decdatasystem

COS310

new user's guide

digital

COS-310
New User's Guide

Order No. AA-D758A-TA

November 1978

This is an introductory manual to the operating procedures, DIBOL language, logical units, and programming conventions associated with the COS-310 Operating System. This information will allow a new user to be operating with COS-310 in a minimal amount of time.

Utility programs are herein introduced, but their actual operating instructions are contained in the COS-310 System Reference Manual.

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, November 1978

The information in this document is subject to change without notice and should not be construed as a commitment by Digital Equipment Corporation. Digital Equipment Corporation assumes no responsibility for any errors that may appear in this document.

The software described in this document is furnished under a license and may only be used or copied in accordance with the terms of such license.

No responsibility is assumed for the use or reliability of software on equipment that is not supplied by DIGITAL or its affiliated companies.

Copyright © 1978 by Digital Equipment Corporation

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			v
		ORGANIZATION OF THE MANUAL NOTATIONAL CONVENTIONS FOR THIS MANUAL	
CHAPTER	1	INTRODUCTION TO COS-310	1-1
	1.1	OVERVIEW	1-1
	1.2	HARDWARE REQUIREMENTS	1-1
	1.3	FUNCTIONAL VIEW OF COS-310 PROGRAMS	1-2
	1.3.1	The System Monitor	1-2
	1.3.2	A High-level Programming Language	1-2
	1.3.3	System Utility Programs	1-2
CHAPTER	2	OPERATING COS-310	2-1
	2.1	USING THE KEYBOARD	2-1
	2.2	ERROR CORRECTION	2-3
	2.3	ERROR MESSAGES	2-3
	2.4	CALLING THE KEYBOARD MONITOR	2-3
	2.5	USING MONITOR AND EDITOR COMMANDS	2-4
	2.5.1	DIRECTORY, FETCH, and LIST Commands	2-4
	2.5.2	RUN, SAVE, and WRITE Commands	2-5
	2.5.3	ERASE Command	2-7
	2.5.4	Line Number Command	2-7
	2.5.5	Number and RESEQUENCE Commands	2-8
	2.5.6	DELETE Command	2-9
CHAPTER	3	LOGICAL UNITS	3-1
	3.1	LOGICAL UNIT TABLE	3-1
	3.2	LOGICAL UNIT NUMBERS	3-2
	3.3	HOW LOGICAL UNITS ARE ASSIGNED	3-2
	3.3.1	Assignments Through the Keyboard (DFU/K)	3-2
	3.3.2	Assignments From the Edit Buffer (DFU/B)	3-3
	3.3.3	Assignments From a Named File (DFU,filnam)	3-3
	3.4	HOW LOGICAL UNITS ARE DISPLAYED AND LISTED	3-4
	3.4.1	Display Assignments on the Screen (DFU/D)	3-4
	3.4.2	List Assignments on the Printer (DFU/DL)	3-4
	3.4.3	Display an Expanded Table on the Screen (DFU/E)	3-4
	3.4.4	List an Expanded Table on the Printer (DFU/EL)	3-4
	3.5	ARRANGEMENT OF LOGICAL UNITS ON MEDIA	3-5
	3.6	HOW A DIBOL PROGRAM USES LOGICAL UNIT NUMBERS	3-6

			Page
CHAPTER	4	THE DIBOL LANGUAGE	4-1
	4.1	DATA DIVISION	4-1
	4.1.1	RECORD and Field Labels	4-2
	4.1.2	Field Types - A or D	4-3
	4.1.3	Initial Values	4-3
	4.2	PROCEDURE DIVISION	4-4
	4.2.1	Data Manipulation Statements	4-4
	4.2.1.1	Moving Alphanumeric Data	4-5
	4.2.1.2	Moving Numeric Data	4-6
	4.2.1.3	Moving Records	4-6
	4.2.1.4	Calculating Arithmetic Expressions	4-7
	4.2.1.5	Data Conversion	4-9
	4.2.1.6	Data Formatting	4-11
	4.2.1.7	Clearing Fields and Records	4-12
	4.2.1.8	Using Literals to Implement Data	4-13
	4.2.1.9	Incrementing Data	4-14
	4.2.2	Input/Output Statements	4-14
	4.2.2.1	DISPLAY - An Input/Output Statement	4-14
	4.2.2.2	XMIT - An Input/Output Statement	4-15
	4.2.2.3	INIT and FINI - Input/Output Statements	4-16
	4.2.2.4	READ and WRITE - Input/Output Statements	4-18
	4.2.2.5	ACCEPT - An Input/Output Statement	4-19
	4.2.2.6	FORMS - An Input/Output Statement	4-19
	4.2.3	Program Control Statements	4-20
	4.2.3.1	IF - A Program Control Statement	4-20
	4.2.3.2	STOP - A Program Control Statement	4-21
	4.2.3.3	GO TO - A Program Control Statement	4-21
	4.2.3.4	CALL and RETURN - Program Control Statements	4-21
	4.2.3.5	ON ERROR - A Program Control Statement	4-22
	4.2.3.6	CHAIN - A Program Control Statement	4-23
	4.2.3.7	TRAP and RETURN - Program Control Statements	4-24
	4.2.4	Debugging Statements	4-25
	4.2.4.1	TRACE - A Debugging Statement	4-25
	4.2.4.2	NO TRACE - A Debugging Statement	4-26
APPENDIX	A		A-1
GLOSSARY			Glossary-1
INDEX			Index-1

FIGURES

3-1	Arrangement of Logical Units on Devices	3-5
3-2	Flowchart of INIT Operation	3-7

PREFACE

This manual introduces you to the operating procedures, DIBOL language, logical units, and programming conventions associated with the COS-310 Operating System. Actual installation procedures related to the hardware and mass storage media configurations are contained in the COS-310 Release Notes and Installation Guide (AA-D759A-TA).

This manual does not attempt to teach fundamental programming or computer concepts. It is written for people who fall into one or more of the following categories:

- Experienced programmers unfamiliar with DIBOL or COS-310
- New programmers of DIGITAL's equipment configurations
- Inexperienced-but-interested people seeking to understand the operation of COS-310

The COS-310 New User's Guide is a supplement to the COS-310 System Reference Manual (AA-D647A-TC).

ORGANIZATION OF THE MANUAL

This manual is written to meet the needs of a diversified audience. You may, therefore, find familiar information; don't feel obligated to read through such material. Three of the four chapters contain examples and/or programs to guide you in practicing the skills associated with chapter information. Information on particular or advanced applications is found in the COS-310 System Reference Manual.

CHAPTER 1 provides a general introduction to COS-310, a listing of the hardware associated with the COS-310 system, and a functional view of the programs within COS-310.

CHAPTER 2 describes the procedures for operating a COS-310 system. Emphasis is given to the Monitor and editor commands.

CHAPTER 3 explains the use of logical units in the COS-310 Operating System. The logical unit within a mass storage device, the logical unit table, and logical unit numbers are explained according to manner of assignment, use, and arrangement.

CHAPTER 4 introduces and explains the DIBOL language. Executable examples illustrate many DIBOL statements.

NOTATIONAL CONVENTIONS FOR THIS MANUAL

Press the RETURN key following information you input through the keyboard. No special RETURN symbol is used in this text. It is assumed that you will press the RETURN key at the end of each statement line, after a response to a program display, and at the end of each command.

To minimize the probability of confusion with respect to the format and content of example dialog between you and COS-310, characters which you must input are printed in red.

Enter uppercase alphabetic characters within program statements exactly as shown. Lowercase characters in a program statement are symbolic representations of specific names, numbers, or characters which are required in a particular application.

Although COS-310 will not recognize lowercase characters, the comment fields in the example programs in this manual are printed in upper and lower case characters to make reading easier.

Monitor and editor commands are spelled out rather than indicated by their two-character designations. However, when you type these commands, you only need to use the first two letters. The exceptions to this first-two-letter convention are the RUN command, the Line Number command, and the Number command. RUN only requires R; Line Number requires LN; the Number command requires the statement line number.

The following symbolic representations are used:

Symbol	Meaning
␣	Insert one character space.
{ }	Make a choice between the items contained within the braces.
[ ]	Decide whether to use the optional items contained within the brackets.
dev	Use the three-character designation of the mass storage media. The first two characters designate the type of

Symbol	Meaning
	media. The third character indicates the drive on which the mass storage media is operating. DK indicates an RK05 disk. RX indicates an RX01 diskette. DY indicates an RX02 diskette.
cmndfl	Command file name.
pronam	Program name.
filnam	Unique name assigned to a data file.
channel	Numeric expression associating a number to a logical unit or to a character-oriented input/output device.
label	Reference label assigned to a statement in a DIBOL program.

Other symbols are explained at the time of their use.

The following terms are of particular importance in this text.

Term	Meaning
Device media	are also called mass storage media. These are the diskettes or disks on which the operating system and/or data are stored.
Device type	refers to the hardware drive which contains the mechanisms to read or write information onto or from a mass storage media.
Drive number	identifies the order in which the device types are located in the cabinets.
Device	refers to the media when it is loaded into a device type.

Other terms are explained in the Glossary.

CHAPTER 1

INTRODUCTION TO COS-310

1.1 OVERVIEW

COS-310 is a disk-resident operating system that operates on the Datasystem 308 (D308), DECstation 78, Datasystem 310 (D310), or DECstation 88 hardware configuration. COS-310 is an applications development tool for data processing users who do such typical business applications as order entry, inventory control, back-order processing, sales and profit analysis, and accounting.

1.2 HARDWARE REQUIREMENTS

Minimum hardware required: One of the following (with a minimum of 16K bytes of memory).

- Datasystem 310 (D310).
- Datasystem 308 (D308).
- DECstation 78/50, 78/70, 88/70, or 88/80. (The RL01 on the DECstation 88/80 is not supported by COS-310.)

Optional hardware with D308 or DECstation 78:

- Up to 4 RX01 floppy disk drives, or up to 4 RX02 floppy disk drives. (RX01 and RX02 drives are not supported on the same system.)
- One LA8, LQP, or LA120 printer.

Optional hardware with D310 or DECstation 88:

- Additional memory up to a system total of 64K bytes.
- Up to 4 RX01 floppy disk drives, or up to 4 RX02 floppy disk drives (RX01 and RX02 drives are not supported on the same system.)

- Up to 4 RK05 cartridge disk drives (D310 only).
- One LA35, LA36RO, LQP, LA8, LA8A, LA120 or LP05 printer.

1.3 FUNCTIONAL VIEW OF COS-310

COS-310 provides the Operating System needed for applications development and execution on the D308, DECstation 78, D310, and the DECstation 88 hardware configurations. The COS-310 Operating System includes a system Monitor, a high-level programming language (DIBOL), and system utility programs.

1.3.1 The System Monitor

Software operation is controlled through the system Monitor. The Monitor is divided into two parts: one residing in memory (a portion of the hardware) and the other residing on the operating system media. The Monitor controls program execution, maintains file directories, stores all of the I/O handlers necessary for the system, and controls the source text editor.

The source text editor uses line numbers as a reference for inserting, deleting, and changing program information. The editor can sequence and resequence line numbers. Input to the editor comes through the keyboard. Output from the editor can be displayed on the screen, listed on the printer, or written to a mass storage device.

1.3.2 A High-level Programming Language

COS-310 uses DIGITAL's Business Oriented Language (DIBOL) as its programming language. DIBOL consists of statements which are divided between the Data Division and the Procedure Division in a program. These statements are much like English verbs and are mnemonic in nature. Each of these statements is introduced in Chapter 4 of this manual and treated in detail of Chapter 1 of COS-310 System Reference Manual.

1.3.3 System Utility Programs

COS-310 includes various programs written to perform specialized functions in the overall operation of COS-310. These are introduced below and are described in the COS-310 System Reference Manual.

A utility program called SYSGEN lets you copy the operating system onto another mass storage device for installation start-up and backup, and lets you change the I/O handlers to adapt to a variety of disk and printer configurations. You perform these operations using conversational statements prompted by the SYSGEN program.

The COS-310 Operating System depends upon the proper use of logical units (See Chapter 3 of this manual). A Data File Utility program (DFU) lets you make and examine logical unit assignments. Logical unit assignments can be input to DFU from the keyboard, from a file stored on the system device, or from a table in the edit buffer. DFU lets you display current logical unit assignments on the screen or output them to the printer in either of two formats.

You can create a DIBOL program that will generate a report by using the COS-310 utility program PRINT. You first create a command file to describe the report and then run the command file through PRINT.

A flowchart generator program (FLOW) lets you generate a flowchart which illustrates the sequence of program logic.

The Peripheral Interchange Program (PIP) transfers files from one device to another, replaces an existing file with a new file, and combines data files. PIP accepts input from the keyboard or from the disk and produces output on the screen, the disk, or the printer.

COS-310's multiphase SORT lets you reorder a data file containing fixed-length records into a specified sequence. SORT will independently sort each volume of a multivolume file and then merge the volumes within the file.

The File Conversion program (FILEX) can copy a COS-310 file onto a flexible diskette in a format directly readable by the IBM 3740. IBM files on flexible diskettes can also be converted to COS-310 format. FILEX allows various file transfers not usually possible in functions of COS-310.

A MENU program lets you select and execute commands from a previously created command file. The MENU program can potentially eliminate many operator errors.

DIBOL Debugging Technique (DDT) aids in program debugging. A dump-and-fix technique (DAFT) lets you search for, examine, list, and change records. DAFT is also used to make minor changes to a data file. A cross-referencing program (CREF) provides an alphabetical listing of all labels used in a DIBOL program, the line number where each label is defined, and the line numbers where each label is used.

