DEC DATASYSTEMS 300 SERIES

COMMERCIAL OPERATING SYSTEM

SYSTEM REFERENCE MANUAL

digital

DEC-08-OCOSA-E-D

COS 300 SYSTEM

REFERENCE MANUAL

SOFTWARE SUPPORT CATEGORY

The software described in this document is supported by Digital Equipment Corporation under Category I , as defined on page iii of this document.

For additional copies, order No. DEC-08-OCOSA-E-D from Software Distribution Center, Digital Equipment Corporation, Maynard, Massachusetts 01754

First Printing, Feb. 1972
Second Printing, Mar. 1972
Third Printing, Apr. 1972
Fourth Printing, Jun. 1972
Revised, Mar. 1973

Your attention is invited to the last two pages of this document. The "How to Obtain Software Information" page tells you how to keep up-to-date with DEC's software. The "Reader's Comments" page, when filled in and mailed, is beneficial to both you and DEC; all comments received are acknowledged and are considered when documenting subsequent manuals.

The material in this document is for information purposes and is subject to change without notice.

SOFTWARE SUPPORT CATEGORIES

Digital Equipment Corporation (DEC) makes available four categories of software. These categories reflect the types of support a customer may expect from DEC for a specified software product. DEC reserves the right to change the category of a software product at any time. The four categories are as follows:

CATEGORY I
Software Products Supported at no Charge

This classification includes current versions of monitors, programming languages, and support programs provided by DEC. DEC will provide installation (when applicable), advisory, and remedial support at no charge. These services are limited to original purchasers of DEC computer systems who have the requisite DEC equipment and software products.

At the option of DEC, a software product may be recategorized from Category I to Category II for a particular customer if the software product has been modified by the customer or a third party.

CATEGORY II
Software Products that Receive Support for a Fee

This category includes prior versions of Category I programs and all other programs available from DEC for which support is given. Programming assistance (additional support), as available, will be provided on these DEC programs and non-DEC programs when used in conjunction with these DEC programs and equipment supplied by DEC.

CATEGORY III
Pre-Release Software

DEC may elect to release certain software products to customers in order to facilitate final testing and/or customer familiarization. In this event, DEC will limit the use of such pre-release software to internal, non-competitive applications. Category III software is only supported by DEC where this support is consistent with evaluation of the software product. While DEC will be grateful for the reporting of any criticism and suggestions pertaining to a pre-release, there exists no commitment to respond to these reports.

CATEGORY IV
Non-Supported Software

This category includes all programs for which no support is given

PREFACE

This manual provides a reference source for users interested in writing and operating DIBOL language and system programs in the COS 300 environment.

Chapter 1 summarizes the use of the DIBOL language in program preparation. Chapters 2, 3, 4 and 5 cover the main system programs which handle the loading and editing of programs, software system configuration and logical unit assignment, and movement of files and programs between devices. Chapters 6, 7 and 8 contain the utility programs which are useful in building, sorting and maintaining data files. Chapters 9-14 contain additional utility programs.

Appendices A through L provide additional reference material and summaries.

A glossary is provided at the end of this manual for those users who are unfamiliar with the terms used.

The system analysts, planners and user programmers who read this manual are expected to have a basic knowledge of DIBOL. Additional background in DIBOL and the COS 300 system can be obtained from the COS 300 Self-Teaching Manual (DEC-08-OCSTA-A-D).

CONTENTS

TABLES

FIGURES

INTRODUCTION

THE LANGUAGE

DIBOL (Digital Equipment Corporation Business Oriented Language) is a COBOL-like language used to write business application programs. The DIBOL language consists of data definition and procedures specified in easy to learn statements. Figures INTRO-1 and INTRO-2 illustrate a possible application work flow using DIBOL and system programs.

THE COS SYSTEM

Basic hardware requirements are:

PDP-8, 8/E, 8/I, 8/L or PDP-12 computer with 8K of core
4 DECtapes, or LINCtapes
User terminal
Line printer

For a detailed description of the basic hardware and additional supported equipment, refer to Appendix F of this manual.

Also provided as part of COS are the following system programs:

MONITOR/EDITOR	controls the calling and executing of all other programs in the COS system; provides the I/O control for the peripheral devices and an editing capability for correcting user programs. This program is referred to throughout the manual as Monitor.
SYSGEN	specifies the system configuration and assigns logical device numbers for system startup.
COMP	translates DIBOL language source statements into a binary object program which can be run on the COS 300 computer.
PIP	transfers data, source or binary files between two devices.
BUILD	creates data files based on the contents of data fields typed by the user.
SORT	sequences records according to key characters or fields specified by the user. Records may be sorted into ascending (0-9 or A-Z) or descending (Z-A or 9-0) sequence.
UPDATE	performs maintenance on data files. UPDATE makes the specified changes, creates a new file and prints a report showing all changes, deletions and insertions.

Figure INTRO-3 suggests possible usage of these system programs in DIBOL program development.

COS FILE STRUCTURE

There are four types of files used in the COS system: source, data, compiler binary, and system program.

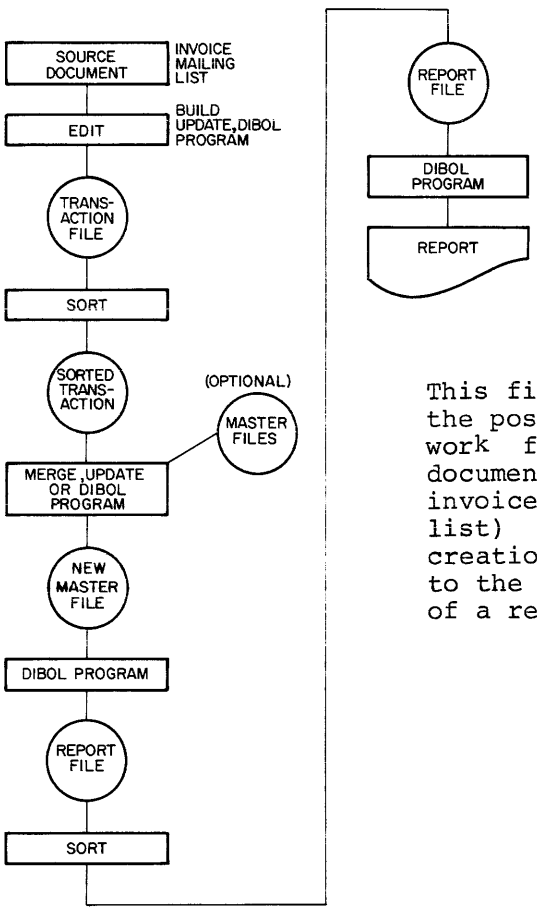
Data files are completely devoted to the storage of data to be processed by DIBOL or system programs.

Source files contain control programs (for BUILD, SORT, etc.) or user DIBOL programs.

Binary files are the output of the compiler and contain user DIBOL programs translated into a code which can be executed by COS.

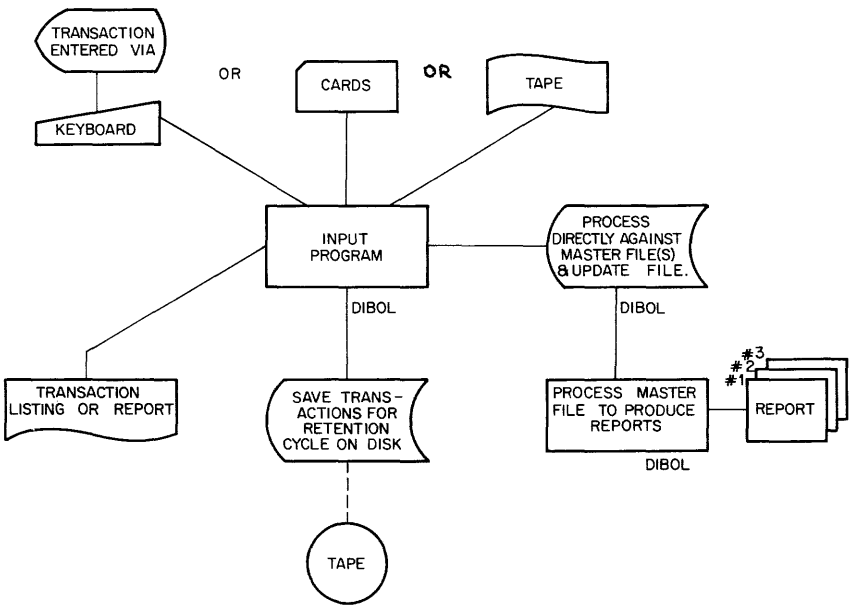
System program files include those programs (MONITOR, SYSGEN, PIP, BUILD, COMP, SORT and UPDATE) supplied as part of the COS package.

DIBOL source programs, control programs and compiled binary programs can also be saved on system devices. (Data files cannot be saved as system files.)



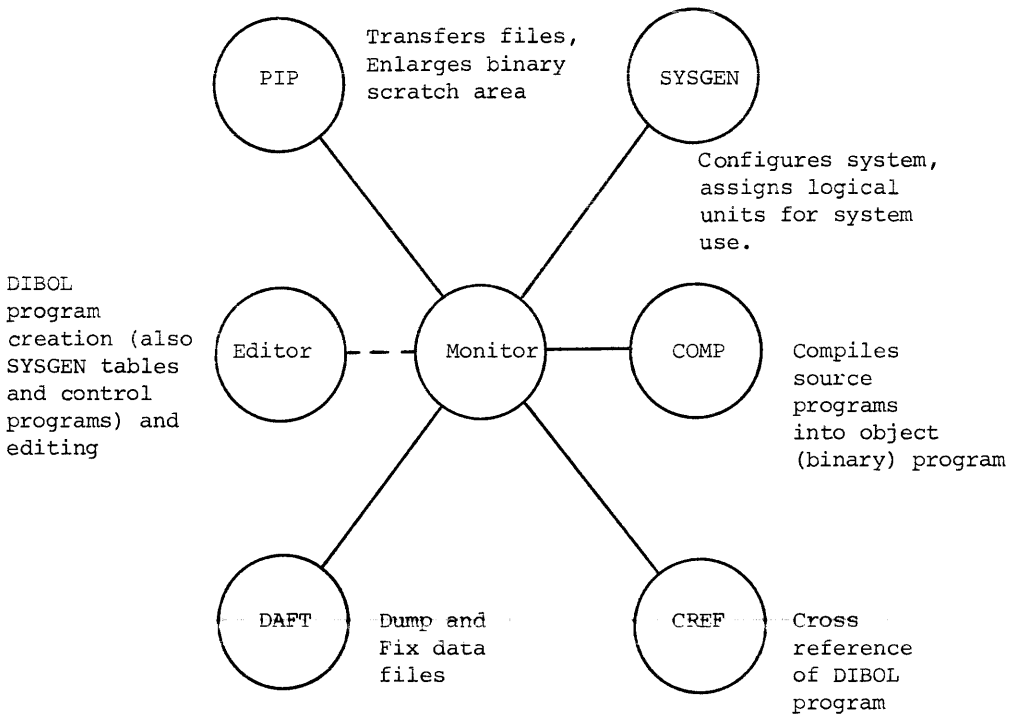
This figure illustrates the possible flow of work from a source document (such as an invoice or mailing list) through the creation of a data file to the eventual output of a report or listing.

Figure INTRO-1. Business Application System



This figure illustrates the input of data (transaction file) from terminal, card reader of DECtape; the processing against the master file by a user created DIBOL program and the output (depending on the DIBOL program being run) of reports and files.

Figure INTRO-2. Typical Disk Operation Direct Processing



This figure illustrates the COS system programs of particular interest to the programmer.

Figure INTRO-3. Programmer's System

MANUAL AND SYSTEM CONVENTIONS

The symbols and terms used throughout this manual are described below.

SYMBOLS USED	EXAMPLE	EXPLANATION
Lower-case characters	PROC n	Represent information that must be supplied by the user, such as values, names and other parameters.
Upper-case characters	NO TRACE	Words or characters that must be used exactly as shown.
...(Ellipsis)	RU prog(,...,progn)	Indicates optional continuation of arguments.
Underscored characters	COS MONITOR 2.1108	Indicates output from the system.
␣	RU␣PIP	Indicates a space or blank.
{ }	START { /T /N /L }	Braces indicate a choice of one of the items enclosed.
[ ]	START [{ /T /N /L }]	Brackets indicate an optional feature.
↵ (carriage return key)	.R COMP, PROG ↵	Line and command terminator.

TERMS

Expressions

Are variables, constants or arithmetic expressions (made up of variables, constants and the operators #,+,-,*,/).

Filnam,pronam,label and keyword

Are used to identify names assigned to files, programs statements, and input lines, these names may be of any length but only the first 6 characters are recognized.

Pronam can contain any characters except space, slash, comma, plus, period and minus.

PART I

DIBOL LANGUAGE

CHAPTER 1

DIBOL LANGUAGE

This chapter provides a reference source on the DIBOL language. If more basic information is desired, refer to the COS 300 Self-Teaching Manual (DEC-08-OCSTA-A-D).

A DIBOL program consists of a series of statements arranged in two sections, Data and PROCedure.

The Data section contains the RECORD statements which specify the type and size of the information to be used in the program and where it is to be stored.

The PROCedure section consists of:

ACCEPT
CALL
CHAIN
DISPLAY
END
FINI
FORMS
GO TO
IF
INCR
INIT
ON ERROR
PROC
READ
RETURN
STOP
TRACE/NO TRACE
TRAP
WRITE
XMIT

and data manipulation statements. These statements are arranged in a logical order for program execution. Refer to the sample program shown in Figure 1-1. Use the editing commands (refer to Chapter 2) to enter newly created DIBOL programs into the COS system. Spaces and tabs may be inserted in the program for clarity and are ignored by the Compiler except when enclosed in single quotation marks (').

Statement length is limited to 120 characters and spaces. When the 121st character is typed, an error message LINE TOO LONG is displayed and that line is lost. CTRL/U deletes the entire line. If the terminal line is filled before the 120 characters are entered, the Editor executes a carriage return/line feed to the terminal and the line can be continued.

```
START
RECORD  INBUF
        STOCKN,     D4
        DESC,       A25
        UCOST,      D5
        QORDER,     D4
        ,           D9
RECORD  OUTBUF,X
        ,           D4
        ,           A25
        ,           D5
        ,           D4
        ECOST,      D9
RECORD  LABEL,      A6,P
PROC 2
        INIT(1,IN,LABEL)
        INIT(2,OUT,'OUTPUT')
LOOP,   XMIT(1,INBUF,EOF)
        IF(STOCKN.LT.1000)GO TO LOOP
        ECOST=UCOST*QORDER
        XMIT(2,OUTBUF)
        GO TO LOOP
EOF,    FINI(2)
        FINI(1)
        STOP
END
```

Figure 1-1. Sample Program

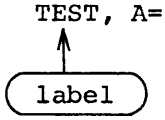
1.1 STATEMENT NUMBERS

Statement numbers are assigned manually by the user when typing in the program or automatically by the Monitor when the program is entered for editing (Refer to Chapter 2). These numbers control the order of the statements in the file. The DIBOL language makes no use of these numbers, but error messages and the DIBOL debugging refer to them.

1.2 STATEMENT LABELS

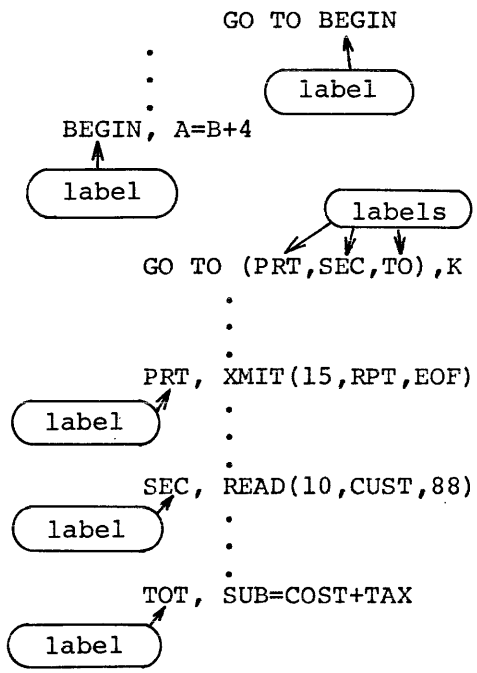
Labels are the symbolic names assigned to identify the location of a statement in the procedure section of a program.

```
TEST, A=B
```



Labels consist of a sequence of letters or digits the first of which must be a letter and only the first six characters are significant. Labels are separated from the statement by a comma. Statements with labels are the transfer point for other statements such as GO TO, IF and CALL.

Examples:



1.3 COMMENTS

Comments and notes to explain the source program may be added following a semicolon (;) on any statement line. A program line that begins with a semicolon contains only comments.

Examples:

```
RECORD CUST ; THIS IS THE CUSTOMER RECORD
; THIS PROGRAM PRINTS THE ACCOUNTS
;PAST DUE REPORT
```

Table 1-1 summarizes DIBOL source program limitations.

TABLE 1-1. SOURCE PROGRAM LIMITATIONS

Maximum characters	about 8,000 per file
maximum number of source files per program	7
Maximum number of symbols in 8K systems	365
in 12K and larger:	511
Line numbers available	0-4095
Maximum characters per line	120

1.4 STATEMENTS

There are six types of statements in DIBOL:

1. Compiler statements (START, END and PROC) which label the beginning, end and procedure section of the program. These statements are non-executable and START and END are optional.

2. Data specification statements (RECORD) which describe the type and size of data to be stored and the location in which it is to be stored.

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

4. Control statements, (GO TO, IF, CALL, ON ERROR, RETURN, TRAP, CHAIN, STOP) which govern the sequence of execution of statements within a program. Without these statements, the program would be executed in the order written.

5. Input/Output statements (ACCEPT, DISPLAY, INIT, XMIT, READ, WRITE and FINI) which control data movement within memory or between memory and peripheral devices, and to open and close the files used by the program. Line spacing to the line printer is controlled by the FORMS statement.

6. Debugging statements, (TRACE, NO TRACE) which permit tracing of program execution.

1.4.1 ACCEPT

The ACCEPT statement takes input from the keyboard, stores it in the specified alphanumeric field and causes the decimal equivalent of the last key typed to be stored in the specified decimal field. ACCEPT is primarily used with the DISPLAY statement (Section 1.4.5) and has the form:

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

where

dfield	is a decimal variable where the decimal equivalent of the last key typed is to be stored.
afield	is an alpha variable indicating where the keyboard input is to be stored.

The code stored in the decimal field depends on the last key typed (special terminator character). The keys and equivalent codes for the allowable terminator characters are shown in Table 1-2. For example, when all characters are entered, filling up afield, the code 00 is stored in the specified decimal field, which should be defined as two or more characters.

This code can be tested later in the DIBOL program.

When an ACCEPT statement is encountered during program execution, the system waits for keyboard input. Execution continues when the characters typed fill the afield or when a special character such as carriage return, line feed, altmode, rubout, CTRL/U, etc. is typed. Any non-COS character terminates the ACCEPT and sets dfield to the decimal equivalent of the last character typed. When the ACCEPT is terminated before afield is full, the remaining character positions in the afield are unchanged.

Examples:

Legal	Illegal
ACCEPT(A,B)	ACCEPT(5,CHAR)
ACCEPT(A(3),B(4,5))	ACCEPT(TCHAR,CHAR+3)

See Appendix L for more examples.

TABLE 1-2. TERMINATOR CODES

Decimal Equivalent	Teletype Console Character	VT05 Console Character
00	NULL OR END OF FIELD	
01	CTRL/A	
02	CTRL/B	
04	CTRL/D	
05	CTRL/E	
06	CTRL/F	
07	CTRL/G	
08	CTRL/H	← or CTRL/H
09	CTRL/I OR TAB	
10	CTRL/J OR LINE FEED	
11	CTRL/K	↓ or CTRL/K
12	CTRL/L	
13	CTRL/M OR CARRIAGE RETURN	CTRL/M, CARRIAGE RETURN, OR CTRL/=
16	CTRL/P	
17	CTRL/Q	
18	CTRL/R	
19	CTRL/S	
20	CTRL/T	
21	CTRL/U	
22	CTRL/V	
23	CTRL/W	
24	CTRL/X	CTRL/X or →
25	CTRL/Y	
26	CTRL/Z	↑
27	ALTMODE	ALTMODE or CTRL/[
28	CTRL/\	
29	CTRL/]	HOME or CTRL/]
30	CTRL/↑	CTRL/^
31	CTRL/←	
32	RUBOUT	
61	ALTMODE	not used
62	ALTMODE	not used
63	↑	^
94	ESCAPE	not used
95	PREFIX	not used

NOTE

Unless otherwise indicated, VT05 characters and codes same as for Teletype.

1.4.2 CALL

The CALL statement causes control to branch to a subroutine and has the form:

```
CALL   label
```

where label is the label of the first statement of a subroutine in the PROCedure section of the program.

The CALL statement saves the return location in a (pushdown) list. Additional subroutine CALL and RETURN statements may be nested within a subroutine to a depth of 50.

Examples:

```
  Legal          Illegal

CALL SET           CALL
.                  CALL(SUBR+3)
.
.
SET, PROFIT=PRICE-COST
.
.
.
RETURN
```

1.4.3 CHAIN

The CHAIN statement allows a DIBOL program which will not fit in available memory to be split up into two or more programs.

The form of the CHAIN statement is:

```
CHAIN decimal expression
```

where decimal expression refers to the position of the CHAIN program in the Monitor RUN command and is in the range 0-7.

When the CHAIN statement is encountered in a DIBOL program, execution of the current program is halted and the CHAIN program referred to by the decimal expression in the CHAIN statement is loaded. Execution of the CHAINed program begins with the statement immediately following the PROC statement. All CHAIN programs must be properly declared in a .RUN command. The program initially run is chain 0, the next program chain 1, etc. For example, a CHAIN 2 statement refers to program TRY2 in the .RUN command.

```
.RUN START+HELP1+TRY2+STOP3
```

Refer to section 2.2.6 for a description of the RUN command, section 1.4.16 for the RECORD ,C statement and to Appendix L for more information about the CHAIN statement.

1.4.4 Data Manipulation Statements

Data manipulation statements clear data fields, move data between fields, calculate decimal expressions, convert data from decimal to alphanumeric and vice versa and format data. The form of this statement is

```
destination field = source field
```

where source field is a field name, variable, constant or expression, and destination field is the field where the data is to be stored. The contents of the source field are moved to the destination field. The destination field must have been defined in the data definition section as either an alphanumeric or decimal field. If the destination or source field is decimal, the data from the source field is right-justified; if both fields are alphanumeric, left-justified. The source data is always converted to the type defined for the destination field. Record names can be used in data manipulation statements.

Arithmetic Expressions:

Arithmetic expressions are allowed on the right side of the equals sign. The expression can contain decimal elements, subscripted data elements, constants, variables and arithmetic operators. There are five binary arithmetic operators:

# (rounding)	Order of priority:
/ (division)	1. rounding
* (multiplication)	2. multiplication and division
+ (addition)	3. addition and subtraction
- (subtraction)	

These operators require decimal operands.

Operators with the same priority are executed left to right.

The result of a division operation is truncated toward zero. For example, 5/3 is 1 and -14/5 is -2.

An error occurs when a divide by zero is attempted or when the result of an addition, subtraction or multiplication is too large or too small.

There are three unary operators (-,+,#). The (+) has no effect; the unary - negates the decimal value to which it is specified. More than one unary minus can be specified for a number, e.g., A= - -3 is equivalent to A = +3 and A = --+-4 is equivalent to A = -(-(+(-4))) or A = -4.

The number sign (#) operator is also used to convert an alphanumeric character to its equivalent internal code and make that decimal number (0-63) available to the program.

The # operator takes that value from the alphanumeric or decimal field on the right side of the expression, checks the first character (left-most), determines the equivalent internal code and returns the code as a decimal number. The result is a decimal value. In this application, the number sign (#) appears before the character.

Examples:

A = #B	A is a decimal field and B is an alphanumeric field. If B contains the characters XYZ, X is converted to its internal code, 71, and the decimal equivalent, 57, is stored in A.
A = 3 + #B	A is a decimal field and B is an alphanumeric field. After conversion, the decimal equivalent of the first character of B, 57, is added to 3 and stored in A.

```
A = 10

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

The order of priority can be altered by the use of parentheses, since expressions in parentheses are executed first. The above expression equals 502.

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

The result of this example is 700.

```
Illegal

A+(B*C
C*3)-F
INTRST*((4*AMT+2)-PAYMNT/2
LENGTH+WIDTH+
6/3-
0054-
34+'ABCDE'
```

Rounding:

Variables may be truncated and rounded. The resulting variable is rounded upward if the digit which followed the remaining variable before truncation is 5 or more. The variable is rounded in terms of magnitude; sign is unchanged.

The format for rounding is:

```
A#B
```

where A is truncated by B places and rounded in magnitude. B cannot be greater than 7 and should be a positive integer. If B is negative, it is treated as positive. When used for rounding, the number sign (#) appears after a character. For example:

```
        .
        .
        .
MONEY,  D6
 .
 .
 .
TEMP=MONEY#2
```

If MONEY equals 123456, TEMP equals 1235.

If MONEY equals 123446, TEMP equals 1234.

If MONEY equals -1473, TEMP equals -15.

Typically, this feature would be used for rounding to the dollar.

Variables:

Variables may take on one of the following three formats:

a. name
b. name (subscript)
c. name (position 1, position 2)

a. name - consists of a sequence of letters and digits beginning with a letter, only the first 6 characters are significant. Used in record name, field name, etc.

b. name(subscript) - name should be a subscripted array and the value of the subscript must be between 1 and the dimension specified in the array. The subscript must be decimal and can be an expression. If either name is not an array or the value of the subscript exceeds the dimension other locations in core outside the limits of name are referenced. No error message is generated unless the user tries to reference data so far away from his data area that he is referencing his program or the system.

c. name (position1, position2) -position1 must be less than or equal to position2; Position2 should be less than or equal to the maximum size of the array associated with name. Positions 1 and 2 must be both positive and decimal. This form of subscripting references those characters from position1 to position2 inclusive. If the variable name is subscripted, you may consider the successive array elements strung out left to right. For example, in referencing A(3,9) in the array A declared as 4D4, the 7 digits are referenced as follows: the last 2 digits of A(1), the entire A(2), and the first digit of A(3).

Decimal Literals:

These consist of a sequence of from 1 to 15 decimal digits.

Alpha literals:

These consist of a sequence of any legal COS characters (except single quotes) enclosed in single quotes.

Example:

```
'ABCD'
```

RECORD Literals:

A RECORD literal is a constant that can be used anywhere in the PROCedure section of a DIBOL program where a record is required. It is similar to an alpha literal except it begins with a double quote (") and ends with a single quote ('). For example:

```
REC="THERE'
INIT (8,TTY)
XMIT (8,"HELLO')
XMIT (8,REC)
```

1.4.4.1 Clearing Fields

Data manipulation statements clear fields when used in the form:

```
field name =
```

The destination can be a single field or an array. If alphanumeric, it is cleared to all spaces; if decimal to all zeros. Fields, parts of fields or fields in an array, can also be cleared by the use of subscripts (expressions in parentheses). Any part of an array can be accessed in a program statement by listing the position of the field of character(s) in parentheses. Specifying an array name without any subscripts, clears the first array element only.

Examples:

Field Name		Meaning
F1(5,7)	=	clears character positions 5, 6, and 7 in field F1
F1(5)	=	clears the fifth element in an array
F1(A)	=	clears the Ath element in array F1
F1(1,1)	=	clears the first character.

To clear a RECORD area to all spaces use the form:

```
RECNAM=
```

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

```
RECORD Z
  ,A3
RECORD Z2
  ,A3
    .
    .
    .
Z(2)=
```

Clears the next record defined after record Z in the data section of the program.

1.4.4.2 Moving Alphanumeric Data

To move the contents of one alphanumeric field to another alphanumeric field, use the following form:

```
Alpha destination = Alpha source
```

If the source is shorter than the destination, the data is left-justified with the right-most characters of the destination field undisturbed. If the source is longer than the destination, the right-most characters of the source field are not moved into the destination field. The source field retains the original data.

For example:

```
RECORD
    A,A5,'ABCDE'
     B,A3,'FGH'
PROC
    A=B
```

A now contains FGHDE

```
RECORD A
    NAME,A4,'FRED'
    NAME1,A7,'JOHNSON'
PROC
    NAME=NAME1
```

The name in RECORD A is now JOHNJOHNSON.

1.4.4.3 Moving Decimal Data

Use a statement of the following form to move the contents of one decimal field to another decimal field.

```
decimal destination = decimal source
```

If the value in the source field is shorter than the length of the destination field, zeros are inserted on the left. If longer, the most significant digits are not moved.

1.4.4.4 Moving Records

An assignment statement may be of the form

REC1=REC2

where REC1 and REC2 are record names or subscripted record names or REC2 may be a record literal. Its effect is to move the contents of REC2 into the space reserved by REC1. If REC2 is smaller than REC1, the right-most characters of REC1 are undisturbed. If REC2 is larger, the right-most characters of REC2 are not moved. Extreme caution must be used if the destination record is subscripted.

1.4.4.5 Converting and Formatting Data

Decimal values can be converted to alphanumeric and vice versa for the purposes of I/O and calculations. Any data field may be reformatted to contain spaces and punctuation marks which are not stored with the records on DECtape or disk and cannot be present during calculations.

Alphanumeric to Decimal Conversion - The form used for this conversion is:

Decimal variable = Alpha expression

Alphanumeric values to be converted to decimal must be 16 or less characters in length. The source is an alphanumeric expression which contains an optionally signed representation of a number. The destination is a simple or subscripted decimal variable.

Example:

DEC=ALPHA

The numeric alphanumeric field (ALPHA) is converted to decimal and stored in the decimal field (DEC). If the decimal field is larger, the data is right-justified and zero-filled. If the decimal field is smaller, the excess characters to the left are not moved. If the alpha field contains characters other than digits, spaces and signs (+ or-), the message BAD DIGIT results at run time. The statement should be preceded by an ON ERROR statement if the contents of ALPHA may contain BAD DIGIT's (refer to the ON ERROR statement for more information). Spaces and signs are not counted as characters that are moved. Spaces in the alpha field are ignored. Signs may be imbedded anywhere in the alpha field. Two minus signs are equivalent to a plus; three minus signs are equivalent to a minus; etc.

Decimal to Alphanumeric Conversion - The form of this conversion is

Alpha variable = decimal expression

The source is a decimal expression and the destination is a simple alphanumeric variable. Example:

ALPHA = DEC

The contents of the decimal field (DEC) are placed in an alphanumeric field (ALPHA). If the alphanumeric field is larger, the data is right-justified and space-filled on the left. If the alphanumeric field is smaller, the excess characters on the left are not moved. Leading zeroes in the decimal field are converted to spaces. However, if the decimal field equals zero, the alpha field is set to spaces except for the right-most character, which is a zero.

Data Formatting - Decimal fields can be reformatted and stored in alphanumeric fields using the form:

```
Alpha variable = decimal expression, format
```

The source is a decimal expression and the destination a simple alphanumeric variable. Format specifies special characters to be inserted with numbers.

Example:

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

The eight-digit decimal number at D is converted to alphanumeric code, reformatted with specified punctuation and stored in alphanumeric field A. The format string must be an alpha expression.

The formatted value is placed in the destination field, right-justified and padded with spaces if necessary. If the formatted value is larger than the destination field, the left-most characters are not moved.

Most printing characters on the line printer or terminal can be used in a format string but the following characters must be used with care: X,Z,*,-,.,', and comma. Table 1-3 shows the special characters to be used in format strings.

Examples:

```
TOTL=TEMP,'XXX,XXX.XX-'
```

If TEMP contains 12345678, TOTL will contain 123,456.78

```
AMT=PAY,'*XXX,XXX.XX'
CDATE=DATE,'XX/XX/XX'
GTOT=TOT1+TOT2,'-XXX,XXX.ZZ'
WAGES=RATE*HOURS,'XXXX.XX'
RATIO=(TOT1/TOT2)#2,'XXX.XX'
```

TOT1 is divided by TOT2; the result is rounded two places; that result is formatted and stored in RATIO.

```
A1=NUM,'XXX0'
```

If NUM contains 987, A1 will contain 9870.

