  command file, 18-1
  error messages, 18-4
  operating procedures, 18-1
  program, 18-1
  program file, 18-1
Merge pass, combine volumes, 9-1, 9-4
Messages, MOUNT, 2-3
Messages, show on the screen, 1-18
Mode,
  associates, 1-26
  designations, 1-26
  disassociate, 1-21
  input (FILEX), 10-4
  output (FILEX), 10-9
Modulo 16, 1-26
Monitor, 2-1
  commands, 2-4
  copy files and (SYSGEN), 3-3
  error messages, 2-23
  keyboard commands, 2-21
  organization, B-3
Monitor commands,
  BATCH,2-4, 2-5
  DATE, 2-4, 2-6
  DIRECTORY, 2-4, 2-8
  PLEASE, 2-4, 2-10
  RUN, 2-4, 2-11
  SAVE, 2-4, 2-13
Monitor commands, sequential execution of, 2-5
Monitor, copy (SYSGEN), 3-3
Monitor dot, commands in response to, 2-14
Monitor, editing features of, 2-1
MONITOR/EDITOR Programs, xiii
Monitor Keyboard commands,
  CTRL/C, 2-2
  CTRL/O, 2-2
  CTRL/Q, 2-2
  CTRL/S, 2-2
  CTRL/U, 2-2
  CTRL/Z, 2-2
  DELETE, 2-2
  RETURN, 2-2
Monitor loaded via bootstrap, 2-4
Monitor, ON ERROR prevents return to, 1-28
Monitor error messages, 2-23
Monitor operating procedures, 2-4
Monitor organization, B-3
Monitor, to patch, 11-1
Monitor, return to, 8-2, 8-9
MOUNT messages, 2-3
Move data between fields, 1-9
Moving,
  alphanumeric data, 1-14
  numeric data, 1-14
  records, 1-15
Multiple CHAIN entry points, D-18
Multiple definition of fields, 1-31
Multiple files passed as one file, 2-12
Multiplication (*), 1-11, 1-13
Multivolume universal interchange files, 10-1

N

Name (defined), data manipulation, 1-9
Negative numbers, characters representing, A-1
Nested, CCP sections to any depth, 5-6
Nested, subroutine CALLS, 1-6
Nested to depth of 50, 1-6
No index file, rough table, D-12
NO TRACE statement, 1-3, 1-34
Nonsystem device, order of logical units, 4-8

Notation conventions,
braces, xv
brackets, xiv
manual, xiv
RETURN, xv
symbols, xiv
Number commands, 2-14, 2-20
Number, editing functions refer to, 1-1
Numbered line begins statement, 1-1
Numbers, debugging aids refer to, 1-1
Numbers, error messages refer to, 1-1
Numbers, negative, A-1
Numeric,
data, moving, 1-14
data verification, 1-11
(definition), xvi
destinations cleared to zeros, 1-13
fields, formatting to alphanumeric, 1-16
fields, special effect in DISPLAY, 1-18
field verification, D-16
fields, right justification, D-6
literals, 1-10
values converted to alphanumeric, 1-11
variable, add one to, 1-25
variable, CHAIN, 1-7

O

Octal (definition), xv
Old file, replace, 2-13, 2-22
ON ERROR - Control statement, 1-3, 1-28
ON ERROR, preceding a data conversion statement, 1-11
ON ERROR presents return to Monitor, 1-28
Operating procedures,
BOOT, 12-1
COMP, 5-1
CREF, 7-1
DAFT 15-1
DDT, 6-1
DFU, 4-1
DKFMT, 14-1
DYFMT, 14-2
FILEX, 10-4
FLOW, 17-1
LINCHG, 13-1
MENU, 18-1
Merge, 9-4
Monitor, 2-4
PATCH, 11-1
PIP, 8-1
PRINT, 16-2
SORT, 9-1
SYSGEN, 3-1
Operations, basic arithmetic, 1-13
Operator interaction, BATCH may require, 2-5
Operators,
binary arithmetic, 1-11
binary and unary, 1-12
binary, priority of execution, 1-11
Order of program execution, 1-36
Orderly execution of commands (MENU), 18-1
Organization of the Monitor, B-3
OS/8,
ASCII Input (FILEX), 10-5
ASCII Output (FILEX), 10-7
compatibility, 10-4
file name extensions for compatibility, 10-4
files on RK05 disk, 10-1
Output,
COS-310 source file (FILEX), 10-8
DAFT, 15-5
Modes (FILEX), 10-7
OS/8 ASCII (FILEX), 10-7
Universal diskette (FILEX), 10-9
Overlay record, 1-31