CHAPTER 2

OPERATING COS-310

This chapter provides the basic information needed to operate the COS-310 system. Emphasis is given to the use of the keyboard and Monitor commands during program creation and editing.

This basic information is presented in a logically applicable order.

Because COS-310 Operating System software is applicable to various hardware configurations, starting instructions for particular hardware and loading instructions for various devices are not contained here. The COS-310 Release Notes and Installation Guide is the best written source for this information.

2.1 USING THE KEYBOARD

The keyboard allows you to enter data and interact with the computer. Most of the keyboard is identical to a typewriter keyboard. Unlike on a typewriter, pressing a key on the keyboard does not automatically display a character. Pressing a key causes the selected character to be sent to the Central Processing Unit (CPU). A program will then display this character on the screen. Consequently, pressing a key will have no effect if the CPU is not running or is running a program that is not waiting for input from the keyboard.

The CAP LOCK key must be locked for alphabetic characters to be used by COS-310. If it is not locked, only numbers will be accepted.

Adjacent to the main keyboard may be a smaller numeric key panel. COS-310 recognizes these keys as another set of numeric keys.

The following keys have special functions when used with COS-310. All other keys operate as if on a normal typewriter keyboard. The shift register must still be pressed to use the top character on any double-character key.

- RETURN key

Pressing the RETURN key indicates the end of COS-310 Monitor and editor command lines, of program statements, and of responses to program inquiries.

- DELETE key

Pressing the DELETE key erases typing errors made while entering Monitor and editor commands. The DELETE key erases the last character or space typed and thus allows you to make character-by-character corrections. It can be used on a line prior to pressing the RETURN key. If you find an error on a line after you have pressed the RETURN key, make corrections with the Monitor Number commands.

- CTRL key

The CTRL (control) key is held down while another key is pressed. These combinations are represented in this manual by using a slash (/) between CTRL and the designation for the other key. The combinations and their effect are shown below.

CTRL/C Terminates execution of the currently running program and returns control to the Monitor.

CTRL/O Stops the display of characters on the screen. Characters sent to the screen after an initial CTRL/O are discarded. Another CTRL/O or a CTRL/C will stop this loss of characters.

CTRL/Q Resumes output suspended by CTRL/S.

CTRL/S Suspends output to the screen but neither loses characters nor terminates the program or command. No input is possible while a CTRL/S is in control. Output is resumed by CTRL/Q.

CTRL/U Deletes an entire line if pressed before RETURN.

CTRL/Z Indicates the end of input and terminates automatic line numbers. Deletes the contents of a line if used before pressing the RETURN key.

The preceding combinations require neither a RETURN key nor any other terminator key; the system performs the function as soon as the combination is typed.

- BACKSPACE, COPY, BREAK, ESC, LINE FEED keys

These keys and their characters are not part of the COS-310 character set. They produce unpredictable results.

2.2 ERROR CORRECTION

Use the DELETE key or CRTL/U to correct errors on a line before you press the RETURN key. The DELETE key erases individual characters; the CTRL/U combination erases an entire line. If you find an error or want to make a change after you press RETURN, use the Number command.

2.3 ERROR MESSAGES

COS-310 is programmed to display error messages for certain kinds of incorrect information. These messages are referenced in Appendix C of the COS-310 System Reference Manual.

2.4 CALLING THE KEYBOARD MONITOR

Once the diskettes and/or disks are mounted and the system is turned on and booted (all these instructions are in the COS-310 Release Notes and Installation Guide), the Monitor asks for a date by displaying:

```
COS MONITOR V 8.00
DATE?
.
```

Enter the date in the form:

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

where:

dd	is the number representing the day of the month
mmm	are the first three letters of the name of the month
yy	are the last two digits of the year designation

If you type anything before the date, the Monitor displays:

```
DATE?
```

If you enter the date in the wrong form, the Monitor displays:

```
BAD DATE
```

You must enter a date whenever the Monitor is booted or whenever you change the system date.

The Monitor indicates that it is ready to accept further commands by displaying the COS MONITOR message.

2.5 USING MONITOR AND EDITOR COMMANDS

Enter all Monitor and editor commands immediately following the dot (.) displayed by the system. An error message is displayed if a character or a blank space occurs between the dot and the command.

The cursor will flash after the last character or space on a line and will wait for you to either enter more information, press CTRL/Z, or press the RETURN key. Pressing CTRL/Z will erase whatever else is on the same line.

2.5.1 DIRECTORY, FETCH, and LIST Commands

The three commands (DI, FE, and LI) are used to display or print the information contained on a system device.

Type DI/T to display the directory of all files on the system device. A directory similar to the following will appear on the screen.

```
•DI/T

DIRECTORY        03-AUG-78

NAME    TYPE LN    DATE

COMP      V  14  18-JUL-78
PIP       V  10  19-JUL-78
MENU      V  05  19-JUL-78
SYSGEN    V  19  19-JUL-78
PATCH     V  05  19-JUL-78
CREF      V  07  19-JUL-78
BOOT      V  02  19-JUL-78
SORT      V  15  19-JUL-78
LINCHG    V  02  19-JUL-78
FILEX     V  23  19-JUL-78
DKFMT     V  02  19-JUL-78
DYFMT     V  02  19-JUL-78
DFU       V  07  19-JUL-78
DAFTA     S  12  19-JUL-78
DAFTB     S  15  19-JUL-78
PRINT0    S  16  19-JUL-78
PRINT1    S  15  19-JUL-78
PRINT2    S  04  19-JUL-78
PRINT3    S  12  19-JUL-78
PRINT4    S  05  19-JUL-78
PRINT5    S  15  19-JUL-78
PRINT6    S  09  19-JUL-78
PRINT7    S  13  19-JUL-78
PRINT8    S  06  19-JUL-78
PRINT9    S  09  19-JUL-78
FLOW1     S  11  19-JUL-78
FLOW2     S  06  19-JUL-78
FLOW3     S  10  19-JUL-78
FLOW4     S  11  19-JUL-78
KRFSRT    S  01  19-JUL-78
KREF      S  06  19-JUL-78
TRMTST    S  05  19-JUL-78
LPTEST    S  06  19-JUL-78
FLOPXX    S  07  19-JUL-78
 <0173 FREE BLOCKS>
```

Type DI to list the directory on the printer rather than to display it on the screen.

The directory gives the name, the type, the length in blocks, and the creation date of each file.

There are three types of files: binary (B), source (S), and system (V). The binary and the system files cannot be edited or altered by the usual editing commands; source files can be called into the edit buffer and edited. Do not edit or alter programs which come with your initial system unless you are advised to do so in an official notification (a patch) from DIGITAL. Unauthorized changes to you software will void your software warranty.

Type FE and the file name to clear the edit buffer (a work area in memory) and copy a source program from the directory into the edit buffer. An LI command displays the contents of the edit buffer.

```
.FE KREF
.LI
```

Use CTRL/S and CTRL/Q respectively to start and stop output to the screen. Use CTRL/C to stop the output to the printer and return control to the Monitor.

2.5.2 RUN, SAVE, and WRITE Commands

The RUN (R) command is used to execute binary (B) or system (V) programs. Other commands and option switches are used with the RUN command to compile and execute a DIBOL source program.

The commands SAVE and WRITE are used to store programs on a mass storage device. SAVE (SA) stores a binary program and WRITE (WR) stores a source program. Example exercise and explanation:

The following exercise illustrates the use of the RUN, SAVE, and WRITE commands. Initialize the system, enter the date, and step through this exercise to become familiar with the order and interrelationship of commands.

In this exercise, you will make a duplicate of a distributed source program, compile the source program, rename the program to prevent alteration of the source, and work with the duplicate.

Use the FETCH (FE), LIST (LI) combination to display the source program TRMTST (Terminal Test); this is a program distributed with the COS-310 software.

```
.FE TRMTST
.LI
```

Before a source program can be executed, it must be compiled (a special program that converts a high-level programming language into an executable binary program). It is good programming practice to include the name of the file whenever you use that file in a Monitor command. If the source file is in the edit buffer, the file name is optional. Compile the source program by typing:

```
.RUN COMP,TRMTST
```

After a slight delay while COS-310 loads the compiler program, the printer will output a two-part compilation listing (Data and Procedure Divisions) and a storage-map listing. To suppress the printing of these listings or if you are without a printer, use the /N option immediately after the RUN COMP,TRMTST command (leave no space).

```
·RUN COMP,TRMTST/N
```

When the source program has been compiled into a binary program and is temporarily stored in the binary scratch area (a work area in memory), the COS MONITOR message will appear.

To store this newly compiled program as a binary file on the system device, type the following command (LEARN becomes its new name):

```
·SA LEARN
```

The program can now be executed with the RUN command. The file name is needed following this RUN command because the program is being executed from the system device rather than from the binary scratch area. It is good programming practice to include the name of the binary file whenever you use that file in a Monitor command. If the binary file is in the binary scratch area, the file name is optional.

```
·RUN LEARN
```

Questions from the TRMTST program (you are calling it LEARN) should appear on the screen. The first question is as follows:

```
DO YOU WISH TO ENTER PARAMETERS?
```

If something else happens, either type CTRL/C and return to the Monitor or boot the system and start over by entering the date.

Because TRMTST is a program written to test the terminal, feel free to answer the displayed questions in a number of ways. YES (Y) and NO (N) responses as well as other characters and numbers may be used. Use your imagination, but remember what happens with each response.

Use CTRL/S to temporarily halt output and CTRL/Q to resume output. CTRL/O will stop output but data will be lost. After a CTRL/O command, only another CTRL/O or a CTRL/C will get things going again.

The TRMTST program (renamed LEARN in this excercise) is written in a closed loop so it continues to execute until you either stop it with CTRL/C or you turn the system off. CTRL/C returns control to the Monitor; turning off the system requires a complete power up, boot, and correct date entry before the COS MONITOR will appear.

If you use the SAVE command to copy LEARN back into the directory, REPLACE? will be displayed on the screen. The directory already contains a binary file named LEARN so the system wants to know what to do about an attempted duplication.

Type N (NO) at this time. A Y (Yes) will replace the file.

```
·SA LEARN
REPLACE?
N
```

The WRITE command followed by LEARN will store the newly named program without any questions because no source (S) file named LEARN is in the directory. Names must be used with the SAVE or WRITE command, or the Monitor will display ERROR IN COMMAND. A name not found in the directory will initiate the message FILE NOT FOUND.

```
·WR LEARN
```

Type DI/T or DI to see this new file name (LEARN) in the directory.

```
·DI/T
```

After a source file is copied from the edit buffer onto a storage device, the file remains in the edit buffer until an ERASE (ER) command erases it or a FETCH (FE) command replaces it.

A binary file will remain in the binary scratch area until another program is compiled. If a program is too big to fit in the binary scratch area, see Chapter 8 of the COS-310 System Reference Manual.

2.5.3 ERASE Command

The ERASE (ER) command clears the edit buffer. Erasing makes it impossible to retrieve erased information from the edit buffer. Always store (WRITE) information that you want to keep before using ERASE.

If the buffer is not erased, the old contents of the buffer will interfere with new information entered through the keyboard.

2.5.4 Line Number Command

Use line numbers to write a program with COS-310. These numbers are necessary for editing and are used for error location and cross-referencing in conjunction with system programs.

Line numbers are entered manually or with a Line Number (LN) command.

During program development, enter either a number or the Line Number (LN) command whenever the system displays a dot (.). The highest line number that the system will accept is 4095. Since the edit buffer is not large enough to hold 4095 lines, line numbers are incremented.

You can designate the starting number and the increment number in the LN command. If you don't designate values, the numbering begins with 100 and is incremented by 10.

Leave one space after the LN and enter your starting number. Enter your increment value after a comma which follows your starting number. If you want the increment to be the same as the starting number, only enter the starting number.

Automatic line numbers will continue until you press CTRL/Z to indicate that data input has stopped. Press RETURN before CTRL/Z or a line of input will be lost. After CTRL/Z, the system displays a dot (.). If you press LN after a CTRL/Z, the incremented numbers continue as they were before you pressed CTRL/Z. Use LN and new parameters if you want to change increment values.

```
.LN         ;Line numbers begin at 100 and increment by 10, or
            ;numbers continue to increment according to parameters
            ;established before the CTRL/Z.

.LN 50      ;Line numbers begin at 50 and increment by 50.

.LN 50,     ;Line numbers begin at 50 and increment by 10.

.LN 50,15   ;Line numbers begin at 50 and increment by 15.
```

2.5.5 Number and RESEQUENCE Commands

Use the Number command to edit source programs.

Only entire lines can be edited with the Number command. Automatic line numbering must be stopped with a CTRL/Z before editing with the Number command can be done. Type the line number and the entire line just as you want it; if you are replacing a line, the old line will be completely deleted. Use CTRL/U to erase an entire line before you press RETURN. Use the DELETE key to erase individual characters before you press RETURN. A line number followed by RETURN will delete the contents of the line.

Type LI to display the edited program on the screen.

Use the RESEQUENCE (RE) command to make increments consistent. This RESEQUENCE standardizes the increment between line numbers. The RESEQUENCE command uses the same numbering procedure as the Line Number command. In place of LN, you type RE.

```
.RE         ;Line numbers begin at 100 and increment by 10.

.RE 20      ;Line numbers begin at 20 and increment by 20.

.RE 20,     ;Line numbers begin at 20 and increment by 10.

.RE 20,5    ;Line numbers begin at 20 and increment by 5.
```

Be careful that the highest line number does not exceed 4095. Type the LI command to display the results of the RE command.

2.5.6 DELETE Command

The DELETE (DE) command is used to erase programs from the directory. Once a program is deleted, it is gone; the information is wiped out. Great caution should be exercised when using this command.