```
A2=DATE,'.XX/XX/XX'
A=Q, 'Q=XX'
```

Also:

```
TOTFMT, All, '-XXX,XXX.ZZ'
GTOT=TOT1+TOT2, TOTFMT
```

TOTFMT is specified in data section and is used to specify format in the PROC section.

When using a comma, period, slash, minus sign or any other notation, each must be counted as a character position. In the first example above, for instance, TOTL must be defined in a RECORD statement as an eleven-character alphanumeric field.

In the example A=B,C, B contains a decimal value being converted to alpha and stored in A (an alpha field). The C represents an alpha field which contains the format strings to be used in the command.

TABLE 1-3. SPECIAL CHARACTERS

Character	Explanation
X	Used in a format to arrange a decimal field for printout. Each X represents a digit and leading zeros are automatically suppressed.
Z	Special character used to suppress a digit when formatting output.
*	Used in a format string to replace leading zeros and eliminate trailing spaces on printout. If the * is not the first character in the string. digits may also be replaced.
-	Inserts an arithmetic sign in a number to be printed. The sign may be placed before or after the number. If the number is positive, a space is inserted where the - is placed. If the hyphen is placed in a position following the first significant digit, but previous to the last position of a format string, it is printed like any other insertion character.
.	Inserts a decimal point in a format string and forces zeros to the right of the decimal point to be significant.
,	Used to insert a comma in a format string if there are significant digits to the left.
	All other characters are treated as insertion characters.

1.4.5 DISPLAY

The DISPLAY statement is used primarily with the VT05 terminal to display a message on the scope and to move the scope cursor to the specified line and character position. DISPLAY is used in conjunction with ACCEPT to display questions on the scope and store the replies.

The form of the DISPLAY statement is:

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

where

y — is a decimal expression representing the scope line number (1-20). If the line number specified is not within 1-20, the results are unpredictable when used with VT05.

NOTE

If y is zero, no positioning is done and x is ignored. The statement with Y equal to 0 is used to output a message without a carriage return/line feed i.e. DISPLAY(0,0,'NAME').

x — is a decimal expression representing the character position (1-72). If the character is not within 1-72 the results are unpredictable.

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

A decimal literal must be one of the following codes:

```
 0=position cursor.
 1=clear to end of scope.
 2=clear to end of line.
25=emit a bell or beep sound.
```

These are the only codes recognized by COS and use of any other codes causes unpredictable results.

afield — is a field defined as alphanumeric which contains a message to be displayed.

dfield is a field defined as decimal which contains a code that causes a particular operation.

```
 0=position cursor.
 1=clear to end of scope.
 2=clear to end of line.
25=emit a bell or beep sound.
```

These are the only codes recognized by COS and use of any other codes causes unpredictable results.

Examples:

```
DISPLAY(0,0,25)          ;ring terminal bell.

DISPLAY(1,1,1)           ;clear entire scope.

DISPLAY(10,1,1)          ;clear from character position 1
                         ;line 10 to end of scope.

DISPLAY(2,20,DAY)        ;display the contents of
                         ;alpha field DAY on line
                         ;2 starting at character
                         ;position 20.

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

DISPLAY(11,1,2)          ;clear line 11 starting at
                         ;character position 1.

DISPLAY(11,37,2)         ;clear second half of line
                         ;11.

DISPLAY(11,12,25)        ;position cursor at line 11,
                         ;character position 12 and
                         ;emit beep.

DISPLAY(Y,X,BEEP)        ;position cursor at coordinates
                         ;defined in X and Y and
                         ;execute code defined in
                         ;BEEP.  If the programs
                         ;data division contains
                              RECORD
                               Y,D2,20
                               X,D2,36
                              BEEP,D2,25
                         ;the cursor is set at line 20,
                         ;character positon 36 and an
                         ;audible beep is sounded.
```

```
DISPLAY(20,36,FIRST)      ;if the data division contains
                               RECORD
                                FIRST,A4,'NAME'
                          ;NAME is displayed at line
                          ;20 character
                          ;position 36.  The cursor is
                          ;left in the 40th position of
                          ;line 20.

DISPLAY(1,1,'HELLO')      ;display HELLO in the
                          ;upper left hand corner
                          ;of the screen.

DISPLAY(10,10,BEEP)       ;If the data division contains
                               RECORD DISP
                               POST,D2,00
                               CSCOPE,D2,01
                               CLINE,D2,02
                               BEEP,D2,25
                          ;the cursor is placed at
                          ;character position 10 on line
                          ;10 and an audible beep
                          ;is emitted.
```

See Appendix L for more examples.

1.4.6 END

This is an optional statement and is used to indicate the termination of a program. The form of the statement is

```
END[/x]
```

where

/x is an optional switch to the compiler

- /N no compiler listing of source program storage map
- /L List source program storage map on line printer
- /T List source program storage map on terminal.

If no options are specified, a compiler listing is produced as usual. The END option switch or lack of it can be overridden with an option switch in the compiler RUN command. If no symbol table is printed, the symbol count and number of free locations are not printed.

1.4.7 FINI

The FINI statement writes an end-of-file mark, closes the file, rewinds the tape (if DECtape) and writes the file length in the file label (write only). (In multi-reel files the file length is written only on the last reel.) The form is:

```
FINI (n)
```

where n is a decimal expression whose value is 1-15 specifying a channel number which was previously specified in an INIT statement. FINI disassociates the channel number from the device specified in the INIT statement. (See Section 1.4.12.) Assignments of channel numbers and devices can then be changed with another INIT statement. FINI is necessary for mass storage output files only but it is considered good programming practice to FINI each data file INITed. If specified for an input file (other than from SYS), reading of that file stops. The file may again be read (from the beginning), if another INIT statement is executed.

Examples:

Legal	Illegal
FINI(1)	FINI
FINI(005)	FINI(
FINI(A+B)	FINI(4
	FINI,3
	FINI(0)
	FINI(16)

If an output file is not INITed, the end-of-file mark is not written and records may be lost. If an update file is not FINIed, the last few records written may be lost.

1.4.8 FORMS

The FORMS statement is used to format line printer (only) output and has the form:

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

where

channel is a decimal expression (value 1-15) associated with the line printer in a previous INIT statement (refer to Section 1.4.12). If the channel specified is not associated with a line printer, the statement is ignored.

skip-code is a decimal expression with a value of 0 to 4095 which causes the line printer to go to the top of the page or skip the number of lines specified. The codes are:

0	go to top of next page
1-4095	skip this number of lines
-1	skip to channel 2 (LS8E printer only) and move paper forward n lines as specified on vertical forms control tape.
-2	print enlarged characters (LS8E printer only) for next XMIT statement. Allows only half as many characters per line as normal size printing.
-85	sound alarm bell (LS8E printer only).

Other negative numbers cause unpredictable results. The skip-code is compiled modulo 4096. For example, FORMS(6,4096) causes a top of form to be executed.

Example:

```
INIT(1,L)
.
.
.
FORMS(1,3)
```

1 is the channel number specified in a previous INIT statement and 3 is the number of lines to be left blank.

```
INIT(5,L)
.
.
.
FORMS(5,0)
```

5 is the channel number and 0 specifies top of line printer page.

```
INIT(5,L)
.
.
.
FORMS(5,-1)
```

5 is the channel number and -1 causes the page to move up the number of lines specified on the vertical forms control tape of the LS8E printer.

1.4.9 GO TO

The GO TO statement has two forms: (1) unconditional and (2) computed. It transfers control to the statement whose label is specified in the GO TO statement.

1.4.9.1 Unconditional GO TO

The form of the unconditional GO TO is:

```
GO TO label
```

where

label	is a statement label assigned to the statement in the PROCedure section where control is to be transferred.

Example:

Legal	Illegal
.	GO TO
.	GO TO,TARGET
.	
GO TO SET	
.	
.	
.	
SET, TOT=TOT+SUB	

1.4.9.2 Computed GO TO

The computed GO TO has the form:

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

where

label1,label2,...,labeln	are statement labels. (There can be any number of labels.)
variable	is a decimal variable or expression representing a value.

Control is transferred to the statement labeled label1, label2,...labeln if variable has the value 1,2,...n respectively. If variable is negative, zero or greater than the number of labels, control passes to the next statement in sequence.

Example:

```
GO TO (LOOP,LIST,TOT),KEY
```

KEY acts as a switch transferring control to LOOP if KEY=1, LIST if KEY=2, or TOT if KEY=3.

Other examples:

Legal

```
GO TO(LAB1,LAB2,LAB3),VAR
GO TO(TRY,LOOP2,TRY,TRY,TOP),Q+1
GO TO(TEST),IN
GO TO(L1,L2,L3,L1,L2,L3,OVER,L2,L3,LA83,W,L1),NOW
GO TO(EI,EI,C),(TEST+2)*1-FLAG
```

Illegal

```
GO TO(
GO TO(LAB1,LAB2
GO TO(L1,L2)
GO TO(D1,D3),
GO TO(D1,D4)KEY
GO TO(),COUNT
GO TO(S0,L0,),PEAR
GO TO(L1,,L3),KIT
```

1.4.10 IF

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

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

where

expression1,expression2	are a combination of constants, variables or arithmetic expressions of the same type.
.rel.	is one of the following relational operators .EQ. Equal .NE. Not equal .LT. Less than .LE. Less than or equal .GT. Greater than .GE. Greater than or equal
stmnt	is a control statement which is executed if the relation is TRUE. "stmnt" must be one of the following GO TO label STOP CALL label TRACE RETURN NO TRACE ON ERROR label

The two variables must be of the same type: both decimal or both alphanumeric. Numeric data may be in decimal or alphanumeric form. In a decimal comparison the shorter field is internally filled to the length of the longer field, then the comparison is made between the longer field and the zero-filled field. In an alphanumeric comparison the compare is made on the number of characters in the shorter field, i.e. in comparing a 3 character field and a 5 character field, only the 3 left-most characters of each field are used.

If the result of comparison is false, the next statement in sequence is executed.

Examples:

Legal

```
IF (A .EQ.  B) GO TO LABEL3
IF (SLOT.NE.  2) CALL BAD
IF(SALES.LT.PROFIT+TAX-RENT) NO TRACE
IF(A+B+C  .GT.  D+E*F) RETURN
IF(NUMBER.LT.D+3) CALL FIXUP
```

Illegal

```
IF (C.LE.76)
IF (D2.EQ.'ABC') THEN GO TO RAL
IF (3.NE.5) D='FALSE'
IF (Q.LE.' ), GO TO TRADE
IF (DIST.GT.RATE*TIME) GO TO (L1,L2),KEY

IF
IF )E.GT.8) GO TO LAND
IF (ABC ,GT.  DEF) GO TO CHI
IF (TIME.GT:10) GO TO BED
IF (P-1 .LT.  0, GO TO UPDATE
IF (D.LT.1 .AND.  D.GT.-3) GO TO NOW

IF (COST .GR.  EXPNSE) GO TO WR
IF (TAX .LTE.  100) GO TO BUY
```

1.4.11 INCR

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

```
INCR  variable
```

where

variable — is a decimal variable to be incremented by 1.

INCR can be used with positive numbers only and cannot be the object of an IF statement. INCR is typically used to add one to a counter and is faster than a data manipulation statement. For example:

```
   A2=A2+1
or
   INCR A2
```

Legal	Illegal
INCR A2(3)	IF (A.LT.16)INCR B
INCR B(5)	INCR A+B
INCR C(H,6)	INCR 3

1.4.12 INIT

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

```
INIT (channel, dev[,data file name][, unit])
```

where

channel is a decimal expression which evaluates to a number 1-15 specifying a channel to be associated with a logical or physical device. (Refer to section 3.3 for logical unit assignments.) This number is then used in other statements such as XMIT to refer to the same device.

If the number specified exceeds 15, it is taken modulo 16.

If not specified, the following channels are initially associated with devices at program startup. (The user may change these assignments with an INIT statement.)

5 = Paper Tape Reader
6 = Line Printer
7 = Keyboard
8 = Terminal scope or printer

A channel that is associated with a logical unit must be FINIed prior to another INIT.

dev is the designation for the COS device to be associated with the channel number. These designations can be abbreviated, since only the first character is checked. The designations are:

Designation	Abbreviation	Meaning
IN	I	Mass storage device to be used for input.
OUT	O	Mass storage device to be used for output
UPDATE	U	Mass storage device to be used for both input and output (direct access) (See READ and WRITE)
KBD	K	Input from terminal keyboard
TTY	T	Output to terminal printer or display

Designation	Abbreviation	Meaning
LPT	L	Line printer
CDR	C	Card reader
PTP	P	Paper tape punch
RDR	R	Paper tape reader
SYS	S	Input from a file located on the system device. This file name must have been specified in a RUN statement. (Refer to Appendix L, Section 6.0.)

data file name is an alpha constant or variable that identifies the data file on the logical COS unit. A data file name is necessary with the I, O, and U devices and illegal with other devices. If the name is more than six characters only the first six are used. If less than six, the name is left-justified and padded with spaces on the right. It can also be the name of a field defined in the data section with the P option which allows flexibility in the files specified to the program.

unit is an optional decimal expression used with I,O and U device codes which specifies the logical unit (1-15) where the data file name is stored or to be stored. (If the number specified exceeds 15, it is taken modulo 16) On input this unit is checked for the data file name if it is not present, a MOUNT message is displayed; on output this unit is checked for the data file name or a temporary (scratch file) name, if another file is present, a REPLACE? message is displayed.

The Monitor uses the name specified in the INIT statement in the run time MOUNT message (refer to Chapter 2). The reply to the MOUNT message is the logical COS unit where the file is stored (INPUT) or to be stored (OUTPUT). The logical unit is then checked by the MONITOR to be sure the named file is indeed stored there. In the case where a logical unit is specified in the INIT statement, the Monitor checks the logical unit for the named file and if found bypasses the MOUNT message. If not found on that logical unit the Monitor displays the usual MOUNT message.

Examples:

```
INIT (15,I,'RENEW')
```

initializes channel number 15 for input, (I). "RENEW " is the name of the input file.

```
RECORD D
   TAPEID,A6,P
.
.
.
.
PROC
.
.
.
INIT (3,IN,TAPEID)
```

The previous code causes the message ENTER TAPEID to be displayed. The user responds with the name of the file; a MOUNT message is then displayed and must be answered with the proper logical unit number. This allows different file names to be specified each time the program is run.

```
INIT(1+1,IN,FILE1)
```

initializes channel 2 for input (I). Other examples:

Legal	Illegal
INIT(1,IN,'FILE01')	INIT
INIT(2,INPUT,'FILE02')	INIT(
INIT(3,I,'FILE05')	INIT(4
INIT(4,IXYDREQW,'LABEL')	INIT<4>
INIT(5,OUT,'TAPE1')	INIT,3
INIT(6,OUTCH,'TAPE2')	
INIT(7,O,RCRD)	
INIT(1,UPDATE,'LABEL')	
INIT(3,UPD,'L')	
INIT(6,U,'L3')	
INIT(6,UP,'T453')	
INIT(5,KBD)	
INIT(5,TTY)	
INIT(4,K)	
INIT(1,TELETYPE,NAME)	
INIT(9,LPT)	
INIT(10,CDR)	
INIT(11,RDR)	
INIT(12,PTR)	
INIT(N,SYS)	
INIT(15,IN,'MASTER',A+B)	

1.4.13 ON ERROR

This statement can be inserted in the source program to cause a branch to the statement with the specified label when a non-fatal executed error occurs in the next program statement. The form of this statement is:

```
ON ERROR    label
```

where label is a statement label assigned to the statement in the PROC section where control is to be transferred.

The ON ERROR statement can be the object of an IF statement.

Examples:

```
.
.
.
ON ERROR ERRTRP
TEMP=100*ORIG/NEW
.
.
.

ERRTRP,

.
.
.
ON ERROR TRAP
DEC = ALPHA
.
.
.

TRAP,
```

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

zero divisor

bad digit in an alpha to decimal conversion

more than 15 digits in decimal field used in calculation

an end of file label was not specified in an XMIT statement

input line overflowed block it was read into

no file specified in RUN command to satisfy INIT (SYS) statement

in a random-access operation (see READ and WRITE) the file contains variable length records, record number is too large or 0, or length specified in the record header word does not match the length of the XMIT block.

1.4.14 PROC

The PROC statement separates the two sections of a DIBOL program, the data definition section and the procedure section. It is of the form:

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

where

n — is a single digit, 0-7, (not an expression) indicating the maximum number of mass storage logical units which the program will have open simultaneously. If no number is specified, the compiler assumes 7. The available core is divided into buffers to handle the number of logical units specified. The more buffers necessary, the smaller they must be and the slower the I/O process.

/x — is an optional listing switch to the compiler

- /N temporarily halts compiler listing of source program
- /L resume listing source program and errors on line printer
- /T resume listing source program and errors on terminal

The PROC option switch is active until disabled by a START or END statement with an option switch. There can be only one PROC statement. If no option switch is specified, a compiler listing is produced as usual. The PROC option switch can be overridden with an option switch at compile time. (Refer to Chapter 4.)

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

If /N is specified in the RUN COMP command, it overrides any /L's or /T's in the source program. However, /L and /T determine on what device errors are listed. If the program simultaneously opens more mass storage logical units than were specified in the PROC statement, a run time error occurs.

Examples:

Legal	Illegal
PROC	PROC 32
PROC 3;BILLING V3	PROC, 4
PROC 3/N	PROC 5+2
PROC;	PROC BILLING V3
PROC 4/L; TEST PROG.	PROC A

1.4.15 READ

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

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

where

channel	is a decimal expression with a value of 1-15 specifying a channel number which links the READ statement to the related INIT statement. (The INIT statement must specify INPUT or UPDATE as the COS device.)
record	is the record into which data is to be read.
rec#	is a decimal arithmetic expression specifying the sequence number of the record to be read.

If the program READS past the end-of-file mark, the results are undefined. See Section 1.4.22 for restrictions on usage.

Examples:

```
READ (5,REX,88)
```

reads the 88th record of the device linked to the channel which was INITed with the INIT (5,...) statement and places it in the core area labeled REX.

```
READ (6,BLT,EXPR)
```

reads the record specified by the expression EXPR and stores it in the core area labeled BLT.

Refer to Appendix L for more examples and a discussion of direct access techniques.

1.4.16 RECORD

The RECORD statement reserves areas of core where records are stored during processing. Block can be used interchangeably with RECORD.

The RECORD statement is of the form:

```
RECORD [name]
```

and is followed by statements of the form

```
fldnam,xn
```

where name labels the record area; fldnam names a field within the record and xn specifies whether the field contains alphanumeric or decimal data, and the number of characters in the field. Storage is allocated in the order that the data appears. The RECORD name can be any number of alphanumeric characters in length (only the first 6 characters are significant) and the first character must be alphabetic. The name is optional unless the program is to transfer (XMIT,READ or WRITE) the data. If no name is specified, the data is assigned to contiguous locations but is not available for input or output. Therefore, RECORD statements with no names are used for temporary storage.

Examples:

Legal	Illegal
RECORD	RECORD $
RECORD ALPHA	RECORD, ABC
RECORD T1376	RECORD 564
RECORD R	RECORD,

Each RECORD statement should be followed by one or more data field definitions. The data field name (fldnam) is optional and may be any number of alphanumeric characters in length (only the first 6 characters are significant). The first character must be alphabetic. A comma may be used without a field name if the program does not reference the individual field. This specification is used when formatting an output line for the printer, for example, so that intercolumn spaces do not require a data name, only a comma followed by the type and size.

Data field type (x) is alphanumeric (A) or decimal (D). All input/output data to the line printer, terminal or high-speed paper tape reader should be in alphanumeric form (A). Decimal (D) is specified for data to be used internally by the program. However, the alpha representation of a positive decimal number is the same as its decimal representation. This is not true of negative numbers (refer to Appendix A). Data stored in alpha fields is generally left-justified and if necessary padded with blanks on the right. Data stored in decimal fields is always right-justified and padded with zeros on the left if necessary. The data field size (n) is a decimal

number 1-510 specifying the number of characters in the field. Decimal fields to be used in arithmetic operations have a maximum length of 15 digits. If data size is omitted, 1 is assumed. The total size of all data fields in a record must not exceed 510, if the record has a name (i.e. can be input or output), otherwise the total size must not exceed 4094.

Examples:

Legal

```
RECORD MASTER
    A, D3
    M1, D4
    M2, A6
      , A6

RECORD TRANS
    CUST, A16
    ADD,  A30
    KEY,  D
       ,  D03
```

Illegal

```
RECORD NUMBER
   SETNO  D11
   ORDERS, D632

RECORD NUMBER
   B,     A-6
```

Inserting Initial Values

Initial values can be assigned to data fields as part of the field statements following a RECORD statement. The initial value may be set as part of the data field definition or marked for insertion via the terminal when the program is ready for execution. The maximum size (limited by the Monitor) of an initial value is approximately 110 characters. If no initial value is assigned to a field it is initially set to all blanks (if alphanumeric) or zeroes (if decimal).

If entered as part of the program, the alphanumeric or decimal constant is placed after the type and size specification. A comma must separate the constant from the type and size specification. The initial value to be entered must agree with the size and type specified for the field. The alphanumeric constant must be enclosed in single quotation marks (' ') and may contain any printable characters except \,',or ← . Tabs inside the single quotation marks are treated as spaces, 1 space per tab. A decimal constant is a string of decimal digits optionally preceded by a sign (+ or -) and optionally followed by a minus sign. The signs (+ or -) and quotes do not count as part of the size of the initial value. The compiler does not insert leading zeros or trailing spaces in initial values.

Examples:

Legal	Illegal
,A4, 'TEXT'	,A4: 'NOTE'
M,D6,000400	,A4, '3Q'
PA,A2,'11'	,A3,345
AA,D3,123	,D4, 'WEED'
AC,A3,'123'	,D4, 12
A,D3,-146	,D4,12345
B,D3,146-	A3 '2'
C,D3,+123	A3, 'HELLO'
	AlT
	AD,A3,

If the initial value is to be entered when the program is executed, put a comma and the character P after the type and size specification.

Example:

```
fldnam,A8,P
```

The character P is used in the data field specification to allow the user to enter data at run time. Before execution, the message

```
ENTER fldnam
```

is displayed on the terminal. The specification

```
QZB135,A6,P
```

causes the message

```
ENTER QZB135
```

to be printed at the terminal at run time. The reply may use six characters

```
999999
```

The terminal entry is stored in the field named. If the value entered at run time in response to the ENTER message is shorter than the field, the value is padded with blanks, if longer, an error will occur. The P should only be used with named fields and is executed once each time the program is run.

NOTE

When using the P option all information entered from the terminals is treated as alphanumeric e.g.,

```
LABEL,A6,P
```

in the data section causes the message

```
ENTER LABEL
```

If the user replies

```
ABCD )
```

LABEL will equal 'ABCD '. The sequence

```
LABEL,D7,P
ENTER LABEL
AB123 )
```

causes LABEL to equal AB12300.

A variable to be initialized with the P option can also be given an initial default value.

Example:

```
NAME,A3,'DOE',P
```

The default value must agree in type and size with that specified for the field. The program uses the default value if CTRL/Z is typed in answer to the run time ENTER message.

Examples:

Legal	Illegal
ENT,A6,P	,D2P
DUP,D8,P	
ZIP,A5,'00000',P	

The character D is used in the data field specification to insert the current date (as specified to the Monitor at start up) in a field to be used in a report heading. The date is stored in the form mmddyy, e.g., June 25, 1972 would be stored as 062572. The field must be exactly six characters long or the date stored in it will be incorrect. This eliminates the necessity of specifying the P option and entering the current date each time the program is run. The P and D options cannot be used on the same field.

Example:

```
RECORD A
  DATE,D6,D
```

Arrays

A number of values can be represented with a single field name by placing a repetition count before the type and size specification. All elements of the array must be the same size and type. For example:

```
TAG, 4D2
```

defines TAG to be 4 decimal fields, each 2 digits long. Data can be initially entered in an array as a continuous string separated by commas. For example,

```
TAG, 4D2,88,44,22,55
```

It is not necessary to initialize all fields of an array. However, those fields to be initialized must be the first field and any continuous number of fields after the first field, for example:

```
TAG, 4D2,88,44
```

initializes the first two fields. The P option can be used to initialize the first array element only.

Examples:

Legal	Illegal
C,4D3	,2D3, 34
D,5A2	,2D3,4545
,2A4,'BLUE'	,2A2,'AMERICA'
,1D7	LABEL,103,A1
,16A1	,3E4
GOAL,3D2,22,23,33	,0D6
UP,4A2,'AB','CD','EF','GH'	,-3A4
INPUT,3D5,P	,6D-3
	GOAL, 3D2,222333
	UP,4A2,'ABCDEFGH'

,S

The character S is used in the data field to assign the value of a variable (A2) equal to the value of the options used at run time.

Example:

```
VAR,A2,S
```

will set VAR to 'XY' if .RU pronam/XY is run.

RECORD Overlay Option

The X option in a RECORD statement allows multiple definition of fields within a RECORD. Specify the overlay option as follows:

```
  RECORD,X
or
  RECORD name,X
```

The X option must be preceded by a comma. There can be one or more overlays defining the same record but there must be a preceding RECORD statement without an X.

The overlay (X) record must be equal to or less than the size of the record being overlayed.

Examples:

Legal	Illegal
RECORD ONE	RECORD TIME,X,
RECORD X,X	RECORD X (X IS A LEGAL RECORD NAME BUT IN THIS FORM DOES NOT INDICATE AN OVERLAY)
RECORD, X	

The X overlay option is useful in reformatting a previously defined record area. For example,

```
RECORD COLOR
     F1,A4,'BLUE'
       ,A1
     F2,A3,'RED'
```

has three alphanumeric fields with a total of eight characters. Fields F1 and F2 can be referenced individually.

```
RECORD RAINBO,X
     ALL,A8
```

overlays the previously described record and permits treatment as one field. The variable ALL contains BLUE RED and setting ALL to blanks in the PROC section sets F1 and F2 to blanks. The X option also allows access to a particular portion of the record by specifying

```
RECORD B,X
       ,A4
    Q6,A4
```

which skips the first four characters and uses the last four. Q6 will contain the quantity ' RED'.

The most common use for record overlaying is with print records. Each unique format for a report is kept in a separate record overlay. For example:

```
RECORD HEAD1
H1, A10, 'HEADING #1'
RECORD HEAD2
H2, A23, 'THE SECOND HEADING LINE'
RECORD PRINT
PR132, A132
RECORD ,X
CUSTNO, A10          ;CUSTOMER NUMBER
,       A20
CUSTNM, A30          ;CUSTOMER NAME
RECORD, X
,       A35
CUSADD, A20          ;CUSTOMER ADDRESS
,       A5
CUSZIP, D5
.
.
.
PR132=H1
XMIT(6,PRINT)
PR132=H2
XMIT(6,PRINT)
PRINT=
CUSTNO=NO
CUSTNM=NAME
XMIT(6,PRINT)
PRINT=
CUSADD=ADDR1
XMIT(6,PRINT)
CUSADD=ADDR2
CUSZIP=ZIP
XMIT(6,PRINT)
.
.
.
```

Records in programs which are loaded directly by the .RUN command are automatically cleared. That is, all decimal fields are set to zero and all alpha fields are filled with blanks, unless an initial value is specified.

However, if the record is loaded as the result of a CHAIN statement, the record will retain whatever contents it may have had on the previous CHAIN unless the clear option (,C) is specified in the record.

See Appendix L and Section 1.4.3 for more detail on CHAINing records.

1.4.17 RETURN

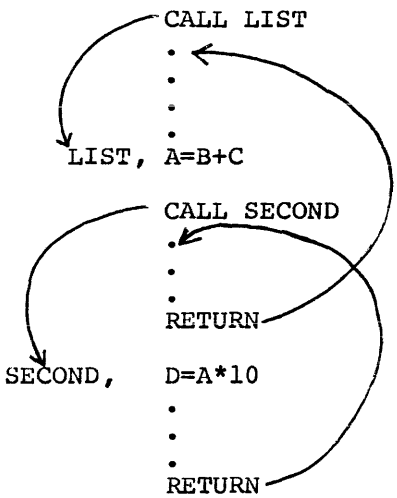
The RETURN statement is placed at the logical end of a subroutine and has the form:

```
RETURN
```

The RETURN statement causes control to return to the next statement after the last CALL (or call implied by a TRAP statement).

It is illegal to attempt a RETURN if no CALL has been executed previously and the message RETURN WITHOUT CALL results.

Example:



1.4.18 START

This is an optional statement and can be inserted anywhere in the program. It is of the form:

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

where

/x — is an optional listing switch to the compiler

- /N temporarily suspends listing of source program
- /L resumes listing of the source program and errors on the line printer
- /T resumes listing of the source program and errors on the terminal

The START option switch is active until disabled by another option switch. The START/x statement may be inserted in a source program as often as necessary. If no option switch is specified a compiler listing is produced on the line printer. The switches can be overridden with an option switch at compile time. (Refer to Chapter 4.)

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

Each time a START statement is encountered during compilation, a page eject occurs and a new page heading is printed.

Examples:

Legal

```
START
START;KR3766-CUST UPDATE
START/L
START/T;TEST PROG.  LISTING
START;
```

Illegal

```
ST
STARTK176-REPORT
START/
```

If /N is specified in the RUN COMP command, it overrides any /L's or /T's in the source program. However, /L and /T will then determine on what device errors are listed.

1.4.19 STOP

STOP terminates program execution and returns control to the Monitor. The form of the statement is:

```
STOP
```

and it may be inserted anywhere in the PROCedure section. There may be more than one STOP statement in a program. STOP does not close files; a FINI statement must be inserted before the STOP to close files previously INITed.

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

1.4.20 TRACE/NO TRACE

These statements are helpful debugging tools and can be inserted anywhere in the PROCedure section of a program. The form of the statement is:

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

Execution of the TRACE statement enables program tracing.

Execution of the NO TRACE statement discontinues the TRACE function and is optional. The initial mode is NO TRACE.

The appearance of TRACE statements in a program does not cause any TRACE output to be generated unless the /T option is specified when running the program.

When program tracing is enabled and the /T switch is on, each DIBOL statement executed causes the line

```
AT LINE xxxx
```

where xxxx is the source line number of the statement to be output on the line printer. This tracing continues until execution of a NO TRACE. If the statement is a data manipulation statement, the value which is produced and stored by the statement is printed on the following line.

For example:

```
AT LINE 0200
000006
```

NOTE

Indiscriminate placement of TRACE statements may cause excessive output to be printed on the line printer - this is often as bad as no output at all. To use the TRACE statement to best advantage an attempt should be made to "pin down" the problem to a certain part of the program flow and then, using conditional (IF) statements, trace only that part of the program.

Example:

If a certain DIBOL program works correctly on records 1-53 of a file, but produces incorrect results on the 54th record and an inspection of the program fails to reveal the reason, the following statements can be added:

In the data definition section:	RECNO,D6
at the beginning of the procedure section:	RECNO=0
after reading an input record:	INCR RECNO NO TRACE
	IF (RECNO.EQ.54)TRACE

This will provide a complete trace of the operations performed when processing record 54, and only that record.

1.4.21 TRAP

The TRAP statement allows a DIBOL program to spool output to the line printer while executing the program. The format of the statement is:

```
TRAP label
```

where

label is the label of a line printer routine.

Whenever the line printer becomes free, DIBOL statement execution temporarily terminates and a call is made to the label specified in the last TRAP statement executed. (If no TRAP statement was executed, no call is made.) When a RETURN is made from this call, normal program execution resumes. The TRAP statement normally precedes a FORMS or XMIT statement.

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

```
          RECORD A
N,        D3
          PROC
          TRAP SUB
          FORMS(6,0)   ;START LPT
          .
          .
          .            ;PERFORM TASK
          .
          .
LOOP,     IF (N.LT.500) GOTO LOOP
          STOP
SUB,      N=N+1
          IF (N.GT.500) RETURN
          XMIT(6,A)
          RETURN
```

Typically, the TRAP statement would be used in a program as follows:

```
        RECORD  LPREC
,       A80
        RECORD
LINCT,  D2
LPFLAG, D1
        RECORD  HDG
,       A60,'.....'
           .
           .
           .
        PROC n
           .
           .
           .
        INIT (1,IN,'INFILE')
        INIT (6,LPT)
        CALL PGTOP
           .
           .            ;PERFORM TASK
           .
DONE,   IF(LPFLAG.EQ.0) GO TO DONE   ;LPFLAG=0 IF TASK IS
                                     ;COMPLETED BEFORE
                                     ;PRINTING
        STOP

LPTRTN, IF(LPFLAG.NE.0) RETURN
        LINCT=LINCT-1
        IF(LINCT.EQ.0) GO TO PGTOP
        XMIT (1,LPREC,LPEOF)
        TRAP    LPTRTN
        XMIT (6,LPREC)
        RETURN
           .
           .
           .
PGTOP,  LINCT=50         ;INITIALIZE FOR NEXTPG
        TRAP    PGTOP1   ;WHERE TO GO NEXT
        FORMS(6,0)       ;SKIP TO NEW PAGE
        RETURN  ;GO BACK
PGTOP1, TRAP    PGTOP2   ;WHERE TO GO NEXT
        XMIT(6,HDG)
        RETURN
PGTOP2, TRAP    LPTRTN
        FORMS(6,3)
        RETURN
LPEOF,  LPFLAG=1
        FORMS(6,0)
        RETURN
```

NOTES

1. If the line printer buffer becomes empty during execution of an INIT, XMIT, READ, WRITE, DISPLAY or FINI statement while I/O is in progress the trap will be delayed until execution of the statement is complete.

2. If the line printer buffer becomes empty during execution of an ACCEPT statement the ACCEPT statement will be interrupted and resumed. The user typing in response to the ACCEPT statement will not be aware that the statement was interrupted and the ACCEPTed data will be correct provided the TRAP subroutine takes no longer than approximately 150 milliseconds to execute. In the example above the execution time of the TRAP routine is much shorter than 150 milliseconds if INFILE is on disk. If INFILE were on DECtape the TRAP routine might take up to one-half second to execute and the ACCEPT statement might lose some input.

3. Always construct a TRAP subroutine so that output to the line printer is immediately followed by a RETURN statement (as in the example).

4. Due to a lack of line printer buffer space in the Monitor, users with TD8/E DECtape and Centronix line printer are limited to output lines of 76 characters in length when using the TRAP statement. Outputting longer lines will result in the program spending all of its time servicing line printer TRAPs, thus cancelling the advantage of the TRAP statement.

5. The TRAP statement will not work on Analex line printers.

6. In the above example, the main DIBOL program will be slowed down by approximately 5 to 10% by the TRAP processor providing that INFILE is on disk.

1.4.22 WRITE

A direct access WRITE statement allows a specified data record to be moved from an area in core to a specified file and has the form:

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

where

channel	is a decimal expression with the value of 1-15 specifying a channel which links the WRITE statement to the related INIT statement. (The INIT statement must specify UPDATE as the COS device.)
record	is the record from which data is output.
rec#	is a decimal expression specifying the sequence number of the record to be written.

Example:

```
WRITE(5,REX,88)
```

returns the 88th record from the core area REX to the device associated with the channel specified with the INIT (5,...) statement.

NOTE

Because of the harm which the READ and WRITE statements could cause if improperly used, several restrictions have been placed upon their use:

1. The channel involved must refer to a mass storage device.
2. The file to be accessed with READ or WRITE operations must contain records of a uniform size.
3. Only one reel of a multi-reel file (the one currently mounted) may be accessed by a READ/WRITE statement.
4. The record which is specified must be exactly the same size as the records of the file being accessed.
5. Attempting to READ or WRITE over the end-of-file mark results in an error message and termination.
6. In general, READing or WRITing a record which comes after the end-of-file results in an error message; however, a fortuitous configuration of words on the unit being accessed may cause the operation to succeed and results in garbled data (on a READ) or the loss of the output record (on a WRITE) but does not crash the COS system.

7. Unless the user provides a FINI statement before terminating his DIBOL programs which have UPDATE files, the data from the last few WRITE statements may not be output properly.

Refer to Appendix L for more examples and a discussion on direct access techniques.

1.4.23 XMIT

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

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

where

channel — is a decimal expression whose value is 1-15 specifying a channel number which associates the XMIT statement with the related INIT statement.

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

One must be very careful when using subscripted record names.

A single subscript, such as REC(3) should only be used if there are consecutive records, beginning with REC and all of the same length as REC, i.e.,

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

If a double subscript form is used, i.e., REC (n,m), then n must be less than m-1, n must be odd and m must be even (or the last character in the record). This double subscript form refers to characters n-2 through m-2 inclusive in record REC. (If n=1, it refers to characters 1 through m-2.) Whenever an XMIT occurs referring to the record, two characters before character n in the record are destroyed. This is the COS 300 word count (see Appendix G). The user need not be concerned about this if n=1.

eof label — is the label of a statement to which the program branches if an end-of-file is read. Used with input files only. (Refer to Statement Labels at the beginning of this chapter.) Label is optional but if not specified, an error message is output when an end of file occurs.

Examples:

```
XMIT (3,INV,EOF)
```

transfers a record from the input file associated with the statement INIT (3,IN,...), to the RECORD area in core labeled INV. If end-of-file is reached, control branches to PROCedure section statement labeled EOF. If the length of the record being read is greater than the defined size, an error message is output at run time. If the length is less than the defined size, the record is left-justified and padded with spaces on the right.