P

,P (RECORD) (information insertion), 1-32
Parentheses in arithmetic expressions, 1-12
PATCH,
cmndfl, 11-1
error correction, 11-3
error messages, 11-5
operating procedures, 11-1
program, 11-1
restart, 11-3
Perform a Read/Check, 8-2, 8-7
PIP (Peripheral Interchange Program),
accessed by RUN, 2-11
error messages, 8-9

operating procedures, 8-1
PIP options,
B, 8-2, 8-3
C, 8-2, 8-3
D, 8-2, 8-4
E, 8-2, 8-5, 8-6
I, 8-2, 8-7
R, 8-2, 8-7
S, 8-2, 8-8
V, 8-2, 8-8
X, 8-2, 8-9
PLEASE command, 2-4, 2-10
PRINT (Report program generator), 16-1
compiling procedure, 16-1
error messages, 16-8
Print logical unit table, 4-1
Printer,
contents of edit buffer output to, 2-17
limitations because of TRAP, 1-38
on-line, 2-8
slower than processor, 1-37
without forms hardware, 3-2
without forms hardware (SYSGEN), 3-4
Priority of execution, binary operators, 1-11
PROC, comment with, 1-29
PROC statement, 1-3, 1-29
Procedure, conditional compilation (CCP), 5-5
Procedure Division, 1-1, D-20
Program,
binary copied and stored, 2-13
binary executed by RUN, 2-11
CHAIN, example of, 1-7
control, branches with GO TO, 1-23
development aid, 7-1
DIBOL binary sequence, 1-7
DIBOL CHAIN, 1-7
DIBOL slowed by TRAP, 1-38
erase with DELETE, 2-7
examination, DDT, 6-3
execution, Monitor, 2-1
execution, order of, 1-36
execution terminated with STOP, 1-35
last statement in, 1-20
master control, 2-1
maximum number of files in, 2-12
readability, 2-19
renumber lines within, 2-21
size of binary, 5-6
source (DIBOL), 1-1
source files per, 1-3
source compilation listing, 5-2
source limitations, 1-3
system executed by RUN, 2-11
table of labels used in DIBOL, 7-1
tracing, 1-36
Programs,
ACCEPT and DISPLAY in, D-1
binary debugged with DDT, 6-1
certain terminate BATCH, 2-5
directory of stored, 2-8
list for review, 2-8
segment sections, 1-34
type of, 2-7
Programming, DIBOL source files, D-19
Programming techniques, advanced, D-1
Pronam (definition), xvi
Pushdown, order of logical units, 4-8
Pushdown stack, 1-6

R

READ - Input/Output statement, 1-3, 1-30
READ, restrictions on use, 1-39
Read/Check, perform a, 8-2, 8-7
Readability of program, 2-19
Record,
COS-310 file (defined), 10-1
count, D-14
Descriptor Division (SORT), 9-2
moved to memory, 1-30
names in data manipulation, 1-9
names, subscripted, 1-40
names subscripted in an array, 1-13
overlaying, 1-31
size dependent on logical units, 4-7
RECORD - Data definition statement, 1-3, 1-31, 5-6
literals, 1-10
Records,
arrange COS-310 data, 9-1
clear, 1-9
clearing, 1-13
master file, D-14
moving, 1-15
subscripted, use with care, 1-40

Red characters, xiv
Relational comparisons between expressions, 1-24
Renumbering program lines, 2-14, 2-21
Repetition count character, 1-32
Replace an old file, 2-13
Replace an old source file, 2-22
Report programs, creation of, 16-1
Report program generator (PRINT), 16-1
RESEQUENCE command, 2-14, 2-21
Restart (PATCH), 11-4
Restrictions on READ and WRITE, 1-39
Return after LN, 2-19
RETURN at end of subroutine, 1-33
RETURN key, xv, 2-2, 2-19
RETURN - Control statement, 1-3, 1-33
RETURN, control branches to, 1-6
Return to Monitor, PIP, 8-2, 8-9
RETURN without CALL or TRAP, 1-33
Rewriting a file, D-16
Right arrow key, D-1
Right justification of numeric fields, D-6
RK05 disk, format (DKFMT), 14-1
RK05 disk, logical unit assignments on, 4-10
Rough table, no index file, D-12
Rounding of numbers (#), 1-12
RUN command,
  CHAIN declared in, 1-7
  Monitor, 2-4, 2-11
Run-Time error messages, 2-24
RX02 diskette, format (DYFMT), 4-2