Because you created, named, and stored LEARN, you can delete it without altering your operating system. Do not delete any of the programs distributed with your COS-310 Operating System software. When using DELETE, you must stipulate both the file name and the type of file (preceded by /) that you are deleting. The type of file is listed in the directory.

```
•DI/T
•DE LEARN/B
•DI/T
```

A review of the directory before and after the DELETE command will assure you that the proper file has been erased. Once the file is erased it cannot be referenced or accessed; it is gone.

CHAPTER 3

LOGICAL UNITS

A logical unit is a data file storage area located on a mass storage device. Logical units are sequentially ordered on a mass storage device. The DIBOL language uses a logical unit number to reference a logical unit. A logical unit table maintained on the system device links a logical unit number to a logical unit.

The logical unit number identifies an entry in the logical unit table. The entry contains information which points to a specific data file storage area (logical unit). One logical unit table can access storage areas contained on many different devices.

This accessing of data through the logical unit table enhances storage flexibility because the same data file storage areas can be referenced by different programs in different ways. Logical units may be reassigned (given different numbers) without changing the contents of the data file, and logical unit assignments can be made or remade at each system start-up or between programs being executed.

3.1 LOGICAL UNIT TABLE

The logical unit table maintained on the system device is a centralized index for logical unit assignments. This table contains and displays a list of unit numbers, device designations for devices being used, and the length in segments of each logical unit (1 segment equals 16 blocks which equals 8192 bytes). A logical unit table display is similar to the following example.

```
UNIT DEV. SEGS.
  1  RX0  0001
  2  RX0  0001
  3  RX0  0001
  4  RX0  0001
  5  RX0  0001
  6  -UNDEFINED-
  7  -UNDEFINED-
  8  -UNDEFINED-
     .
     .
     .
 15  -UNDEFINED-
```

The logical unit table also contains but does not display the starting address of the first block and the address of the handler.

The logical unit table only knows where a data file storage area begins and how many segments are reserved in that area. The data file's name, volume number, and creation date are stored on the first block of the data file.

3.2 LOGICAL UNIT NUMBERS

Unlike data file directory information which is referenced and accessed by file name, information on a logical unit is referenced by logical unit number and accessed through the logical unit table. A maximum of 15 logical unit numbers can be assigned in a table.

3.3 HOW LOGICAL UNITS ARE ASSIGNED

Assigning logical units and associating them with numbers in the logical unit table is done with the Data File Utility (DFU) program.

DFU is an interactive program which allows you to designate the device on which you want the data to be stored and the number of segments you want reserved for the data. The DFU program goes to the device that you designate and determines where on that device your data storage area will be located. The starting address of the first segment in the storage area and its length in segments is then recorded in the logical unit table. The starting address of the first segment is not displayed by any of the options of DFU.

Option switches (/K, /B, filnam) used with DFU allow you to make logical unit assignments in different ways.

3.3.1 Assignments Through the Keyboard (DFU/K)

DFU/K allows you to make or to change logical unit assignments by inputting information through the keyboard. The following exercise makes logical unit assignments with the /K option.

```
·RUN DFU/K
DFU V8.00
1=RX0,10
2=RX1,10
3=RX0,15
  .
  .
  .
10=END
```

3.3.2 Assignments From the Edit Buffer (DFU/B)

DFU/B allows you to create a logical unit table in the edit buffer and then automatically make the logical unit assignments and copy the table onto the system device. The following exercise creates a table in the edit buffer and then makes logical unit assignments with the /B option. The .ER command clears the edit buffer to prevent mixing of old and new entries. It is good practice to use END to indicate your last entry to the logical unit table.

```
·ER
·0100 RX0,2
·0200 RX0,1
·0300 RX1,1
·0400 RX1,1
   .
   .
   .
·0900 END
·RUN DFU/B
DFU V8.00
COS MONITOR V 8.00
·
```

3.3.3 Assignments From a Named File (DFU,filnam)

Another way to assign logical units is to create and name a file containing the input for DFU. This named file can then be used at any time by using the DFU command followed by the file name. The following exercise makes logical units assignments with the filnam option. The .ER command clears the edit buffer to prevent mixing of old and new entries.

```
·ER
·0100 RX0,1
·0110 RX1,1
·0120 RX2,15
·0130 RX0,05
   .
   .
   .
·0200 END
·WR TABLE1
·RUN DFU,TABLE1
DFU V 8.00
COS MONITOR V 8.00
·
```

3.4 HOW LOGICAL UNIT ASSIGNMENTS ARE DISPLAYED AND LISTED

The option switches (/D, /DL, /E, /EL) used with DFU allow you to display or list logical unit tables in different ways.

The /D and /DL options output a table containing the logical unit number, the device on which the unit resides, and the number of segments reserved in each unit.

The /E and /EL options output an expanded table containing the logical unit number, the device on which the unit resides, the number of segments reserved in each unit, the name of the file contained on the unit, the sequence number of the file, the date the file was created, and the actual number of segments in the file.

When you use the /E or /EL options, you must have all devices which contain logical unit assignments mounted on the system. If a device is not mounted, an error message is displayed.

3.4.1 Display Assignments on the Screen (DFU/D)

To display a table of current logical assignments on the screen, type:

```
.RUN DFU/D
```

3.4.2 List Assignments on the Printer (DFU/DL)

To list a table of current logical assignments on the printer, type:

```
.RUN DFU/DL
```

3.4.3 Display an Expanded Table on the Screen (DFU/E)

To display a list of data files and the logical units on which the files are assigned, type:

```
.RUN DFU/E
```

3.4.4 List an Expanded Table on the Printer (DFU/EL)

To print a list of data files and the logical units on which the files are assigned, type:

```
.RUN DFU/EL
```

3.5 ARRANGEMENT OF LOGICAL UNITS ON MEDIA

Logical units are data file storage areas sequentially ordered on mass storage media. The method of storage differs between devices containing only data and devices containing the operating system. Sequential order on data media is from the beginning of the media moving in incremental order toward the end of the media. The first logical unit on the data media is at the beginning and data storage units are sequentially numbered through the remainder of the storage media.

Sequential order on the operating system media is different from the sequential order on data media. Rather than beginning at the front of the media, the logical units are input from the end of the media. Numbers are assigned in a push-down sequence. The first storage unit created is temporarily placed at the end of the media. When a second unit is created, the first unit is pushed toward the front of the media and the second unit remains at the end. This can continue until the disk area is full or the maximum number of units has been designated. The first unit always goes nearer to the front as more units are added at the end. This arrangement on the system device is to allow the unused space to be between the programs and the data files. Any new programs can then be added in the unused space. This is illustrated in Figure 3-1.

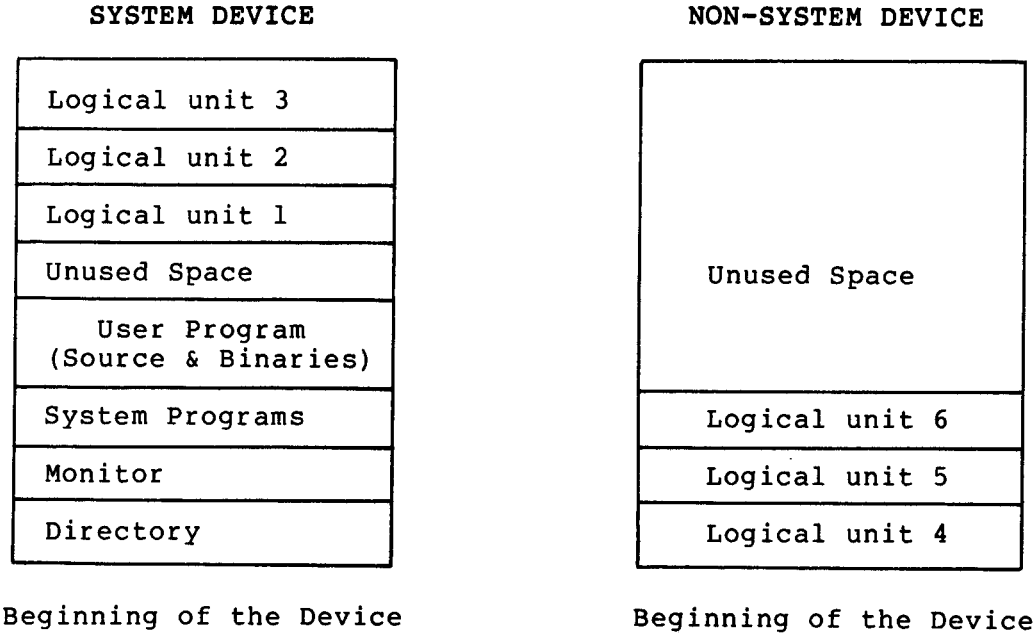


Figure 3-1 Arrangement of Logical Units on Devices

3.6 HOW A DIBOL PROGRAM USES LOGICAL UNIT NUMBERS

The following explanation illustrates how an INIT statement uses a logical unit number to identify a location in the logical unit table. The table location contains information which points to the first block of an assigned logical unit on mass storage device. The first block of data on a logical unit is reserved for the file name, volume number, and creation date. A flowchart of an INIT operation is shown in Figure 3-2.

If a logical unit number is specified at compilation time, when the INIT statement is executed, the program uses the logical unit number to access the logical unit table. The information stored in the table finds the device where the logical unit (the storage area) is located and reads the file name and volume number from the first block on the logical unit. The file name from the logical unit is compared to the file name as specified in the INIT statement.

If the file names are the same, the program verifies that volume one of the file is on the specified logical unit. If volume one is on the logical unit, the program associates the logical unit with the specified channel number. This association of channel with logical unit completes the purpose of the INIT statement.

If a logical unit number is not specified at compilation time, when the INIT statement is executed the program checks to see what mode was specified in the INIT statement. The program then displays a mount message to prompt the operator to specify a logical unit number. After the operator specifies a logical unit number, the device is found, the first block is read, comparisons and verifications are made, and the channel number is associated with the logical unit.

If the file name on the first block of the logical unit is different than the file name specified in the INIT statement, the program checks to see what mode is specified in the INIT statement. If the specified mode is input, the program displays a mount message to prompt the operator to specify another logical unit number. After the operator specifies a logical unit number, the device is found, the first block is read, comparisons and verifications are made, and the channel number is associated with the logical unit.

If the file names are different and the specified mode is output, the program checks to see if the file name on the logical unit is a temporary file name. If the file name is temporary, the program replaces the temporary name with the name specified in the INIT statement and associates the specified channel number with the logical unit.

If the file names are different, the specified mode is output, but the file name is not temporary, the program displays a replace message to prompt the operator to decide whether to replace the file name on the logical unit. If the name is replaced, the specified channel number is associated with the logical unit. If the file name is not replaced, the entire operation goes back to the mount messages which ask for a new logical unit number.

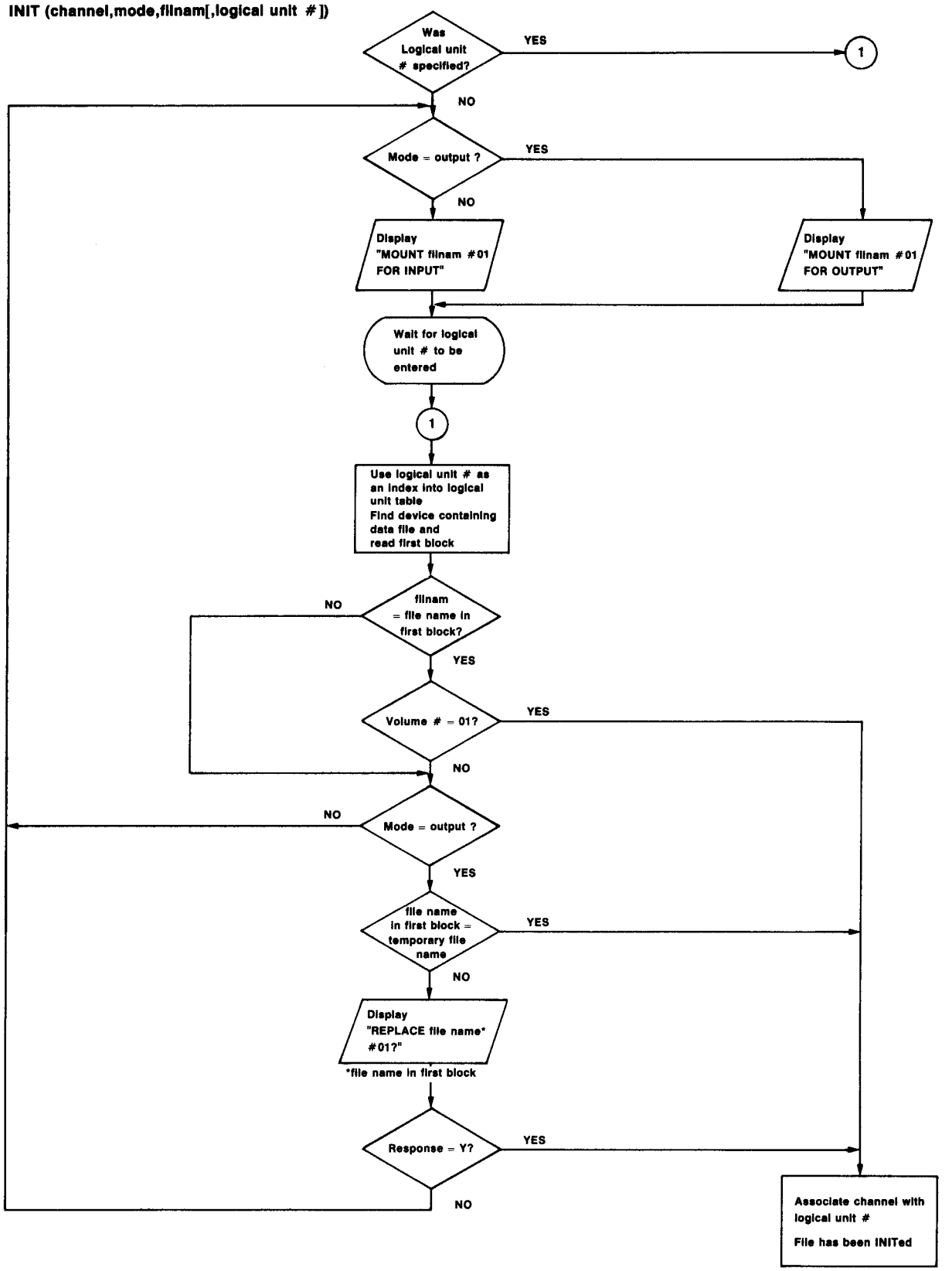


Figure 3-2 Flowchart of INIT Operation

CHAPTER 4

THE DIBOL LANGUAGE

DIGITAL'S Business Oriented Language, DIBOL, is designed to run commerical applications on COS-310 based systems. COS-310 will recognize no other high-level language programs.