```
XMIT (1,CUST,NEXT)
```

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

The statement

```
XMIT (2,BUFF)
```

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

```
XMIT (8, "HI THERE')
```

would output the message HI THERE on the operator's terminal if channel 8 was INITed to the TTY.

```
XMIT (8,CUST(1,7),EOF)
```

accesses the first five characters of the record area CUST.

Other examples:

Legal

```
XMIT(N,B2)
XMIT(FUNCT(I)+2,FOO)
XMIT(7,ROD,LAB3)
XMIT(9,BLKNUM,L)
XMIT(3,B1)
XMIT(M,B23A,I)
XMIT(1,HHAA,FFPP)
XMIT(8,"MOUNT INVOICE FORMS')
XMIT(15,REC(1,33))
XMIT(CHAN,REQ(1,INVOC+6))
```

The XMIT statement may be used with direct access READ and WRITE statements for operations on files. See Appendix L for details.

PART II

SYSTEM AND UTILITY

PROGRAMS

CHAPTER 2

MONITOR

The Monitor is the master control program for the COS system. It contains all the system I/O handlers (TTY, DECtape, RF08, RK08, RK8E, Line Printer, Card Reader, High-speed Reader and High-speed Punch) and enables the user to edit, compile, save and execute user programs. The Monitor maintains a directory of all programs stored on the system device, labels files and opens and closes files as needed.

During program execution the Monitor produces the messages which instruct the user to mount files. It also provides the means for BATCHing several programs for sequential execution.

Table 2-1 lists the special key commands that are available in on-line operations.

The editing feature of the Monitor can be used to create SYSGEN tables and source files on the system device for later use.

2.1 OPERATING PROCEDURES

The Monitor must be loaded via a bootstrap routine each time the system is restarted. (Refer to Appendix B.) Monitor signals that is is loaded into memory by displaying the message

```
COS MONITOR 2.1108 (or current version number)

DATE?

.
```

type the DAte command, a space, the current date, and the CR key in the form .DA mm/dd/yy. For example, .DA 1/25/72. The date must be entered before proceeding. This date is used during program execution to date reports, files, and new programs created. The Monitor responds to the date with a dot to indicate it is ready to accept any of the commands shown in Section 2.2.

TABLE 2-1. MONITOR KEY COMMANDS

Key	Function
CTRL/C	Returns control to the Monitor. Monitor displays a dot and awaits a command. If already in the Monitor, CTRL/C has the same effect as a CTRL/U.
CTRL/N	Turns on imbedded numeric keypad at terminal and can be used in a BUILD or DIBOL program for numeric input. Type CTRL/N a second time or return to Monitor and the feature is turned off. When the feature is turned on certain keys are interpreted as numbers as indicated below:

Key	Value
J	1
K	2
L	3
U	4
I	5
O	6
M	0

NOTE

Keys 7, 8, and 9 have values 7, 8, and 9. These ten keys form a numeric keypad.

CTRL/O	Suppresses terminal echo of typed output. If echo is already suppressed, CTRL/O restores the terminal echo. CTRL/O is also used to halt and resume output from an LI command or the compiler. The echo always resumes the next time the dot is printed.
CTRL/U	Deletes the current input line.
CTRL/V	Suppresses terminal echo of Monitor message, COS MONITOR 2.1108. The Monitor displays a dot (.) when CTRL/V is typed. If message is suppressed, type CTRL/V to enable message. CTRL/V has the same effect as CTRL/U.
CTRL/Z	Signals the end of input and returns control to the Monitor. Halts output of line numbers from LN command.
RUBOUT	Erases the last character typed and echoes the deleted character. RUBOUT cannot be used in all instances; if the character to be deleted does not echo when RUBOUT is typed, type CTRL/U and retype the line.

2.2 MONITOR COMMANDS:

The commands

BATCH
DATE
DELETE
DIRECTORY
PLEASE
RUN
SAVE

apply to Monitor functions. Only the first two characters of the command need be typed. (R is sufficient for the RUN command.) Any additional characters up to the first blank are ignored. All commands must be terminated by typing the CR key. With the exception of RUN, the Monitor commands have no effect on system programs (COMP, SORT, etc.). Refer to Section 2.3 for editing commands.

2.2.1 BATCH

With the BATCH command a string of Monitor commands can be executed sequentially. As soon as the execution of one program ends, another is automatically started. The form of the BATCH command is:

```
.BATCH pronam
-
```

where

pronam	is the name assigned to a list of Monitor commands previously created via editing commands and saved with a WRITE command.

For example:

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

All programs batched must be on the system device. The Batch program can contain another BATCH command only as its last line.

When a Monitor command is read from the BATCH file, it is displayed on the terminal and executed. Type CTRL/C to terminate a BATCH program. The BATCH must then be restarted at the beginning.

All of the necessary programs and data files must be available; if an error occurs, BATCH Monitor mode terminates, control returns to interactive mode, and a dot is displayed on the terminal.

Correct the error in the program being executed and restart the BATCH program at the beginning or individually type each command left to be executed.

When the BATCH program is finished, control returns to the Monitor and a dot is displayed on the terminal.

2.2.2 DATE

The DATE command stores a date to be assigned to any program created or reports printed until a new DATE command is issued or the system is restarted. The DATE command is of the form:

```
.DA  mm/dd/yy
-
```

where

```
mm  is a number 1 - 12
dd  is a number 1 - 31
yy  is a number 72 - 79
```

If the date entered does not conform to the above restrictions, an error message is displayed.

This command must be entered whenever the Monitor is restarted. It may also be used whenever it is necessary to change the system date.

Examples :

```
.DA 1/25/72
-
.DATE 12/5/73
-
```

2.2.3 DELETE

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

The DELETE command is of the form:

```
.DE pronam[,dev]/X
-
```

where

pronam	is the name of the file to be removed from the directory.
dev	is the 3-character designation for the physical device where the file is stored DECtape = DT0-DT7 disk = DK0-DK3 If no device is specified, the system device is assumed.
/X	Is a 1- or 2-character code indicating whether the file to be deleted is a source (S), binary (B) or system (SV) file. This code is necessary to differentiate between three files with the same name but of different types. The code SV is used rather than V to make it more difficult to inadvertently delete a system file.

Examples:

Legal	Illegal
.DE JOB1, DT1/B -	.DE JOB1 -
.DE PROGA,DK3/S -	.DE CONVEX/V -
.DE INV/S -	
.DE CONVEX/SV -	

See Chapter 3 for a description of device designations.

2.2.4 DIRECTORY

The DIRECTORY command prints a list of programs stored on a physical device or the label of the file stored on a logical unit. The form of the command is:

```
.DI   { [physical device] [/T] }
-     { /logical unit#         }
```

where

physical device	is one of DT0-DT7 or DK0-DK3 and must be preceded by a comma or space. If not specified the system device is assumed.
logical unit#	is the number (0-15) of the logical unit assigned with SYSGEN. A logical unit # must be preceded by a /. Specifying a logical unit causes a data file label to be printed. If the unit is a physical device which has more than one logical unit, the DI command must be repeated for each logical unit to be labeled.
/T	is an optional switch which causes the directory to be output to the terminal.

A directory contains the current date, names of programs, types of programs, length in 256-word blocks (LN) and the date the program was stored. Be sure the line printer is on-line before issuing the DI command.

A file label contains the file name, sequence number (if a multi-reel file), the date the file was created, file length in segments, and logical unit number where the file was stored when the label was requested. Segments are 16 256-word blocks long.

See Chapter 3 for a description of device designations and logical unit numbers.

Examples:

```
.DI DK0
-
```

causes a directory similar to the following to be printed on the line printer:

```
DIRECTORY      15-FEB-72

NAME    TYPE LN    DATE

COMP      V  14  19-JAN-72
MORE      S  10  15-FEB-72
 <0006 FREE BLOCKS>
TST2      S  07  12-FEB-72
 <0007 FREE BLOCKS>
TST4      S  07  15-FEB-72
GLOP      S  10  15-FEB-72
 <0579 FREE BLOCKS>
```

The command

```
.DI/3
```

outputs a file label similar to one below.

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

2.2.5 PLEASE

The PLEASE command outputs a message to the terminal during a BATCH program execution.

The form of the PLEASE is:

```
PLEASE text string
```

The message is displayed exactly as entered and the terminal alarm is sounded. After taking the action requested in the PLEASE message, type any one key to continue the BATCH program. To make a two-line PLEASE command, the first line can be terminated with an AND and the second line begun with another PLEASE. This lets the operator know more message is to follow.

For example:

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

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

If a PLEASE command is given when in a non-BATCH mode, following the CR, the bell rings and the system waits for a key to be typed.

2.2.6 RUN

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

SYSGEN	To change or check system configuration
BUILD	To build data files
SORT	To sort data files
UPDATE	To update data files
PIP	To move information between physical devices.
COMP	To compile a user source program into the binary program.

The RUN command has the form:

```
.RU [pronam] [+chain1+...+chainn] [,filnam1,...,filnamn] [/xx]
-
```

where

pronam is the name of the system program to be run or pronam+chain1... are binary files which are part of a program and constitute one large program broken up into several chains. For example:

```
.RUN PROG+CHAIN1+CHAIN2+CHAIN3
-
```

would execute program PROG. PROG would then determine whether programs CHAIN1, CHAIN2 or CHAIN3 would next be run. See section 1.4.4 for a description of the CHAIN statement. If the program name is omitted, Monitor loads and executes the last compiled DIBOL program from the compiler's scratch area.

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

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

/xx — is one or a combination of the option switches including /T which enables the TRACE feature and /D which enables the DDT mode.

Other options are described with the programs to which they apply. Refer to the appropriate chapter for more detailed information on these programs.

If the program specified is not found, an error message is produced.

The RUN command is the only Monitor command which can be abbreviated to one character, R.

Examples:

Command	Explanation
.RU -	executes previously compiled DIBOL program from the Compiler's scratch area.
.RU JOB1 -	runs program called JOB1
.RUN COMP, CHECK	this command compiles the source program CHECK, prints a compiler listing and temporarily saves the binary on the system device.
.R BUILD, FD, IO,FILE1,FILE2 -	this command builds a data file in the format specified in the Field Descriptor (FD) and I/O section (IO) of the BUILD control program with the data in FILE1 and FILE2.

MOUNT Messages

The Monitor outputs MOUNT messages to the terminal whenever an input or output unit must be specified. These messages have the form:

```
MOUNT xxxxxx #nn FOR INPUT:

MOUNT xxxxxx #nn FOR OUTPUT:
```

where

xxxxxx — is the name of the data file desired by the program currently executing.

#nn — is a sequence number from 1 to 63.

Answer the MOUNT message with the appropriate logical COS unit (1-15). If an error is made in the reply, type CTRL/U and the correct reply.

The MOUNT message is displayed (and the sequence number incremented) whenever the program reaches the end of the device and more information remains to be read or written.

When default units specified in control programs are not available, a question mark precedes the MOUNT message, i.e.,

?MOUNT xxxxxx #01 FOR INPUT:

The message

REPLACE xxxxxx #nn ?

is displayed when a file is already stored on the logical unit specified to a MOUNT message. Answer REPLACE with a Y and the CR key to replace the old file or any other key to keep the old file. File labels beginning with any character other than A-Z, [,↑, or] are considered to be temporary files, and no REPLACE message is displayed when such files are about to be written over. The following error messages can occur:

IN USE
ILLEGAL UNIT

NOTE

Occasionally, if output is to a scratch unit, a garbled filnam or sequence number in the "REPLACE" message may be typed. This is because the area on the unit where the label should be is simply random characters. Just answer "YES" to the REPLACE message as usual.

2.2.7 SAVE

This command stores the binary program from the compiler scratch area on the named device. The form of the SAVE command is:

```
.SAVE pronam[,dev][/Y]
```

where

pronam	is the name to be assigned to the compiler's binary output.
dev	is any mass storage device DT0-DT7 or DK0-DK3 which has a directory. If no device is given, the system device is assumed.
/Y	is used to bypass the REPLACE? message and YES response when a duplicate name would be encountered. Normally used in a BATCH mode to bypass operator response.

The SAVEd program must be moved to the system device before it can be executed.

2.3 EDITING COMMANDS

The COS Monitor contains a line editor; that is, all insertions, changes, and deletions are done with line numbers. All text input to the Monitor is assigned a sequence of line numbers. For example:

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

The commands:

```
ERASE
FETCH
LIST
Line Number
Number Commands
RESEQUENCE
WRITE
```

apply to the editing functions. These commands can be entered at any time in response to the dot displayed by the Monitor. Only the first two characters of the command need be typed and all commands must be terminated by typing the CR key.

2.3.1 ERASE

This command erases text from the edit buffer (that portion of core where the Monitor stores text being edited). The form of the command is:

```
.ER [n1][,n2]
-
```

where

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

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

Examples:

```
.ER       clears the entire edit buffer.
-
.ER 5     clears line 5.
-
.ER ,5    clears from the start of the buffer to line 5.
-
.ER 5,10  clears from line 5 to line 10, inclusive.
-
```

2.3.2 FETCH

The FETCH command loads the named source file into core from the specified device. The form of the FETCH command is:

```
.FE pronam[,dev]
-
```

where

pronam	is a previously created program which is to be brought into core.
dev	is the 3-character designation of the physical device, DT0-DT7 or DK0-DK3 where the program is stored. dev may be separated from pronam by a space or comma. If device is omitted, the system device is assumed.

Monitor searches for the named program. If the program is not found, Monitor displays the message

FILE NOT FOUND

Retype the command with the correct program name or device.

If another program is run while this program is in the active work area, Monitor temporarily stores this file in the editing scratch area on systems device, then returns it to core when program execution is complete.

Examples:

.FE RICH -	Moves program RICH from the system device to the edit buffer.
.FE PAYROL,DT2 -	Moves program PAYROL from DECtape unit 2 to the edit buffer.
.RUN XYZ -	Runs a program called XYZ. On completion of this program, the PAYROL source file is in the edit scratch area.

The FETCH command erases the edit buffer before inserting the FETCHed program.

2.3.3 LIST

The LIST command outputs the specified lines or the entire edit buffer to the high-speed punch, line printer, or terminal.

The LIST command has the form:

```
.LI [n1][,n2][/x]
-
```

where

n1 is the line number of the line to be listed or the first line number of two which delimit the lines to be listed. If omitted, listing starts at the beginning of the text buffer.

n2 is the second of two line numbers indicating where listing ends. If omitted, but the comma is included, listing continues to the end of the edit buffer.

/x is the one letter code which indicates the output device

- /L line printer
- /P high speed punch (line numbers are omitted)

If no device is indicated, the list is output to the terminal.

If no line numbers are specified, the entire edit buffer is output to the specified device. Type CTRL/O to stop output from an LI command.

Examples

.LI /L -	List entire edit buffer on line printer.
.LI 5 -	List line 5 on the terminal.
.LI ,5 -	List from start of buffer to line 5 on the terminal.
.LI 5,/P -	List from line 5 to the end of the buffer on the high-speed punch.
.LI 5,10 -	List lines 5-10, inclusive, on the terminal.

2.3.4 Line Number

The Line Number command automatically outputs line numbers at specified increments so new lines can be entered without typing each line number.

The Line Number command has the form:

```
.LN [n][,inc][/x]
```

where

n	is the starting line number. If no starting line number is specified, 100 is assumed. If the comma after the starting number is omitted, the starting number and increment number are the same, i.e., LN 100 causes a starting number and an increment of 100. If the command is LN 100, the start line number is 100 and the increment remains unchanged unless the Monitor is read back into core; at which time the increment returns to 10.
inc	is the increment between line numbers. If no increment is specified, 10 is assumed.
/x	is the input device. /R high-speed reader (All rubouts are ignored so that paper tapes with TABS followed by RUBOUTs will be read correctly.) /C card reader /T low-speed paper tape reader. If no device is specified, the terminal is assumed. Line numbers are specified by the LN command and cannot be part of the text read from the input device.

If Line Number mode is terminated and some editing has been done, type the line number command, LN, with no arguments to display the next number in sequence. When inputting from the card reader, all 80 columns are stored (40 columns with mark-sense cards).

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

The maximum number of characters on a line, including the line number and space, is 120. In automatic line number mode the line number and space are counted as two characters. When the terminal line is full,

Monitor executes carriage return/line feed and since this is a continued line does not display the next line number.

If the 120 characters limit is exceeded, the Monitor gives the error message LINE TOO LONG and the line is lost.

If a CR is the first character typed, no line is entered and line number increments. To obtain a blank line, type space and CR.

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

Type CTRL/Z to indicate the end of input and halt the automatic line numbers. When input is coming into the Editor from the card reader, a beep sounds to indicate no more cards. Load more cards into the reader and type any character to continue the input or type CTRL/Z to indicate the end of input.

Examples:

.LN — Requests line numbers starting at 100 with increments of 10. The terminal is assumed to be an input device.

The terminal display would be

```
.0100
```

Type the first line of the program and the CR key. Editor displays the next line number each time the CR key is typed.

```
.0100 START
.0110
```

.LN 10,5/R — Requests line numbers starting at 10 with increments of 5 to be assigned to input from the high-speed reader.

.LN ,100 — Request line number starting at 100 (default) with increment of 100. Input is from the terminal.

.LN 10/C — Requests line number starting at 10 with increment of 10 (default). Input is from card reader.

If an error is made when using automatic line numbers, type RUBOUT or CTRL/U, RUBOUT erases the last character typed and echoes the erased character. CTRL/U erases the entire line. Monitor redisplays the line number. CTRL/C acts the same as CTRL/U.

If the edit buffer is full, the error message TEXT AREA FULL appears, and the last line entered is lost. The edit buffer must be separated into two or more source files. This is done by:

1. WRiting the edit buffer as file B.

2. ERasing the last half of the edit buffer.

3. WRiting the edit buffer as file A.

4. FEtching file B.

5. ERasing the first half of the edit buffer.

6. WRiting the edit buffer as file B.

2.3.5 Number Commands

Any line beginning with a number inserts the line number and text in the edit buffer.

The form is:

```
     nnnn
or
     nnnn text
```

where

nnnn	is a line number.
text	is the data to be inserted in the buffer. The data must be separated from the line number by a space or tab. (If tab is used, it becomes the first character of text.)

If text is already at that line number, the new text replaces it. Files are stored in increasing line number order and a new line number is inserted in numerical order. If a CR is typed right after the line number, data at that line number is cleared from the buffer.

The input line is limited to 120 characters including the line number which counts up to four characters.

Examples:

```
.40            erases text and number at line 40
.45 RECORD     inserts RECORD at line 45
```

Text before editing:

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

Editing commands:

```
.35  PROC 1
.40  INIT(1,IN,'LABEL',2)
.47
```

Text after editing:

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

2.3.6 RESEQUENCE

The RESEQUENCE command renumbers the program lines to adjust for additions and deletions.

The form of the command is:

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

where

- n is the starting line number. If no starting line number is specified, 100 is assumed.
- , If the comma after the starting number is omitted, the starting number and increment number are the same, i.e., RE 100 causes a starting number and increment of 100. If the command is RE 100, the starting line number is 100 and the increment remains unchanged unless the Monitor is read back into core; at which time the increment returns to 10.
- inc is the increment between line numbers. If no increment is specified, 10 is assumed.

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

Examples:

- .RE Resequences line numbers of program in the edit buffer using 100 as starting line number and 10 as increment.
- .RE 10,5 Resequences line numbers of program in edit buffer using 10 as starting line number and 5 as the increment.
- .RE ,100 Resequences line numbers of program in edit buffer using 100 or the last specified line number as the starting line and 100 as the increment.
- .RE 50, Resequences line numbers of program in edit buffer using 50 as the starting number and the increment remains unchanged.

2.3.7 WRITE

The WRITE command stores a new or modified source program on the specified device so it can later be compiled or FEtched for editing.

The form of the command is:

```
.WR pronam[,dev][/Y]
-
```

where

pronam	is the six character name assigned to the program to be stored.
dev	is the three character designation of the physical device (DT0 - DT7, DK0 - DK3) where the program is to be stored. If no device is specified, the system device is assumed.
/Y	is used to bypass the REPLACE? message and YES response when a duplicate name would be encountered. Normally used in a BATCH mode to bypass operator response.

If the name specified is a duplicate name, Monitor asks

REPLACE?

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

2.4 EDITING EXAMPLE

```
.FE  ACC08S
-
.LI            ;LIST PROGRAM TO BE EDITED
-

0010    START    ;ACC08S -
0020    ;THIS PROGRAM UPDATES A MASTER FILE OF STOCK RECORDS, WITH A
0030    ;TRANSACTION FILE OF RECEIVALS AND WITHDRAWALS.
0040    ;A NEW MASTER FILE AND A REPORT ARE PRODUCED.
0050    ;
0060    ;
0070    ;
0080             RECORD  MASTI    ;MASTER INPUT RECORD
0090    MIPART,  D9
0100    MICOST,  D10      ;UNIT COST
0110    MIQTY,   D10
0120    MIDATE,  D6       ;DATE OF LAST TRANSACTION (MMDDYY)
0130             BLOCK,X
0140    MIWHOL,  A35
0150    ;
0160             RECORD  MASTO    ;MASTER OUTPUT RECORD
0170    MOPART,  D9
0200    MOCOST,  D10
0210    MOQTY,   D10
0220    MODATE,  D6
0230    ;
0240             RECORD  TRAN     ;TRANSACTION FILE
0250    TYPE,    A1               ;TYPE (R=RECEIVAL, W=WITHDRAWAL)
0260    TPART,   D9       ;PART NUMBER
0270    TQTY,    D7       ;QUANTITY
0280    TDATE,   D6       ;DATE (MMDDYY)
0290    ;
.
.
.
0500    END

                          ;Editing
.150
.230
.290
.0090   MIPART,  D9       ;PART NUMBER
.0110   MIQTY,   D10      ;QUANTITY ON HAND
.0228            BLOCK,X  ;REDEFINITION
.0229   MOWHOL,  A35
```

```
.LI                                                    ;LIST EDITED PROGRAM

0010    START   ;ACC08S
0020    ;THIS PROGRAM UPDATES A MASTER FILE OF STOCK RECORDS, WITH A
0030    ;TRANSACTION FILE OF RECEIVALS AND WITHDRAWALS.
0040    ;A NEW MASTER FILE AND A REPORT ARE PRODUCED.
0050    ;
0060    ;
0070    ;
0080            RECORD  MASTI   ;MASTER INPUT RECORD
0090    MIPART, D9      ;PART NUMBER
0100    MICOST, D10     ;UNIT COST
0110    MIQTY,  D10     ;QUANTITY ON HAND
0120    MIDATE, D6      ;DATE OF LAST TRANSACTION (MMDDYY)
0130            BLOCK,X
0140    MIWHOL, A35
0160            RECORD  MASTO   ;MASTER OUTPUT RECORD
0170    MOPART, D9
0200    MOCOST, D10
0210    MOQTY,  D10
0220    MODATE, D6
0228            BLOCK,X         ;REDEFINITION
0229    MOWHOL, A35
0240            RECORD  TRAN    ;TRANSACTION FILE
0250    TYPE,   A1      ;TYPE (R=RECEIVAL, W=WITHDRAWAL)
0260    TPART,  D9      ;PART NUMBER
0270    TQTY,   D7      ;QUANTITY
0280    TDATE,  D6      ;DATE (MMDDYY)
.
.
.
0500    END
```

2.5 BATCH EDITING

As the following example illustrates, editing can be done via a BATCH program.

```
.ERASE
-
.LN 5,5/R                          ;AUTOMATIC LINE NUMBERS START AT
-                                  ;5 AND INCREMENT BY 5.  INPUT FROM
                                   ;PAPER TAPE READER.
.LIST                              ;LIST THE FILE READ IN.
-
0005 FETCH TEMP
0010 LIST
0015 ERASE 4
0020 ERASE 30,34
0025 35 PROC 1
0030 40 INIT(1,IN,'LABEL',2)       ;BATCH Program
0035 47 FORMS (6,0)
0040 RE 5,5
0045 LIST
0050 WRITE TEMPA
0055 DELETE FILE/S

.WRITE FILE                        ;WRITE BATCH PROGRAM CALLED "FILE".
-
.BATCH FILE                        ;RUN BATCH PROGRAM "FILE"
-

FETCH TEMP                         ;FIRST COMMAND OF BATCH PROGRAM LOADS
                                   ;TEMP FOR EDITING
LIST                               ; SECOND COMMAND LISTS TEMP
0000 RECORD A
0004           ,D1
0005           ,A5
0010           ,A6
0015           ,D6
0020           ,D10
0025           ,A25
0030           ,D2
0031 RECORD B                        ;TEMP PROGRAM
0032           ,D2
0033           ,A1
0034           ,D2,      70
0035 PROC
0040           INIT(1,V,IN)
0045           XMIT (1,A,EOF)
0046           XMIT (6,A)
0047           XMIT (6,B)
0050 EOF,      FINI (1)
0055           STOP
0060           END
```

```
ERASE 4                          ;THIRD COMMAND ERASES LINE 4.
ERASE 30,34                      ;FOURTH COMMAND ERASES LINES 30-34.
35 PROC 1                        ;FIFTH COMMAND INSERTS NEW TEXT
                                 ;AT LINE 35.
40 INIT(1,IN,'LABEL',2)          ;SIXTH COMMAND INSERTS NEW TEXT
                                 ;AT LINE 40.
47 FORMS (6,0)                   ;SEVENTH COMMAND INSERTS NEW TEXT
                                 ;AT LINE 47.
RE 5,5                           ;EIGHTH COMMAND RESEQUENCES LINE
                                 ;NUMBERS STARTING AT 5 AND
                                 ;INCREMENTING BY 5.
LIST                             ;NINTH COMMAND LISTS EDITED TEMP.
0005 RECORD A
0010           ,A5
0015           ,A6
0020           ,D6
0025           ,D10
0030           ,A25
0035 PROC 1
0040 INIT(1,IN,'LABEL',2)
0045           XMIT (1,A,EOF)
0050           XMIT (6,A)
0055 FORMS (6,0)
0060 EOF,      FINI (1)
0065           STOP
0070           END

WRITE TEMPA                      ;STORE TEMPA ON SYSTEM DEVICE
DELETE FILE/S                    ;DELETE SOURCE FILE OF BATCH
                                 ;PROGRAM

.
-
```

2.6 ERROR MESSAGES

Monitor Errors

Message	Explanation
BAD COMPILATION	User tried to SAVE a compiled binary that had errors.
BAD DIRECTORY	Directory does not look right (for example, requesting a directory of a data tape).
BAD LABEL	Tape has no label, or its form is incorrect.
DT n?	The expected tape is not available on a DECtape drive. For example, if following a MOUNT message the operator has specified that a file should be written on unit 3, and unit 3 is not selected, or is WRITE LOCKed, this message is output. CTRL commands are not operative at this point, i.e., CTRL/C does not work. To recover, set switch to ENABLE the unit or set the dial selector to the proper number. Then type any key to continue. On LINCtapes there is no message.
ENTER "DATE MM/DD/YY"	Typed an unrecognizable date.
ERROR IN COMMAND	Miscellaneous.
FILE NOT FOUND	After, for example, FETCH FILEX.
LINE TOO LONG	Greater than 120 characters.
LINE # TOO LARGE	Greater than 4095.
REPLACE?	Duplicate file names. Type Y to replace.
TEXT AREA FULL	Greater than 8,150 characters.

?ILLEGAL UNIT	The unit specified is not DK0-DK3 or DT0-DT7; for example,
	
	`WRITE FILE,XY3`
	`WRITE FILE, RK9`

!*!MEMORY FAIL!*!	A hardware error has occurred. Restart the program.

Message	Explanation
?NO FILE TO SAVE	Nothing in the edit work area when WRITE command is issued.
*NO INIT DONE	Program attempted to read or write on a device that was not opened by the system program.

Run-Time Error Messages

BAD DIGIT	A character other than +, -, space, or the digits 0-9 was encountered in an alpha to decimal conversion.
*BAD PROGRAM	Attempting to run a binary program which contains a compilation error. Check compilation listing for error flags. Correct flagged errors, and recompile.
*DIBOL FILE NUMBER IN USE	In INIT, the channel number is already associated with a device.
*DIBOL FILE NUMBER NOT INITED	An attempt was made to XMIT, READ, or WRITE with a channel number that was not associated with a device.
END OF FILE	The last record of an input file has been read and the end of file mark encountered, but no EOF label was specified in the XMIT statement.
*I/O ERROR ON xx, RETRY?	System failed in three attempts to read from or write to a device. xx = DT or DK. Type Y if retry is desired; any other character if no retry and return to Monitor is desired. In either case, type only one character.
*ILLEGAL DEVICE	Attempt to WRITE on a file that was not INITialized for UPDATE or attempt to READ from a file that was not INITialized for INPUT or UPDATE.
*ILLEGAL SUBSTRING	A DIBOL PROCedure section statement has attempted to access a data field, F1 (m,n), but m=0 or m>n.
ILLEGAL RECORD #	Record number is 0, past the end of the physical unit or the length specified in the record header word does not match the length of the XMIT block (records in data file are all not the same length).

Message	Meaning
LINE TOO LONG	An input line (record) overflowed the block it was read into.
*NO BUFFERS LEFT	Not enough core available for I/O buffers. An I/O buffer of some multiple of 512 characters is set up for each active mass storage file. Another possibility: too few files were specified in the PROC statement.
NO FILE	No file specified in RUN statement to satisfy INIT (SYS) command.
NUMBER TOO LONG	A decimal field longer than 15 digits was used in a calculation.
*PROGRAM TOO BIG	Binary program does not fit in available core. Reduce program size.
*PUSHDOWN OVERFLOW	Either (1) a statement is too complex or (2) subroutines are nested to a depth greater than 50, or a combination of the two.
*RETURN WITHOUT CALL	The program tried to execute a RETURN, but there was no place to go: there was no corresponding CALL statement.
*SUBSCRIPT TOO BIG	Program attempted to destroy the run-time system or itself by using a subscript larger than that defined in the Data Definition section. Note that the run-time system does not detect all illegal subscripts; only those which would cause the user's program or the system to be destroyed.
ZERO DIVISOR	The program attempted to divide by zero.

NOTES

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

2. AT LINE nnnn is displayed under all run-time error messages. nnnn is a DIBOL source program line number in which the error occurred. If COMP/O were specified for this program, nnnn is meaningless.

CHAPTER 3

SYSGEN

SYSGEN is a conversational program used to reconfigure the software system to handle the hardware available, change the logical unit assignments and print a table of those assignments for reference.

3.1 OPERATING PROCEDURES

To load SYSGEN type:

```
.RU SYSGEN[, filnam]
-
```

or

```
.RU SYSGEN/x
-
```

where

filnam — is optional and is a file previously created and stored with the Editor which contains the table of logical unit assignments to be made.

/x — is one of three options used with SYSGEN.

- /C change the hardware handlers in the system.
- /T take new logical unit assignments from terminal.
- /L print table of current logical unit assignments.

If no file or options are specified SYSGEN takes the information stored in the edit buffer as the new logical unit assignments.

Refer to the appropriate section for further explanation of reconfiguration and logical unit assignments.

3.2 SYSTEM SOFTWARE CONFIGURATION

As shipped, the COS software system is set up to handle the following hardware configuration on the PDP-8:

TC08 with 8 DECtapes (or TD8E for the 8/E)
RK8E, RK08 or RF08 disks
Terminal
LP08 80-column Line Printer
Card Reader
High-speed Paper Tape Reader and Punch

or

LINCtape (PDP-12)
RK8E, RK08 or RF08 disks
LP08 80-column Line Printer
Terminal
Card Reader
High-speed Paper Tape Reader and Punch

on the PDP-12.

NOTE

No change is necessary if your COS hardware configuration matches the one described here.

If, however, the system has either a 132 column line printer or an Analex line printer or the system device is to be changed, type the command

```
.RU SYSGEN/C
```

and the CR key. (Refer to Figure 3-1 for a flow chart of the question and answer sequence.)

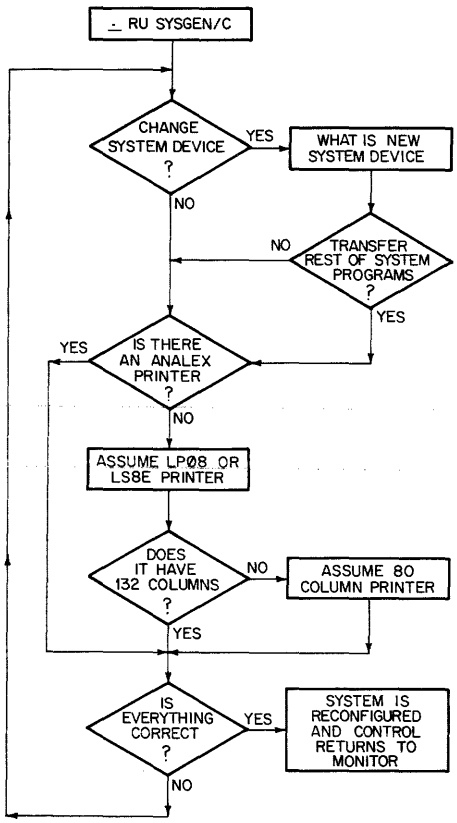


Figure 3-1. SYSGEN Flow Chart

SYSGEN displays the following questions on the terminal:

DO YOU WANT TO CHANGE THE SYSTEM DEVICE?

Type NO and the CR key to keep the current system device. SYSGEN skips to the questions on the line printer.

Type YES and the CR key to change the system device from DECtape to Disk or Disk to DECtape. This is the only way to move the Monitor and DIBOL run time system (RSYS). If the system device is changed, the system must be restarted with the bootstrap appropriate to that device (see Appendix B or Chapter 9, BOOT). SYSGEN then asks:

WHAT IS THE NEW SYSTEM DEVICE?

Type the three character designation of the new system device, DK0-DK3 or DT0-DT7.

NOTE

Since only an RK08, or RK8E, or RF08 disk may be resident on the system, disks are commonly addressed as DK0-DK3. An RF08 disk must always be addressed as DK0.

DO YOU WANT TO TRANSFER YOUR FILES?

Answer YES and the CR key to move the rest of the system programs (PIP, COMP, etc.) to the new system device. Transfer of the system programs destroys anything stored on the new system device. Answering NO and the CR key will zero the directory of the new device.

DO YOU HAVE AN ANELEX PRINTER?

NOTE

The Anelex printer must be configured into the system with SYSGEN before it is turned on or it will not function properly.

Answer YES and the CR key and the next question is

IS EVERYTHING CORRECT?

Answer NO and the CR key and SYSGEN assumes an LP08 or LS8E printer and the question

132 COLUMN PRINTER?

is displayed. Answer YES and the CR key if there is a 132 column line printer. Answer NO and the CR key and SYSGEN assumes an 80 column line printer. The next question is:

IS EVERYTHING CORRECT?

Check all answers made to the above questions. Type YES and the CR key if the answers are correct. Type NO and the CR key to correct any errors. SYSGEN repeats the questions starting at the beginning.

SYSGEN/C does not reset the logical unit assignments to reflect the new area occupied by the system on a disk. Run SYSGEN/T to reassign logical units if the system is moved to a disk or the system may be destroyed by a data file.

3.3 LOGICAL UNIT ASSIGNMENT

The assignment of logical units to physical mass storage devices provides greater utilization of the storage area and a certain device independence. The device independence is available at run time; any mass storage device can be specified for I/O and the program executes using the specified devices.

The COS system handles storage using the following hierarchy

```
2 characters = 1 word
256 words    = 1 block
16 blocks    = 1 segment
```

Refer to sections 3.3.1 and 3.3.2 for background on logical unit assignment before proceeding. Devices not currently part of the system configuration can not be assigned a logical unit.

To make logical unit assignments from the terminal, type

```
.RU SYSGEN/T
```

and the CR key. Type the 3 character device code, a comma and length in segments in the order they are to be assigned to the logical units. The comma and length in segments can be omitted and SYSGEN assigns all the space left on the specified device to the next logical unit (DECtape = 46 segments; RK08 Disk, 202 segments; RK8E, 404 segments; RF08 disk, 64 segments times the number of platters). The first device entered is automatically assigned to logical unit 1, the second to unit 2, etc. For example,

```
.RU  SYSGEN/T
DT1,46
DT2,46
DT3,46
DT4,46
DT5,46
DT6,46
DT7,46
DK0,46
DK0,46
DK0      ;logical unit 10 is 110 segments (assuming DK is
         ;the RK08 disk and DECtape is the systems device)
```

assigns DT1 to logical unit 1 with a length of 46 segments, DT2 to logical unit 2, etc. Logical units 11-15 are left unassigned. Be sure to follow the format shown in the example. If an error is made when typing the table, type CTRL/C then retype the RUN SYSGEN/T command and start the table over again.

Logical assignment can also be accomplished by typing the table into the Monitor. This method allows error correction with the Monitor commands. If the following logical unit assignments are typed to the Monitor,

```
0010  DT1
0015  DT2
0020  DT3
0025  DT4
0030  DT5
0040  DT6
0050  DT7
0060  DK0,46
0070  DK0,46
```

and the command

```
.RUN SYSGEN
-
```

and the CR key is typed, SYSGEN makes the logical unit assignments according to the table in the edit buffer. Logical units 1-7 are DECtape units 1-7 with a length of 46 segments and logical units 8 and 9 are portions of RK08 disk (DK0), each 46 segments long. A listing of the SYSGEN assignments is always printed on the line printer upon successful completion of the SYSGEN program.

The table created with the Monitor can also be stored with the WRITE command for future use. Use the command

```
.RU SYSGEN, file
-
```

to assign the logical units with a previously created file. For example

```
.RU SYSGEN,TABL1
-
```

where TABL1 contains

```
0010  DT1
0020  DT2
0030  DT3
0040  DT4
0050  DT5
0060  DT6
0070  DT7
0080  DK1,202
0090  DK2,101
0100  DK2,101
```

Occasionally, it is desirable to check the logical device assignment table. To obtain a printout of the table, type

```
.RUN SYSGEN/L
-
```

and the SYSGEN table of the current logical device assignments is printed on the line printer. For example:

LOGICAL UNIT #	DEVICE NAME	LENGTH (SEGMENTS)
01	DT1	0046
02	DT2	0046
03	DT3	0046
04	DT4	0046
05	DT5	0046
06	DT6	0046
07	-UNDEFINED-	
08	-UNDEFINED-	
09	-UNDEFINED-	
10	-UNDEFINED-	
11	-UNDEFINED-	
12	-UNDEFINED-	
13	-UNDEFINED-	
14	-UNDEFINED-	
15	-UNDEFINED-	

This table should be made available for reference by all persons using the COS system.

Physical mass storage devices are limited by the hardware; but can be subdivided into logical units. Logical units are composed of segments; the number of segments per unit is up to the user. COS allows one file per logical unit.

The COS system is shipped with logical units 1-7 assigned to DECtape units 1-7 and each unit is originally assigned 46 segments. Logical units 8-15 are left unassigned. The user may assign and reassign any of the 1-15 logical units as desired. Logical unit 0 is always the system device and cannot be reassigned.

It is advisable to create logical units of varied sizes since file lengths vary and a short file in a long logical unit wastes storage.

3.3.1 DECtape Users

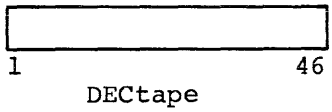
Although more than one logical unit can be assigned to a physical DECtape unit, it is advisable to assign one logical unit per DECtape. The use of two or more units per DECtape may cause excessive tape spinning; for example, if reading from one unit and writing to another located on the same physical device. In the same example, if the output unit filled and had to be replaced, it might be necessary to dismount the tape, thereby also removing the needed input unit.

A DECtape is 46 segments long so the logical unit assigned to the physical DECtape unit should be assigned 46 segments.

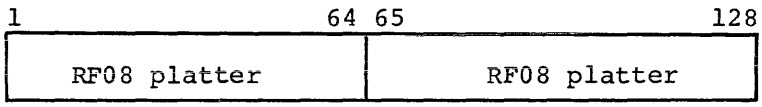
3.3.2 Disk Users

The easiest method of assigning logical units to a disk is to think in terms of a sequence of DECtapes.

A DECtape is 46 segments long.

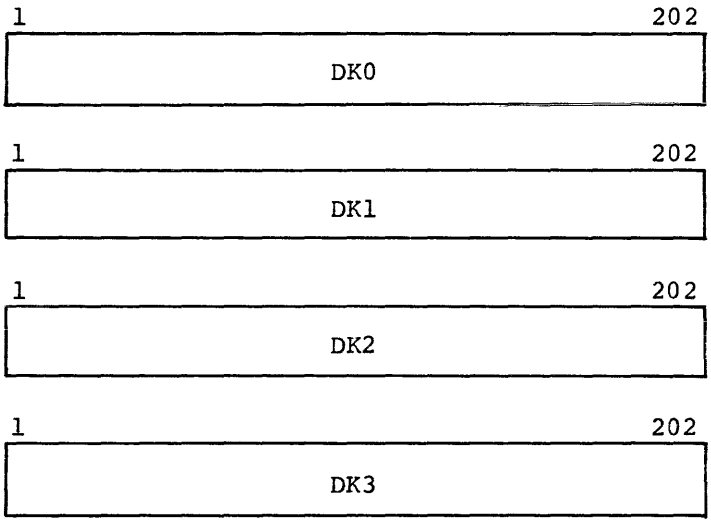


A RF08 platter is 64 segments; and a second platter adds another 64 segments.



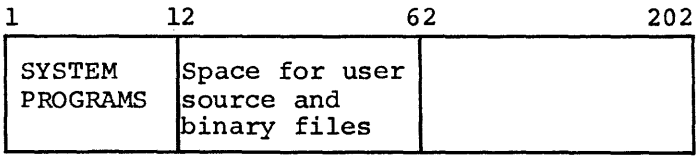
The second platter is treated as a continuation of the first.

The RK08 disk cartridges are each 202 segments and there can be up to four drives (0-3).



The RK8E disk cartridges are each 404 segments and there can be up to four drives (0-3). If the disk is the system device, approximately 200 blocks (or 12 segments) must be left unassigned to hold the operating system and system programs. In addition, space should be left on the system device to store source programs, control programs and binary programs. The remainder of drive 0 may be used for this purpose. However, fifty segments are suggested which leaves approximately 140 segments (342 for RK8E) for logical unit assignments. Logical units on the system device are automatically assigned at the end of that device (the opposite is true of all other devices). For example:

DK0 (assuming RK08)

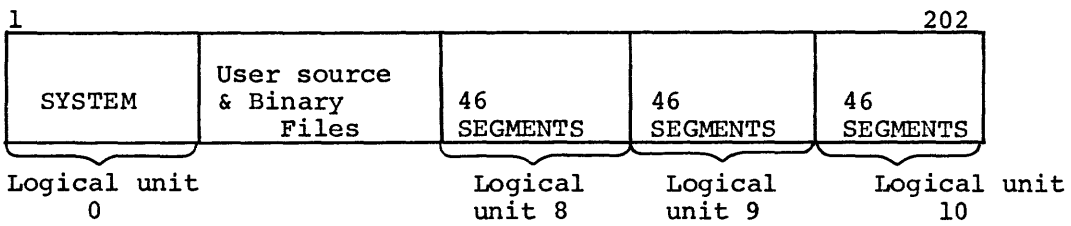


Logical Unit 0

which leaves approximately 140 segments for logical unit assignments.

A sample assignment might be

DK0 (assuming RK08)



The area between logical unit 8 and the system can be left unassigned for system program overflow, if desired. On devices other than the system device SYSGEN starts assigning logical units at the beginning of the device.

3.4 ERROR MESSAGES

Message	Explanation
# TOO LARGE	Number entered was greater than 4095.
ALL UNITS ASSIGNED	Attempt to assign more than 15 units.
BAD SWITCH	Not /T, /C, or /L.
ILLEGAL DEVICE	Device other than DT0-DT7 or DK0-DK3.
INSUFFICIENT SPACE ON DEVICE	Attempt to allocate more segments than there are available on a device.
NOT ENOUGH ROOM FOR SYSTEM AND FILES	Device specified too small to accommodate system programs and files.
SYNTAX ERROR	Missing comma, extra characters, etc.

CHAPTER 4

COMP

The Compiler takes a DIBOL language program and converts it to a binary language program which can be executed by the run time system (RSYS). In the process of creating the executable program, the Compiler outputs a listing of the source program and a storage map of the records and fields which are used by the program.

The Compiler sets aside storage space for the constants, variables and statements used by the program. Refer to Appendix C for an explanation of the amount of code generated by the Compiler.

Be sure to document any new programs completely and accurately and to prepare a set of instructions for execution including any necessary changes to logical device assignments, names of files, and where each is to be mounted, the format of any entries to be made to the program, restart procedures and any other essential information.

4.1 OPERATING PROCEDURES

The command to run the Compiler has the form:

```
.RU COMP[, file1,...,file7][/xx]
-
```

and the CR key. (Be sure the line printer is turned on.)

Where

file1,...,file7 — are files on the system device to be compiled into one binary. These files contain segments of one program, not several programs. If no files are specified, the program in the edit scratch area is compiled.

/xx — is one or a combination of the following options:

- /N suppresses the output of the Compiler listing and storage maps.
- /G compiles the source program and if no errors are detected, executes the binary program with no user intervention. /N is implied. The message LOADING is displayed when compiling is successfully completed. The program must have an END statement to be compiled and executed with the /G option, if INIT SYS is used in the program.
- /1-7 specifies a physical unit (DECtape only) for temporary storage during compilation. The unit specified should contain a scratch tape. File is recopied on the system device

when compilation is complete. Use of temporary storage units minimize tape motion during compilation.

/T enables the TRACE function and implies the /G option. For example,

```
RUN COMP/T2
```

/D performs an implied /G option and then chains to the DDT (DIBOL Debugging Technique) mode which is explained in Appendix H.

/O compiles the source program in less than the amount of memory space required by the other options, at the expense of eliminating the TRACE feature (if stated in the source program) and accurate error reporting. Execution speed of the compiled program is increased by as much as 20%. This option can be combined with either the /N, /G, /D, /T, or /1-7 option.

Use the SAVE command to store the binary program prepared by the Compiler.

To calculate how much memory space is saved by using the /O option, the following rules apply:

1. Saves 1 location for each executable statement.

2. Saves 1 additional location for each label.

3. Costs 1 location for each ON ERROR statement. (However, ON ERROR statements are executable.)

Since the TRACE statement in a program compiled with the /O option has no useful effect, and since any run-time error reported in such a program will be reported to have occurred in the PROC statement, it is suggested that the /O option only be used on those programs which have been thoroughly debugged.

Unless the /N or /G option is specified in the RUN command, the Compiler produces a listing of the source program (Figure 4-1) and a storage map (Figure 4-2) on the line printer or the device specified in the START, PROC or END statements. Most errors detected by the Compiler are indicated by underscoring the number of the line where the error exists and inserting a caret ( ) pointing in most cases to the error. Other errors are listed in the symbol map. These errors must be corrected before an attempt is made to execute the program.

The number of errors detected by the compiler is displayed on the terminal as

```
nn ERRORS
```

```
COS DIBOL    01-MAR-73 THUR    COMPILATION LISTING     V12I28  PAGE 01
            DATA DIVISION   S Q U A R E  R O O T  S U B R O U T I N E

0010              START   ; S Q U A R E  R O O T  S U B R O U T I N E
0020
0030              RECORD PRINT
0040     A,       A15
0050     S,       A15
0060
0070              RECORD
0080     X,       D15
0090     SQRTX,   D15
0100     TEMPX,   D15
0110     N,       D3,001
0120
```

FIGURE 4-1. COMPILER LISTING
(sheet 1 of 2)

```
COS DIBOL    01-MAR-73 THUR    COMPILATION LISTING     V12I28  PAGE 02
    PROCEDURE  DIVISION

0130     PROC 1
0140
0150               INIT(1,LPT)
0160     LOOP,     X=N*1000000
0170               CALL SQRT
0180               A=N
0190               S = SQRTX,'XXX'.XXX'
0200               INCR N
0210               XMIT(1,PRINT)
0220               IF (N,LE,99) GOTO LOOP
0230               STOP
0240
0250     SQRT,     TEMPX=X/2                          ;TRIAL VALUE
0260     SQLOOP,   SQRTX=(TEMPX+(X/TEMPX))/2          ;NEWTON'S METHOD
0270               IF(SQRTX-TEMPX ,EQ, 0) RETURN
0280               TEMPX=SQRTX
0290               GOTO SQLOOP
0300     END
```

FIGURE 4-1. COMPILER LISTING
(sheet 2 of 2)

```
COS DIBOL    01-MAR-73 THUR     STORAGE MAP LISTING      V12I28  PAGE 03
#         NAME       TYPE        DIM  SIZE     ORIGIN

0001      PRINT      RECORD       01     32      20000
0002      A          ALPHA        01     15      20002
0003      S          ALPHA        01     15      20021
0004      X          DECMAL       01     15      20042
0005      SQRTX      DECMAL       01     15      20061
0006      TEMPX      DECMAL       01     15      20100
0007      N          DECMAL       01     03      20117
0010      ,,1        DECMAL       01     01      20122
0011      LOOP       LABEL        00     01      10071
0012      ,,1000     DECMAL       01     07      20123
0013      SQRT       LABEL        00     01      10132
0014      ,,XXX,     ALPHA        01     07      20132
0015      ,,99       DECMAL       01     02      20141
0016      ,,2        DECMAL       01     01      20143
0017      SQLOOP     LABEL        00     01      10141
0020      ,,0        DECMAL       01     01      20144

0016 SYMBOLS

NO ERRORS DETECTED,  08 K CORE REQUIRED [3940 FREE LOCS = 14 BUFFERS]
```

FIGURE 4-2. STORAGE MAP LISTING

The storage map lists the names of all the records, fields, and labels as they were processed by the Compiler. The information is arranged in six columns as follows:

this column contains the internal number of the symbol named in column 2 and may be ignored.

NAME Column 2 gives the name of any symbol (field name, record name, program label) or constant used in the compiled program. Constants are decimal constants or alpha characters which appear in the PROC section of the source program. Only the first six characters of a symbol name are used; any characters after the sixth are ignored. Only the first four characters of a constant are used and each constant is preceded by two periods (..) to distinguish it from a symbol. Constants with four characters or less appear only once on the symbol map even though they may occur more than once in the program. Constants with more than four characters are listed each time they occur in the program. Record literals begin with a "..".

TYPE Column 3 states the symbol type. The types are:

- ALPHA Symbol is an alpha field name or an alpha constant (character string).
- DECMAL Symbol is a decimal field name or decimal constant.
- RECORD Symbol is a record name or record literal.
- LABEL Symbol is a program label.
- REDEF Symbol is multiply defined (redefined). All attempts at definition after the first are flagged as errors in the source listing.
- UNDEF*** Symbol is a label referenced by the program but not defined. For example, GO TO TAG1 in a program where TAG1 does not appear as a label.

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

DIM Column 4 contains the array dimension (number of fields) of the alpha or decimal symbols. The column is meaningless for other types of symbols.

SIZE Column 5 lists the size of the symbol in characters. The size of a RECORD symbol is the number of the characters of all its symbols plus 2.

ORIGIN Column 6 gives the 15-bit (octal) byte address where the symbol appears in core and for most users should be ignored. Octal addresses are shown for labels.

The number of symbols used, number of errors detected, core required and free locations are listed at the bottom of the symbol map.

Maximum number of symbols allowed in an 8K system is 365; in 12K or larger systems, 511.

4.2 CONDITIONAL COMPILATION PROCEDURE

The Conditional Compilation Procedure (CCP) is a feature which permits the DIBOL programmer to include statements in a source program which will be compiled by the compiler only if the programmer elects for those statements to be compiled.

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

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

Notice that the left angle bracket is followed by a control variable, in this case PROMPT. Until the control variable is turned "on" somewhere in the program before the left angled bracket is encountered by the compiler, those statements between the angled brackets will be ignored. Notice too, that the right angled brackets appears on a line by itself. The command to turn on a control variable is as follows:

```
=control variable
```

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

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

Conditional compilation can also be used, for example, to conditionalize debugging statements into a source file. Once the program has been tested, the control variable can be shut off by deleting the command to turn it on, effectively removing the debugging routines from the source.

CCP can also allow you to combine several similar (but not identical) source programs into one source.

If the control variable used in a CCP statement is undefined, the compiler will automatically set aside space for it. This, however, is wasteful of space and it is to the programmer's advantage to use, for CCP symbols, symbols that are already being used for some other purpose.

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

If a CCP variable is used in the middle of a record definition (in the record section of a DIBOL program) it must be a variable that has previously been defined, because if not, the compiler will allocate space for the variable in the middle of your record (where not desired).

CCP sections may be nested to any depth. Any sections inside a CCP section that is off will be completely ignored by the compiler. This includes other CCP statements. To indicate that certain statements are being ignored (conditionalized out), the compiler listing will not print the line number for that statement. There must be a matching > for each < used. If this condition is not met, the compiler generates a CCP ERROR message. This error is fatal if angled brackets do not match by the end of the program.

4.3 ERROR MESSAGES

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

The Compiler error messages are:

Message	Explanation
BAD ALPHA VALUE	Initial value in an alpha data specification statement did not begin or end with a single quotation mark.
BAD DECIMAL VALUE	The initial value for a decimal data specification was incorrectly formed.
BAD PROC #	The number in a PROC statement was not a digit from 0 to 7.
BAD RELATIONAL	An illegal relational occurs in an IF statement. For example, a .GX. instead of a .GT.

Message	Explanation
CCP ERROR	Matching angled bracket (< or >) missing.
COMMA MISSING	No comma appeared where one was expected.
DATA INITIALIZATION MISSING	No data initialization followed a comma in a data specification statement.
EXPECTED LABEL IS MISSING	A required label is missing.
EXPRESSION NOT ALLOWED	A complex expression or bad character occurs to the left of an = or where only a variable is allowed.
EXTRA CHARS AT STMNT END	Extra characters occur at the end of a legal statement.
FIELD TOO LARGE OR 0	In a data description statement, the dimension was 0 or more than 3 digits long, or the field size was 0 or larger than 511.
ILLEGAL OPERATOR	A bad character was encountered in an expression where an operator would be expected.
ILLEGAL STMNT	The statement was not a data manipulation statement (it had no =) nor did it start with a recognizable keyword.
INITIAL VALUE WRONG SIZE	The initial value in a data specification statement had a length different from the field size specified.
LABEL NOT ALLOWED	A symbol in an expression was not of type alpha or decimal, or a record or a symbol which had been redefined was used.
MISSING CLOSE PAREN	No close parenthesis occurred where one was expected.
MISSING OPEN PAREN	No open parenthesis occurred where one was expected.
MISSING OPERAND	A binary operator occurs in an expression with no operand following it; or no expression at all occurs where one is expected.
MISSING OR BAD DEVICE	The device in an INIT statement was missing or started with an illegal character.

Message	Meaning
MISSING QUOTE	The statement contained an odd number of quotes (').
MISSING RELATIONAL	No relational appeared in an IF statement.
NAME PREVIOUSLY DEFINED	The name used was previously defined and this statement tries to redefine it.
NOT A OR D	A character other than A or D occurred in a data specification statement where A or D was expected.
NOT LABEL	A symbol which was not a 'label' occurred where a label was required.
PROGRAM TOO BIG	Binary output too big for the binary scratch area. Remedy: Run PIP with /n option, (refer to Chapter 5).
RECORD TOO BIG	A named record exceeded 510 characters in size.
STMNT TOO COMPLEX	The statement generated too much code and overflowed the Compiler's code buffer, or it had too much nesting and overflowed the Compiler's push-down stack. Remedy: break up the statement into smaller parts.
SUBSCRIPT ERROR	No comma or close parenthesis occurs after a subscript.
SUBSCRIPT NOT DECIMAL	The type of a subscript was not decimal.
TOO MANY SYMBOLS!	A fatal error message. Only 365 symbols allowed in symbol table in 8K system, and only 511 symbols allowed in larger systems. The compiler stops compiling; no storage map can be produced.
TOO MUCH DATA	Program's Data division exceeds 32K bytes.
UNDEFINED NAME	A name is used which was never defined in the data section.
WRONG DATA TYPE	Mixed modes occurred in an expression; or an argument which was supposed to be decimal was not or one of the three arguments in a data manipulation statement was of the wrong type.

CHAPTER 5

PIP

PIP (Peripheral Interchange Program) handles file tranfers between two logical devices, i.e., moves data, source and binary files, and system programs between two file-oriented devices; copies the contents of one device to another, and consolidates files to remove free blocks. PIP can also be used to allocate more space to the binary scratch area.

5.1 OPERATING PROCEDURES

Type the Monitor command:

```
.RU PIP[/n]
```

and the CR key to load PIP. /n is optional and is a number 0-9 which indicates the number of segments to allocate to the binary scratch area. The /n switch is used in conjunction with the E option. Refer to section 5.1.4 for further information.

PIP responds to the RUN command with:

```
PIP 2.1108 (or current version number)
OPT-
```

requesting the type of transfer and I/O devices. Answer with one of the options, B, C, D, E, S, V, X, described on the following pages. After the option is specified, PIP displays the IN and OUT questions requesting the input and output files. If an incorrect character is entered when responding to the OPT, IN or OUT questions, type CTRL/U and the correct response. Type CTRL/C to return to the Monitor.

5.1.1 Transfer Binary File

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

Answer IN with the name of the binary program to be transferred and, optionally, a comma, and the input device, DT0-DT7, DK0-DK3. If no device is specified, the system device is assumed.

Answer OUT with the name to be assigned to the output file, and, optionally, a comma, and the output device, DT0-DT7, DK0-DK3. If no device is specified, the system device is assumed.

If an attempt is made to transfer to or from a non-file-oriented device, the IN or OUT message is repeated.

Example:

```
OPT-  B
IN-   TEST,DT1
OUT-  TEST,DK1
OPT-            (CTRL/C typed here)
.
```

5.1.2 Copy Device

Type C in reply to OPT- to copy the contents of an entire device onto another similar device, i.e., disk to disk or DECtape to DECtape.

Answer IN with the three-character physical device, DT0-DT7, or DK0-DK3.

Answer OUT with one of the physical devices as explained for IN.

Example:

```
OPT-  C
IN-   DT0
OUT-  DT7
OPT-          (X or CTRL/C typed here)
.
```

5.1.3 Transfer Data Files

Type D to move data files between the devices specified after IN and OUT. Answer IN with a label (up to six characters, first character must be alphanumeric), an optional logical unit number (1-15) preceded by a slash or comma, or a device switch:

```
/R  for paper tape reader
/C  for card reader
/K  for keyboard
/F  BCD paper tape in high speed reader
```

If /R is specified as the input device, records stored are variable length. If /C, the record produced is the same size as the card even if a smaller number of columns are punched.

If a label is specified without a logical unit number or if both are specified and a data file of another name is found, PIP asks

```
MOUNT  xxxxxx  #01 FOR INPUT:
```

answer with the appropriate logical unit number (1-15). Answer OUT with a label, number or device as described for IN. Output devices are:

```
/L  line printer
/P  paper tape punch
/T  terminal
```

If a label is specified without a default unit specification, the message

```
MOUNT  xxxxxx  #01 FOR OUTPUT:
```

is displayed. Answer with the appropriate logical unit number (1-15).

If a file with a different name already exists, Monitor asks

```
REPLACE  xxxxxx#nn?
```

Type Y and the CR key to destroy old file and replace with new. Type N and the CR key to return to the OPT- question. If output device specified is a logical unit (1-15) and the unit is filled, an error message is produced.

When at the end of the input file or card deck, PIP asks

```
MORE?
```

Answer with the CR key if there is no more input, or specify another label, logical unit or device.

NOTE

PIP transfers data in alphanumeric mode only. A negative number is treated as the letter which has the equivalent code. (Refer to Appendix A, COS Character Codes.)

Examples:

		Explanation
OPT-	D	
IN-	EMPNAM	Dump EMPNAM
MOUNT	EMPNAM #01 FOR INPUT:	file on logical
1		unit 1 on the
OUT-	/L	line printer.
MORE?	N	CTRL/C is typed to
OPT-		return to the
.		Monitor.
OPT-	D	
IN-	HRPAY,2	Combines 2
OUT-	PAYFIL/1	data files into
MORE?	Y	one output
IN-	SALPAY,3	file.
MORE?	N	CTRL/C is typed
OPT-		to return to the
.		Monitor.
OPT-	D	Input from
IN-	/C	card reader which
OUT-	LIST1	creates a
MOUNT	LIST1 #01 FOR OUTPUT:	multi-unit
10		file.
MORE?	Y	
IN-	/C	
MOUNT	LIST1 #02 FOR OUTPUT:	Logical unit 10
11		is full.
MORE?	N	
OPT-		CTRL/C is typed
.		to return to the
		Monitor.

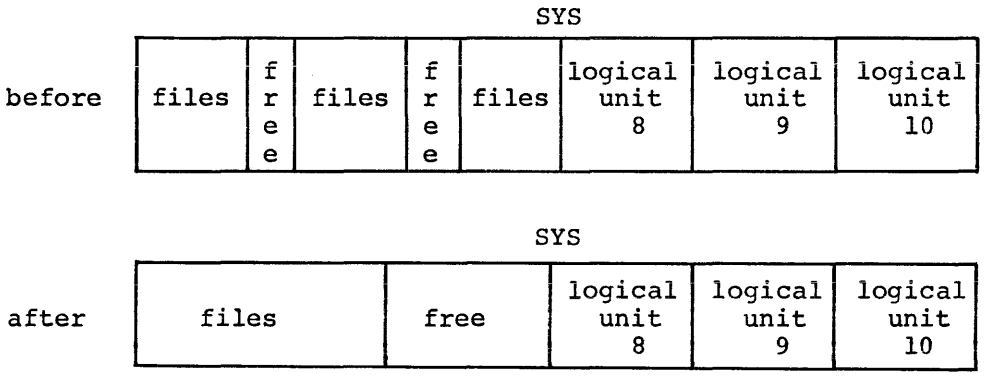
5.1.4 Eliminate Spaces in Directory

Type E to consolidate the free blocks of the file on the input device, and store the consolidated files on the output device. In addition, it is possible to eliminate one or two of the types of files (source, binary, and system) during the consolidation. These free blocks are shown in the file directory and occur when a file is deleted from the directory. The Monitor is not copied.

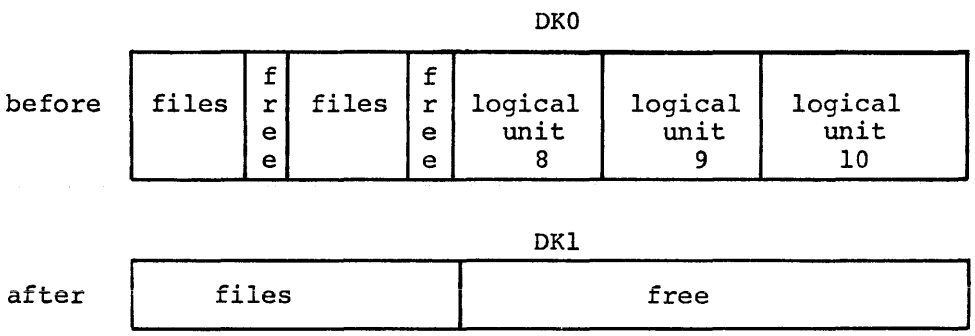
Answer IN with DT0-DT7, or DK0-DK3 and the CR key.

Answer OUT with DT0-DT7, or DK0-DK3 and the CR key.

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



However when consolidating a device (other than the system device) onto another device PIP eliminates the free space as follows



Example:

```
OPT-  E
IN-   DT0
OUT-  DT0
```

The message

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

is displayed. Answer with the CR key if all files are to be transferred, or answer by typing one or a combination of two of the characters S, V, B separated by a comma and followed by the CR key if one or two of the types of files are to be eliminated.

For example:

```
.R PIP)
PIP 2.1108
OPT- E)
IN- DT0)
OUT- DK0)
TYPES OF FILES TO BE SKIPPED (S,V,B): S )
```

This would consolidate free space and eliminate source files located on DT0 and store the consolidated files on DK0 assuming there was a directory on DK0.

NOTE

All control keys are ignored until the option E operation is completed.

Binary Scratch Area Modification:

As mentioned in section 5.1, the E option can be used in conjunction with /n to change the size of the binary scratch area. The binary scratch area, normally two segments (32 blocks) long, is adequate for most applications but extremely large DIBOL programs (containing many statements) may need more space.

A typical sequence for expanding the binary scratch area is:

```
.R PIP/2)
PIP 2.1108
OPT- E)
IN- DT0)
OUT- DK0)
TYPES OF FILES TO BE SKIPPED (S,V,B): )
```

This would allocate 64 blocks to the binary scratch area on DK0, doubling its original size.

In general,

1. If the output device is not the input device, the binary scratch area on the output device equals 2+n segments.

2. If the output device is the input device, the binary scratch area is set either to the size of the current area or to 2+n, whichever is less. That is, compressing a device onto itself can shrink the binary scratch area, if 2+n is less than the size of the original area. The binary scratch area cannot be expanded if a device is being compressed onto itself since that would require writing over existing files.

3. If no /n is specified, the size of the binary scratch area on the input device is assumed.

5.1.5 Transfer Source Files

Type S to transfer source files between two file-oriented devices. Answer IN with the name of the source file to be transferred and, optionally, a comma, and the output device DT0-DT7, or DK0-DK3. If no device is specified the system device is assumed.

Answer OUT with the name to be assigned to the output file and, optionally, a comma, and the output device, DT0-DT7, or DK0-DK3. If no device is specified, the system device is assumed.

If an attempt is made to transfer to or from a non-file-oriented device, the IN or OUT message is repeated.

Example:

```
OPT-  S )
IN-   TEST,DT1 )
OUT-  TEST,DK1 )
OPT-            (X or CTRL/C typed here)
.
```

5.1.6 Transfer System Program

Type V to move a system program between two file-oriented devices. Answer IN with the name of the system program to be transferred and, optionally, a comma, and the input device, DT0-DT7 or DK0-DK3. If no device is specified, the system device is assumed.

Answer OUT with the name to be assigned to the output file, and, optionally, a comma, and the output device, DT0-DT7 or DK0-DK3. If no device is specified, the system device is assumed. If an attempt is made to transfer to or from a non-file-oriented device the IN or OUT message is repeated.

Example:

```
OPT-  V )
IN-   SORT,DT1 )
OUT-  SORT,DT4 )
OPT-            (X or CTRL/C typed here)
.
```

```
OPT- X
```

5.1.7 Return to Monitor

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

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

5.2 ERROR MESSAGES

Message	Explanation
BAD DIRECTORY	Attempt to reference or store a file on a device with no directory (or a directory that has been destroyed).
ILLEGAL DEVICE SWITCH	A switch was specified that was not /R, /C or /K for input or /P, /L or /T for output.
NO ROOM	Attempt to store a file on a full device.

CHAPTER 6

BUILD

BUILD is a utility program used to create data files containing fixed length records. Creating data files requires careful consideration of their proposed usage. (Designing BUILD records is discussed in Appendix D.) BUILD allows input from the keyboard, card reader, high-speed paper tape reader, or system device.

A BUILD control program must be written before BUILD can be run. This control program provides the field description and order for input and output of the data to be used to create the file. Be sure to prepare a set of instructions on input preparation so data input will be according to specifications in the control program.

During execution, BUILD checks each data input line to be sure the data entered conforms exactly to specifications in the control program. If no errors are found, the input line is written on the new data file and maybe printed on the line printer. If an error is detected, the record is not written to the data file.

BUILD outputs a listing of the control file and data records created on the line printer, if desired. A summary of the number of records written, number of records not written and any hash totals kept is printed on the line printer for each file created.

6.1 BUILD CONTROL PROGRAM

The BUILD control program is created and written on a mass storage device with editing commands.

This program conveniently falls into three sections, Field Descriptor, INPUT, and OUTPUT. Briefly, the Field Descriptor section defines all the fields to be used, the type and size and any initial values to be assigned.

The INPUT section describes the order in which the data will be received. The input format specified in this section must be followed when preparing data for input. The OUTPUT section describes the order of the output of data. There may be up to seven output formats for one set of input so several different output files can be created with one input file.

Figure 6-1 is an example of a simple BUILD control program which is explained in detail in the following sections.

Errors in the BUILD control program cause a return to the Monitor. Spaces and tabs may be inserted anywhere in the program for readability. As with any COS program, comments inserted on a line must be preceded by a semicolon.

The control program may be stored as two separate source programs: the Field Descriptor section as one and the INPUT and OUTPUT sections as the other.

```
.LN 10,10                ;AUTOMATIC LINE NUMBER COMMAND,
                         ;START AT 10, INCREMENT OF 10
.0010  DEFINE
.0020  F1,D6             ;PART NUMBER
.0030  F2,A30            ;PART NAME
.0040  F3,D7             ;VENDOR NUMBER
.0050  F4,D6             ;DATE
.0060  F5,D6             ;WHOLESALE PRICE
.0070  T1,D10            ;TEMPORARY STORAGE
.0080  C1,D2,25          ;CONSTANT FIELD
.0090  C2,D6,500000      ;CONSTANT FIELD
.0100  INPUT CDR         ;INPUT FROM CARD READER
.0110  PART              ;KEYWORD
.0120  F1                ;PART NUMBER
.0130  F2                ;PART NAME
.0140  F5                ;WHOLESALE PRICE
.0150  DATE              ;KEYWORD
.0160  F4                ;DATE
.0170  VENDOR 1          ;KEYWORD
.0180  F3                ;VENDOR NUMBER
.0190  OUTPUT 1 PRTFIL   ;OUTPUT TO PRTFIL FILE
.0200  F3 C              ;VENDOR NUMBER
.0210  F1 C              ;PART NUMBER
.0220  F2                ;PART NAME
.0230  F5                ;WHOLESALE PRICE
.0240  F4                ;DATE
.0250  END
```

Sample input lines for above control program

```
:PART  126789  'HEX BOLT' 200
:DATE  30772
:VENDOR  1476116
:PART  176117  WASHER 100
:VENDOR  7346177
:PART  176118
:VENDOR  3177615
:PART  126788  '#6 MACHINE SCREW' 300
:VENDOR  2061311
```

Figure 6-1. Sample BUILD Control Program

Records created according to control program in Figure 6-1 would look like:

```
1476116126789HEX BOLT       22 spaces      000200030772
7346177176117WASHER         24 spaces      000100030772
```

6.1.1 Field Descriptor Section

The Field Descriptor section specifies all fields to be referenced in later sections of the Control program. It gives their type (alpha or decimal), size (1-510) and initial values, if any.

The format for this section is

```
DEFINE
fieldx, typesize, initial value
.
.
.
```

where

DEFINE — is the section heading and must be the first statement in the program.

fieldx — is an F, T, or C and a number. The F, T or C indicates data fields (F), temporary fields (T), and constant fields (C). These names are for the convenience of the user and are treated equally by BUILD. All F fields must be listed first, all T fields second, and all C fields last. The number (x) must be a positive, non-zero decimal number not greater than 511 and may begin at any number. Numbers may be skipped but within a field group (F, T or C) must be in ascending order. Temporary fields are used to accumulate hash totals. Constant fields are used to enter a default value or range check.

Remember, however, that F1 is assigned to the first field, F2 to the second, etc. If the field names assigned are, for example,

```
F1
F5
F6
```

Blank fields will be left in core. References to F2 would cause an error.

typesize — is A for alphanumeric fields or D for decimal fields. Use A to describe fields which contain other than numbers. Size is an unsigned decimal number in the range 1-510.

initial value — Initial values can be assigned to any field. Alphanumeric values consist of any sequence of legal COS characters (excluding single quotation marks) and are enclosed in single quotation marks. Initial decimal values consist of a sequence of digits optionally preceded by a + or - sign or followed by a - sign. These initial values must be of the same size and type as the field for which they are specified. The sign and quotation marks do not count in the size of the initial value. If an initial value is not specified, the field is initially set to 0 if decimal or blank if alpha.

Remember, there is a difference between decimal and alpha constants consisting of the same digits and if a constant is to be referenced as alpha and decimal, it must be defined both ways. For example,

```
C1,D3,234
C2,A3,'234'
```

Field description statements must be entered on consecutive lines and each line is terminated with the CR key. Use the editing commands to create this file.

Example:

```
.0010  DEFINE            ;STARTS BUILD CONTROL PROGRAM
.0020  F1,D6             ;PART # FIELD, DECIMAL, 6 DIGITS
.0030  F2,A30            ;PART NAME FIELD, ALPHANUMERIC, 30 CHARS
.0040  F3,D7             ;VENDOR # FIELD, DECIMAL, 7 DIGITS
.0050  F4,D6             ;DATE FIELD, DECIMAL, 6 DIGITS
.0060  F5,D6             ;WHOLESALE PRICE
.0070  T1,D10            ;TEMPORARY FIELD
.0080  C1,D2,25          ;CONSTANT FIELD CONTAINING VALUE OF 25
.0090  C2,D6,500000      ;CONSTANT FIELD CONTAINING VALUE OF 500000
```

6.1.2 INPUT Section

The INPUT section specifies the device (i.e., card reader, keyboard, etc.) and arrangements of the data to be input. The format of the INPUT section is

```
INPUT dev
keyword, output format#
fieldx
.
.
.
```

where

INPUT — is the section header and must be the first statement of the INPUT section.

dev — is the 3 character designation for the input device:

KBD - terminal keyboard
CDR - card reader
RDR - high-speed paper tape reader
SYS - a file on the system device

In the case of CDR, RDR and SYS, the input data is prepared in advance and stored on cards, paper tape, or the system device.

keyword — is the name assigned to a group of fields which will make up one input line to the BUILD program. Keyword is an alphanumeric character string of any length but only the first six characters are significant. Field names cannot be keywords. Keywords provide the connection between the data entered at BUILD run time and the data format specified in the control program.

(comma) — is optional and specifies comma as the field delimiter on input lines. This allows the alphanumeric data input for that keyword to contain embedded spaces and tabs. In most cases, however, spaces are suggested as delimiters.

For example,

if control program specifies

```
PART,
```

then the input line is

```
PART,126789,HEX BOLT,200
```

However, if the control program specifies

```
PART
```

then the input line is

```
PART 126788 '#6 MACHINE SCREW' 300
```

output format# — is a number (1-4 for 8K systems or 1-7 for 12K systems) which specifies the output format to be used for this input line. This number corresponds to an OUTPUT section number (described in section 6.1.3). If an output format number is not specified, there is no output for fields under this keyword. If an output # is specified, no output takes place until input for the corresponding keyword fields is completed.

fieldx — is one of the previously described fields (F, T, or C) and a number. These fields must be listed in the order in which they will be input.

To enter the BUILD input line without keywords, specify an asterisk (*) in place of a keyword. For example,

```
INPUT  CDR
*1
F1
F2
```

All field descriptions beginning with the * must be listed before those beginning with keywords.

NOTE

> BUILD input lines without keywords must be entered in the exact order specified in the INPUT section of the control program because there are no keywords to relate an input line to the fields described in the control program.

Data is inserted in * fields in order of input.

The INPUT section is terminated by the OUTPUT heading for the next section of the control program.

Example:

```
.0010  DEFINE
.0020  F1,D6
.0030  F2,A30
.0040  F3,D7
.0050  F4,D6
.0060  F3,D6
.0070  T1,D10
.0080  C1,D2,25
.0090  C2,D6,500000
.0100  INPUT CDR          ;INPUT IS FROM CARD READER
.0110  PART               ;1ST INPUT LINE HAS KEYWORD PART
.0120  F1                 ;1ST FIELD OF PART LINE IS PART NUMBER
.0130  F2                 ;2ND FIELD OF PART LINE IS PART NAME
.0140  F5                 ;3RD FIELD OF PART LINE IS WHOLESALE PRICE
.0150  DATE               ;2ND INPUT LINE HAS KEYWORD DATE
.0160  F4                 ;1ST FIELD OF DATE LINE IS DATE
.0170  VENDOR 1           ;3RD INPUT LINE HAS KEYWORD VENDOR
                          ;AND ALL INPUT IS TO BE OUTPUT
                          ;ACCORDING TO FORMAT 1
                          ;IF VENDOR IS MISSING; NO OUTPUT OCCURS
.0180  F3                 ;1ST FIELD IS VENDOR NUMBER
```

OPTIONS

The INPUT section of the control program also allows assignment of default fields, skips, or errors for missing fields, flag fields, range checks, hash total fields, and checkdigits.

These options may be specified for any field and have the order and format shown:

fieldx[SKIP / =dfldx / ERROR] [flag] [RANGE([lfldx],[ufldx])] [HASH(hfldx)] [CHECK]

Spaces may be used to separate the options for readability. Any combination of options can be used but they must appear in the order shown above.

Use one of the SKIP, default (dfldx) or ERROR options to specify an alternative for a missing field in a BUILD input line. SKIP causes the value previously stored in fieldx to be used again, assuming it was not cleared.

=dfldx assigns the current value of the default field (dfldx) to fieldx. The default field (usually a C field) must have the same type and size as fieldx. ERROR causes an error message on the terminal at BUILD run time. If none of these is specified, SKIP is assumed when fieldx is missing from the BUILD input line. A constant field (C) cannot be used as fieldx.

Use the flag option to set the flag field to 1 when fieldx is present and zero when not present. The flag field can be checked later in the user's DIBOL program. The flag field must be a decimal field.

Use the RANGE option to specify a high and low limit for the data in the destination field. The destination field and the fields specified as the limits must be decimal fields. The low limit field is specified first and must be less than or equal to the size of the destination field. If the low limit field is omitted, there is no lower limit. The high limit field is specified second and must be larger than or equal to the size of the destination field. If the high limit field is omitted, there is no high limit check. The value of the data entered at run time must be within the range specified.

Use the HASH option to request a hash total on the data in the destination field. The destination field and the hash total field (hfldx) must be decimal. Since the hash total field (usually a T field) is used to store an accumulating sum of the destination field data, it cannot be smaller than the destination field and should be larger. If the hash total exceeds the storage field , the high order (leftmost) digits are lost. The field used to store the hash total should not be used for any other purpose within the BUILD control program.

Use the CHECK option to request a checkdigit calculation on the data entered in the destination field. The destination field must be decimal. The checkdigit is the rightmost digit and is not stored. For example, if the field is a D6 and the number 1234561 is entered, the rightmost digit (1) is the checkdigit and is not stored. Using the same field size, D6, if the number 12345 is entered, 5 becomes the checkdigit. The checkdigit formula is calculated on 001234 and 5 is not stored. If the checkdigit is not valid, an error results. The formula for calculating the checkdigit is described in Appendix E.

Example:

```
0010   DEFINE
0020   F1,D6
0030   F2,A30
0040   F3,D7
0050   F4,D6
0060   F5,D6
0065   F6,D1
0070   T1,D15
0080   C1,D2,25
0090   C2,D6,500000
0100   INPUT CDR
0110   PART,                ;PART INPUT LINE HAS COMMA AS DELIMITER
0120   F1 ERROR             ;ERROR MESSAGE IF PART NO. NOT ENTERED
0130   F2 F6                ;F6=0 IF F2 NOT PRESENT
0140   F5 RANGE (C1,C2) HASH T1     ;WHOLESALE PRICE MUST BE
0150   DATE                 ;WITHIN RANGE SHOWN IN C1 AND C2. STORE
0160   F4                   ;HASH TOTAL IN T1
0170   VENDOR 1
0180   F3 CHECK             ;DO CHECKDIGIT ON VENDOR NUMBER
```

EXPLANATION

RANGE check on amount $.25 to $5,000.00 HASH total on amount.

If there is not a part number, output an error. Do a checkdigit on vendor number. Data input for keyword PART must be separated by commas.

6.1.3 OUTPUT Section

The OUTPUT section specifies the name and sequence of the data to be stored on a mass storage device. The OUTPUT section is composed of OUTPUT blocks, each of the following form

```
OUTPUT output format# label/default unit
fieldx
.
.
.
```

where

OUTPUT	is the heading of the last section of the control program and must appear even if no output is desired. Be sure that OUTPUT is not used as a keyword in the control program.
output format#	is the number (1-4 for 8K systems, 1-7 for 12K systems) which links a keyword in the INPUT section to a particular OUTPUT format. This number must be in ascending order (beginning with 1) and must appear after OUTPUT.

label	is the name to be assigned to the newly created data file. The name can be any unique sequence of characters (only the first six are significant).
default unit	is an optional entry specifying the logical unit (1-15) where the output file is located. (If the number specified exceeds 15, it is taken modulo 16.) If the file is not found on the default unit when the control program is executed, a MOUNT message is displayed.
fieldx	The F, T or C fields in the order in which they are to be output.

The OUTPUT section ends with the specification of another OUTPUT section (up to 4 for 8K, 7 for 12K) or an END statement.

If another OUTPUT section is specified, its format number must be the next number in sequence. Different OUTPUT format sections can contain the same fields so that several data files can be created from one input. All fields between OUTPUT and END or OUTPUT and another OUTPUT form one record; record length must not exceed 510 characters.

OPTIONS

The OUTPUT section of the control program has two options, C or +, which are specified as follows:

```
OUTPUT 1 datal
F10  C
T50  +
T75  +C5
```

Use C to clear a field after output of the record. Decimal fields are cleared to zeros, alpha fields to blanks. During input the contents of fields not cleared with C options can be repeated in following records by skipping fields in subsequent input lines.

Use + as a counter or sequence check. The field to be incremented is usually a temporary field (T) and must be decimal. This field is output as part of the record being created. + increments the field by one; to increment by any other number specify an increment field (+fieldxx). The increment field (usually a C field) must be decimal and must not be larger than the destination field. If the increment sum exceeds the destination field, the high order digit is lost. The C and + options cannot be used on the same field.

Example:

```
0010   DEFINE
0020   F1,D6
0030   F2,A30
0040   F3,D7
0050   F4,D6
0060   F5,D6
0070   T1,D10
0080   C1,D2,25
0090   C2,D6,500000
0100   INPUT CDR
0110   PART,
0120   F1 ERROR
0130   F2
0140   F5 RANGE (C1,C2) HASH T1
0150   DATE
0160   F4
0170   VENDOR 1
0180   F3 CHECK
0190   OUTPUT 1 PRTFIL    ;OUTPUT FORMAT 1; FILE NAME, PRTFIL
0200   F3 C               ;VENDOR NUMBER
0210   F1 C               ;PART NUMBER
0220   F2 C               ;PART NAME
0230   F5 C               ;WHOLESALE PRICE
0240   F4                 ;DATE
0250   END
```

EXPLANATION

Clears fields with "C" each time record is written.

6.1.4 Storing the BUILD Control Program

When the BUILD control file is complete, it should be stored on a mass storage device for future use. Use the command

```
.WRITE BLDSPC
```

for example, to store the control program.

6.2 OPERATING PROCEDURES

To load the BUILD program, type

```
. RUN BUILD[, file1,...,file7][/xx]
```

in reply to the dot displayed by the Monitor

where

file1,...,file7 are the BUILD control programs. If the BUILD control file specified the INPUT device as SYS, then the rest of the files are names of data files on the system device to be input. If no files are specified, the file in the edit buffer is used to create a master file.

Multiple files are concatenated and passed to the program as one large file.

/xx are optional switches which affect the listing printed by BUILD. The first switch is:

/L list the BUILD control program, or

/N do not list the control program.

The second switch is:

/L list the data records created

/N do not list the data records

If a second switch is not specified, a data records list is not produced. (/N is assumed.) If the no list condition is specified, errors encountered in the control program or data records are listed on the line printer as is the summary of hash totals, records created, etc.

Examples:

```
. RU BUILD,BLDSPC/NL
```

lists the data records and summary of totals but not the control program.

```
. RU BUILD,BLDSPC/L
```

lists the control program and summary of totals, while

```
. RU BUILD,BLDSPC/LL
```

outputs both listings and the summary.

When BUILD is loaded, the message

MOUNT xxxxxx #01 FOR OUTPUT:

is displayed. xxxxxx is the output file name specified in the control program and #01 is the sequence number.

Continuing the example, the message is

MOUNT PRTFIL #01 FOR OUTPUT:

Answer with the number (1-15) of the logical device where the output file is to be stored. When BUILD is complete, the Monitor DIRECTORY command can be used to obtain a label of the data file. The next steps depend on the device (KBD, CDR, RDR, SYS) specified in the INPUT statement of the control program. If the data output exceeds the storage capacity of the logical unit, the following message is displayed if default messages were not specified:

MOUNT xxxxxx #02 FOR OUTPUT:

where xxxxxx is the label of the output file. Mount the new output device if necessary. Type the number of the logical unit where the rest of the output is to be stored.

BUILD can be aborted at any time by typing CTRL/C, to return to the Monitor. All output is lost. The BUILD program must not be run if the line printer is inoperative.

6.2.1 KBD - Data Input from Keyboard

If the INPUT statement specifies KBD, BUILD responds to the RUN command with

DO YOU PREFER PROMPTING?

Answer YES and a CR key to have the input line keywords printed automatically by BUILD. To use prompting, the data input must consist of successive input lines whose keywords (if present) appear in exactly the order in which they are presented in the INPUT section of the control program with no omissions. Monitor displays

:

after each keyword to indicate it is ready for input.

Answer NO and a CR key to allow each input line keyword to be typed. BUILD displays a colon (:) when it is ready to accept input. Input lines can then be typed in any order and BUILD will match the input line keyword to the keyword in the INPUT section of the control program.

If the reply is YES or NO, the dialogue is over and data input can begin. BUILD assumes errors will be edited on line.

When all the input lines have been entered, type CTRL/Z to indicate the end of data input. BUILD closes the output files and prints number of records written, records not written and any hash totals requested and returns control to the Monitor.

6.2.2 SYS, CDR or RDR - Data Input from System Device, Card, or Papertape

If the INPUT statement specifies SYS, CDR or RDR,

The question

DO YOU WISH TO EDIT ON-LINE?

is displayed. Answer NO and the CR key and future errors are printed on the line printer and all fields in the bad input line will become undefined. Records with undefined fields are not written to the output file. They remain undefined unless cleared in the output section or redefined with subsequent input.

Answer YES and the CR key to have an error message displayed on the terminal whenever an error is encountered. BUILD displays the incorrect input line, the error message and the message

RETYPE:

Type the correct input line. For details on data input refer to Section 6.3. If just the CR key is typed, the field in error becomes undefined.

When BUILD reaches the end of the paper tape in the reader, it asks

MORE?

Answer NO and the CR key if there is no more input. Otherwise, put additional paper tape in the reader and type Y and the CR key to continue reading data.

When input is complete, BUILD closes the output data files and prints the number of records written and records not written and any hash totals requested and returns control to the Monitor.

When BUILD reaches the end of cards in the reader, the terminal alarm sounds. If there are more cards put them in card reader and type any key to resume operation. Otherwise type CTRL/Z to terminate input.

6.3 BUILD INPUT LINE

The data input line is free form, i.e., specific data fields need not be in prespecified column positions. However, the fields must be input in the same order within a keyword line as specified in the INPUT section. Data input lines entered at the keyboard may or may not contain keywords depending on the answer to the prompting question. Data input lines prepared on paper tape or cards must contain keywords, unless an asterisk was specified in the control program.

The following conventions must be observed when entering input lines.

1. The first field in an input line may be preceded by spaces or tabs if desired.

2. Each input line must be terminated with the CR key.

3. The fields, if more than one is entered per line, must be separated from each other by spaces, tabs, or commas. For example, an input line described in the control program as

```
DEFINE
  F1,D6
  F2,A30
  F5,D6
INPUT KBD
PART,
F1
F2
F5
```

can be entered as

```
PART,126789, HEX BOLT, 500
```

or if described as:

```
.
.
.
PART
F1
F2
F5
```

can be entered as

```
PART␣126789␣'HEX BOLT'␣500
```

4. A field can be omitted by actually omitting it if there are no fields to the right or by typing two consecutive commas if there are fields to the right. For example

```
PART,126789,HEX BOLT
PART,, HEX BOLT,100
```

Omitting a field may cause an error depending on the options (SKIP, default field or ERROR) specified for the field in the control program.

5. Decimal data must be entered in one of the following forms:

```
digits
+digits
-digits
digits-
```

The plus sign is optional and minus can be placed at the beginning or end of the data. Decimal data cannot have embedded blanks, i.e., 12 34 is illegal. The number of digits must be less than or equal to the field size specified in the control program in the corresponding field.

If the number entered is smaller than the field size, it is right justified and zeros are inserted on the left. If the number is too long, an error results.

6. Alphanumeric data must be entered in one of the following forms:

    'data' or data

    In the first form ('data') any legal COS character (including spaces and tabs) is permitted except a single quotation mark (apostrophe). BUILD treats tabs inside quotes as single spaces, but echoes them as backslashes on the terminals.

    In the second form any legal COS character except spaces, tabs, and commas is permitted. The first character, however, must not be a single quotation mark. If the keyword PART did not have a comma, then input would be

    PART␣175␣WASHER␣100

    The number of alphanumeric characters must be less than or equal to the size of the corresponding field specified in the control program. If the data is smaller than the field, it is left justified and padded with blanks. If the data is too large, an error results.

7. If comma was specified as the field delimiter in the INPUT section of the control program, then alphanumeric data not enclosed in single quotation marks may have embedded spaces and tabs but no commas and must be separated from the next field by a comma. For example, if the control program specified:

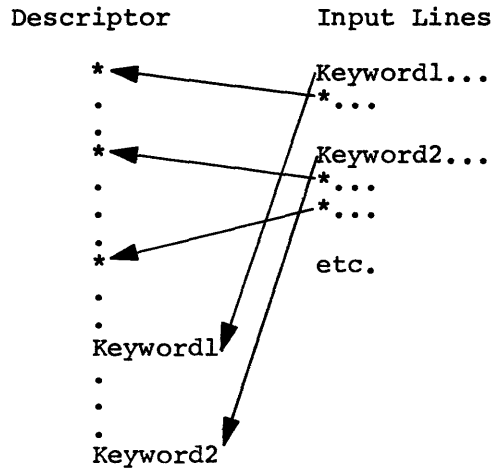
```
DEFINE
F1,A30
F2,A30
F3,A30
INPUT KBD
NAME,1
  F1
  F2
  F3
```

    the input line would be typed as

    NAME, JOHANSEN'S MARKET, 2 MAIN ST., 'MAYNARD, MASS.'

    Extra text at the end of a data record is illegal except for comments which must be preceded by a semicolon.

8. The arrangement of * and keyword records is immaterial at run time. BUILD inserts records in the asterisk fields by the order of input and in keyword records by matching keywords. For example,



There is a current asterisk record pointer in BUILD which keeps track of the next asterisk record to be filled. After inputting the last * record, BUILD goes back to the first one.

6.4 ERROR MESSAGES

BUILD error messages are output to the line printer as the errors are encountered.

BUILD checks each section of the control program and the input data for errors.

If editing is being done on line, the line in error, error message, and RETYPE message are displayed on the terminal. The keyword is displayed if prompting is in effect. Otherwise, the keyword must be retyped. If no input editing is being done, data input errors are logged on the line printer in the form

```
input line
****
error message
```

Failure to correct a field in error causes the field to become undefined. The record containing an undefined field is not written on a storage device.

Error messages followed by an explanation point (!) indicate fatal errors. The error messages for the Field Description section of the control program are:

Message	Meaning
BAD DIGIT IN DECIMAL INITIAL VALUE	Alpha character in a decimal initial value.
DATA TABLE OVERFLOW!	Too much data. Maximum = 960 characters.
DESCRIPTOR TABLE OVERFLOW!	Too many F, T and C fields defined. Maximum = 160.
EXTRA CHARS AT STMNT END	Characters not relating to statement appear with statement.
FIELD NUMBER MISSING OR 0	Field number or default unit number is missing, is 0 or greater than or equal to 512.
INITIAL ALPHA VALUE DOESN'T BEGIN WITH QUOTE	Beginning quotation mark missing for initial alpha value.
INITIAL VALUE TOO BIG	The initial value specified is larger than the field size.
INITIAL VALUE TOO SMALL	The initial value specified is smaller than the field size.
MISSING CLOSE QUOTE ON ALPHA INITIAL VALUE	Quotation mark not specified where necessary.
MISSING INITIAL VALUE	Comma was inserted after type and size but initial value was not specified.
NO COMMA AFTER FIELD NAME	No comma or a character other than comma was specified after the field name.
NOT A OR D	A character other than A or D occurred in a data specification statement where A or D was expected.
NOT BUILD FILE!	File did not start with DEFINE statement.
NOTHING AFTER FIELD NAME	Field type and size are not specified after field name and comma.
NUMBER REPEATED OR OUT OF ORDER	A field sequence number is used more than once or is out of ascending order sequence.

Error messages for the INPUT/OUTPUT sections of the control program are:

Message	Meaning
BAD END STATEMENT	Missing or misspelled END statement.
BAD DEFAULT FIELD	Default field does not agree with size or type of object field.
BAD DEPICTOR	OUTPUT format in error.
BAD DEVICE	Not KBD, SYS, RDR or CDR.
BAD FIELD NUMBER	Field number specified is greater than 2047.
BAD INPUT STATEMENT	Statement missing or misspelled.
BAD OPTION	Something other than C or + is specified in OUTPUT section.
BAD OUTPUT FORMAT #	Format number is wrong or out-of-range (1 or 7).
BAD RANGE CONSTRUCTION	In RANGE portion of statement an open or close parenthesis or comma is missing.
C-FIELD NOT PERMITTED	C-type fields cannot be specified in the INPUT section.
DESTINATION FIELD NOT DECIMAL	The destination field specified is not defined as decimal.
FIELD NOT DECIMAL	Field not defined as decimal.
HASH FIELD MISSING	No field specified for storage of hash total.
INCREMENT TOO BIG	Length of increment field bigger than length of output field.
MISSING FIELD NAME	Field name is missing.
OUT OF ROOM!	Too many descriptors in INPUT section. Maximum keywords, approximately 121. Approximate maximum fields for all keywords = 192. Too many fields in OUTPUT section. Approximate maximum = 128.
OUTPUT RECORD TOO BIG	Record exceeds 510 characters.

Message	Meaning
UNDEFINED FIELD	Field referenced was not defined in Field Descriptor section of control program or a decimal field was being equated to an alpha field, or an alpha field was specified for a flag.
UNDEFINED FORMAT	Format number referenced but not defined (error given on END statement).
UNEXPECTED END OF FILE	Missing END statement.

Error messages for data input are:

Message	Meaning
BAD ALPHA FIELD	Illegal entry in alpha field.
BAD CHECKDIGIT	Checkdigit calculated by BUILD does not match the one entered.
BAD DECIMAL FIELD	Illegal entry in decimal field.
BAD DELIM	Fields are separated by an illegal delimiter, i.e., by a ; or !.
BAD KEYWORD	Keyword entered not described in the control program.
EXTRA CHARS AT END OF DATA	More data was entered than was defined in the control program, i.e., 4 fields entered when only 3 were defined.
FIELD OUT OF RANGE	Data entered is not within the range specified in the control program.
NECESSARY FIELD MISSING	Field specified as necessary in control program was not entered.
TOO BIG	Data entered is greater than the size of the field as defined in the control program.

CHAPTER 7

SORT

SORT arranges COS data files in order according to a key specified in the SORT control program. The key consists of certain fields or parts of fields which appear in each record of the file being sorted. The data records can be sorted in ascending or descending order based on the key.

Before SORT can be run, a SORT control program must be written. The control program defines the fields of the records to be sorted, specifies labels for input and output files, and the key to be used in the sort.

SORT handles one file per run and there must be a separate SORT control program for each file sorted. Files to be sorted must contain fixed length records.

If a multi-reel file is to be sorted, the SORT program sorts each reel separately. A separate merge pass of the same SORT program must then be done to combine the file. To sort a multi-reel DECtape file on disk without going through a merge phase, PIP the data file onto disk before sorting. (See PIP option D.)

7.1 SORT CONTROL PROGRAM

The SORT control program is created and written on a mass storage device with editing commands. It consists of a file Descriptor section and an INPUT/OUTPUT section.

7.1.1 File Descriptor Section

The File Descriptor section has the form:

```
DEFINE
Fx,typesize
.
.
.
```

where:

DEFINE	is the section heading and should be the first statement in the program.

Fx	is the first field in the record to be sorted. All fields in the record to be sorted must be defined in the File Descriptor section in the order they appear in the record. The number (x) must be a positive, non-zero decimal number not greater than 511 and should begin at 1 with no numbers skipped.
typesize	is an A for alphanumeric fields or a D for decimal fields and an unsigned decimal number in the range 1 - 510.

File Descriptor statements must be entered on consecutive lines and each line is terminated with the CR key. Use the editing commands to create the control program.

NOTE

The BUILD Field Descriptor written previously to create the data file can be used as the SORT File Descriptor if the input and output fields are exactly the same and if it was stored as a separate program. T and C fields and initial values are ignored by SORT.

Example:

```
0010   DEFINE
0020   F1,D6              ;PART NUMBER
0030   F2,A30             ;PART NAME
0040   F3,D7              ;VENDOR NUMBER
0050   F4,D6              ;DATE
0060   F5,D6              ;WHOLESALE PRICE
```

This example is based on the file created with the BUILD control program. (See Chapter 6.)

7.1.2 INPUT/OUTPUT Section

This section specifies the input and output file labels and the number of work units available for the SORT.

These work units become part of the MOUNT message displayed during SORT run time.

Format of the INPUT/OUTPUT section of the control program is:

```
INPUT filnam/d,filnam/d
SORT n/w1,w2,w3
KEY fieldx(n,m)-,fieldx(n,m)...
OUTPUT filnam/d
END
```

where

INPUT filnam/d,filnam/d	is the name of the file containing the records to be sorted. If no name is specified, SORTIN is assumed. /d is optional and is the unit (1-15) to check for the named file. If the control program is used for a separate merge operation, the additional filnams are the files to be merged.
SORT n/w1,w2,w3	is the number (n) of logical units to be used as scratch areas during the sort. n is any number from 3 to 7. There must be at least three units to execute a sort. If the SORT statement is not present 4 units are assumed. It is recommended that the size of the scratch units be as large as one "reel" of the input file. /w1,w2,w3 are optional default work units to be used if available. Using default units bypasses the mount work unit message.
KEY fieldx(n,m)-,...	fieldx is the field name and number of the field to be used as the SORT key. (n,m) are optional and specify the nth through mth characters of the field. If no characters are specified, the entire field is used as a SORT key. - requests a SORT on that field in descending order. If - is not specified, ascending order is assumed (+ may also be used to indicate ascending order). Up to eight fields or parts of fields can be specified for the sort key. There is no restriction on the size of the sort field as long as the total size of the fields which comprise the key is not larger than 510 characters. The sort is done left to right on the key using the COS codes shown in Appendix A. The leftmost key is most significant, and the leftmost character in each key field is most significant for sorting purposes. n and m are unsigned integers with n less than or equal to m. Each m must be less than or equal to the length of the record.

OUTPUT filnam/d — is the data file name to be given to the sorted records. If this statement is missing SORT assigns the name SRTOUT to the output. For multi-reel files the names $TMP00 - $TMPnn are used. /d is the default unit to check for the output file. If the file is not there, the mount message is displayed.

NOTE

For a multi-reel input file, SORT sorts the first reel onto the output file and then asks for the second reel of input. For example, when sorting a 3-reel input file, the output files (which are to be input to MERGE) are called $TMP00, $TMP01, and $TMP02.

END — Terminates control program.

Example:

```
0010    DEFINE
0020    F1,D6
0030    F2,A30
0040    F3,D7
0050    F4,D6
0060    F5,D6
0070    INPUT PRTFIL        ;DATA FILE NAME
0080    SORT 4              ;WORK UNITS
0090    KEY F1-,F4          ;SORT KEYS = PART
                            ;NUMBER IN DESCENDING ORDER
                            ;DATE IN ASCENDING
                            ;ORDER
0100    OUTPUT PRTFIL       ;SORTED DATA FILE NAME
0110    END
```

7.2 SORT OPERATING PROCEDURES

SORT is called via the Monitor command:

```
.RU SORT[, file1,...,file7]
```

where

file1,..,file7 are the SORT control program which can be stored as more than one file. If none are specified, the file in the edit scratch area is used.

Mount the input file and n-1 (where n is the number of logical units to be used for SORT) of the scratch units. The Monitor outputs the message:

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

where filnam is the input file name specified in the SORT control program. Type the number of the logical unit (1-15) where the data file is stored. Monitor outputs the messages :

```
MOUNT $WORK2 #01 FOR OUTPUT:
.
.
.
MOUNT $WORKn #01 FOR OUTPUT:
```

Type the number (1-15) of the logical unit where all but the last of the SORT scratch units are located. The Monitor checks the labels of the files currently on these units. If a label exists, Monitor asks:

```
REPLACE xxxxxx #nn?
```

Type YES and the CR key to use the unit as a scratch area and destroy what is currently stored there. Type NO and the CR key to save the present label and specify another unit for a work area.

When the first sort phase is complete, the input is no longer needed. Monitor displays the message:

```
MOUNT $WORK1 #01 FOR OUTPUT:
```

In a tape SORT the input tape can be dismounted and the last work unit mounted or the input file can become the last scratch area. (Monitor will ask REPLACE? if this is done; type YES and the CR key).

Type the number (1-15) and the CR key of the logical unit where the last work area is located.

When the main part of the SORT is complete, Monitor displays:

```
UNIT xx IS FREE
MOUNT filnam #01 FOR OUTPUT:
```

xx is one of the units previously assigned as a work unit. SORT no longer needs the unit and it may be used for the output unit if desired.

Mount the output file. Type the logical unit number (1-15) where the output is to be stored. SORT reads the sorted records from the work units and stores them on the specified output unit.

If a default output unit is specified in the OUTPUT line of the SORT control, the MOUNT...FOR OUTPUT message may not appear.

NOTE

Sometimes the SORT program does not request that the last work unit be mounted - it types UNIT xx

IS FREE where xx is the unit where the input file is mounted, and asks that the output file be mounted. Do not be alarmed - SORT has decided that the file is sufficiently small and well-ordered that the main part of the sort is unnecessary.

7.3 RUNNING SORT AS PART OF AN UPDATE PROCEDURE

If the SORT control program has a line the first six non-blank characters of which are UPDATE, then the contents of the INPUT and OUTPUT parameter lines are ignored and the SORT program uses the label $UPD00 for both its input and its output file. (See Chapter 8 for UPDATE.)

7.4 MERGE OPERATING PROCEDURE

To run SORT as a merge only operation, type:

```
.RU SORT[, file1,file2,...,file7]/x
-
```

where

file1,...,file7	are the control program (possibly stored in two or more segments).
/x	is one of the following switches:
/A	specifies that names of files to be merged are to be entered from the terminal in answer to the INPUT FILE LABELS: message. The output data file and default unit name are specified in the OUTPUT clause of the control program.
/M	specifies that names of files to be merged are listed in the INPUT line of the SORT control program. This option bypasses the INPUT FILE LABELS: message.
/n	specifies that the name of the files to be merged is listed in the INPUT line of the SORT control program. Checks for n files of the same name on n(1-6) default units as specified. If the number of units specified is more than the number of units shown on the SORT control INPUT line, a MOUNT message is displayed for those files not on the INPUT line.
/L	can be used with any of the above switches and lists the SORT control program on the line printer.

When called with the /A option, SORT requests the names of the data files to be merged with the message:

INPUT FILE LABELS:

Enter the 1-6 character data file names separated by commas up to a maximum of six and default units if desired. At least two names must be entered or the error message NO INPUT is displayed.

For example:

```
PRTFIL/10,FILE1/3
```

If default units are not specified, Monitor outputs the message:

MOUNT filnam #01 FOR INPUT:

(where filnam is the data file name) for each data file specified.

For example:

MOUNT PRTFIL #01 FOR INPUT:

Type the logical unit number (1-15) where the data file is located.

Example:

Sample Control Program called PAYKEY

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

To SORT the data file described in the control program PAYKEY,

```
.RUN SORT, PAYKEY
```

The input data file is PAYROL located on logical unit 4. There are three sort work units, logical units 1, 2 and 3. The output data file is PAYROL and the sorted file is to be put back on unit 6.

The /A option in SORT is used as follows:

```
.RUN SORT, PAYKEY /A)
INPUT FILE LABELS:
PAYROL/4, PAY1/2,PAY0)
MOUNT PAY0 #01 FOR INPUT:
10)
```

The input data file names to be merged are:

```
PAYROL on logical unit 4
PAY1 on logical unit 2
PAY0 on logical unit 10
```

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

The /M option in SORT is used as follows:

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

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

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

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

The /n option in SORT is used as follows:

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

The input data files to be merged are:

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

7.5 RULES FOR USING DEFAULT UNITS

If you are using default units in your control programs for SORT, BUILD, etc., the data files should be mounted on those units. If the Monitor cannot find the input file on the default unit the message

```
? MOUNT xxxxxx #xx FOR INPUT:
```

is displayed. In the above MOUNT message the ? indicates that the input data file "xxxxxx" was not on the default unit specified.

```
MOUNT PAYROL #01 FOR OUTPUT:
6 )
```

If output of a data file using a default unit is attempted and another file already exists, the REPLACE message is displayed:

```
REPLACE LIST #02 ?
Y )
```

If output of a data file onto a logical unit that is already being used for input or output is attempted, the message IN USE is displayed.

```
IN USE
MOUNT PAYROL #01 FOR OUTPUT:
10 )
```

7.6 ERROR MESSAGES

Message	Explanation
File descriptor errors:	
BAD DIGIT IN DECIMAL INITIAL VALUE	Alpha character in a decimal initial value.
BAD WORK UNIT COUNT	Number of work units not in range 3-7.
EXTRA CHARS AT STMNT END	Characters not relating to statement appear on statement line.
FIELD NUMBER MISSING OR 0	Field number is missing or is 0 or greater than or equal to 512.
ILLEGAL SORT KEY	Bad syntax on KEY statement, KEY too complex, or KEY statement missing.
ILLEGAL UNIT	Default unit is 0 or greater than 15.
INITIAL ALPHA VALUE DOESN'T BEGIN WITH QUOTE	Beginning quotation mark missing for initial alpha value.
INITIAL VALUE TOO BIG	The initial value specified is larger than the field size.
INITIAL VALUE TOO SMALL	The initial value specified is smaller than the field size.
MISSING CLOSE QUOTE ON ALPHA INITIAL VALUE	Quotation mark not specified where necessary.
MISSING INITIAL VALUE	Comma was inserted after type and size but initial value was not specified.
NO COMMA AFTER FIELD NAME	No comma or a character other than comma was specified after the field name.
NOT A OR D	A character other than A or D occurred in a data specification statement where A or D was expected.
NOTHING AFTER FIELD NAME	Field type and size are not specified after field name and comma.
NUMBER REPEATED OR OUT OF ORDER	A field sequence number is used more than once or is out of ascending order sequence.
TOO MANY FILES	Merge only, more than 6 input files specified.

Message	Explanation
UNRECOGNIZABLE LINE	Parameter line did not start with a good keyword.
SORT Errors:	
BAD RECORD SIZE	File contains records of variable length.
NO INPUT	Input file is null or not enough input files specified for a merge.
OUTPUT ERROR	Indicates a system malfunction.
UNIT xx IS FREE	NOT AN ERROR - merely an informative message to aid the operator. xx is a COS logical unit number.

CHAPTER 8

UPDATE

COS UPDATE is a file maintenance program which makes changes, insertions and deletions to data files.

UPDATE must also have a control program before it can be executed.

UPDATE operates in two passes, pass1 reads the control program and UPDATE commands and checks for errors. It then outputs a modified version of these commands to a scratch file for the user to sort. Pass2 uses the UPDATE commands read from the scratch file to modify the input file, and creates a new output file.

The UPDATE commands are listed on the line printer as they are executed. A summary of the number of records read, changed, inserted and deleted is printed when the UPDATE is complete.

8.1 UPDATE CONTROL PROGRAM

The control program consists of a file dscriptor section followed by UPDATE, INPUT, SORT, KEY, OUTPUT and END statements in that order.

8.1.1 File Descriptor Section

This section defines the fields as they appear in the file to be updated. The restrictions are the same as for the SORT program. T and C field names are accepted but are ignored by UPDATE. Initial values are retained unless specifically changed by an UPDATE command (see section 8.2). Be sure not to skip any numbers when assigning field names. The file descriptor section must start with a DEFINE statement.

Example:

```
0010   DEFINE
0020   F1,D6              ;PART NUMBER
0030   F2,A30             ;PART NAME
0040   F3,D7              ;VENDOR NUMBER
0050   F4,D6              ;DATE
0060   F5,D6              ;WHOLESALE PRICE
```

This file descriptor section is based on the file created with BUILD in Chapter 6.

8.1.2 UPDATE Statement

The form of this statement is:

```
UPDATE filnam[/unit]
```

where

filnam	is the 1-6 character name of the data file to be updated. This name may not contain spaces, tabs, or commas.
/unit	is optional and specifies the logical unit (1-15) which contains the file. If the number specified exceeds 15, it is taken modulo 16. If the unit is omitted or if the file is not on the unit specified, Monitor outputs the usual MOUNT message.

Example:

```
0010    DEFINE
0020    F1,D6
0030    F2,A30
0040    F3,D7
0050    F4,D6
0060    F5,D6
0070    UPDATE PRTFIL/3    ;DATA FILE TO BE UPDATED IS
                           ;PRTFIL LOCATED ON LOGICAL
                           ;UNIT 3.
```

8.1.3 INPUT Statement

The form of this statement is:

```
INPUT dev
```

where

dev	is the physical device (KBD, RDR, CDR, SYS) to be used to input the UPDATE commands

KBD - terminal keyboard
RDR - paper tape reader
CDR - card reader
SYS - system device file

Example:

```
0010   DEFINE
0020   F1,D6
0030   F2,A30
0040   F3,D7
0050   F4,D6
0060   F5,D6
0070   UPDATE PRTFIL/3
0080   INPUT SYS          ;UPDATE COMMANDS
                          ;ARE STORED
                          ;IN A FILE ON THE
                          ;SYSTEM DEVICE.
```

8.1.4 SORT Statement

The form of this statement is:

```
SORT n
```

where n is the number (3-7) of logical scratch units which are to be used if the updated records must be SORTed. This is an optional statement. If the UPDATE commands are sorted, no SORT statement is needed. There must be at least three units to execute a SORT. If no scratch units are specified, 4 is assumed. If no SORT statement is specified, UPDATE executes a combined pass1 and pass2, reading one UPDATE command at a time and performing the update on the input file directly. No scratch unit is needed in this case.

Example:

```
0010   DEFINE
0020   F1,D6
0030   F2,A30
0040   F3,D7
0050   F4,D6
0060   F5,D6
0070   UPDATE PRTFIL/3
0080   INPUT SYS
0090   SORT 4             ;THERE ARE FOUR SCRATCH
                          ;UNITS FOR THE SORT
```

8.1.5 KEY Statement

The form of this statement is:

```
KEY fieldx(n,m)-[,...,fieldx(n,m)-]
```

where

fieldx(n,m)-,... is the SORT key used to sort the data file. (See SORT control program.) There can be up to eight fields or parts of fields in the SORT key.

Example:

```
0010   DEFINE
0020   F1,D6
0030   F2,A30
0040   F3,D7
0050   F4,D6
0060   F5,D6
0070   UPDATE PRTFIL/3
0080   INPUT SYS
0090   SORT 4
0100   KEY F1-,F4          ;THE KEY FIELDS
                           ;ARE PART NUMBER IN
                           ;DESCENDING ORDER
                           ;AND DATE
                           ;IN ASCENDING ORDER.
```

8.1.6 OUTPUT Statement

The form of the statement is:

```
OUTPUT filnam[/unit]
```

where

filnam is the 1-6 character name to be assigned to the updated file. If the statement is omitted the output file has the same name as the input file.

/unit is the logical unit (1-15) where the output will be stored. If no unit is specified, a MOUNT message is displayed.

Example:

```
0010   DEFINE
0020   F1,D6
0030   F2,A30
0040   F3,D7
0050   F4,D6
0060   F5,D6
0070   UPDATE PRTFIL/3
0080   INPUT SYS
0090   SORT 4
0100   KEY F1-,F4
0110   OUTPUT PRTFIL/3     ;CALL UPDATED
                           ;FILE PRTFIL AND
                           ;STORE ON UNIT 3
```

8.1.7 END Statement

The form of the statement is:

```
END
```

END is the terminating statement of the UPDATE control program and must be present.

Example:

```
0010   DEFINE
0020   F1,D6
0030   F2,A30
0040   F3,D7
0050   F4,D6
0060   F5,D6
0070   UPDATE PRTFIL/3
0080   INPUT SYS
0090   SORT 4
0100   KEY F1-,F4
0110   OUTPUT PRTFIL/3
0120   END                ;TERMINATING STATEMENT OF
                          ;CONTROL PROGRAM
```

8.2 UPDATE COMMANDS

The UPDATE commands can be input from the keyboard at UPDATE run time or can be prepared previously on paper tape, cards or the system device.

UPDATE commands have the general form:

```
{I}
{D}␣␣key [Fn=value],...
{C}
```

where

I,D,C	denote Insert, Delete, and Change
key	represents the data in the fields used to sort the record. The character string typed must be exactly the same length as specified for the SORT fields. The string is continuous; no spaces, etc. can separate fields if more than one is in the sort key. If the data file is sorted on an alpha key (A12) all trailing spaces must be typed on an UPDATE command line, for example, C␣JONES JOHN␣␣␣F6=...
Fn	is a field name and sequence number as specified in the UPDATE File Descriptor. Fields are processed from left to right and Field numbers may appear in any order.

=value — is the data to be inserted in the field named (Fn). This data must agree in type with the type specified for Fn and must be less than or equal in size. If the value specified is smaller than the field size, the alpha field data is left justified and padded with blanks; decimal field data is right-justified and padded with zeros.

On an insert, missing fields are set to their initial value if one was specified, otherwise the field is set to blanks (alpha) or zeroes (decimal).

Decimal values have the format

```
[+]
[-]digits[-]
```

Alpha values have the format

```
'characters'
```

No single quotation marks may appear within the alpha value. If no character is a space, tab or comma then the surrounding quotation marks need not be present.

A command which is too long for one UPDATE command line can be continued on the next line. Type a comma at the end of each line to be continued. There can be any number of continue lines.

Example:

```
I␣9988773771761 F1=001122,↵
F2='SCREW,#11',F3=100,F4=7771777,F5=32072
```

An error in an UPDATE command line causes the line to be ignored and an error message is output to the line printer. Execution continues.

8.3 UPDATE EXAMPLE

A file called ACCT has a mistake in the zip code of an employee whose employee number is 63411. The following control file is prepared:

```
0010   DEFINE
0020   F1,A40              ;NAME
0030   F2,D5               ;EMPLOYEE NUMBER
0040   F3,A30              ;STREET ADDRESS
0050   F4,A5               ;ZIP CODE
0060   F5,D8               ;SALARY
0070   UPDATE ACCT
0080   INPUT KBD
0090   KEY F2
0100   OUTPUT ACCT
0110   END
```

This file describes the fields of the file ACCT, specifies that it is sorted on field2 (employee number) and says that input commands are from the keyboard. No SORT statement is included because only one update command is needed and so no sorting is necessary. If more than one change is to be made in the file and if the changes are typed with their keys not in order then a SORT statement would be necessary.

The user then types:

```
.RUN UPDATE/1L
```

to run pass1 and get a listing of the control file.

If no SORT is required, UPDATE asks that the input and output files (both named ACCT) be mounted. If a SORT statement had appeared, the mount message would have requested the scratch tape $UPD00. (ACCT would not be mounted until pass2). The single UPDATE command can then be typed at the keyboard:

```
C␣63411␣F4=11691
```

specifying the new zip code of 11691 for the record whose key is 63411. All other fields remain the same. The user then types CTRL/Z to specify the end of the input from the keyboard.

8.4 OPERATING PROCEDURES

UPDATE is called with the Monitor command:

```
.RUN UPDATE[,file1,...,file7][/s(1)s(2)]
```

where

file1,...file7 are the names of the files which contain the UPDATE control program (possibly stored in two segments, FDR and I/O) and the UPDATE commands (if previously stored on the system device).

If no files are specified, UPDATE uses the file in the edit buffer.

/s(1)s(2) are switches

The first switch is 1 or 2 specifying the UPDATE phase. 1 - read control program and UPDATE commands and check for errors, put in scratch area. 2 - read scratch area after sorting and perform UPDATE. If neither 1 nor 2 is specified, pass1 is assumed.

The second switch is

- /N for no listing (default)
- /L for a listing of UPDATE commands, (pass2) or control program (pass1).

If no errors are found in the control program and the UPDATE commands are sorted, UPDATE continues automatically to pass2. Otherwise, SORT must be used to sort the UPDATE commands into the same order as the file. Type RU SORT,UPCON,UPCOM (control program). SORT asks that $UPD00 be mounted. When the SORT is complete, call the second phase of UPDATE. For example,

```
.RU UPDATE,UPCON,UPCOM/2
```

NOTE

If the input device specified was KBD, there is no indication from UPDATE that it is ready to accept input from the keyboard.

8.5 ERROR MESSAGES

! indicates a fatal error.

Message	Explanation
BAD DEVICE	Illegal device specified on input statement
BAD DIGIT IN DECIMAL INITIAL VALUE	Alpha character in a decimal initial value.
BAD INPUT STMNT	Input statement is missing or out of order.
BAD KEY STMNT	Key in KEY statement has bad syntax.
BAD UPDATE COMMAND	Bad syntax, probably missing F, =, or comma.
BAD UPDATE STMNT	Update statement is incorrect.
BAD VALUE	Value in update command too long or incorrect.
DATA TABLE OVERFLOW!	Too much data, maximum = 960 characters.
DESCRIPTOR TABLE OVERFLOW!	Too many F, T and C fields defined, maximum = 160.
EXTRA CHARS AT STMNT END	Characters other than A or D occurred in a data specification statement where A or D was expected.
FIELD NUMBER MISSING OR 0	Field number or default unit number is missing or is 0 or > = 512.
I RECORD ALREADY EXISTS	Tried to insert a record already present.

Message	Explanation
INITIAL ALPHA VALUE DOESN'T BEGIN WITH QUOTE	Beginning quotation mark missing for initial alpha value.
INITIAL VALUE TOO BIG	The initial value specified is larger than the field size.
INITIAL VALUE TOO SMALL	The initial value specified is smaller than the field size.
MISSING CLOSE QUOTE ON ALPHA INITIAL VALUE	Quotation mark not specified where necessary.
MISSING INITIAL VALUE	Comma was inserted after type and size but initial value was not specified.
MISSING OR OUT OF ORDER	On C or D, record was not there. Probably because scratch file was not sorted.
MISSING UNIT	Unit number specified is a multiple of 16.
NO COMMA AFTER FIELD NAME	No comma or a character other than comma was specified after the field name.
NO DEFINE!	Control file did not start with a DEFINE. Returns to Monitor immediately.
NO END STATEMENT	END statement missing after OUTPUT specification.
NONEXISTENT FIELD	Tried to update Fn where no Fn appeared in file description section.
NOT A OR D	A character other than A or D occurred in a data specification statement where A or D was expected.
NOT I,D, OR C	Bad first character on update command.
NOTHING AFTER FIELD NAME	Field type and size are not specified after field name and comma.
NUMBER REPEATED OR OUT OF ORDER	A field sequence number is used more than once or is out of ascending order sequence.
OUT OF ORDER	SORT statement missing and update commands are not in order.
UNEXPECTED END OF FILE	Missing END statement.

CHAPTER 9

BOOT

BOOT is used to bootstrap the system from one device to another, i.e., if the system has been moved from DECtape to disk, BOOT may be run to start the system on disk.

9.1 OPERATING PROCEDURES

The command to run the BOOT program has the form:

```
.RUN BOOT/xx
```

and the CR key:

where

/xx		is the switch which indicates the device where the system to be bootstrapped resides.
	/RK	is the RK08 or RK8E disk unit 0.
	/RF	is the RF08 disk.
	/DT	is the TC08 DECtape unit 0.
	/TD	is the TD8E DECtape unit 0.
	/LT	is the LINCtape unit 0.

If no switch or an illegal switch is specified, BOOT displays the error message NO and control returns to the Monitor.

Do not try to bootstrap onto a device which is not ready or does not exist.

Other options are available for experienced programmers:

/PT	loads the high-speed RIM loader and BIN paper tape loader into field 0.
/ZE	zeroes all of core in field 0.

BOOT may also be used to bootstrap OS/8.

CHAPTER 10

PATCH

PATCH is used to fix (or patch) system programs or the Monitor on a COS system tape. The data to make the changes is provided by Digital Equipment Corporation as patches in the form of dialogue with the PATCH program with the user input underlined. It would be necessary, therefore, to type only the underlined data exactly as shown and to be sure that the computer output corresponds with what is illustrated. If everything matches, the patch is correctly made. To better understand the workings of PATCH, it is important to note that both system programs (files) and the Monitor consist of blocks of numerical information (program code; machine language instructions) on the system device. Each block consists of words of 4-digit numbers. These words are numbered from 0 to 377 (base 8). PATCH reads one of these blocks, allows user examination and/or change (patch) of individual words in this block and writes the new block back out to the system device.

10.1 OPERATING PROCEDURES

Be sure the system tape has the current version of the program to be patched. The command to run the PATCH program is of the form:

.RU PATCH

and the CR key.

PATCH replies:

COS PATCH SYSTEM VERSION 11.08 (or current version number)

and

FILE NAME:

Respond with one of the following:

(a) The name of the file to be patched.

(b) /N and/or a carriage return, to indicate a patch for the Monitor not for a program. The system responds PATCHING MONITOR.

(c) CTRL/Z or /X. To indicate end of PATCH operation. The message EXIT is printed and control returns to the Monitor (just as if CTRL/C had been typed.)

When typing in letters or digits, a typing mistake can be corrected (before typing the CR key) by typing CTRL/U and retyping the entire line correctly.

The next PATCH question is:

BLOCK:

Type the block number specified on the patch information sheet as sent out by DEC. This specifies which BLOCK is to be patched. Typing END or CTRL/Z at this point causes a return to the FILE NAME question. If at any time before typing END or CTRL/Z it is discovered that the wrong BLOCK number was typed, type R (for RESTART) and PATCH returns to the FILE NAME question. Reenter the file name, then type the correct BLOCK number and continue from there.

PATCH displays:

LOCATION:

Type in the number provided by DEC. This specifies which word in the block is to be patched. PATCH responds with:

OLD VALUE: nnnn

where nnnn is the four-digit old (current) value of this location.

It is very important to verify that this value is the one that DEC said it should be. If it is not, do not proceed; something is wrong.

a. Check that everything previously typed was letter perfect. If the wrong BLOCK number was typed, type R and restart at FILE NAME. If the wrong LOCATION was typed, type the CR key in answer to the NEW VALUE: question. This makes no change to the location specified. Then type the correct location number in answer to the LOCATION question:

b. If everything typed was correct, perhaps an old version of the Monitor or the system program in question is being used. Before starting PATCH, the version of the program in question should be verified.

c. If everything seems correct but the dialogue doesn't agree with what DEC predicted, save all output and consult your software specialist.

When the program asks:

NEW VALUE:

enter the new value as supplied by DEC. This new value replaces the old value in core. The block is not changed on the system device until all patches are made to this block.

If the wrong new value is typed, retype the location, verify that its current (old) value is the one typed in by mistake, and then enter its new corrected value.

If a new value is entered for the wrong location, reenter that location and as the new value, type its original value (determined by looking back on the listing for the OLD VALUE at the time this location was first referenced). If a scope display is used and the original value cannot be determined, type R and redo the entire block.

Answer NEW VALUE: with the CR key or CTRL/Z if the location was just examined and no new value is to be entered.

When all patches to a block are complete, type CTRL/Z or END to the LOCATION: question. At this point, PATCH asks for the RELATIVE CHECKSUM (similar to a hash total). Type the relative checksum, as supplied to you by DEC, and the CR key. If every thing is correct, the new corrected block is written on the system device, and the message:

```
NEW BLOCK PATCHED OK
```

is displayed. The question BLOCK: is displayed again to allow patching of another block. Type in the next block number in this file to be patched or type END or CTRL/Z to terminate patches for the file. In the latter case, PATCH displays the message:

```
nn BLOCK(S) PATCHED IN THIS FILE
FILE NAME:
```

If an error is detected, the message:

```
BAD CHECKSUM
LOCATION:
```

is displayed and the faulty block is not written to the system device. The new block as changed is still in core. Review the numbers typed in and possibly examine the locations changed to see if they are correct. If the error is found, fix it and then type END or CTRL/Z to the LOCATION: message. If an error is not found, type R to restart the program and try patching the entire block again.

10.2 ERROR MESSAGES

Message	Explanation
BAD CHECKSUM	An attempt was made to write a block which was incorrectly patched.
BAD DIRECTORY	Logical unit 0 was not a system device.
BAD NUMBER	A number with more than 4 digits, a non-digit or containing the digits 8 or 9 (these are never used) was typed.
BLOCK TOO BIG	An incorrect block number was typed. It cannot be larger than the length of the file being patched.
FILE NOT FOUND	The program with the name specified was not found on the system device. It must be there. If it is on another tape, it may be mounted on logical unit 0 when PATCH first starts up; but thereafter DO NOT CHANGE UNITS.
LOCATION TOO BIG	A location greater than 377 was typed.
NO CHANGE IN BLOCK	An attempt was made to write a block but no changes were made in it. If this was on purpose, fine, otherwise make the changes to the block again.

CHAPTER 11

DAFT

The DAFT (Dump and Fix Technique) program is similar in function to an editor but is used for data records. DAFT provides the capability to search for, examine and change records and to list records or parts of records on the line printer or terminal. DAFT may be used to make minor changes to a data file.

DAFT allows one input and one output file to be open at the same time. These two files may be the same file when in update mode. At any given time, there is a record from the input file in core known as the current record. This record may be modified by the CHANGE command before it is written on the output file. An output file need not be open if records from the input file are only being examined.

11.1 OPERATING PROCEDURE

The command to run DAFT has the form:

.RU DAFT[, file1,...,filen]

and the CR key.

Where

file1,...,filen	are previously created files which contain a string of DAFT commands to be used to dump or fix the data file. These files are optional and if not specified, commands are entered via the terminal. After the last command in the last file is executed, additional commands may be entered from the keyboard.

11.2 COMMANDS

DAFT commands begin with a keyword (consisting of any number of non-blank characters, of which only the first one is significant). Some commands take arguments, in which case the keyword is followed by one or more spaces and the arguments. These arguments may be separated from each other by one or more spaces.

In Table 11-1, an expression in brackets [ ] indicates that it is optional.

The following symbols are used in Table 11-1 to show the form of DAFT commands:

n	represents an unsigned non-zero positive integer. If it is optional in a command and is omitted, n=1 is assumed.

< a,b > represents a key field consisting of character positions a through b inclusive. a and b must each be unsigned non-zero positive integers and b must not be smaller than a. If this is optional in a command, and is omitted, the key specified in the KEY command is used.

label is a data file name.

unit represents an integer in the range 1-15. If this is specified, the system looks for the specified file on this logical unit before issuing a MOUNT message. If given, /unit must immediately follow the file name.

+ indicates that before a record is read from the input file, the current record in core (if there is one) is written on the output file. The + sign need not have a space before it unless it is the only argument.

data represents a piece of data. Alpha data has the form:

```
' characters... '
```

and decimal data has the form:

```
[-] digits...
```

Before data is used in executing the command, it is adjusted to the same length as the key field. If the data is smaller and alpha, it is left justified in the field and padded with spaces on the right. If it is smaller and decimal, it is right justified and padded with zeroes on the left. If data is larger and alpha, excess characters on the right are ignored. If it is larger and decimal, excess characters on the left are ignored.

TABLE 11-1. DAFT COMMAND SUMMARY

Command	Function
Advance [n] [+]	Advances the input file n records. If n is omitted, 1 is assumed.
Backspace [n]	Backspaces n records if the input file was opened with the UPDATE command. If n is omitted, 1 is assumed.
Change [<a,b>] data	Replaces the data in the current key field of the record currently in core with the data specified. If the key, <a,b>, is used in this command, it temporarily overrides the key specified by the KEY statement.

TABLE 11-1. DAFT COMMAND SUMMARY (Cont'd)

Command	Function
Display [n]	Sets the width of the print line of the listing device (terminal or line printer) to n characters (maximum 130). If n is omitted, this command turns on the grid if it was off and turns it off if it was on.
Exit	Returns control to the COS Monitor providing no output file is open.
Fini [+]	Closes the output file. If + is specified, the current record and the remainder of the input file are first copied to the output file.
Goto n [+]	Makes record n the current record.
Help	Displays a summary of DAFT commands on the terminal.
Input label [/unit]	Opens the specified file (label) for input. The first record is read and becomes the current record.
Key a,b	Sets the current key to character positions a through b inclusive.
List [n] [<a,b>] [+]	Lists on the line printer, n consecutive records beginning with the current record. The key, <a,b>, if specified, represents those consecutive characters in the record that are to be considered.
Output label [/unit]	Opens the specified file (label) for output.
Put [n]	Writes n copies of the record currently in core on the output file.
Query	Lists information about the current status of the program on the terminal. This includes the names of the input and output files, the units where the files are located, the record in core and the version number of the DAFT program.
Rewind	Rewinds the input file. Record 1 becomes the current record.

TABLE 11-1. DAFT COMMAND SUMMARY (Cont'd)

Command	Function
Search [<a,b>] data [+]	Searches the current record and then succeeding records for an occurrence of the specified data appearing in the current key field.
Type [n] [<a,b>] [+]	Same as list except output is to the terminal.
Update label [/unit]	Opens the specified file (label) for updating. This command can only be specified for a file with fixed length records since direct access I/O is used to move records.
Version	Displays the version number of the program on the terminal.
Write [n]	Performs same function as ADVANCE [n] +, i.e., n-1 consecutive records, beginning with the current record are written on the output file. The nth record (after the original one) becomes the current record.
X	Lists the record number and size of the current record on the listing device (either terminal or line printer depending on whether the last record was listed via a TYPE or LIST command). The line printer is the initial listing device.

11.3 DAFT ERROR MESSAGES

Message	Explanation
0 NOT ALLOWED	In some commands, 0 is not a permissable argument.
BAD DIGIT IN DATA	In a CHANGE or SEARCH command, a character other than digits or a minus sign is contained in a numeric data field (a field without quotes).
CANT BACKSPACE PAST BEGIN OF FILE	Attempted to backspace past beginning of file. The first record becomes the current record.
CANT BACKSPACE WITH SEQUENTIAL INPUT	Attempted to backspace with sequential input.

Message	Explanation
END OF INPUT FILE AT RECORD nnnn	An attempt was made to read past the end of file mark on the input file. This is not necessarily an error. nnnn was the last record read. The input file is closed.
EXCESSIVE GRID SIZE	The grid (printer width) may not be greater than 130 characters.
EXTRA CHARS	Extra characters were found after the logical end of a command.
ILLEGAL RECORD - CLOSING FILE	The file being updated contains a bad record (one not the same size as record 1). Only fixed length records are permitted on such files. The file is closed.
KEY ENTIRELY PAST END OF RECORD	The key specified in a LIST or TYPE command began with a character greater than the record size.
KEY EXTENDS PAST RECORD END	A change or search was attempted with a key that extends past the end of a record. However, a list with such a key is possible. In such a case the list is terminated at end of record.
KEY TOO BIG	The range of the key cannot exceed 100 characters.
NO DATA	Data must be specified in a CHANGE or SEARCH command.
NO INPUT FILE	The command requires an input file but one is not open.
NO LABEL NAME	The filnam was omitted in an INPUT, OUTPUT or UPDATE command.
NO OUTPUT FILE	The command requires an output file but one is not open. The command is terminated at the point just prior to writing the current record on the output file.
OUTPUT FILE ALREADY OPEN	A request was made to open an output file while one was already open. Only one output file may be open at a time. The request is ignored.

Message	Explanation
OUTPUT FILE STILL OPEN	An EXIT cannot be made when the output file is still open. The output file can be closed with the FINI command or CTRL/C may be typed.
PUSH DOWN OVERFLOW	The program will abort with this message when too many errors are made.

11.4 DAFT OUTPUT

Records may be listed with a grid above them. This grid consists of two lines of numbers showing the character positions. The lower of the two lines represents the units digits of the column counts. The upper line represents the tens digits. The tens digits are printed for the first and last character in the record (or part of the record) listed or whenever it changes. If there is a hundreds digit, it is printed in column 1 or whenever it changes.

Refer to Figure 11-1 for a sample of a DAFT operation.