S

,S (RECORD) (assign value of variable), 1-32
SAVE command to store binary program, 2-4, 2-13, 5-4
Scratch area (binary) modification, 2-12, 8-6
Screen,
  contents of edit buffer output to, 2-17
  line number, 1-18
  move cursor on, 1-18
  when full, 2-18
Search for records, 15-1
Sectors on universal diskette, 10-1, 12-1
Segment programs, 1-34
Segments, calculating for logical unit, 4-7
Semicolon, comments after, 1-1
Sequential execution of Monitor commands, 2-5
Seven-bit ASCII, 10-4
Signs embedded in alphanumeric fields, 1-11
Size,
  binary program, 5-6
  defined in RECORD statement, 1-1
  determining logical unit, 4-7
Six-bit binary word, A-1
Skip-code (definition), 1-22
Skip (erase) files, 8-5
SORT program, xiv, 9-1
  accessed by RUN, 2-11
  command file, 9-1, 9-5
  error messages, 9-7
  key, 9-3
  operating procedures, 9-1
Sorted file, D-11
SORTIN, 9-3
Source (defined) data manipulation, 1-9
Source area stored in destination, 1-9
Source area converted and justified, 1-9
Source files, xiv, D-19
  COS-310, B-1
  COS-310 output (FILEX), 10-8
  DIBOL programming of, D-19
  creation of, 16-2
  load into edit buffer, 2-16, 2-16
  per program, 1-3
  replace old, 2-22
  stored on device, 2-22
  separate edit buffer into two, 2-19
  transfer, 8-2, 8-8
Source program,
  compilation listing, 5-2
  compiler converts to binary, 5-1
  debug with CCP, 5-5
  DIBOL, 1-1
  erase with DELETE, 2-7
  limitations, 1-3
  maximum characters in, 1-3
Space,
  allocate to binary scratch area, 8-6
  consolidate directory, 8-2, 8-5

Special characters, xv, 1-17
Special characters in formatting, 1-17
Special codes in DISPLAY, 1-18
Special DISPLAY codes, 1-18
START - Compiler statement, 1-3, 1-34
START system on new device, 12-1
START with comment, 1-34
START,
  batch file, 2-4, 3-2
  batch file space required, 3-3
  batch file (SYSGEN), 3-4
  with comment, 1-34
Statement,
  CHAIN encountered, 1-7
  comments in, 1-1
  define size of, 1-1
  define type of, 1-1
  last in program is END, 1-20
  lines with comments, 1-1
  separated from label, 1-1
Statements,
  Data Division, 1-1
  data manipulation, 1-9
  initial values for, 1-1
  reference labels, 1-1
  six kinds in DIBOL, 1-3
Statements, DIBOL
  ACCEPT, 1-3
  CALL, 1-6
  CHAIN, 1-6
  DISPLAY, 1-18
  END, 1-20
  FINI, 1-21
  FORMS, 1-22
  GO TO, 1-23
  IF, 1-24
  ON ERROR, 1-28
  PROC, 1-29
  READ, 1-30
  RECORD, 1-31
  START, 1-34
  STOP, 1-3, 1-35
  TRACE/NO TRACE, 1-36
  TRAP, 1-37
  WRITE, 1-39
  XMIT, 1-40
STOP - Control statement, 1-3, 1-35
STOP,
  SAME EFFECT AS END, 1-35
  terminates program execution, 1-35
Storage hierarchy, 4-7
Storage map, listing, 5-1, 5-3
Store binary program on device, 2-13
Store system date, 2-6
Subroutine, ACCEPT, D-2
Subroutine, branches control to, 1-6
Subroutine,
  CALL statements, 1-6
  call traceback, 6-1
  internal, 1-6
  nested in, 1-6
  TRAP, 1-37
Subroutines, generalized ACCEPT, D-1
Subscripted array in data manipulation, 1-9
Subscripted record names, 1-13, 1-40
Subscripting to access parts of fields, 1-13
Subtraction (-), 1-11, 1-13
Symbol table, internal, 5-8
Symbols,
  DAFT command, 15-2
  maximum in system, 1-3
  notation convention, xiv
SYSGEN (System Generation PROGRAM), XIII, 3-1
SYSGEN default, 3-2, 13-1
SYSGEN/B, 3-1
SYSGEN/C, 3-3
System,
  boot to start on new device, 12-1
  change handlers in, 3-3
  COS-310 logical units on, 4-7
  create on new device, 3-1
  date change, 2-6
  date stored by DATE, 2-7
  device directory, B-4
  device holds BATCH cmndfl, 2-5
  device, order of logical units, 4-8
  encounters ACCEPT, 1-4
  device format, B-2
  files, xiv, B-2
  files, COS-310, B-2
  files, transfer, 8-2, 8-8
  generation program (SYSGEN), 3-1
  I/O handlers in Monitor, 2-1
  maximum symbols per, 1-3
  program, erase with, 2-7
  program, transfer, 8-8
  programs accessed by RUN, 2-11
  programs executed by RUN, 2-11