A DIBOL program is divided into a Data Division and a Procedure Division. The Data Division allocates data storage, designates the names of data records and fields, determines the type of fields being used (alphanumeric or numeric), indicates the number of characters in each field, and may contain initial values assigned to a field. The Procedure Division consists of English-like action statements used with data information to develop programs.

4.1 THE DATA DIVISION

The Data Division optionally begins with START. This nonexecutable statement issues a top-of-page command. A heading optionally follows a semicolon after START. COS-310 prints this heading at the top of the page. Any comment to accompany START follows a second semicolon.

The Data Division contains RECORD statements and accompanying field data information. The RECORD statement designates the beginning of a group of data fields; the group is called a record. RECORD statements control the location in memory where the data is stored during program execution. Unnamed records (records without labels) cannot be used for input/output operations. The RECORD statement should have a space but no punctuation between the word RECORD and the name assigned (optional) to the record.

Field data information always accompanies RECORD statements. Data fields designate the type of information (alphanumeric or numeric), the number of characters, and optionally designate an initial value for each field. Fields optionally contain array information. An array is a series of same-sized entries within the same field.

The following example illustrates a Data Division within an actual DIBOL program. COS-310 ignores comments after the semicolon (except after START and PROC). Comments are optional and are used in source program listings to explain and document the program.

Example:

```
            START
            RECORD XXPLIN           ;Record named XXPLIN.
PF01,       A7                      ;Account number.
,           A2
PF02,       A25                     ;Description.
,           A2
PF03,       A8                      ;Invoice number.
,           A2
PF04,       A8                      ;Date.
,           A2
PF05,       A10                     ;Amount.
            RECORD ACCT             ;Input account record.
ACTNO,      A7                      ;Account number.
DESC,       A25                     ;Description.
VENNO,      5A4                     ;Vendor numbers.
INVNO,      A8                      ;Invoice number.
DATE,       D6                      ;Date.
AMT,        D8                      ;Amount.
            RECORD XXHD01           ;Column headings.
,           A7   ,'ACCOUNT'
.           A5
,           A19,'ACCOUNT DESCRIPTION'
,           A6
,           A7   ,'INVOICE'
,           A4
,           A4   ,'DATE'
,           A6
,           A6   ,'AMOUNT'
            RECORD                  ;Work area.
LINE,       D2,00
```

The variation in spacing in the preceding example is not mandatory (COS-310 ignores it) but is for convenience and ease of reading.

4.1.1 RECORD and Field Labels

Labels are used to identify RECORD and field statements. A label can have any number of alphabetic and numeric characters (the first character is alphabetic) but COS-310 works with only the first six. The labels are referenced in the Procedure Division. A comma must follow a label in a field statement; if no label is used, a comma must begin a field statement. Without proper placement of punctuation, the information within the Data Division will be incorrectly interpreted. Name and label are used interchangeably. The following example contains statements and comments from a Data Division.

Example:

```
            RECORD XXPLIN           ;Record labeled XXPLIN.
PF01,       A7                      ;Field labeled PF01.
,           A2                      ;Unnamed field statement.
PF02,       A25                     ;Field named PF02.
```

4.1.2 Field Types - A or D

Numerics and the alphabetic characters A or D follow the first comma in a field statement. The alphabetic characters indicate the type of characters within the field; A for alphanumeric, D for numeric. Numeric characters before the A or D indicate the number of elements within an array. Numeric characters after the A or D indicate the number of characters in the field or in each element of an array.

Elements in an array can be referenced individually or in combinations with the use of subscripts. A subscripted label can be used in any command or statement where a label is appropriate.

Numeric fields contain up to 15 characters. Only numbers can be used in numeric fields; alphanumeric fields are required if you want to use punctuation. Calculations can only be done in numeric fields.

Alphanumeric fields can contain up to 510 characters. Any legal COS-310 character can be part of an alphanumeric field.

The following example contains statement labels, A and D designations, field size, and array information.

Example:

```
          RECORD
ACTNO,    A7                ;Alphanumeric field.
DESC,     A25               ;Twenty-five character field.
VENNO,    5A4               ;Five elements, four characters each.
INVNO,    A8                ;Eight-character alphanumeric field.
DATE,     D6                ;Six-character numeric field.
AMT,      D8                ;Eight-character field.
```

4.1.3 Initial Values

Initial values in the Data Division follow a comma after the type and character count designations. Alphanumeric initial values begin and end with single quotes. All spaces and characters enclosed between the single quotes are included in the character count; the single quotes are not counted. Numeric initial values do not require quotes. No spaces are allowed between numbers in a D (numeric) designation.

Example:

```
,         A4   ,'DATE'          ;Alphanumeric initial value.
,         A6
,         A6   ,'AMOUNT'        ;Alphanumeric initial value.
          RECORD                ;No initial value on Record.
LINE,     D2   ,00              ;Numeric initial value
```

Notice that only fields contain initial values. Initial values in the Data Division must agree in type and character count with the information defined in the type and character count designations.

4.2 THE PROCEDURE DIVISION

The Data Division is separated from the Procedure Division by the non-executable mandatory statement PROC. This statement indicates that the Data Division is complete and the Procedure Division is to begin.

The Procedure Division contains DIBOL statements for data manipulation, input/output of data, program control, and program debugging.

The following example contains statements and comments from a Procedure Division.

Example:

```
          PROC 1
          INIT(6.LP)                ;Open the printer.
          INIT(1,IN,'ACCTFL',2)     ;Open account file in input mode.
READ,     XMIT(1,ACCT,EOF)          ;Get the next account record.
          PF01=ACTNO                ;Move ACTNO to PF01.
          PF02=DESC                 ;Move description to print record.
          PF03=INVNO                ;Move INVNO to PF03.
          PF04=DATE,'XX/XX/XX'      ;Move date to print record.
          PF05=AMT,'XXX,XXX.XX'     ;Move amount to print record.
          IF(LINE.LE.0)CALL TOP     ;Start new page if needed.
          XMIT(6,XXPLIN)            ;Print this line.
          LINE=LINE-1               ;Decrement line counter.
          XXPLIN=                   ;Clear print record.
          GO TO READ                ;Continue.

EOF,      FINI(1)                   ;Close account file.
          FINI(6)                   ;Close the printer.
          STOP                      ;Stop program execution.

TOP,      FORMS(6,0)                ;Start a new page.
          XMIT(6,XXHD01)            ;Print column headings.
          FORMS(6,1)                ;Skip a line.
          LINE=55                   ;Set line counter to 55.
          RETURN                    ;Return from subroutine.
          END
```

4.2.1 Data Manipulation Statements

DIBOL data manipulation statements move data between fields and between records, calculate arithmetic expressions, convert data from one type of field to another, clear data fields, and format data.

DIBOL uses the following form as a data manipulation statement.

```
destination = source
```

COS-310 interprets this to mean that the contents of the source are moved to the destination. The destination is defined and named in the Data Division. The source is a variable, a literal, or an expression.

When data is moved from source to destination, the source remains unchanged but destination is always altered. The following example contains data manipulation statements.

Example:

```
PF01=ACTNO              ;Move account number to print record.
PF02=DESC               ;Move description to print record.
PF03=INVNO              ;Move invoice number to print record.
PF04=DATE,'XX/XX/XX'    ;Move date to print record.
PF05=AMT,'XXX,XXX.XX'   ;Move amount to print record.
   .
   .
   .
LINE=LINE-1             ;Decrement line counter.
XXPLIN=                 ;Clear print record.
```

The next two examples contain data manipulation statements. These examples are executable and will work if you input them properly.

Example:

```
RECORD
  FLD1, A7, 'DEVICES'    ;Initial value is DEVICES.
PROC
  FLD1= 'SEGMENT'        ;Move SEGMENT into FLD1.
  DISPLAY(0,0,FLD1)      ;FLD1 now contains SEGMENT.
  STOP
```

Example:

```
RECORD
  FLD1, A7, 'NUMBERS'    ;Initial value is NUMBERS.
  FLD2, D7, 1234567      ;Initial value is 1234567.
PROC
  FLD1=FLD2              ;Move 1234567 to FLD1.
  DISPLAY(0,0,FLD1)      ;FLD1 now contains 1234567.
  XMIT(8," ")            ;Execute a carriage return/line feed.
  DISPLAY(0,0,FLD2)      ;FLD2 still contains 1234567.
  STOP
```

4.2.1.1 Moving Alphanumeric Data

An alphanumeric source moved to an alphanumeric destination by a data manipulation statement is left-justified in destination. If the source has fewer characters than the destination, data is left-justified and the rightmost characters in the destination are undisturbed. If the source has more characters than the destination, data is left-justified and the rightmost characters from the source

are not moved into the destination. The following example illustrates the moving of a larger source into a smaller destination.

Example:

```
RECORD ACCT
  HEAD1, A7, 'BALANCE'    ;Seven-character alphanumeric field.
  HEAD2, A5, 'TOTAL'      ;Five-character alphanumeric field.
PROC
  HEAD2=HEAD1             ;Move the seven characters into the
                          ;five-character area.
  DISPLAY(0,0, HEAD2)     ;Display the five-character area.
  STOP
```

Alphanumeric records and fields are used in moving data. Fields can only be moved into fields, and records can only be moved into records.

4.2.1.2 Moving Numeric Data

Numeric source data moved to a numeric destination by a data manipulation statement is right-justified in destination. If source has fewer characters than destination, data is right-justified and zero filled. If source has more characters than destination, the leftmost source characters are not moved.

The following example performs a calculation, moves the sum into a destination area, and displays the answer on the screen. The sum of BILLS + TAXES + MORTG produces a 4 digit result (1075). Since costs is only 3 digits in length, the most significant digit is lost.

Example:

```
RECORD A
  BILLS, D2, 75             ;Two-character numeric field.
  TAXES, D3, 800            ;Three-character numeric field.
  MORTG, D3, 200            ;Three-character numeric field.
RECORD B
  COSTS, D3                 ;Three-character numeric field.
PROC
  COSTS=BILLS+TAXES+MORTG ;Calculate to four-digit source and
                          ;store in three-digit destination.
  XMIT (8,B)              ;Display destination (RECORD B).
  STOP
```

4.2.1.3 Moving Records

Records moved with a data manipulation statement are treated like large alphanumeric fields. Source and destination are record areas. If source has fewer characters than defined for destination, data is left-justified and the rightmost characters in destination are undisturbed. If source has more characters than defined for destination,

data is left-justified and the rightmost source characters are not moved to destination.

The following example moves the contents of the record named FRMR into the record named ENGR and displays ENGR on the screen.

Example:

```
RECORD ENGR                    ;Record named ENGR
     ,A5                       ;Five-character unnamed field.
     ,A6                       ;Six-character unnamed field.
RECORD FRMR                    ;Record named FRMR
  FLD1, A11, 'OCCUPATIONS'     ;Initial value is OCCUPATIONS.
PROC
  ENGR=FRMR                    ;Move OCCUPATIONS into ENGR.
  XMIT (8,ENGR)                ;DISPLAY record named ENGR.
  STOP
```

4.2.1.4 Calculating Arithmetic Expressions

Arithmetic expressions are used as the source in a data manipulation statement. The value of the expression is moved to the destination. Expressions can contain numeric elements, subscripted data elements, literals, variables, and arithmetic operators (#, #, +, -, *, /).

The arithmetic operations of converting to internal code (#), rounding (#), adding (+), subtracting (-), multiplying (*), and dividing (/) are performed on a priority basis. The value of the expression is calculated and the value is moved to the destination.

The following example performs calculations, stores them in a destination, and displays the record named TOTAL.

Example:

```
START
RECORD
  QORDER, D4, 0002                ;Four-digit numeric with initial
                                  ;value.
  UCOST, D4, 0200                 ;Four-digit initial value.
  ECOST, D10                      ;Ten-digit numeric field.
  Y, 5D3,000,007,100,025,023      ;Five array elements with
                                  ;three-digit initial values.
RECORD TOTAL                      ;RECORD named TOTAL.
  X, D5                           ;Only field in RECORD TOTAL.
PROC
  ECOST=UCOST*QORDER              ;Multiply and store in ECOST.
  X=X+1                           ;Add one to X and store in X.
  Y(1)=Y(X)+(25*Y(2)+Y(3))/Y(4)   ;Value of X determines subscript
                                  ;of Y. Solve equation and store
                                  ;in first element of Y.
  X=Y(3)+Y(4)+Y(1)                ;Add elements and store in X.
  XMIT (8,TOTAL)                  ;Display RECORD TOTAL on screen.
  STOP
```

The arithmetic operator # converts an alphanumeric or a numeric character to its equivalent internal code and then returns the code for use as a decimal value (see Appendix A for equivalent internal codes). This converting to internal code is expressed with # inserted prior to the source in the data manipulation statement.

```
destination = #source
```

If a character preceded by # is used in a calculation, the code conversion takes place first and then the arithmetic calculations are done and the value is moved to destination.