```
COS MONITOR 2.0606
.R DAFT
*HELP
ADVANCE N+
BACKSPACE N
CHANGE <A,B> DATA
DISPLAY N
EXIT
FINI +
GOTO N+
HELP
INPUT LABEL/UNIT
KEY A,B
LIST N KEY+
OUTPUT LABEL/UNIT
PUT N
QUERY
REWIND
SEARCH <A,B> DATA +
TYPE N <A,B>+
UPDATE LABEL/UNIT
VERSION
WRITE N
X
*VERSION
DAFT VERSION 6.06
*V
DAFT VERSION 6.06
*VERY
DAFT VERSION 6.06
*INPUT MAILNG
MOUNT MAILNG #01  FOR  INPUT:
1
*TYPE 1

RECORD 000001 OF FILE MAILNG,   RECORD LENGTH=204 CHARACTERS

 DIGITAL EQUIPMENT CORP.  D. F. PAVLOCK

 12-3                     146 MAIN ST.

 MAYNARD        MA017540S/3-1  0012345A11111  06217

 2003SUPERVISOR
```

Figure 11-1. DAFT Sample

```
*DISPLAY 70\
*T 1

RECORD 000001 OF FILE MAILNG,    RECORD LENGTH=204 CHARACTERS

 DIGITAL EQUIPMENT CORP.  D. F. PAVLOCK             12-3

      146 MAIN ST.              MAYNARD        MA017540S/3-1  0012345A11

 111  062172003SUPERVISOR
*D
*T 2

RECORD 000001 OF FILE MAILNG,    RECORD LENGTH=204 CHARACTERS

 0        1         2         3         4         5         6         7
 12345678901234567890123456789012345678901234567890123456789012345678
 DIGITAL EQUIPMENT CORP.  D. F. PAVLOCK             12-3

 7        8         9        10         1         2         3         4
 12345678901234567890123456789012345678901234567890123456789012345678
      146 MAIN ST.              MAYNARD        MA017540S/3-1  0012345A11

14        5         6         7         8         9        20  0
 12345678901234567890123456789012345678901234567890123456789012345678
 111  062172003SUPERVISOR

RECORD 000002 OF FILE MAILNG,    RECORD LENGTH=204 CHARACTERS

 0        1         2         3         4         5         6         7
 12345678901234567890123456789012345678901234567890123456789012345678
 DIGITAL EQUIPMENT CORP.  K. RICHER                 12-3

 7        8         9        10         1         2         3         4
 12345678901234567890123456789012345678901234567890123456789012345678
      146 MAIN ST.              MAYNARD        MA01754COS 300 0001972T 3

14        5         6         7         8         9        20  0
 12345678901234567890123456789012345678901234567890123456789012345678
 00   053172002WRITER
*T 2<25,50>

RECORD 000002 OF FILE MAILNG,    RECORD LENGTH=204 CHARACTERS

 2    3         4         5
 56789012345678901234567890
  K. RICHER
```

Figure 11-1. DAFT Sample (Cont'd)

```
RECORD 000005 OF FILE MAILNG,    RECORD LENGTH=204 CHARACTERS

 DEC                        S. G. WELLCOME
*Q
INPUT FILE: MAILNG     OPEN
UNIT: 00
OUTPUT FILE: /NONE/
UNIT: 00
KEY=<001,050>

RECORD 000005 OF FILE MAILNG,    RECORD LENGTH=203 CHARACTERS

DAFT VERSION 6.06
*K
*Q
INPUT FILE: MAILNG     OPEN
UNIT: 00
OUTPUT FILE: /NONE/
UNIT: 00
KEY=<001,510>

RECORD 000005 OF FILE MAILNG,    RECORD LENGTH=204 CHARACTERS

DAFT VERSION 6.06
*B   ;SHOULD CAUSE AN ERROR MESSAGE
CANT BACKSPACE WITH SEQUENTIAL INPUT
```

Figure 11-1. DAFT Sample (Cont'd)

```
RECORD 000003 OF FILE MAILNG,    RECORD LENGTH=204 CHARACTERS

 2    3         4         5
 56789012345678901234567890
  S. RABINOWITZ
*D
*T 2

RECORD 000003 OF FILE MAILNG,    RECORD LENGTH=204 CHARACTERS

 DIGITAL EQUIPMENT CORP.  S. RABINOWITZ            12-3

      146 MAIN ST.            MAYNARD        MA01754COS 300 0099999C12

 345  062172009COMPILER

RECORD 000004 OF FILE MAILNG,    RECORD LENGTH=204 CHARACTERS

 DIGITAL                  R. LARY                  12-3

      146 MAIN ST              MAYNARD        MA01754         0000000

      061272999SORT
*D
*KEY 1,50   ;SET PERMANENT KEY
*T2

RECORD 000004 OF FILE MAILNG,    RECORD LENGTH=204 CHARACTERS

 0        1         2         3         4         5
 123T5678901234567890 1R34567890123456789012345678 90
 DIGITAL                  R. LARY
*T 2

RECORD 000004 OF FILE MAILNG,    RECORD LENGTH=204 CHARACTERS

 0        1         2         3         4         5
 12345678901234567890123456789012345678901234567890
 DIGITAL                  R. LARY

RECORD 000005 OF FILE MAILNG,    RECORD LENGTH=204 CHARACTERS

 0        1         2         3         4         5
 12345678901234567890123456789012345678901234567890
 DEC                      S. G. WELLCOME
*D
*T
```

Figure 11-1. DAFT Sample (Cont'd)

CHAPTER 12

COS-OS/8 CONVERTER

The COS-OS/8 Converter program, CONVEX, provides the capability to convert source or data ASCII files created with OS/8 to COS source or data files. COS data files can be converted to OS/8 ASCII files.

12.1 OPERATING PROCEDURES

The command to run CONVEX has the form:

```
.RUN CONVEX
```

and the CR key:

CONVEX replies with the program version number:

```
VERSION 12.28          (or current version number)
```

And the message:

```
INPUT MODE-
```

Answer with the letter code:

```
A  (OS/8 ASCII file mode)
D  (COS Data file mode)
```

and the CR key.

It then displays the message:

```
FILE NAME-
```

The answer has two forms depending on the mode.

1. For mode A (OS/8 ASCII) type the OS/8 file name (one to six characters optionally followed by a period and up to a two character extension) followed by a comma and then the device name which is one of:

   ```
   DTn
   DKn
   ```

   where n is a single digit referring to the physical unit # (DT0-DT7 or DK0-DK3). If no device is specified, CONVEX uses the system device.

Examples:

```
ABC,DT5
RICH.TM,DK0
```

2. For mode D (COS data) type the COS data file name (one to six characters) optionally followed by a slash and the logical unit number (1-15). If a logical unit is specified, the system first checks that logical unit for the specified file before issuing a MOUNT message.

   Examples:

   ```
   CUST/5
   LIST/1
   PROSPT
   ```

   Illegal:

   ```
   CLIENT,DT1
   ```

If everything is ok, the program displays the message:

OUTPUT MODE -

Type one of the following one character codes:

A (OS/8 ASCII file)
D (COS Data file)
S (COS Source files)

The next question is:

FILE NAME -

If the option specified was A or D, reply following the rules explained in steps 1 and 2. If an existing file is to be destroyed, CONVEX asks:

REPLACE?

Type Y for YES, N or any other character for NO and the CR key.

If the option specified was S, type the filename, optional comma and COS device e.g. GROUPA,DT3.

This device must be physically the same as the system device, i.e., tape to tape or disk to disk.

When the conversion is complete the message:

INPUT MODE -

is displayed again and another conversion can be specified. Type CTRL/C to return to Monitor.

NOTE

If instead the message:

OUTPUT MODE -

is displayed again after successful output has occurred (MODE A), CONVEX needs more space to continue. Specify an additional output file.

NOTE

OS/8 files are always generated so as to be multiples of 16 blocks long. For this and other reasons, the resulting OS/8 files may be longer than necessary. To recapture the wasted space, use OS/8 PIP to copy the file back to itself using the /A option.

12.2 Error Messages

In some cases, illegal or bad syntax on input causes the question to be reasked.

Other errors generate one of the following:

Message	Explanation
BAD CHAR	Warning message logged for each attempt to convert an OS/8 character you tried to convert to COS which is not in COS character set. Characters such as ↑, CTRL/Q, etc. Note: ← is translated to <, tabs are converted to proper number of spaces. Vertical tabs, line feeds, form feeds, rubouts etc. are ignored. Lower case characters are converted to upper case.
BAD DEV	Specified an output device with -S option and output device was not same type physical device as system device. e.g. FILE,DK1 when system device is DT0.
BAD DIRECTORY	Attempted to reference or store a file on a device with no directory or a directory that has been destroyed.
FULL	Specified an output device with -S option and source file being created was filled.
NO END	Entire OS/8 ASCII input file has been read but no end of file (CTRL/Z) was found.

Message	Explanation
NO ROOM	Directory of output device was full; no room for specified output file.
NONCE ERROR	User tried to use a feature not currently supported or documented (but might be implemented in some future release).
NOT FOUND	File with specified name was not found.

CHAPTER 13

FORMAT PROGRAMS

There are four format programs provided with the COS package which mark disk or DECtape for use with COS.

These programs are:

RK8MRK	Formats an RK08 disk.
RKEMRK	Formats an RK8E disk.
TDMARK	Formats a DECtape on a TD8E drive.
DTMARK	Formats a DECtape on a TC08 drive.

These programs are described in detail in diagnostic documents (RK08 Disk Formatter, DEC-08-D5KB; RK8E Disk Formatter, DEC-08-DHRKD-A-D; TD8-E DECtape Formatter, DEC-8E-EUZC-D; TC01 - TU55 DECtape Formatter, DEC-08-EUFB-D) provided with the system. The procedures in the following paragraphs cover the use of these programs with the COS system.

13.1 RK8MRK

Mount the system tape or disk and the disk to be formatted.

13.1.1 Operating Procedure

The command to run RK8MRK is:

```
.RUN RK8MRK
```

and the CR key.

RK8MRK replies with the question:

```
DRIVE NO.?
```

Type the unit number of the RK08 disk to be formatted. Be sure that the unit specified is not a good system device since formatting destroys the contents of a disk.

```
DRIVE NO.? 0
```

RK8MRK displays the question:

```
WRITE STANDARD FORMAT? (TYPE "Y" FOR YES)
```

Type Y to format the disk for use with COS. Any other reply provides a non-standard format which can not be used with COS.

When formatting is complete RK8MRK displays the messages:

```
DISK FORMATTED
```

and

```
DRIVE NO?
```

At this point specify another drive to be formatted or type CTRL/C to return to the COS Monitor.

13.1.2 Error Messages

All error messages displayed contain the track number and sector number where the error occurred. For example:

```
TRACK 0104 SECTOR 02
PARITY ERROR
```

Message	Explanation
CONTROL BUSY ERROR	Disk IOT issued when control was busy which would effect operation.
DATA RATE ERROR	The processor was busy and did not respond to a data break request within the 13 micro seconds required. The transfer is terminated immediately.
EXEC WORD ERROR	An error occurred while reading the EXEC. word of the sector.
FORMAT ERROR	An error occurred while reading the header word.
FORMAT PARITY ERROR	A parity error occurred in either the Header word or Exec word while reading the sector.
PARITY ERROR	A bit in data, parity or timing has been picked up or dropped on read. Transfer continues to the end of the sector where the error occurred. Word count, current address information can be used to identify the error.
SECTOR NO GOOD	The program attempted to read or write data on a sector whose header words indicated a bad sector. The transfer is terminated immediately.
TIME OUT ERROR	The control did not complete an operation after 32 revolutions.
TRACK ADDRESS ERROR	Track, surface or sector address read from the disk did not agree with the address count registers or the disk drive electronics indicated track position 000 and the track counter did not agree. The transfer is terminated immediately.
TRACK CAPACITY EXCEEDED ERROR	The program attempted to read or write beyond sector 17.
WRITE LOCK ERROR	The program attempted to write a section that was write protected. The write operation is terminated immediately.

Error Recovery

If the program halts with the program counter = 1337 indicating non-existent drive or device not adjusted correctly. If CTRL/C does not work, restart the COS system. (Refer to Appendix B.) If an error reoccurs, the disk may be bad.

At the end of each track in which an error occurred on the write and read operation the message:

RE-WRITE TRACK (Y FOR YES)?

is displayed. Type Y to rewrite the track.

13.2 RKEMRK

Mount the system tape and the disk to be formatted.

13.2.1 Operating Procedure

The command to run RKEMRK is:

```
.R RKEMRK
```

and the CR key.

RKEMRK replies with the following:

```
RK8E DISK FORMATTER PROGRAM
FOR ALL QUESTIONS, ANSWER Y FOR YES OR N FOR NO.
```

The following questions are asked:

```
FORMAT DISK 0?
FORMAT DISK 1?
FORMAT DISK 2?
FORMAT DISK 3?
ARE YOU SURE?
```

After answering Y or N to each of the above questions the disk(s) is formatted and the following message is printed:

```
RK8E DISK FORMATTER PASS COMPLETE
FORMAT SAME DISK(S) AGAIN?
```

Answer Y or N. At this point each of the FORMAT DISK n? questions are printed again. Exit to Monitor by typing CTRL/C or continue procedure for formatting disks.

13.2.2 Error Messages

The following error messages may be displayed when marking an RK8E disk for COS:

```
DISK DATA ERROR
READ STATUS ERROR
WRITE STATUS ERROR
RECALIBRATE STATUS ERROR
```

After the error header mentioned above is displayed, the terminal will print some of the following error information pertaining to the failure.

PC: PROGRAM LOCATION OF FAILURE

CD: EXPECTED INFORMATION

ST: CONTENTS OF THE STATUS REGISTER

CM: SOFTWARE COMMAND REGISTER

DA: SOFTWARE CYLINDER, SURFACE, AND SECTOR REGISTER

AD: ADDRESS OF DATA BREAK

DT: DATA FOUND DURING DATA BREAK

After the error information is displayed, the terminal will display one of the following questions asking the error recovery desired.

1. If the error was a recalibrate error, the following question will be displayed:

   TRY TO RECALIBRATE SAME DISK AGAIN?

   Typing a Y for yes will result in a repeat of the recalibrate sequence on the disk in error. Typing N for no will result in progressing to the next available disk.

2. If the error was a write error, the following question will be displayed:

   TRY TO FORMAT SAME CYLINDER AGAIN?

   Typing Y for yes will result in a repeat of the write sequence on the current cylinder. Typing N for no will result in progressing to the next sequential cylinder.

3. If the error was a READ or check error, the following question will be displayed:

   TRY TO CHECK SAME CYLINDER AGAIN?

   Typing a Y for yes will result in a repeat in the READ and check sequence on the current cylinder. Typing a N for no will result in progressing to the next sequential cylinder.

13.3 TDMARK

TDMARK records the timing and mark tracks on a DECtape mounted on the TD8E transport.

13.3.1 Operating Procedures

The command to run TDMARK is:

.RUN TDMARK

and the CR key. TDMARK replies with the question:

UNIT?

Before replying to this question, mount the DECtapes to be marked with just enough turns on the right hand reel to provide a grip. Make sure that no two tape units are set to the same unit number; set the WRITE ENABLE/WRITE LOCK switch to WRITE ENABLE and the REMOTE/OFF/LOCAL switch to REMOTE. Set the WTM (Write Timing Mark) switch to on (in down position). This red switch is located on the upper right hand side of Module 868 (printed circuit card with wires) within the processor (the black box behind the lights and switches).

Answer the UNIT question with one or two unit numbers, corresponding to the units of the tapes to be marked. For instance, if tapes are mounted on units 0 and 1, type:

0 1)

Spaces are ignored, so it makes no difference if spaces are typed between the unit numbers. Only one specification of a unit is significant, i.e., typing 000111 has the same effect as typing 01. Only tapes on units 0 and 1 may be formatted. If an error is detected in the response to this question, the question is repeated. TDMARK replies with:

FORMAT?

Type MARK and the CR key and the program assumes 201 words 2702 blocks (standard PDP-8 format). Any other reply produces a format which is not compatible with COS.

TDMARK displays the message:

0201 WORDS,2702 BLOCKS.OK?(YES OR NO)

If NO is typed, the program reverts to the FORMAT? question. If YES the program displays:

SET SWITCH TO WTM

Type the CR key and if the switch is set, the tape on the first unit specified begins to move.

Once all of the tapes specified have been marked, the message:

SET SWITCH TO OFF

is displayed. Reset the WTM switch to normal (up position) and type the CR key to start the second pass. Note that during the second pass with multiple DECtape units, as soon as one tape stops, the next tape starts; the first tape is completed and may be replaced with a fresh tape in preparation for formatting additional tapes. When formatting is completed, the message:

FORMAT

is displayed. Typing SAME and the CR key repeats the entire process with the same format. The new DECtapes must be mounted and ready before the CR key is typed in response to the SET SWITCH TO WTM message. When all tapes are formatted, type CTRL/C to return to the COS monitor.

13.3.2 Error Messages

The following error messages may be displayed when marking a tape for COS.

Error messages for response to SET SWITCH TO WTM:

Message	Explanation
SETUP	Indicates an error in the DECtape setup. One of the units specified is in WRITE LOCK position, not selected, or the write flip-flop is unable to be set, or there may be a timing error. (After UNIT message is displayed.)
SWITCH NOT SET TO WTM OR SINGLE LINE FLAG FAILED TO SET. SET SWITCH TO WTM.	Indicates the switch on the M868 module is not set to the WTM position or the timing generator for writing the mark and timing tracks is not setting the single line flag. If the switch was not set to WTM position, set the switch and type the CR key. If the switch was set to WTM position and this error occurred, try again or examine the timing generator circuit.

Error messages for marking and verifying a tape:

PC	xxxx	BLOCK NUMBER ERROR PHASE n
PC	xxxx	CHECKSUM ERROR PHASE n
PC	xxxx	DATA ERROR PHASE n
PC	xxxx	MARK TRACK ERROR PHASE n
PC	xxxx	TIMING ERROR PHASE n
PC	xxxx	WRITE ERROR PHASE n

where xxxx equals the program counter (PC) at the time of the failure. n equals the pass in which the error occurred. (See Recovery.)

Recovery:

Although an error should cause doubt concerning the entire process. a restart may be made (except in phase 0) by typing RETRY. RETRY causes the program to go back to phase 1. Type RESTART to return to the UNIT? question.

Phase 0:	Write timing and mark track forward.
Phase 1:	Reads mark track reverse.
Phase 2:	Write data, forward block and reverse block numbers forward and writes the checksums.
Phase 3:	Displays block numbers in AC (accumulator) reverse.
Phase 4:	Reads data, forward block and reverse block numbers forward and calculates the checksum.
Phase 5:	Reads reverse block numbers in reverse.

The entire program may be restarted at 0200 any time.

13.4 DTMARK

DTMARK records the required timing and mark tracks on a DECtape mounted on the TC01-TU55 DECtape unit. Up to eight DECtapes may be formatted at a time, assuming the system has eight tape transports.

13.4.1 Operating Procedures

The command to run DTMARK is:

```
.RUN DTMARK
```

and the CR key. DTMARK replies with:

```
DTA?
```

Mount the DECtapes to be marked on the tape transports, with just enough turns of tape on the right hand reel of each transport to provide a grip. Make sure that no two tape units are set to the same unit number. Set the WRTM (WRite Timing Mark) switch located on the hardware frame to the down position (on). Set the WRITE ENABLE/WRITE LOCK switch to WRITE ENABLE and the REMOTE/OFF/LOCAL switch to REMOTE.

Reply to the DTA? question with a unit number or series of unit numbers, corresponding to the DECtape units where the tapes are mounted. For instance, if tapes are mounted on units 2 and 3, type 2 3 and the CR key. Spaces are ignored, so it makes no difference if spaces are typed between the unit numbers.

Only one specification of a unit is significant, i.e., typing 2 2 3 3) has the same effect as typing 2 3).

The program displays:

```
DIRECT?
```

Type MARK and the CR key to format the DECtape for use with COS. Any other reply causes a format which is incompatible with COS. (Refer to TC01-TU55 DECtape Formatter Writeup.) The program assumes 201(8) words, 2702(8) blocks (standard PDP-8 format) and displays the message:

```
0201 WORDS,2702 BLOCKS.OK? (YES OR NO)
```

If a NO) is typed, the program reverts to the DIRECT? question. If YES), the tape on the first unit specified begins to move.

Once all of the tapes specified have been marked, the message:

```
SET SWITCH TO NORMAL
```

is displayed. Return the WRTM switch to normal (up position) and type the CR key to start the second pass. Note that during the second pass with multiple DECtape units, as soon as one tape stops, the next tape starts. The first tape is completed and may be replaced with a fresh tape in preparation for rerunning the program to format additional tapes.

When DTMARK has formatted the tapes specified, the message:

DIRECT?

is displayed again. Type SAME and the CR key to repeat the entire process with the same format. The new DECtapes must be mounted and ready before SAME ↵ is typed. (0 means DECtape unit 8).

When all tapes are formatted, type CTRL/C to return to the COS Monitor.

13.4.2 Error Messages

Errors typed in response to the DTA? and DIRECT? questions cause a return to DTA? or DIRECT? respectively.

Error messages for response to OK? (YES OR NO):

Message	Explanation
SETUP?	Indicates an error in the DECtape setup, unit in WRITE LOCK position, nonselectable unit or switch not in WRTM position. After display of this message, the DTA? question is displayed.

Error messages which may occur during marking and verifying a tape:

```
xxxx SHOULD BE yyyy BLK ERROR PHASE n
xxxx SHOULD BE yyyy DATA ERROR PHASE n
END TAPE ERROR PHASE n
LAST INT NOT END ZONE
MARK TRACK ERROR PHASE n
PARITY ERROR PHASE n
SELECT ERROR PHASE n
TIMING ERROR PHASE n
```

Recovery from error messages:

Although an error should cause doubt concerning the entire process, restarts may be made (except when in phase 0), by typing RETRY and the CR key. RETRY causes the program to return to PHASE 1. Type RESTART and the CR key to return to the DTA? question.

```
PHASE 0:    Mark track write.
PHASE 1:    Writing last reverse block number forward.
PHASE 2:    Writing block numbers and data in reverse.
PHASE 3.    Reading and checking block numbers and data.
```

If it is necessary to restart the DTMARK program, type CTRL/C to return to the COS Monitor then type RUN DTMARK.

If CTRL/C does not work, rebootstrap the COS system (refer to Appendix B) and then call DTMARK.

CHAPTER 14

CREF

CREF (Cross REFerence) is a utility program. Primarily an aid to program development, it provides an alphabetical listing of all symbols used in a DIBOL program, along with the line number where each symbol is defined and all the line numbers where each symbol is used.

CREF requires 8K of core and can handle any reasonable 8K program. If 12K of core is available CREF expands its tables to make use of the extra space. CREF expects the source program to be free from errors. Only a very minimal amount of error checking is preformed by CREF and no attempt should be made to CREF programs with compilation errors. If CREF finds a line it cannot handle it will print:

xxxx IS BEING IGNORED

where xxxx is a line number.

For an example of CREF operation, consider the listing in Figure 14-1. It is a short program that computes the square roots of numbers from 1 to 99. (Output of program is shown in Figure 14-3.) Its CREF table, Figure 14-2, illustrates the basic functions of CREF.

The columns of the CREF Table are:

NAME	Name of the symbol.
DEF	Line number of program where symbol is defined.
REFERENCES	Line numbers of program where symbol is referenced.

14.1 OPERATING PROCEDURES

Type:

.RUN CREF[,file1,file2,...]

and the CR key.

Where

file1,file2,...	are the parts of a DIBOL program, just as in running COMP (maximum 7). If no files are specified, the Editor scratch area is used.

CREF reads the DIBOL program, produces its table on the line printer and returns control to the COS Monitor.

```
COS DIBOL    01-MAR-73 THUR    COMPILATION LISTING     V12I28  PAGE 01
             DATA DIVISION   S Q U A R E  R O O T  S U B R O U T I N E

0010              START   ; S Q U A R E  R O O T  S U B R O U T I N E
0020
0030              RECORD PRINT
0040     A,       A15
0050     S,       A15
0060
0070              RECORD
0080     X,       D15
0090     SQRTX,   D15
0100     TEMPX,   D15
0110     N,       D3,001
0120
```

Figure 14-1. Square Root Subroutine
(Sheet 1 of 3)

```
COS DIBOL    01-MAR-73 THUR    COMPILATION LISTING     V12;28  PAGE 02
   PROCEDURE  DIVISION

0130     PROC 1
0140
0150             INIT(1,LPT)
0160     LOOP,   X=N*1000000
0170             CALL SQRT
0180             A=N
0190             S = SQRTX,'XXX,XXX'
0200             INCR N
0210             XMIT(1,PRINT)
0220             IF (N,LE,99) GOTO LOOP
0230             STOP
0240
0250     SQRT,   TEMPX=X/2                          !TRIAL VALUE
0260     SQLOOP, SQRTX=(TEMPX+(X/TEMPX))/2          !NEWTON'S METHOD
0270             IF(SQRTX-TEMPX ,EQ, 0) RETURN
0280             TEMPX=SQRTX
0290             GOTO SQLOOP
0300     END
```

Figure 14-1. Square Root Subroutine
(Sheet 2 of 3)

```
COS DIBOL    01-MAR-73 THUR     STORAGE MAP LISTING      V12128  PAGE 03
#         NAME       TYPE        DIM   SIZE     ORIGIN

0001      PRINT      RECORD      01     32      20000
0002      A          ALPHA       01     15      20002
0003      S          ALPHA       01     15      20021
0004      X          DECMAL      01     15      20042
0005      SQRTX      DECMAL      01     15      20061
0006      TEMPX      DECMAL      01     15      20100
0007      N          DECMAL      01     03      20117
0010      ',1        DECMAL      01     01      20122
0011      LOOP       LABEL       00     01      10071
0012      ',1000     DECMAL      01     07      20123
0013      SQRT       LABEL       00     01      10132
0014      ','XXX',   ALPHA       01     07      20132
0015      ',99       DECMAL      01     02      20141
0016      ',2        DECMAL      01     01      20143
0017      SQLOOP     LABEL       00     01      10141
0020      ',0        DECMAL      01     01      20144

0016 SYMBOLS

NO ERRORS DETECTED.  08 K CORE REQUIRED [3940 FREE LOCS = 14 BUFFERS]
```

Figure 14-1. Square Root Subroutine
(Sheet 3 of 3)

```
COS300 CREF              V11,08   THUR  01-MAR-73        PAGE 1

SYMBOL     DEF     REFERENCES

A           40     180
LOOP       160     220
N          110     160    180    200    220
PRINT       30     210
S           50     190
SQLOOP     260     290
SQRT       250     170
SQRTX       90     190    260    270    280
TEMPX      100     250    260    260    270    280
X           80     160    250    260
```

Figure 14-2. CREF of Square Root Subroutine

1	1,000
2	1,414
3	1,732
4	2,000
5	2,236
6	2,449
7	2,645
8	2,828
9	3,000
10	3,162
11	3,316
12	3,464
13	3,605
14	3,741
15	3,872
16	4,000
17	4,123
18	4,242
19	4,358
20	4,472
21	4,582
22	4,690
23	4,795
24	4,898
25	5,000
26	5,099
27	5,196
28	5,291
29	5,385
30	5,477
31	5,567
32	5,656
33	5,744
34	5,830
35	5,916
36	6,000
37	6,082
38	6,164
39	6,244
40	6,324
41	6,403
42	6,480
43	6,557
44	6,633
45	6,708
46	6,782
47	6,855
48	6,928
49	7,000
50	7,071
51	7,141
52	7,211
53	7,280
54	7,348
55	7,416

Figure 14-3. Output of Square Root Subroutine
(Sheet 1 of 2)

56	7,483
57	7,549
58	7,615
59	7,681
60	7,745
61	7,810
62	7,874
63	7,937
64	8,000
65	8,062
66	8,124
67	8,185
68	8,246
69	8,306
70	8,366
71	8,426
72	8,485
73	8,544
74	8,602
75	8,660
76	8,717
77	8,774
78	8,831
79	8,888
80	8,944
81	9,000
82	9,055
83	9,110
84	9,165
85	9,219
86	9,273
87	9,327
88	9,380
89	9,433
90	9,486
91	9,539
92	9,591
93	9,643
94	9,695
95	9,746
96	9,797
97	9,848
98	9,899
99	9,949

Figure 14-3. Output of Square Root Subroutine
(Sheet 2 of 2)

14.2 ERROR MESSAGES

Although CREF expects the source to be error free, certain types of errors are detected.

xxxx IS BEING IGNORED — CREF has encountered a line it cannot interpret.

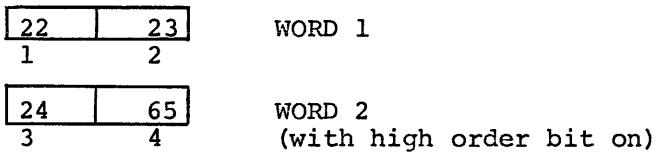
In addition to the above message, CREF prints an informative message:

SYMBOLS DEFINED BUT NOT REFERENCED; xx — Symbols were found that were defined but not referenced by the program.

APPENDIX A

COS CODES

Numbers are stored 2 characters per word in 6-bit binary form (combinations of 1 and 0). 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 form



and would be recognized as 123T. This means that any program where the decimal to alpha conversion is neglected might produce negative numbers with letters. Refer to Table A-1 for a list of equivalent characters for negative numbers.

The sequence of the COS codes is the same as the output from the SORT utility.

TABLE A-1 COS CODES

CHARACTER	EQUIVALENT NEGATIVE NUMBERS	COS CODE	CHARACTER	COS CODE
0		21	!	02
1		22	"	03
2		23	#	04
3		24	$	05
4		25	%	06
5		26	&	07
6		27	'	10
7		30	(	11
8		31	)	12
9		32	*	13
A		42	+	14
B		43	,	15
C		44	-	16
D		45	.	17
E		46	/	20
F		47	:	33
G		50	;	34
H		51	<	35
I		52	=	36
J		53	>	37
K		54		
L		55	?	40
M		56	@	41
N		57	[	74
O		60	\	--
P	-0	61	]	76
Q	-1	62	↑	77
R	-2	63	←	--
S	-3	64	Leader/Trailer	--
T	-4	65	LINE FEED	--
U	-5	66	Carriage Return	--
V	-6	67	Space	01
W	-7	70	RUB OUT	--
X	-8	71	Blank	--
Y	-9	72	Bell	--
Z		73	TAB	75
null		00	FORM	--

APPENDIX B

LOADING COS

The COS system is supplied on DECtape (LINCtape for PDP-12 users). It is suggested that the user copy the COS tape on a certified DECtape (or LINCtape) before using the system (Refer to Chapter 5.) The copy can be used and the original stored in a safe place in case part of the system should be inadvertently destroyed during subsequent use.

The Monitor can be put into core memory in one of three ways. If the computer is equipped with the automatic load feature, the Monitor is loaded by pressing the INITIALIZE switch under the table or by raising the SW switch on the computer console.

Otherwise the Monitor must be loaded into memory using the appropriate Bootstrap routine.

The editing scratch area is not automatically erased during system start-up.

If the system is equipped with an Analex printer, the printer must be turned off until the system is bootstrapped and re-configured.

1.0 AUTOMATIC LOAD

If one of the optional automatic bootstraps

MI8-EC	for the	TC08
MI8-ED	for the	RK8
MI8-EH	for the	TD8E
MI8-E	for the	RF08

was purchased, push the SW switch from down position to up position. Then return it to the down position.

2.0 TC08 DECTAPE AS THE SYSTEM DEVICE

If your system uses DECtape as the system device, use the following procedures:

1. Mount the COS DECtape (DEC-08-OCOSA-D-UO) on DECtape unit 0.

2. Set the LOCAL/OFF/REMOTE switch to REMOTE and the WRITE ENABLE/WRITE LOCK switch to WRITE LOCK. Set SWITCH REGISTER to zero (all switches down) and press EXTD ADDR LOAD.

3. Enter the binary equivalent of the following locations and instructions (refer to Chapter 1 of Introduction to Programming for information on binary numbers). To bootstrap a TC08 system:

LOCATION	INSTRUCTION
7613	6774
7614	1222
7615	6766
7616	6771
7617	5216
7620	1223
7621	5215
7622	0600
7623	0220
7754	7577
7755	7577

Set the SWITCH REGISTER (SR) to enter the first location (7613) and press ADDR LOAD. Set the switches to enter the first instruction and push up on DEPosit switch. Continue entering instructions as described up to and including instruction 0220. Then set 7754 in SR and press ADDR LOAD. Set SR to 7577 and push up on DEP switch twice.

4. Set terminal REM/LOC switch to REM.

5. Set SR to binary equivalent of 7613; press ADDR LOAD, CLEAR and CONT keys in that order. The Monitor should respond with the message