T

TAB key produces 8 spaces, 2-19
Tab settings, 1-1
Table, labels used in DIBOL program, 7-1
Table, logical unit assignments, 4-2, 4-6
Table lookup, direct access, D-10, D-11
Tabs used for readability, 2-19
Terminal alarm is sounded (PLEASE), 2-10
Terminal BATCH, certain programs, 2-5
Terminate program execution with STOP, 1-35
Terminating character, 1-4, 1-5
Terminating value, used by DIBOL, 1-4
Terminators, escape sequences, D-8
Text, erase (clear) edit buffer, 2-15
Text, lines of assigned a line number, 2-14
308, Datasystem, xiii
310, Datasystem, xiii
Top-of-page command, 1-34
TRACE - Debugging statement, 1-3
TRACE,
  indiscriminate placement of, 1-36
  turned off in CHAIN, 8-3
  use of IF to isolate, 1-36
Trace/No Trace statements, 1-36
Traceback, subroutine call (DDT), 6-1
Transfer,
  binary file, 8-2, 8-3
  data (XMIT), 1-40
  data files, 8-4
  data records, 1-40
  files, ASCII format, 10-1
  files, SYSGEN, 3-3
  files, universal format, 10-1
  source files, 8-2, 8-8
  system files, 8-8
Transferring control through IF statement, 1-24
Transferring variable values, D-7
TRAP - Control statement, 1-3, 1-37
TRAP,
  information for effective use of, 1-37
  limitations on printers, 1-38
  normally precedes FORMS or XMIT, 1-37
  slows DIBOL programs, 1-38
  subroutine construct, 1-37
  turned off in CHAIN, 1-8
Trappable error, 2-24
Truncating a file, D-15
Two-line PLEASE command, 2-10
Type, define in RECORD statement, 1-1

U

Unary operators in COS-310, 1-12
Unconditional GO TO, 1-23
Underscores line number with errors, 5-2
Universal diskette,
  (definition), 10-1
  format for, 10-1
  functions of sectors, 10-1
  input (FILEX), 10-4, 10-6
  interchange format directory, 10-2
  output (FILEX), 10-9
Unsorted file, D-10
Up arrow key, D-1
Uppercase characters, xiv

V

Valid directory entry dates, 2-8
Value,
  default (SYSGEN), 3-2
  initial (definition), 1-32
  terminating used by DIBOL, 1-4
Values,
  CCP different than DIBOL, 5-6
  transferring variable, D-17
Variable,
  add one to numeric, 1-25
  examination in DDT, 6-1
  name, 1-9
  numeric in CHAIN, 1-7
  values, transferring, D-17
Variables (defined), 1-9, 5-6
Verfication, numeric field, D-16
Verify data, copy and, 8-2, 8-7
Verify numeric data, 1-11
VT50/VT52 Escape Sequences, D-22
VT52 clearing feature, D-3

W

Word boundry, 5-6
Word count number, B-1
Words of code requirement, 5-7
WRITE - editor command, 2-14, 2-22
WRITE - Input/Output statement,
    1-3, 1-37, 1-39
WRITE,
  move data with, 1-39
  restrictions on use, 1-39

X

XMIT - Input/Output statement,
    1-3, 1-37, 1-40
XMIT statement, extending a file
    with, D-15

Z

Zero, divison by, 1-28, 2-26

READER'S COMMENTS

NOTE: This form is for document comments only. DIGITAL will use comments submitted on this form at the company's discretion. Problems with software should be reported on a Software Performance Report (SPR) form. If you require a written reply and are eligible to receive one under SPR service, submit your comments on an SPR form.

Did you find errors in this manual? If so, specify by page.

Did you find this manual understandable, usable, and well-organized? Please make suggestions for improvement.

Is there sufficient documentation on associated system programs required for use of the software described in this manual? If not, what material is missing and where should it be placed?

Please indicate the type of user/reader that you most nearly represent.

- ☐ Assembly language programmer
- ☐ Higher-level language programmer
- ☐ Occasional programmer (experienced)
- ☐ User with little programming experience
- ☐ Student programmer
- ☐ Non-programmer interested in computer concepts and capabilities

Name _____ Date _____

Organization _____

Street _____

City _____ State _____ Zip Code or Country _____

Please cut along this line.

---------- Fold Here ----------

---------- Do Not Tear - Fold Here and Staple ----------

FIRST CLASS
PERMIT NO. 33
MAYNARD, MASS.

BUSINESS REPLY MAIL
NO POSTAGE STAMP NECESSARY IF MAILED IN THE UNITED STATES

Postage will be paid by:

digital

Business Products
Software Development Group
MK1-2/H32
Continental Blvd.
Merrimack, New Hampshire 03054