Example:

```
RECORD
  CUSNAM, A1, 'Q'           ;Initial value Q equal to decimal
                            ;code 50.
  FACTOR, D2, 02            ;Initial value 02.
  RATING, A3                ;Alphanumeric field.
PROC
  RATING=#CUSNAM*FACTOR     ;Multiply and store in RATING.
  DISPLAY(0,0,RATING)       ;Display alphanumeric field RATING.
  STOP
```

Another use of # is in conjunction with the digits 1 through 7 for rounding numbers and manipulating them into certain formats. When used for rounding, # is placed after the element being rounded and before the digit. The digit indicates how many characters are removed from the right of the number. If a number larger than 7 is used, the number is divided by the maximum number plus one (7+1=8) and the remainder is the number of characters to be removed. This is modulo 8.

The following exercise rounds numbers, moves them to a destination, and displays the record wherein the answer is stored.

Example:

```
START
  RECORD TOTAL1             ;Record named TOTAL1.
    SUB1, 5D5               ;Five-element numeric array named
                            ;SUB1.
  RECORD TOTAL2             ;Record named TOTAL2.
    SUBT, 5D5               ;Five-element numeric array named
                            ;SUBT.
  RECORD                    ;Unnamed record.
    SUB2, D5, 13579         ;Five-digit numeric field named
                            ;SUB2.
    SUB3, D5, 86420         ;Five-digit numeric field named
                            ;SUB3.
PROC
  SUB1(1)=SUB2#2            ;Cut SUB2 by two digits and store in
                            ;the first element of array SUB1.
  SUBT(1)=SUB3#2            ;Cut SUB3 by two digits and store in
                            ;the first element of array SUBT.
  XMIT (8,TOTAL1)           ;Display TOTAL1 (contents of SUB1).
  XMIT (8,TOTAL2)           ;Display TOTAL2 (contents of SUBT).
STOP
```

In addition to cutting off characters, # causes the rightmost remaining number to be incremented by 1 if the leftmost number that was cut off was 5 or greater.

COS-310 executes arithmetic operators in order of priority. Rounding (#) is done first, multiplying (*) and dividing (/) are done next; adding (+) and subtracting (-) are done last. Operators with the same priority are executed left to right. Operations within parentheses will be executed first.

The following example performs calculations using the same numbers but different priorities. Answers are displayed on the screen.

Example:

```
START
  RECORD
    FLD,4A5                  ;Four-character array, five charac-
                             ;ters each.
PROC
    FLD(1)=100*10/2+3-1      ;Calculate expression and store in
                             ;first element of FLD.
    FLD(2)=100*10/(2+3)-1    ;Calculate expression and store in
                             ;second element of FLD.
    FLD(3)=100*(10/2+3-1)    ;Calculate expression and store in
                             ;third element of FLD.
    FLD(4)=100*10/(2+3-1)    ;Calculate expression and store in
                             ;fourth element of FLD.
    DISPLAY (0,0,FLD(1))     ;Display first element in array.
    XMIT (8," ')             ;Carriage return/line feed.
    DISPLAY (0,0,FLD(2))     ;Display second element in array.
    XMIT (8," ')             ;Carriage return/line feed.
    DISPLAY (0,0, FLD(3))    ;Display third element in array.
    XMIT (8," ')             ;Carriage return/line feed.
    DISPLAY (0,0, FLD(4))    ;Display fourth element in array.
STOP
```

4.2.1.5 Data Conversion

Data manipulation statements allow alphanumeric fields to be converted to numeric fields for arithmetic operations and then converted from numeric to alphanumeric for display.

Signs usually associated with arithmetic calculations are alphanumeric in type and cannot be used in numeric fields. The actual display and computation requirements are simplified by this data conversion capability.

The following example converts data from alphanumeric to numeric form, performs a calculation, and displays the results of the calculation and conversion on the screen.

Example:

```
RECORD A
  NUM1,D4                    ;Numeric field.
  ,    A1
  NUM2,D4                    ;Numeric field.
  ,    A1
  NUM3,D4                    ;Numeric field.
RECORD B
  ALF1, A4, '-123'           ;Alphanumeric field.
  ,     A1
  ALF2, A4, '-456'           ;Alphanumeric field.
  ,     A1
  ALF3, A4                   ;Alphanumeric field.
PROC
  NUM1=ALF1                  ;Move contents of ALF1 into NUM1.
  NUM2=ALF2                  ;Move contents of ALF2 into NUM2.
  NUM3=NUM1+NUM2             ;Add NUM1 to NUM2 and store in NUM3.
  ALF3=NUM3                  ;Move NUM3 to ALF3.
  XMIT (8,A)                 ;Display contents of RECORD A.
  XMIT (8,B)                 ;Display contents of RECORD B.
  STOP
```

The least significant character in a numeric field may have a bit set to indicate that the field has a negative value. This saves one character of disk space for each numeric field. If this negative numeric field were displayed or printed, it would have a Q through Y as its last character.

Numeric Display	Alphanumeric Display
P	-0
Q	-1
R	-2
S	-3
T	-4
U	-5
V	-6
W	-7
X	-8
Y	-9
123T	-1234
16W	167-
612Y	-6129

The negative value of 1 appears in an alphanumeric display as -1. The numeric display for the negative value of 1 is Q. All negative numbers are displayed in numeric form using the equivalent code for the negative value of the least significant digit. People are used to seeing -1 which is an alphanumeric character combination.

This direct data conversion from a numeric negative value to an alphanumeric field allows computation and display or printing without the use of special formatting statements.

4.2.1.6 Data Formatting

Data fields can be formatted to contain special characters and punctuation which cannot be present during arithmetic calculations. Data formatting requires converting numeric fields to alphanumeric fields with the use of a data manipulation statement. The format string must begin and end with single quotes. The number of characters and spaces between the single quotes should agree with the character count defined for the destination field. Labels can be used to reference formats contained in the Data Division.

Formatting uses the following form of the data manipulation statement.

```
alphanumeric destination = numeric source, 'format string'
```

Any COS-310 character except X, Z, *, -, ., ', and , may be used in the format string. These restricted characters have special meanings and must be used with care. These meanings are explained in Chapter 1 of the COS-310 System Reference Manual.

The following example formats data and displays it on the screen.

Example:

```
START
  RECORD A                    ;Record labeled A.
    A1,A8
      ,A3                     ;Field to allocate three spaces.
    A2,A4
      ,A3
    A3,A4
      ,A3
    A4,A11                    ;Field to allocate eleven spaces.
      ,A3
  RECORD B
    FMT,A4,'X.XX'             ;Field with initial value of format
                              ;field.
    DATE,D6,103078            ;Numeric field with date informa-
                              ;tion.
    NUM,D3,123
    COST,D3,999
    TOT,D12,000007894211      ;Numeric field with twelve initial
                              ;characters.
PROC
    A1=DATE,'XX/XX/XX'        ;Format for date to be stored in A1.
    A2=NUM,'ZZX'              ;Format field; (Z suppresses lead-
                              ing ;zeros).
    A3=COST,'XXX0'            ;Format with 0 preceded by Xs.
    A4=TOT,'*XXX,XXX.XX-'     ;Format TOT; the * is inserted and
                              ;replaces leading zeros.
    XMIT (8,A)                ;Display contents of Record A.
    A2=NUM, FMT               ;Use format from statement FMT to
                              ;format NUM and store in A2.
    XMIT (8,A)                ;Display contents of RECORD A.
END
```

4.2.1.7 Clearing Fields and Records

This data manipulation statement clears an entire field or record, clears designated characters within fields, or clears designatd elements within an array. Clearing entire fields or records is done with the following manipulation statement. No source is used.

```
destination =
```

Clearing specific characters or elements requires the use of both the field or array name and subscripts. These subscripts set the limits (beginning and ending) of the characters or elements to be used.

The following example clears fields within records and displays the contents of the records.

Example:

```
START
  RECORD ACCTNG                ;Record named ACCTNG.
    ACCTPB,2A4,'0000','1111'   ;Array with two elements.
      ,A3                      ;Three-character temporary storage.
    ACCTRB,3D2,99,88,77        ;Array with initialized values.
  RECORD INVNTR                ;Record named INVNTR.
    FIFO, D7, 1357975          ;Seven-character numeric field.
    LIFO, D8, 24680864         ;Eight-character numeric field.
PROC
  ACCTPB(2)=                   ;Subscripted statement.
  ACCTRB(3)=                   ;Subscripted statement.
  XMIT (8,ACCTNG)              ;Display record ACCTNG on screen
                               ;(channel 8).
  XMIT (8,INVNTR)              ;Display record INVNTR on channel 8.
END
```

Records may be cleared in a similar fashion as arrays. The records used in an array must be in sequential order and must all be the same length in the Data Division. The records do not have to have the same name. Records are cleared to all spaces even if the record contains numeric fields. This is illustrated in the following example.

Example:

```
START
  RECORD BUSNS                 ;First record in Data Division.
    ,A20
  RECORD ENGR                  ;Second record in Data Division.
    ,D20
  RECORD ACCT                  ;Third record in Data Division.
    ,D20
PROC
  BUSNS(2)=                    ;Clears second record.
  BUSNS(3)=                    ;Clears third record.
END
```

4.2.1.8 Using Literals to Implement Data

An alphanumeric literal is a series of characters delimited by single quotes. A numeric literal is a series of up to 15 numbers (not delimited by quotes). A numeric or alphanumeric literal is used as a field anywhere except as the destination in a data manipulation statement.

The following example uses alphanumeric and numeric literals.

Example:

```
START
  RECORD INVNTY                ;Record named INVNTY.
    ITEM1, A5
    NUM1, D3
    ITEM2, A7
    NUM2, D3
    ITEM3, A9
    NUM3, D3
RECORD TOTAL                   ;Record named TOTAL.
    ITEM4, A7
    NUM4, D3
PROC
    ITEM1='POTS'               ;Literal moved to ITEM1.
    NUM1=25                    ;Numeric literal moved to NUM1.
    ITEM2=', PANS '            ;Literal moved to ITEM2.
    NUM2=101
    ITEM3=', DISHES'
    NUM3=125
    ITEM4='ITEMS= '
    NUM4=NUM1+NUM2+NUM3        ;Total moved to NUM4.
    XMIT (8,INVNTY)            ;Display record INVNTY.
    XMIT (8,TOTAL)             ;Display record TOTAL.
END
```

A record literal is a sequence of alphanumeric characters delimited by a double quote at the beginning and a single quote at the end. Record literals are used as the source in a data manipulation statement. The following example shows the use of record literals.

Example:

```
START
  RECORD HDNG                  ;Record named HDNG.
    ,A10                       ;Ten-character field.
  RECORD DATA                  ;Record named DATA.
    ,A14                       ;Fourteen-character field.
PROC
  HDNG="   MONTHLY'            ;MONTHLY preceded by three spaces.
  DATA="BALANCE REPORT'        ;Record literal moved to DATA.
  XMIT (8,HDNG)                ;Display contents of HDNG.
  XMIT (8,DATA)                ;Display contents of DATA.
END
```

Alphanumeric, numeric, or record literals cannot be altered. These are not defined in the Data Division of the program.

4.2.1.9 Incrementing Data

Another kind of data manipulation is done with the INCR statement. A variable is incremented by 1 with INCR faster than with the data manipulation statement which uses destination = source +1. It is often used to increment a counter.

The following example increments a counter to a total of 20.

Example:

```
START
  RECORD COUNT
    CTR,D2
  PROC
  LOOP,                        ;Label where control transfers.
     .
     .
     .
  INCR CTR                     ;Add 1 to CTR.
IF(CTR.LT.20)GO TO LOOP        ;Control transfer statement.
     .
     .
     .
  XMIT (8, COUNT)
  END
```

4.2.2 Input/Output Statements

4.2.2.1 DISPLAY - An Input/Output Statement

The DISPLAY statement is used to move the cursor to a particular location on the screen, to display a message beginning at that particular cursor location, and to clear the screen. The message can either be the contents of an alphanumeric data field or an alphanumeric literal. The DISPLAY statement is also used to display questions on the screen. Numeric fields cannot be displayed. The DISPLAY statement has the following form:

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

The cursor is positioned according to the values of two numeric expressions separated by a comma (x and y in the format above). The value of the first expression indicates a line on the screen; if the value is larger than the number of lines on the screen, the cursor will go to the last line on the screen. The value of the second expression indicates a character position on the screen width; if the value is larger than the number of character positions on the screen width, the cursor goes to the last position on the screen width.

Special effects are generated by a select group of numeric characters inserted as numeric fields after the cursor positioning information.

0	positions the cursor but displays no message.
1	clears from the cursor position to the end of screen.
2	clears from cursor position to the end of line.
7	sounds the terminal alarm.

Any other numeric fields must be converted to alphanumeric fields before they can be used with DISPLAY.

Examples:

```
DISPLAY (5,10,'DATE')      ;Display DATE beginning on line 5, char-
                           ;acter position 10.

DISPLAY (3,5,REC)          ;Display the contents of REC beginning
                           ;on line 3, character position 5.

DISPLAY (0,0,'RESPOND')    ;Display RESPOND at the current cursor
                           ;location.

DISPLAY (10,15,1)          ;Clear to end of screen from line 10,
                           ;character position 15.

DISPLAY (6,7,0)            ;Position the cursor at line 6, char-
                           ;acter position 7.
```

Example:

```
START
  RECORD
  LINE1, A18, 'THIS IS AN EXAMPLE'     ;Field named LINE1.
PROC
  DISPLAY (1,1,1)                      ;Clear the screen.
  DISPLAY (10,31,LINE1)                ;Display contents of LINE1.
  DISPLAY (12,37,'OF THE')             ;Begin display of literal
                                       ;on line 12.
  DISPLAY (14,32,'DISPLAY STATEMENT')  ;Begin display on line 14,
                                       ;character position 32.
  DISPLAY (0,0,7)                      ;Sound terminal alarm.
  STOP
```

4.2.2.2 XMIT - An Input/Output Statement

An XMIT statement transfers a record between memory, storage devices, and peripheral devices. The transfer involves a channel over which the transfer will happen, the name of the record involved in the transfer, and optionally includes a label which indicates a program statement to branch to if an end-of-file is read. If a record exceeds the size of the record into which it is being read, an error message is displayed.

The XMIT statement has the following form:

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

A record literal can be substituted for the record label when the record is being output.

Example:

```
XMIT (3,ACCT,TOTAL)          ;Transfers a record from channel 3
                             ;into a record called ACCT (3 must
                             ;have been open by an INIT statement
                             ;in input mode). Branch to TOTAL at
                             ;end-of-file.

XMIT (8,SCRTST)              ;Output record SCRTST (screen test)
                             ;onto the screen (channel 8).
```

XMIT statements allow records to be displayed on the screen or listed on the printer.

Example:

```
RECORD SHOW                      ;Record named SHOW.
  FLD1,A18,'THIS IS AN EXAMPLE ;Field with initial value.
  FLD2,A11,' OF AN XMIT'       ;Field with initial value.
PROC
  XMIT(8,SHOW)                   ;Transfer content of SHOW over
                                 ;channel 8 (screen).
  XMIT (8, "STATEMENT')          ;Record literal transferred over
                                 ;channel 8.
STOP
```

The XMIT statement also allows data to be read from the keyboard. Because the DELETE key does not work with XMIT statements, use CTRL/U to correct any errors. If you type more data than the record can accept, an error message will be displayed. This use of XMIT is not desirable because the ACCEPT statement does the same thing with fewer complications.

4.2.2.3 INIT and FINI - Input/Output Statements

DIBOL uses channel numbers (1-15) to reference mass storage devices and character-oriented input/output devices during program execution. If you stipulate a number greater than 15, it is treated modulo 16.

The INIT statement opens a device and associates a channel number with the device. The FINI statement closes a device and disassociates the channel number from the device.

Each device, whether mass storage or character-oriented, has a mode designation that is used along with the INIT statement. The designations (I, O, U) indicate the purpose (input, output, or update) for which the storage device was opened. Input allows sequential reading of data, output overwrites the data stored on a logical unit, and update is a random I/O operation. The designations (K, T, L, S) indicate the character-oriented device (keyboard, terminal, printer, source file) and its relationship to program execution.

The channel number and the mode designations are required in an INIT statement. COS-310 is shipped with channel numbers 6, 7, 8 set to default to the printer, the keyboard, and the terminal respectively. If these channel numbers are associated with any other mode, their default is no longer in effect.

It is good programming practice to stipulate the data file name and the logical unit number referencing the logical unit in which the file is stored. The complete INIT statement has the following form:

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

Examples:

```
INIT (9,U,BKLG,3)           ;Opens (initializes) channel 9 for
                            ;update.  BKLG contains the file
                            ;name on logical unit 3.

INIT (1,I,'INVNTY',7)       ;Associates channel 1 for input.
                            ;INVNTY is a file found on logical
                            ;unit 7.

INIT (7,K)                  ;Associates the keyboard with chan-
                            ;nel 7.
```

Because the FINI statement disassociates the channel number from its related information, the FINI statement only needs the channel number. The FINI statement has the following form:

```
FINI (channel)
```

Examples:

Assume that the FINI statements below are in the same program as the INIT statements above.

```
FINI (9)                    ;Disassociates channel 9 as an up-
                            ;date mode; closes BKLG on logical
                            ;unit 3.

FINI (1)                    ;Closes file INVNTY and disassoci-
                            ;ates channel 1 from input mode.

FINI (7)                    ;Disassociates channel 7 from key-
                            ;board.
```

4.2.2.4 READ and WRITE - Input/Output Statements

The DIBOL statements READ and WRITE move data records between data files and areas of working memory during program execution by associating through the relative record number.

The READ statement moves a data record from a specified data file to an area of working memory defined in the program's Data Division.

The WRITE statement moves a data record from an area of working memory to a specified data file.

Both READ and WRITE statements use channel numbers, records, and record numbers. The READ and Write statements have the following forms:

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

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

The channel number (1-15) in the READ statement is associated with devices in input (I) or update (U) mode. The record label in the READ statement is the name of a record into which data is to be read. The record number (rec#) in the READ statement specifies the location within a logical unit from which data is to be read.

Example:

```
READ (3,ACTPBL,5)          ;Read record 5 of the file on chan-
                           ;nel 3 into record named ACTPBL.

READ (10,ACTRCB,11)        ;Read record 11 of the file on chan-
                           ;nel 10 into record named ACTRCB.

READ (11,STKINV,REC+5)     ;Read record determined by the ex-
                           ;pression REC+5 into STKINV.

READ (5,CUSNAM,EXAM)       ;Read EXAM from the file associated
                           ;with channel 5 into CUSNAM.
```

The channel number (1-15) in the WRITE statement is associated with devices in output (O) or update (U) mode. The record label in the WRITE statement is the name of a defined record area in the program from which data is to be output. The record number (rec#) in the WRITE statement is a number or expression specifying the location within a logical unit into which data is to be written.

Example:

```
WRITE (5,ADDRES,PLC+1)     ;Write record PLC+1 from ADDRES to
                           ;file associated with channel 5.

WRITE (11,PHONES,41)       ;Write record 41 from PHONES to file
                           ;associated with channel 11.

WRITE (15,ZIPCOD,EXPR)     ;Write record EXPR from ZIPCOD to
                           ;file associated with channel 15.
```

4.2.2.5 ACCEPT - An Input/Output Statement

The ACCEPT statement causes program execution to pause and remain dormant while you input information through the keyboard. The ACCEPT statement is most often used with the DISPLAY statement.

The ACCEPT statement has the following form:

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

The ACCEPT statement stores the keyboard entry in an alphanumeric field and stores the decimal equivalent of the terminator character (the last character typed) in a numeric field. The decimal equivalent of COS-310 terminator characters is shown with the ACCEPT statement in the COS-310 System Reference Manual.

Example:

```
ACCEPT (FLD1,DETAIL)        ;Store input in field named DETAIL
                            ;and terminator character in FLD1.

ACCEPT (DIG1,SUMMRY)        ;Store input in field named SUMMRY
                            ;and terminator character in DIG1.
```

4.2.2.6 FORMS - An Input/Output Statement

The FORMS statement causes the printer to skip lines or to start new pages. The FORMS statement has the following form:

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

The channel is any number (1-15) previously associated with a printer. The skip-code is one of the numbers 0-4095. If 0, the form goes to the top of the page. Any skip-code other than 0 causes the printer to skip that many lines before printing characters.

Negative numbers cause unpredictable results. If the skip-code exceeds 4095, COS-310 will begin reading 4096 as 0, 4097 as 1, and so on. This is modulo 4096.

Example:

```
FORMS (6,18)                ;Printer is associated with channel
                            ;6.  Begin printing 18 lines from
                            ;the top.

FORMS (15,4050)             ;Printer is associated with channel
                            ;15.  Begin printing after skipping
                            ;to a new page.

FORMS (12,4099)             ;Printer is associated with channel
                            ;12.  Begin printing after skipping
                            ;3 lines (modulo 4096).
```

4.2.3 Program Control Statements

4.2.3.1 IF - A Program Control Statement

DIBOL uses the IF statement to compare expressions and to determine a course of action. The expressions must be of the same data type to be compared. The IF statement has the following form:

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

The following two-letter codes set between periods are used to show the relationships. No spaces should separate the expressions from the relational comparison codes.

```
.EQ.    Equal
.NE.    Not Equal
.LT.    Less Than
.LE.    Less Than or Equal
.GT.    Greater Than
.GE.    Greater Than or Equal
```

If the relationship is true, the rest of the statement is executed. If the relationship is not true, program execution continues to the next line in sequence.

The statement at the end of an IF statement must be one of the following:

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

The expressions being compared must be of the same data type and can be a combination of literals, variables, or arithmetic expressions. The expressions are made the same size before a comparison is made.

Example:

```
IF (INCOME.LT.OUTGO)CALL HELP ;Branch control to HELP if INCOME
                              ;is less than OUTGO.

IF (EOJSW.EQ.1) STOP          ;Stop program if EOJSW equals 1.

IF (DEBIT.GT.CREDIT+10)RETURN ;RETURN to first statement after
                              ;previous CALL or TRAP statement if
                              ;DEBIT is greater than CREDIT + 10.

IF (ITAX.GE.RSTAX)TRACE       ;Branch to TRACE if ITAX is greater
                              ;than or equal to RSTAX.
```

4.2.3.2 STOP - A Program Control Statement

Use STOP to terminate program execution and return control to the Monitor. This statement does not close files (FINI closes files) nor does it have to be the last statement in a program (the optional statement END is the last statement). If you do not close files that were opened for output or update, the STOP statement may cause records to be lost.

A STOP statement can be used anywhere and any number of times in the Procedure Division of a DIBOL program.

4.2.3.3 GO TO - A Program Control Statement

A GO TO statement branches to the location indicated. There are two forms of the GO TO statement, unconditional and computed. The unconditional GO TO branches immediately to a location identified by a label. The unconditional GO TO statement has the following form:

```
GO TO label
```

The computed GO TO contains a number of possible referencing labels and a variable which indicates which label is to be referenced. The computed GO TO statement has the following form:

```
GO TO (label1  label2,...labeln),variable
```

The variable is a numeric value or an expression representing a value.

Example:

```
GO TO LOOP                      ;Branch control to LOOP when this
                                ;statement is executed.

GO TO(LOOP,STOP,RETURN),EXPR    ;Branch control to LOOP if value of
                                ;EXPR is 1, to STOP if value of EXPR
                                ;is 2, to RETURN if value of EXPR is
                                ;3, and the next statement if EXPR
                                ;has any other value.
```

4.2.3.4 CALL and RETURN - Program Control Statements

The CALL and RETURN statements branch to, and return from, subroutines. A subroutine is a routine to be used a number of times in a program or to be used only under certain conditions.

Write subroutines in places where they will not be run as part of sequential program execution. A CALL statement branches program execution to a subroutine; a RETURN statement branches execution from a subroutine.

The CALL statement includes a label to identify the subroutine to which it is to branch. The CALL statement has the following form:

```
CALL label
```

Example:

```
CALL LOOP                    ;Branch control to subroutine LOOP.

CALL SUBR1                   ;Branch control to subroutine SUBR1.
```

A CALL statement may be used within subroutines. As many as 50 CALL statements can be used at a time. This multiple CALL procedure (CALL1 branches to CALL2 which branches to CALL3, and so forth) is called nesting.

The RETURN statement is the last statement executed in a subroutine. Do not use a GO TO statement to branch out of a subroutine. Failure to use a RETURN statement will eventually overflow the pushdown stack arrangement and will cause the program to abort. No label is needed with the RETURN statement because it automatically branches control to the next statement after the last CALL statement that was executed. The RETURN statement has the following form:

```
RETURN
```

Do not use a RETURN statement unless it corresponds to either a CALL or a TRAP statement.

4.2.3.5 ON ERROR - A Program Control Statement

Fatal and nonfatal errors can occur during program execution. A fatal error crashes a program and requires a complete program restart. A nonfatal error is one where COS-310 detects an error, displays an error message, and waits for you to make the correction.

An ON ERROR statement branches execution to another location when a nonfatal error is encountered. This branching prevents COS-310 from issuing error messages and waiting for corrections. No automatic correction of the error is made; the DIBOL program must be written to take some corrective action or at least make an orderly shut-down (FINI) of the files before stopping. Execution continues from the statement to which control was branched.

Write an ON ERROR statement into a program just preceding (on the line above) a statement where an error might occur. The ON ERROR statement requires a label to identify the location where execution is to branch.

Example:

```
ON ERROR LOOP                ;Branch control to LOOP if the next
                             ;statement creates an error.
```

A common use of the ON ERROR statement is to verify that keyboard entry is numeric data. This verification involves using a data manipulation statement to move keyboard data from an alphanumeric to a numeric field. The following example illustrates this verification process.

```
Example:
   RECORD
      ALF, A10                ;Alphanumeric field.
      NUM, D10                ;Numeric field.
   PROC
      ACCEPT (NUM,ALF)        ;Enter data.
      ON ERROR NONNUM
      NUM=ALF                 ;Move ALF to NUM.
      XMIT (8,"NUMERIC')
      STOP
   NONNUM, XMIT (8, "NONNUMERIC')
      STOP
```

4.2.3.6 CHAIN - A Program Control Statement

Programs whose size exceeds the capacity of working memory cannot be executed unless they are divided into smaller programs. A CHAIN statement sequentially executes these smaller programs to produce the effect desired from the oversized program.

List the programs to be chained following the RUN command. The order of this listing determines the number (0-7) associated with the program. A number larger than 7 results in an error message.

A CHAIN statement written into a program must include a number 0-7. This number indicates which of the programs in the RUN command is executed next. The CHAIN statement has the following form:

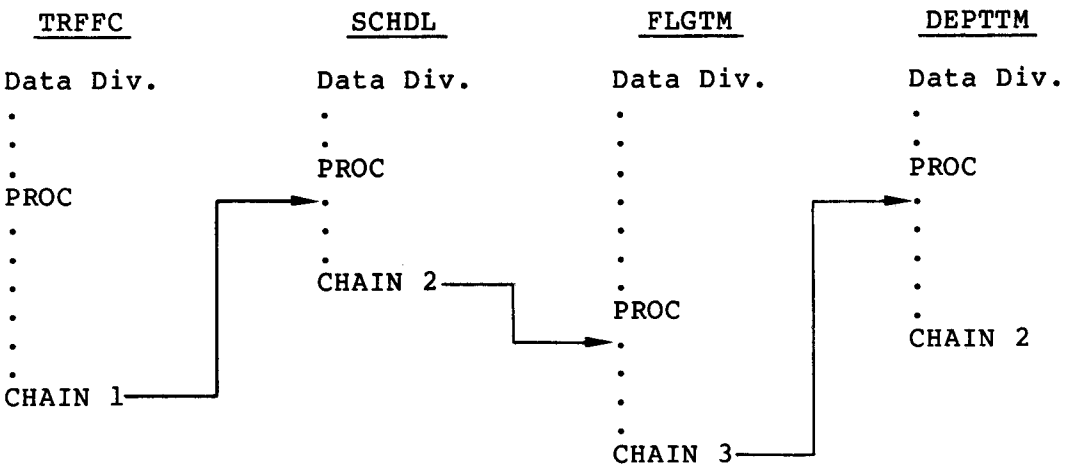
```
CHAIN number
```

A program loaded by a CHAIN statement does not automatically return to the calling program. Once you have chained out of a program, the only way to return to that program is with an appropriate CHAIN statement. Programs loaded by a CHAIN statement always begin execution immediately after the PROC statement in a program. It is good practice to close (FINI) files before chaining to another program.

Any DIBOL record storage area 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.

Example:

```
.RUN TRFFC+SCHDL+FLGTM+DEPTTM
```



4.2.3.7 TRAP and RETURN - Program Control Statements

A TRAP statement coordinates a printer and its buffer with overall program execution. Information output for the printer is put into a buffer one record at a time. The buffer holds the data until it can be printed. If information is output for the printer but the buffer is filled, a program without TRAP halts further statement execution and waits until the printer makes the buffer available for more characters.

The TRAP statement allows two separate operations to take place simultaneously. TRAP allows the DIBOL run-time system to interrupt the other task and to output more data to the printer whenever the printer buffer has more room for data.

When the buffer empties, TRAP implements the following procedure.

- Program execution temporarily halts.
- Dibol run-time system stores the location of the halt.
- Control is transferred to the location specified in the TRAP statement hwere (most likely) the DIBOL program outputs another record to the printer and executes a RETURN.
- Control returns to the location where program execution was halted.
- Program statement execution then runs concurrently with the printer.

The TRAP statement includes a label which identifies a printer routine. The TRAP statement has the following form:

```
TRAP label
```

The last statement in the printer routine is a RETURN statement. This RETURN statement returns to the statement where TRAP interrupted program execution.

The following example program prints numbers 1-500 on the printer while other program statements are being executed.

Example:

```
            RECORD A
N,          D3
            PROC
            TRAP SUB
            FORMS (6,0)                 ;Start the printer.
            .
            .
            .                           ;Perform a task.
            .
            .
LOOP,       IF(N.LT.500)GO TO LOOP      ;If the other task finishes
                                        ;first, wait for the
                                        ;printing to finish before
                                        ;stopping the program.
            STOP
SUB,        N=N+1
            IF(N.GT.500)RETURN
            XMIT (6,A)                  ;Print record A 500 times.
            RETURN
```

4.2.4 Debugging Statements

4.2.4.1 TRACE - A Debugging Statement

When properly written and activated within a DIBOL program, TRACE provides a record of program execution by causing the line number of each statement in the Procedure Division to be printed as it is executed. These line numbers are printed in the following form:

```
AT LINE xxxx
```

If a statement line covered by TRACE contains a data manipulation statement, the result of the manipulation will be printed with the line number in the following form:

```
AT LINE xxxx
result
```

The TRACE function is activated by a /T option in the RUN command which begins the execution of a program.

4.2.4.2 NO TRACE - A Debugging Statement

The NO TRACE statement is written into a DIBOL program following the lines that are to be traced. The NO TRACE statement stops the printing of line numbers under the TRACE statement.

The following example includes TRACE and NO TRACE statements.

Example:

```
0110 RECORD
0120       ITEM,  D5
0130       HOURS, D2
0140       SALARY,D5
0150       WAGES, D7
0160 PROC
0170       HOURS=40
0180       SALARY=300
0190       TRACE
0200       WAGES=HOURS*SALARY
0210       IF(WAGES.EQ.10000)NO TRACE
0220       HOURS=10
0230       IF(HOURS.EQ.10)GO TO NEXT
0240       NO TRACE
0250 NEXT,WAGES=HOURS*SALARY
0260       NO TRACE
0270       HOURS=20
0280       WAGES=HOURS*SALARY
0290       STOP
```

When the TRACE statement is activated by the /T option in the RUN command for the preceding program, the following information will be listed on the printer.

```
AT LINE 0200
0012000
AT LINE 0210
AT LINE 0220
10
AT LINE 0230
AT LINE 0250
0003000
AT LINE 0260
```

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	↑

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 a specified number equals or exceeds the base (the modulo number). The base is then divided into the specified number and the remainder is used as the variable. In modulo 16, if 17 were specified, 17 would be divided by 16 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, alphanumeric field type, 4-3
ACCEPT,
  see XMIT, 4-16
  input/output statement, 4-19
  often used with DISPLAY, 4-19
Adding (+), 4-7, 4-8
Alarm, 7 sounds terminal, 4-15
Alphabetic characters,
  CAP LOCK key locked for, 2-1
  lowercase, vi
  uppercase, vi
Alphanumeric data, moving, 4-5
Alphanumeric fields, records treated like, 4-6
Alphanumeric literal, 4-13
Array,
  clear elements in, 4-12
  elements in, 4-3
  information, fields contain, 4-1
  number of characters in, 4-3
  subscripts reference elements in, 4-3
Arithmetic expressions,
  calculating, 4-7
  data manipulation statements calculate, 4-4
Arithmetic operators, expressions contain, 4-7
Assignments, logical unit displayed and listed, 3-4
Automatic line numbers,
  continue until CTRL/Z, 2-8
  CTRL/Z terminates, 2-2

B

B (binary) files, 2-5
/B, DFU, 3-3
BACKSPACE key, not part of COS-310, 2-2
Backup, installation start-up and, 1-3
Binary (B) files, 2-5
Binary scratch area, 2-6
  program too big for, 2-7
Block, address of first, 3-2
Braces, choice of items within, vi
Brackets, optional items within, vi
BREAK key, not part of COS-310, 2-2
Buffer, TRAP coordinates printer, 4-24

C

,C,
  in a CHAIN statement, 4-23
  record retains contents without, 4-23
Calculations in numeric fields only, 4-3
CALL and RETURN, program control statements, 4-21
CALL statement, form of, 4-22
Calling the keyboard Monitor, 2-3
CAP LOCK key, 2-1
Central Processing Unit (CPU), character sent to, 2-1
CHAIN statement, form of, 4-23
Chaining, close files before, 4-23
Channel,
  FINI disassociates device from, 4-16
  in XMIT statement, 4-14, 4-15
  INIT associates device with, 4-16
  numeric expression, vii
  with FORMS statement, 4-19
Channel number,
  COS-310 shipped with, 4-17
  in READ statement, 4-18
  in WRITE statement, 4-19
Character,
  count designations, 4-3
  DELETE key erases, 2-2
Characters,
  CAP LOCK key to input alphabetic, 2-1
  Data Division indicates number of, 4-1
  label can have number of, 4-2
  lowercase alphabetic, vi
  number in field, 4-3
  numeric after A or D, 4-3
  numeric before A or D, 4-3
  numeric generate special effects, 4-15
  restricted in format string, 4-11
  up to 15 in numeric field, 4-3
  uppercase alphabetic, vi
Clear data fields, data manipulation statements, 4-4
Clearing fields and records, 4-12
Cmndfl, command file name, vii
Code (#), converting to internal, 4-7, 4-8

Codes, relational comparision, 4-20
Comma,
  must follow label, 4-2
  in unlabeled field statement, 4-2
Command,
  DELETE (DE), 2-9
  DIRECTORY (DI), 2-4
  DI/T, 2-4
  ERASE (ER), 2-7
  FETCH (FE), 2-4
  file name, vii
  Line Number (LN), 2-7
  LIST (LI), 2-4
  RESEQUENCE (RE), 2-8
  RUN (R), 2-5
  SAVE (SA), 2-5
  WRITE (WR), 2-5
Commands,
  editor, vi
  interrelationship of, 2-5
  Monitor and editor, 2-3
  number, 2-8
Comment fields in example programs, vi
Comments, 4-1
Comparison, relational codes, 4-20
Compilation Listing, 2-6
Compile source program, 2-5
Conventions for manual, notational, vi
Conversational statements, 1-3
Convert data, data manipulation statements, 4-4
Converting to internal code (#), 4-7, 4-8
COPY key, not part of COS-310, 2-2
Corrections, character-by-character, 2-2
COS-310,
  default channel numbers, 4-17
  introduction to, v
  multiphase SORT, 1-3
  procedures for operating, v
  use of logical units with, 1-3
  utility programs, 1-3
CPU, central processing unit, 2-1
Cross-referenceing program (CREF), 1-3
CTRL key, 2-2
CTRL/C,
  terminates execution of program, 2-2
  to return to Monitor, 2-5
CTRL/O, stops display of characters, 2-2
CTRL/Q, resumes terminal output, 2-2, 2-5
CTRL/S, terminates screen output, 2-2, 2-5
CTRL/U,
  deletes a line, 2-2
  to correct an error, 2-2
CTRL/Z,
  line numbers continue until, 2-8
  terminates automatic line numbers, 2-2
Cursor,
  after last character, 2-3
  location, display message at, 4-14
  move to location on screen, 4-14
  position according to values, 4-14

D

D, numeric field type, 4-3
/DL, DFU, list table on printer, 3-4
DAFT, dump-and-fix technique, 1-3
Data,
  data manipulation statements convert, 4-4
  data manipulation statements format, 4-4
  fields, data manipulation statements clear, 4-4
  fields used in moving, 4-6
  file name (filnam), vii
  files, display a list of, 3-4
  files, print a list of, 3-4
  formats referenced by labels, 4-11
  formatting, data manipulation allows, 4-11
  moving alphanumeric, 4-5
  moving numeric, 4-6
  on logical unit, first block of, 3-6
  RECORDS used in moving, 4-6

using literals to implement, 4-13
XMIT reads from keyboard, 4-16
Data conversion, data manipulation allows, 4-9
Data devices, arrangement of storage on, 3-5
Data Division,
allocates data storage, 4-1
designates names of files, 4-1
destination defined in, 4-5
determines types of fields, 4-1
in a program, 1-2
indicates number of characters, 4-1
initial values in, 4-3
literals defined in, 4-3
may contain initial values, 4-1
optionally begins with START, 4-1
Data File Utility Program (DFU), 1-3
Data Manipulation statements,
allow data conversion, 4-9
allow data formatting, 4-11
calculate arithmetic expressions, 4-4
clear fields, 4-12
clear records, 4-12
convert records, 4-12
DIBOL form of, 4-4
examples of, 4-5
format data, 4-4
INCR, 4-14
in TRACE, 4-25
move data, 4-4
Datasystem 308 (D308), 1-1
Datasystem 310 (D310), 1-1
Date, Monitor asks for, 2-3
DDT, DIBOL debugging technique, 1-3
DE (DELETE) command, 2-9
Debugging statements, 1-3, 4-25
Decimal equivalent, 4-19
DECstation 88, 1-1
DECstation 78, 1-1
Default channel numbers, COS-310 shipped with, 4-17
Delete a line, CTRL/U, 2-2
DELETE (DE) command (DE), 2-9
DELETE key to correct error, 2-2
DELETE key, will not work with XMIT, 4-16
Designations,
character count, 4-3
of mass storage media, vi
three-character, vi
two-character, vi
Designations,
data manipulation statement in, 4-4
defined in Data Division, 4-5
example of source to, 4-6
Device,
arrangement of storage on data, 3-5
arrangement of storage on system, 3-5
FINI disassociates channel from, 4-16
INIT associates channel with, 4-16
loading instructions for, 2-1
logical unit on mass storage, 3-1
media loaded into, vii
sequential order on nonsystem, 3-5
sequential order on system, 3-5
type, vii
DFU/B, 3-3
DFU, data file utility program, 1-3
DFU/D, display table on screen, 3-4
DFU/E, display a list of data files, 3-4
DFU/EL, print a list of data files, 3-4
DFU, filnam, 3-3
DFU/K, 3-2
DFU option switches, 3-2
DI (DIRECTORY) command, 2-4
DI/T command, 2-4
DIBOL, 4-1
program, labels reference, vii
program, logical units in, 3-6
DIBOL Debugging Technique, (DDT), 1-3
Directories, Monitor maintains file, 1-2
DIRECTORY (DI) command, 2-4
Directory, erase programs from, 2-9
Disk, DK indicates RK05, vii

Diskette,
  DX indicates RX02, vii
  RX indicates RX01, vii
DISPLAY statement, form of, 4-14
Dividing (/), 4-7, 4-8
DK indicates RK05 disk, vii
Dot displayed by the system, 2-3
Double-character key, shift register on, 2-1
Double quotes, record literal delimited by, 4-13
Drive number, vii
D308 (Datasystem 308), 1-1
D310 (Datasystem 310), 1-1
Dump-and-fix technique (DAFT), 1-3
DY indicates RX02 diskette, vii

E

/E, DFU, display a list of data files, 3-4
Edit buffer,
  ER clears, 2-7
  LI displays contents of, 2-4
  logical unit assignments through, 3-3
Edited, entire lines by number command, 2-8
Editor commands, Monitor and, 2-3
Editor commands spelled out, vi
Editor,
  input to, 1-2
  Monitor controls source text, 1-2
  output from, 1-2
/EL, DFU, print a list of data files, 3-4
Elements,
  in array, 4-3
  in array, clear, 4-12
  subscripts reference, 4-3
END, last entry in logical unit table, 3-3
ER command, prevent mixing of entries, 3-3
ERASE (ER) command, 2-7
Erase programs from directory, 2-9
Erase characters, DELETE key, 2-2
Error Messages, 2-3
Errors,
  correct after RETURN key, 2-2
  correction of, 2-2
  CTRL/U to correct, 2-2
  DELETE key to correct, 2-2
  fatal and nonfatal, 4-22
  MENU eliminates many operator, 1-3
ESC key, not part of COS-310, 2-2
Example programs, comment fields in, vi
Expression, source is an, 4-5
Expressions,
  calculate arithmetic, 4-4
  calculating arithmetic, 4-7
  compare with IF, 4-20

F

Fatal errors, ON ERROR with, 4-21
FETCH (FE) command, 2-4
Field,
  comment, vi
  data manipulation statements clear, 4-4, 4-12
  number of character in, 4-3
  RECORD statement designates group of, 4-1
  used in moving data, 4-6
Field data information accompanies RECORD, 4-1
Field labels, 4-2
Field statement, comma must follow label in, 4-2
Field Types, A or D, 4-3
File conversion program (FILEX), 1-3
File directories, Monitor maintains, 1-2
File name,
  command, vii
  data, vii
Files,
  binary (B), 2-5
  source (S), 2-5
  STOP does not close, 4-21
  system (V), 2-5
  three types of, 2-5
FILEX, file conversion program, 1-3
Filnam,
  data file name, vii
  DFU, 3-3

FINI,
  closes device and
    disassociates channel, 4-16
  form of statement, 4-17
Flexibility, table enhances
    storage, 3-1
Flowchart generator program
    (FLOW), 1-3
Flowchart of an INIT operation,
    3-7
Format data, data manipulation
    statements, 4-4
Format string, characters
    restricted in, 4-11
FORMS, form of statement, 4-19

G

GO TO,
  form of statement, 4-21
  out of subroutine, 4-22

H

Handlers,
  address of, 3-2
  Monitor stores I/O, 1-2
Hardware,
  drive, vii
  minimum required, 1-1
  starting instructions for, 2-1
Heading, COS-310 prints at top
    of page, 4-1

I

I, mode designation for input,
    4-17
IBM 3740, directly readable by,
    1-3
IF, form of statement, 4-20
Implement data, using literals
    to, 4-13
INCR, form of statement, 4-14
Incorrect information, error
    messages for, 2-3
Increment by 1, # causes, 4-9
Incremented, Line numbers are,
    2-7
Incrementing data, 4-14
Increments, use RE to make
    consistent, 2-8
Index, centralized, 3-1
INIT, form of statement, 4-17
INIT operation, flowchart of an,
    3-7
Initial values,
  Data Division contains, 4-1,
    4-3
  only fields contain, 4-3
Input/Output statements, 4-14
Internal code, converting to
    (#), 4-7
I/O handlers, Monitor stores,
    1-2
Input to editor, 1-2
Installation procedures, v
Installation start-up and
 backup,
    1-3
Instructions, loading and
    starting, 2-1
Interrelationship of commands,
    2-5

K

K, mode designation for
    keyboard, 4-17
/K, DFU, 3-2
Key panel adjacent to keyboard,
    2-1
Keyboard,
  data to be read from, 4-16
  input information through,
    4-19
  interact with computer, 2-1
  logical unit assignments
    through, 3-2
  Monitor, calling the, 2-3
Keys, special function, 2-1

L

L, mode designation for printer,
    4-17
Label,
  comma must follow, 4-2
  interchangeable with name, 4-2
  RECORD and field, 4-2
  reference data formats, 4-11
  reference DIBOL program, vii
  reference in Procedure
    Division, 4-2
Language, (DIBOL), high-level
    programming, 1-2
LINE FEED key, not part of
    COS-310, 2-2

Line Number (LN) command, 2-7
Line numbers,
  as references, 1-2
  CTRL/Z terminates automatic, 2-2
  incremented, 2-7
Literals,
  alphanumeric, 4-13
  defined in Data Division, 4-13
  expressions can contain, 4-7
  implement data, 4-13
  numeric, 4-13
  record, 4-13
  source is a, 4-5
LIST (LI) Command, 2-4
Listing,
  storage-map, 2-6
  two-part compilation, 2-6
Loading instructions for devices, 2-1
Logical unit assignments,
  displayed and listed, 3-4
  from a named file, 3-3
  from the edit buffer, 3-3
  through the keyboard, 3-2
Logical unit numbers, 3-2
Logical units,
  arrangement on media, 3-5
  COS-310 depends on, 1-3
  explains use of, vi
  how assigned, 3-2
  reassigned, 3-1
  referenced by number, 3-1
Logical unit table, 3-1
  displayed on screen, 3-4
  END is last entry in, 3-3
  listed on printer, 3-4
Lowercase characters, vi

M

Manual,
  notational conventions for, vi
  organization of this, v
Mass storage device, logical unit on, 3-1
Mass storage media
  designations of, vi
  also called device, vii
Media,
  also called mass storage, vii
  designations of mass storage, vi
  device, vii
  loaded into device, vii
Memory, minimum of 16K bytes of, 1-1
MENU, eliminate operator errors, 1-3
Message, display at cursor location, 4-14
Mode designation, 4-17
Modulo 16, 4-16
Monitor,
  and editor commands, 2-3
  commands spelled out, vi
  divided into two parts, 1-2
  number commands, 2-2
  system, 1-2
Move data, data manipulation statements, 4-4
Moving data,
  alphanumeric, 4-5
  fields used in, 4-6
  numeric, 4-6
  records used in, 4-6
Multiplying (*), 4-7, 4-8

N

Name,
  command file, vii
  Data Division designates, 4-1
  interchangeable with label, 4-2
  program (pronam), vii
Named file, logical unit assignments from, 3-3
Negative Numbers,
  in numeric form, 4-10
  skip-codes with, 4-19
Negative value, 4-10
Nesting, 4-22
Nonfatal errors, ON ERROR with, 4-22
Notational conventions for manual, vi
NO TRACE, form of statement, 4-26
Number commands, 2-8
Number, drive, vii
Numbers,
  lines as reference, 1-2
  skip-codes with negative, 4-19
Numeric characters,
  after A or D, 4-3
  before A or D, 4-3
  generate special effects, 4-15
Numeric data,
  moving, 4-6

ON ERROR, verifies, 4-23
Numeric elements, expressions can contain, 4-7
Numeric fields,
calculations only in, 4-3
up to 15 characters in, 4-3
Numeric form of negative numbers, 4-10
Numeric key panel, adjacent to keyboard, 2-1
Numeric literal, 4-13

O
O, mode designation for output, 4-17
ON ERROR, form of statement, 4-22
Operating COS-310, procedures for, v
Operators, expressions can contain arithmetic, 4-7
Option switches, DFU, 3-2
Optional hardware, 1-1
Organization of manual, v
Output, CTRL/S suspends terminal, 2-2
Output from editor, 1-2

P
Pages, FORMS starts new, 4-19
Parentheses, operations within executed first, 4-9
Patch, official notification from DIGITAL, 2-5
Peripheral interchange program (PIP), 1-3
PRINT, COS-310 utility program, 1-3
Printer,
records listed on, 4-16
routine, 4-25
TRAP coordinates with program, 4-24
Priority, calculations on, 4-7
PROC statement, nonexecutable mandatory, 4-4
Procedure Division,
contains DIBOL statements, 4-4
develop programs in, 4-1
example statements from, 4-4
in a program, 1-2
labels referenced in, 4-2
Procedures, installation, v
Program,
comments explain and document, vi, 4-1
compile the source, 2-5
control statements, 4-20
CTRL/C terminates execution of, 2-2
labels reference DIBOL, vii
Monitor controls execution, 1-2
name (pronam), vii
too big, 2-7
use of logical units, 3-6
Program execution,
TRACE provides record of, 4-25
TRAP coordinates printer with, 4-24
Programs,
erase from the directory, 2-9
functional view of DIBOL, v
perform functions with, 1-2
system utility, 1-2
Programming language (DIBOL), 1-2
Punctuation,
in RECORD statement, 4-1
proper placement of, 4-2
Push-down sequence, numbers assigned in, 3-5

Q
Questions, to display on screen, 4-14
Quotes,
double, 4-11, 4-13
single, 4-13

R
R (RUN) command, 2-5
READ, form of statement, 4-18
RE (RESEQUENCE) command, 2-8
Reassigned, logical units, 3-1
Record label, 4-2
in READ statement, 4-18
in WRITE statement, 4-18
Record literal, 4-13
Record number (rec#),
in READ statement, 4-18
in WRITE statement, 4-18
RECORD statement,
designates a group of fields, 4-1
punctuation in, 4-1

Records,
  cleared as in an array, 4-12
  data manipulation statement clears, 4-12
  display or list, 4-16
  field data information accompanies, 4-1
  treated like alphanumeric fields, 4-6
  used in moving data, 4-6
Red, characters printed in, vi
Relational comparison codes, 4-20
Report, DIBOL program will generate, 1-3
Representations, symbolic, vi
RESEQUENCE (RE) Command, 2-8
Resume terminal output, CTRL/Q, 2-2
RETURN key, 2-1
  correct errors after, 2-2
  no special symbol for, vi
  press following input, vi
RETURN, form of statement, 4-21, 4-24
  last statement in printer routine, 4-25
  with CALL, 4-21
  with CALL or TRAP, 4-22
  with TRAP, 4-24
RK05 disk, DK indicates, vii
Rounding (#), 4-7, 4-8
RUN (R) command, 2-5
  programs to be chained follow, 4-23
  /T activates TRACE, 4-25
RX indicates RX01 diskette, vii
RX01 and RX02 drives, 1-1
RX01 diskette, RX indicates, vii
RX02 diskette, DY indicates, vii
RX02 drives, RX01 and, 1-1

S

S, mode designation for source file, 4-17
S (source) files, 2-5
SAVE (SA) command, 2-5
Scratch area, binary, 2-6
Screen,
  clear the, 4-14
  move cursor to location on, 4-14
  records displayed on, 4-16
  suspend output to, 2-2
Segments on logical units, 3-1
Shift register on double-character key, 2-1
Skip-code, 4-19
Skip lines, FORMS causes printer to, 4-19
Software, unauthorized changes to, 2-5
SORT, COS-310's multiphase, 1-3
Source,
  data manipulation statement, 4-4
  example of, 4-6
  expression as, 4-5
  files (S), 2-5
  literal as, 4-5
  program, compile a, 2-5
  record literal as, 4-13
  text editor, Monitor controls, 1-2
  variable as, 4-5
Spacing, variation for ease of reading, 4-2
START, nonexecutable statement, 4-1
Starting address of first block, 3-2
Starting instructions for hardware, 2-1
Start-up and backup, installation, 1-3
Statements,
  comma must follow label in, 4-2
  conversational, 1-3
  DIBOL are mnemonic, 1-2
  from Procedure Division, 4-4
Stop display of characters, CTRL/O, 2-2
STOP,
  does not close files, 4-21
  program control statement, 4-21
Storage,
  arrangement of data device, 3-5
  arrangement of system device, 3-5
  Data Division allocates, 4-1
  designations of mass, vi
  flexibility, 3-1
  media, vii
Storage device, logical unit on mass, 3-1
Storage-map listing, 2-6

Subroutines,
  CALL statement with, 4-22
  CALL and RETURN from, 4-21
  where to write, 4-21
Subscripted data elements,
    expressions can contain, 4-7
Subscripts,
  reference elements in array,
    4-3
  to clear elements in array,
    4-12
Subtracting (-), 4-7, 4-8
Symbolic representations, vi
SYSGEN, utility program called,
    1-3
System,
  date, change the, 2-3
  devices, arrangement of
    storage on, 3-5
  files (V), 2-5
  Monitor, 1-2
  utility programs, 1-2

T

T, mode designation for
    terminal, 4-17
/T, TRACE function activated by,
    4-25
Table,
  display on screen, 3-4
  END in logical unit, 3-3
  listed on printer, 3-4
  output an expanded, 3-4
Terminal alarm, 7 sounds, 4-15
Terminal output, CTRL/Q resumes,
    2-2
Terminates program execution,
  CTRL/C, 2-2
  STOP, 4-21
Terminator character, 4-19
Text editor, Monitor controls
    source, 1-2
Three-character designation, vi
Top-of-page command, START
    issues, 4-1
TRACE, form of statement, 4-25
Transfer, of records, 4-15
TRAP, form of statement, 4-24
Type of fields, Data Division
    determines, 4-1

U

U, mode designation for update,
    4-17
Unnamed records, 4-1
Utility programs, system, 1-2
Uppercase alphabetic characters,
    vi

V

V (system) files, 2-5
Values, Data Division may
    contain initial, 4-1

W

WRITE (WR) command, 2-5
WRITE, form of statement, 4-18

X

XMIT statement, form of, 4-15

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
MK 2/H32
Camp Sargent Road
Merrimack, New Hampshire 03054

digital equipment corporation