```
COS MONITOR 2.1108      (or current version
                         number)
DATE?
.
```

Enter the DATE command as described in Chapter 2. At this point the SYSGEN program can be used to change the system unit number assignment (refer to SYSGEN, Chapter 3).

Set the switch to WRITE ENABLE.

6. If the Monitor does not respond correctly, start at step 1 and reload the bootstrap routine.

2.1 SYSTEM RESTART ON DECTAPE

If the system halts or CTRL/C is inoperative, use the automatic bootstrap switch to restart the system. Each time the Monitor is reloaded, the old date is erased and the Monitor requests a new date. Use the DATE command to enter the new date.

For users without a bootstrap switch, set SR to 7600 and press LOAD ADDR, CLEAR and CONT, assuming the core resident monitor has not been destroyed.

3.0 DISK AS THE SYSTEM DEVICE

If your system uses an RK08, RK8E, or RF08 disk as the system device, the programs must be loaded on the disk from the DECtape supplied.

1. Follow steps 1-6 described in section 2.0 on DECtape as a system device.

2. Enter the DATE command in response to the Monitor's DATE? message. Then enter the command:

   .RU SYSGEN/C

   in response to the Monitor's period. SYSGEN asks the following questions:

   DO YOU DESIRE TO CHANGE THE SYSTEM DEVICE?

   Type YES and a CR key.

   WHAT IS THE NEW SYSTEM DEVICE

   Type the 3 character designation for the disk system device (DKn) and the CR key.

   DO YOU WANT TO TRANSFER YOUR FILES?

   Type YES and the CR key.

   DO YOU HAVE AN ANELEX PRINTER?

   Type YES and the CR key and SYSGEN skips to the last question. Type NO and the CR key and SYSGEN assumes an LP08 or LS8E printer and asks:

   132 COLUMN PRINTER ?

   Answer YES and the CR key if there is a 132-column line printer. If NO is the answer, SYSGEN assumes an 80-column printer.

   IS EVERYTHING CORRECT?

   Type YES or NO and the CR key. If NO is entered the questions are reasked.

The files are transferred to the disk specified. To operate from the disk as the system device, refer to BOOT, Chapter 9, or restart the system with one of the following bootstraps.

RK08 AND RK8E BOOTSTRAP ROUTINE

Location	RK08 Instruction	RK8E Instruction
0030	6733	6743
0031	5031	5031

Set the SWITCH REGISTER to 0030, press the ADDR LOAD, CLEAR, and CONT keys in that order.

NOTE

If a PDP-12 is being used, execute an I/O PRESET in 8-mode before starting at location 30.

The Monitor should respond with the version number and the DATE? message. Enter the DATE command as described in Chapter 2.

RF08 BOOTSTRAP ROUTINE

Location	Instruction
7750	7600
7751	6603
7752	6622
7753	5352
7754	5752

Set SR to 7750; press ADDR LOAD, CLEAR, and CONT in that order. The Monitor should respond with the version number and the DATE? message. Enter the DATE command as described in Chapter 2.

3.1 SYSTEM RESTART ON DISK

If the system halts and CTRL/C is inoperative, use one of the bootstrap routines to restart the system. Each time the Monitor is restarted the Monitor requests a new date. Use the DATE command to enter the new date.

For users without the bootstrap switch if the core resident portion of the Monitor is intact, set SR to 7600 and press LOAD ADDR, CLEAR and CONT.

4.0 TD8E BOOTSTRAP

If your system uses TD8E's use the following bootstrap routine to load the COS software.

1.

Location	Instruction
7300	1312
7301	4312
7302	4312
7303	6773
7304	5303
7305	6777
7306	3726
7307	2326

7310	5303
7311	5731
7312	2000
7313	1300
7314	6774
7315	6771
7316	5315
7317	6776
7320	0332
7321	1327
7322	7640
7323	5315
7324	2321
7325	5712
7326	7354
7327	7756
7330	7747
7331	7400
7332	0077

Mount the COS 300 DECtape (DEC-8E-OCOSA-D-UO) on DECtape unit 0. Set the LOCK/OFF/REMOTE switch to REMOTE and the WRITE ENABLE/WRITE LOCK switch to WRITE LOCK. Set the switch register (SR) to zero (all switches down) and press EXTD ADDR LOAD.

Set the SR to the binary equivalent of the first location and press ADDR LOAD. Set the switches to the binary equivalent of the first instruction and push up on the DEP switch. Continue entering the instructions as described, the location is incremented automatically and need not be entered again.

2. Set terminal REM/LOC switch to REM.

3. Set SR to binary equivalent of the first location; press ADDR LOAD, CLEAR and CONT keys in that order. The Monitor should respond with the message

```
COS MONITOR 2.1108      (or current
                         version)
DATE?
.
```

WRITE ENABLE unit 0 and enter the DATE command as described in Chapter 2.

4. If Monitor does not respond correctly start at step 1 and reload the bootstrap.

5.0 PDP-12 USERS

For a PDP-12 with LINCtape,

1. Set left switches to 0700.
2. Set right switches to 0000.
3. Press I/O PRESET with mode switch set to LINCmode.
4. Mount system tape (DEC-12-OCOSA-D-UO) on drive 0, WRITE LOCKED, REMOTE.
5. Press DO.
6. When LINCtape stops moving, press the START 20 switch.

   Monitor displays

   ```
   COS MONITOR 2.1108     (or current version)
   DATE?
   .
   ```

7. WRITE ENABLE unit 0 and enter the DATE command as described in Chapter 2.
8. If Monitor does not respond correctly, start at step 1 and reload the system.

Any LINCtapes to be used with the COS system must be marked with MARK12 (DEC-12-SE2E-UO) (P option, 129 words).

APPENDIX C

SIZE OF CODE PRODUCED BY DIBOL COMPILER

DIBOL source statements are compiled by the DIBOL compiler and produce code which can later be executed by the run time system (RSYS). Memory is also set aside for the constants and variables that are used by the program.

Each variable (field name) takes up as much memory (in bytes) as specified in its data specification statement. For example, a variable defined by a 6D3 specification requires 18 (decimal) bytes of storage. This is 9 words since a computer word consists of 2 bytes.

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

Each RECORD statement requires one additional word of storage. This word is reserved at the beginning of the record for storing the COS word count during I/O operations.

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

Each constant used in the procedure section of a DIBOL program requires storage (in bytes) equal to the length of the constant. The length of a decimal constant is equal to the number of digits in the number, including leading zeros. There are no negative constants. The length of a character string is the number of characters in it.

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

The code produced by statements in the procedure section must start on a word boundary. Each statement requires an overhead of one word. Each statement label used requires one word. Unlabeled statements with line numbers 1000 more than the previous line number require one additional word each.

The number of words of code generated by an expression can be determined by the following rule:

1. Add together the number of variables and constants used.

2. Add in the number of operators which appear (do not count unary + signs). The operators include +, -, /, *, #.

3. Add one for each subscript reference (be it of the form V(A) or V(A,B)).

The following table shows how much code is used by various DIBOL statements. E denotes the number of words required by the expression E.

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

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

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

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

APPENDIX D

DESIGNING A RECORD

The file design begins with the set of related information shown in blank form. It is a simplified form which could be processed to print out customer invoices.

```
DATE
PART#          PART NAME
WHOLESALE PRICE
VENDOR#
```

This information is used in a BUILD control program such as described in Chapter 6. First, the file designer notes the type and size of each field in this record. Numerical fields on which calculations will be performed later by a program are defined as decimal fields; other fields are defined as alphanumeric fields.

```
0010    DEFINE
0020    F1, D6 ;PART #
0030    F2, A30 ;PART NAME
0040    F3, D7 ;VENDOR #
0050    F4, D6 ;DATE
0060    F5, D6 ;WHOLESALE PRICE
```

The designer now assigns BUILD input line keywords. Here, he should group related items such as part #, name and wholesale price.

```
0100    INPUT
0110    PART
0120    F1
0130    F2
0140    F5
0150    DATE
0160    F4
0170    VENDOR 1
0180    F3
```

A sample data entry form is now completed, showing the line names and the size and type of each field. This can be used for reference when data is entered. When this record design has been proven, it will be more efficient to print these line names on the forms which are used to collect the original data.

```
DATE (D6)_____
PART# (D6)_____NAME(A30)_____WHOLESALE PRICE (D6)_____
VENDOR# (D7)_____
```

APPENDIX E

BUILD CHECKDIGIT FORMULA

In most applications involving identification numbers, each number may be verified for accuracy by a checkdigit, which is essentially a redundant digit added to the normal number. The checkdigit is determined by performing an arithmetic operation on the number in such a way that the usual errors encountered in transcribing a number are detected. The checkdigit is determined as follows:

1. Start with a number without the checkdigit...5764.

2. Multiply the first digit and every other digit by 2 (left to right). 10 12

3. Sum the digits in the resulting numbers and the digits not multiplied. 1+0+7+1+2+4 =15

4. Subtract sum from next higher number ending in zero. 20-15=5

5. Add checkdigit to the end of the number. 57645
   (This is the correct checkdigit if the number is entered in a D4 field.)

Note that a checkdigit procedure is not completely error proof. In the example given above, 5764 or 5673 give the same checkdigit. It is unlikely, however, that transpositions of this sort will occur. The checkdigit does not guard against the possible assignment of an incorrect but valid code, such as the assignment of a wrong valid identification code to a customer.

If the number entered for a checkdigit calculation is shorter than the field, the rightmost digit is used as the checkdigit and the remainder of the number is right-justified and padded with zeroes on the left. The zeroes are considered when the checkdigit formula is calculated.

APPENDIX F

HARDWARE DESCRIPTIONS

The hardware components for COS are:

PDP-8/E	Computer with at least 8K of core
TD8E	DECtape drives, (or TC08 drives, or RK08, RK8E, or RF08/RS08 disk)
VT05	Terminal
PC8-E	Paper tape Reader/Punch (optional)
CR8-E	Card reader (optional) or
CM8-E	Card reader (optional)
LP08	Line printer or
LS8-E	Centronics line printer

The above hardware components are described in the following paragraphs.

Figure F-1 shows one of the possible system configurations.

The central processor unit (CPU) of the COS system is located under the desk area of the system. There are two switches necessary for the operation of the COS system.

OFF/POWER Switch	In the OFF position the switch disconnects all primary power to the machine. In the ON position all manual controls are enabled and primary computer power is applied.
INITIALIZE Switch	Push this switch to initialize the system Bootstrap Loader.

Additional information on the CPU can be obtained from Introduction to Programming.

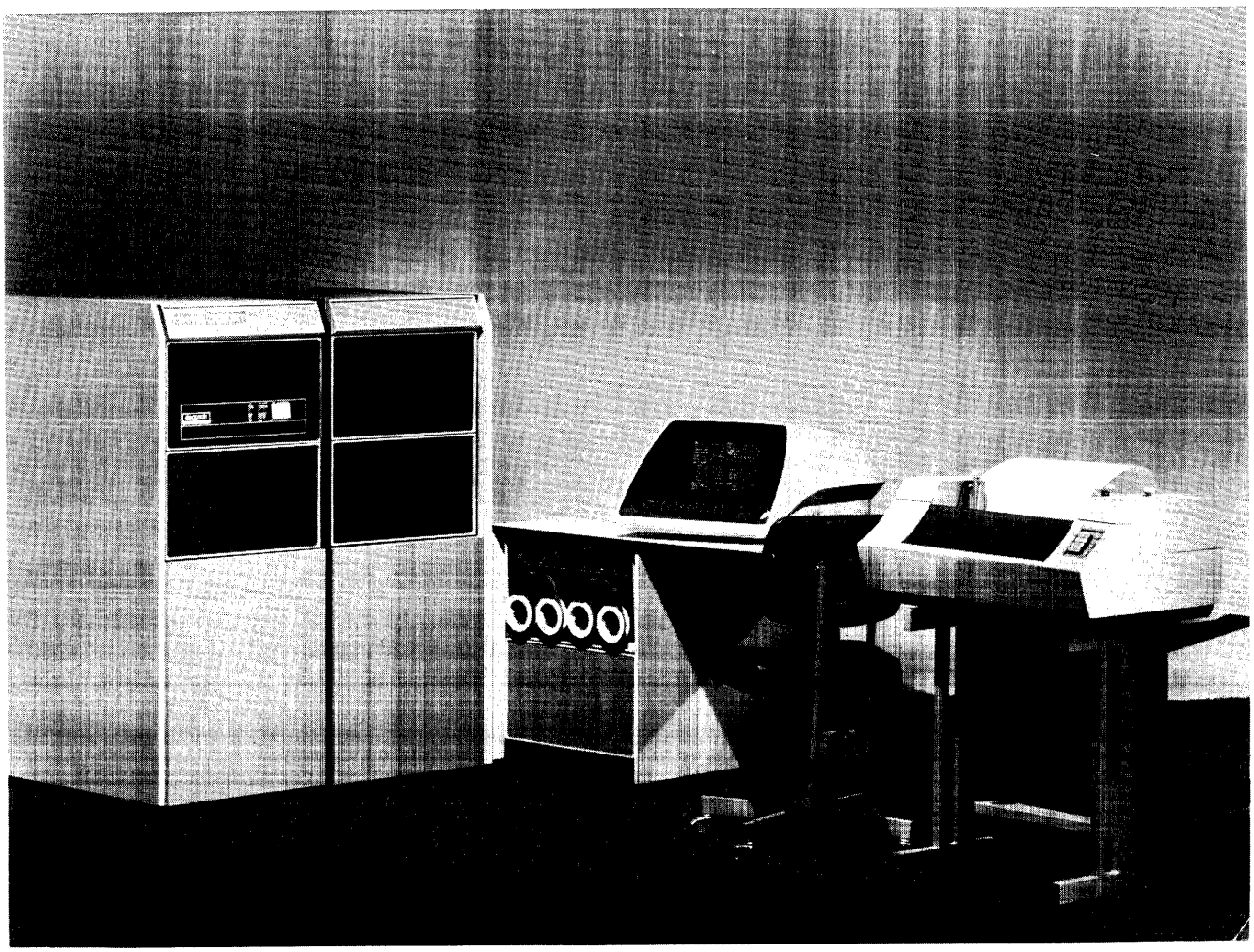


Figure F-1. Possible COS Hardware Configuration

1.0 VT05 TERMINAL

Basically, the VT05 terminal (Figure F-2) consists of a keyboard and CRT display.

The basic function of the keyboard is to provide a convenient, on-line means of transmitting ASCII characters to the computer to be processed and displayed. An ASCII code is transmitted to the computer each time a key is depressed.

The VT05 display allows direct viewing of all displayable characters contained in the VT05 character set that are transmitted from and/or received by the VT05.

The VT05 displays up to 20 lines, 72 characters per line, or a maximum of 1440 characters full screen. A blinking cursor indicates the position of the next character to be generated and moves to the right automatically as each character is displayed. The VT05 can be operated in LOCal or REMote mode. In LOCal mode the terminal is off line and there is no communication with the computer. In REMote mode (the normal operating mode) data is transmitted between the terminal and the computer.

VT05 controls and keys and their respective functions are listed in Tables F-1 and F-2. Keys which are operative off-line only are not covered here; refer to the hardware manual for the VT05 for a description of their function.

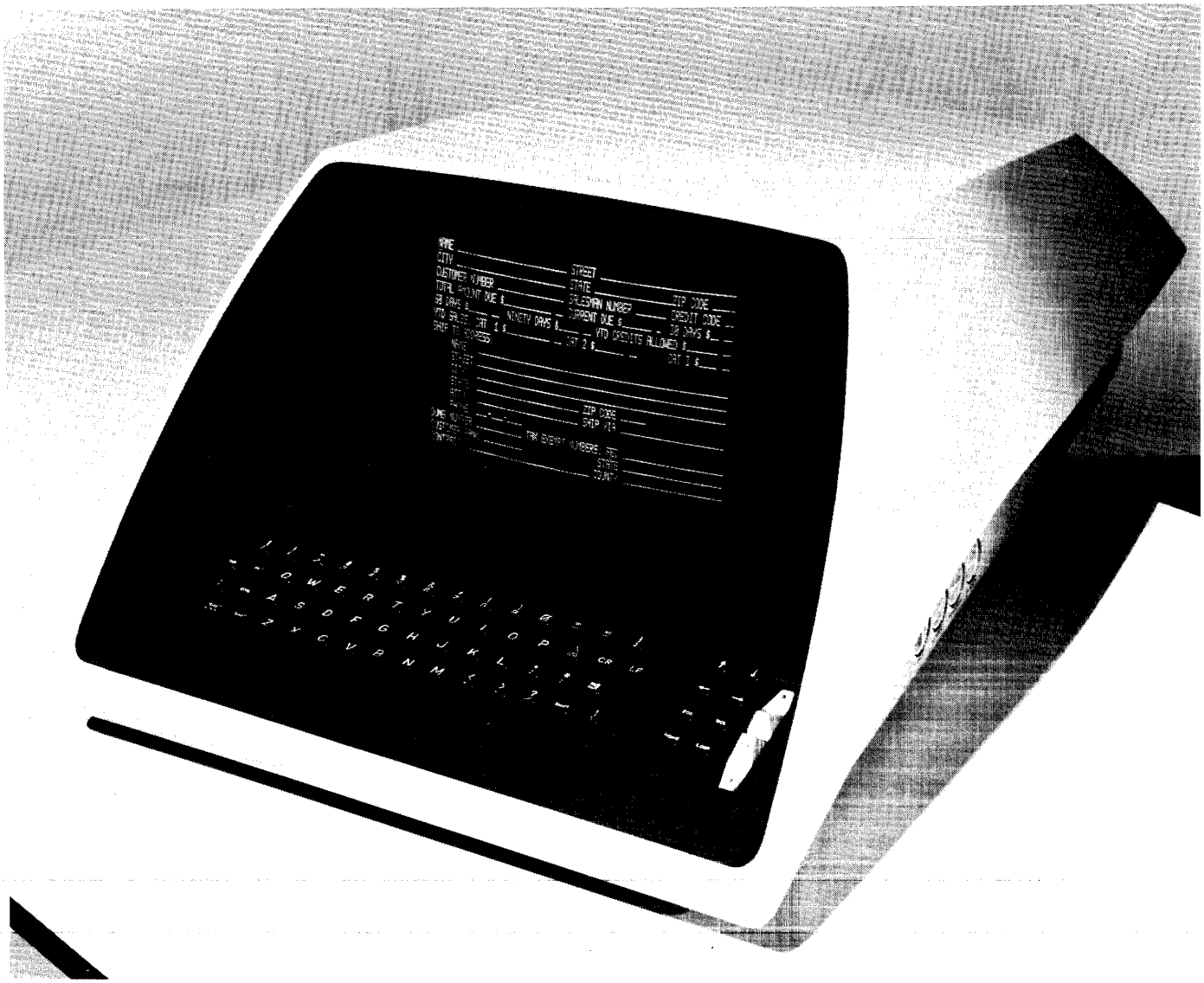


Figure F-2. VT05 Terminal

TABLE F-1. VT05 CONTROLS

Control	Function
Power ON/OFF Switch	This switch is located on the right front of the VT05. When the ON/OFF switch is turned ON, power is applied to the complete system and the VT05 display refresh memory is cleared. After approximately one minute has elapsed, the cursor (bright dash should appear in the upper left-hand corner (HOME position) of the screen.
REM/LOC Switch	This switch is located on the right front of the VT05. In LOCal mode, the terminal is off-line and data can be typed at the keyboard but not entered into memory. In REMote mode, keyboard data is transmitted from the VT05 to the processor. NOTE The CONTRAST, BRIGHTNESS, VERTICAL and HORIZONTAL controls are adjusted in the same way as those of a commercial television receiver.
CONTRAST Control	The CONTRAST control is located on the right-hand side of the VT05 and is used to adjust for display contrast (clarity).
BRIGHTNESS Control	The BRIGHTNESS control is located on the right-hand side of the VT05 and is used to adjust for CRT display brightness or intensity. NOTE To correctly adjust the VT05 character presentation, turn the CONTRAST control counterclockwise to minimum, then adjust the BRIGHTNESS control decreasing the intensity until the raster brightness is barely intensified (just above the CRT cutoff point). As a final step, adjust the CONTRAST control to the desired level, according to ambient lighting conditions.
VERTICAL Sync Control	The VERTICAL sync control is located on the right-hand side of the VT05 and is used to properly synchronize the screen in the vertical direction.
HORIZONTAL Sync Control	The HORIZONTAL sync control is located on the right-hand side of the VT05 and is used to adjust the screen picture for proper synchronization in the horizontal direction.

TABLE F-2. VT05 KEYS

Key	Function
CR	Returns cursor to the left margin of screen.
LF	Moves cursor down one line in same location. If pressed when cursor is on the bottom line (line 20), the display scrolls (top line disappears and rest of lines move up one line position).
RUBOUT	Works in REMote mode in conjunction with the Monitor to erase characters. Erases the character above the cursor from memory and echoes the deleted character on the screen, i.e., 
TAB	Moves the cursor to the right to the next TAB stop each time TAB is pressed.
ALT	Has no effect on the display and is ignored by the terminal. It is included to provide an alternate escape character (replaces Carriage Return or Line Feed) for use in the user program.
CTRL	Used simultaneously with character keys to perform special functions as follows:
CTRL/C	Causes an immediate return to the Monitor.
CTRL/N	Enables and disables imbedded numeric keypad. See Table 2-1 in Chapter 2 for details.
CTRL/O	Suppresses terminal output. If output is already suppressed, enables terminal output.
CTRL/U	Erases the current input line.
CTRL/V	Suppresses Monitor version number message. If output is already suppressed, enables output of message.
CTRL/Z	Indicates no more input. When used in answer to an ENTER message it forces the program to use a default value. When specified while editing, it signifies end of paper tape or card deck.

TABLE F-2. VT05 KEYS (Con't)

Key	Function
SHIFT	Produces the character in upper case portion of the key pressed simultaneously with SHIFT, e.g., ! rather than 1. Control characters are "SHIFT inhibited" and alpha characters are permanently set to upper case.
SHIFT LOCK	Enables and holds the shift function or releases it if already set. When enabled the SHIFT LOCK key is lit.
Space	Produces a blank character position (wherever the cursor is located) each time it is pressed. Cursor moves one position to the right.

Additional features:

The VT05 sounds a warning beep when the line reaches character position 65. This alerts the user that there are 8 positions before the end of the line. The cursor moves to the right automatically as each character is displayed until it reaches position 72.

1.1 VT05 Start Up Procedures

1. Press the OFF portion of the ON/OFF power switch.

2. For local operation (used for training, maintenance and alignment purposes) press the LOCal portion of the REM/LOC switch.

   For on-line operation, press the REMote portion of the REM/LOC switch.

3. Press the ON portion of the ON/OFF power switch. The blinking cursor (bright dash) appears in the HOME position (the first character position of the top line), and a beep is emitted.

4. Allow approximately one minute for the CRT filament warmup.

5. If the cursor does not appear as specified, press the HOME key.

   If the cursor still does not appear, check the BRIGHTNESS control to be sure it is not set too low. Contact the local DEC Field Service Office if these steps do not make the cursor appear.

6. In REMote mode all keyboard and control functions are operational and the VT05 is ready for on-line operation. To verify correct system operation, press the various controls and displayable character keys individually and observe the display. In LOCal mode all keyboard functions are operational. To verify this, press the keys individually and observe the screen.

2.0 HIGH-SPEED PAPER TAPE READER AND PUNCH UNIT

The high-speed paper tape reader and punch unit shown in Figure F-3 performs paper tape input and output. The high-speed paper tape reader is used to input data into core memory from eight-channel, fan-folded (non-oiled), perforated paper tape. The reader inputs information photoelectrically at a rate of 300 characters per second.

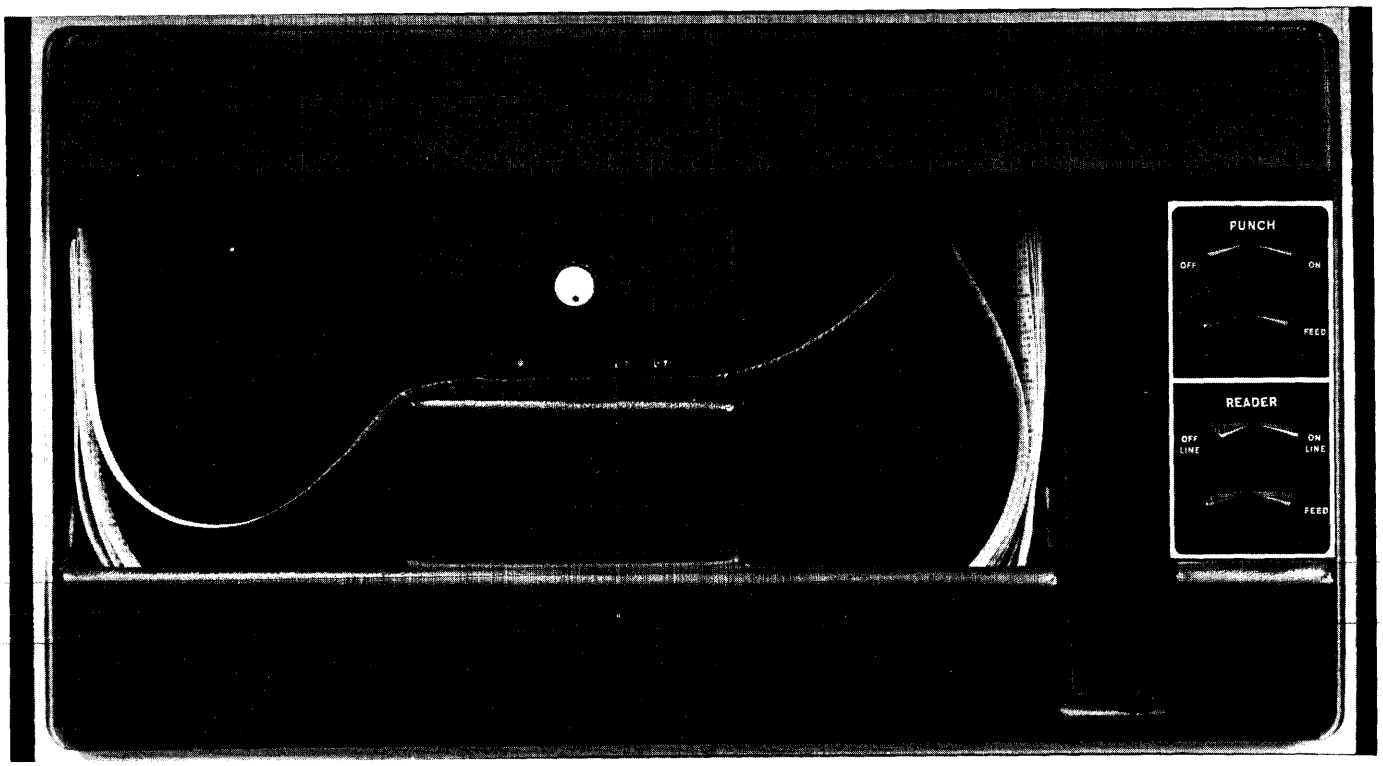


Figure F-3. High-Speed Paper Tape Unit

The reader controls are shown in Figure F-3 and described in Table F-3.

TABLE F-3. HIGH-SPEED READER CONTROLS

Control	Function
ON/OFF	Press down on the ON portion of this switch to supply power to the reader unit. The OFF position setting disconnects the reader power supply.
FEED	Press the right side of this switch to advance tape through the reader without recording by the photoelectric sensors.
Control knob	Turn this knob to the left to raise the tape retainer lever so paper tape can be inserted or removed. Turn the knob to the right to lower the lever to hold the paper tape over the sprocket wheel.

The reader is controlled by the computer, although the operator may indirectly control the reader from the keyboard through the computer.

Paper tapes are manually positioned in the high-speed reader with the following steps.

1. Turn the control knob to raise the tape retainer lever.
2. Place the paper tape in the right-hand bin so the beginning of the tape passes over the sensors first.
3. Place several folds of leader tape in the left-hand bin with the tape passing over the sprocket wheel.
4. Turn the control knob to lower the retainer lever over the tape so the feed holes are engaged in the teeth of the sprocket wheel.
5. Press the tape FEED button until leader tape is over the reader head.
6. Tape is advanced and read by programmed computer instructions.

Once the paper tape has been properly placed in the reader and the leader/trailer has been positioned as outlined in the preceding steps, the tape is normally read under control of system software.

The high-speed paper tape punch is used to record computer output on eight-channel, fan-folded paper tape at 50 characters per second. All characters are punched under program control from the computer.

The punch controls are shown in Figure F-3 and are described in Table F-4.

TABLE F-4. HIGH-SPEED PUNCH CONTROLS

Control	Function
ON/OFF	Press down on the ON portion of this switch to supply power to the punch unit. The OFF position setting disconnects the punch power supply.
FEED	Press the right side of this switch to advance feed-hole-only punched tape for leader/trailer purposes.

3.0 DECTAPE TRANSPORT UNIT

The DECtape transport unit shown in Figure F-4 is a bidirectional magnetic tape transport which reads and writes the 10-channel magnetic tape. Tape movement can be controlled by programmed instructions from the computer or by the manual operation of switches located on the front panel of the transport. Data is transferred only under program control.

The transport controls are described in Table F-5.

TABLE F-5. DECTAPE CONTROLS

Control	Function
REMOTE	This switch position energizes the DECtape transport and places it under program control. The REMOTE SELECT light comes on whenever the unit is selected for use.
OFF	This switch position disables the DECtape transport.
LOCAL	This switch position energizes the DECtape transport and places it under operator control from external transport switches.
WRITE ENABLE	This switch position enables the DECtape for search, read, and write activities. The WRITE light comes on when the switch is put in the WRITE ENABLE position.
WRITE LOCK	This switch position limits the DECtape transport to search and read activities only. (This prevents accidental destruction of permanent data.)

TABLE F-5. DECTAPE CONTROLS (Con't)

Control	Function
Unit Selector	The value specified by this eight-position dial (0-7) identifies the transport to the control unit. On TD8E devices, the first controller is hardwired to accept only 0 or 1 and the second controller only 2 or 3.
	With the transport in LOCAL mode, depressing this switch causes tape to feed onto the right-hand spool.
	With the transport in LOCAL mode, depressing this switch causes tape to feed onto the left-hand spool.

DECtapes are mounted on the transport unit as follows:

1. Set REMOTE/OFF/LOCAL switch to OFF.
2. Place DECtape on left spindle with DECtape label out.
3. Wind four turns on empty DECtape reel on right spindle.
4. Set switch to LOCAL.
5. Depress the  switch for a few seconds to make sure tape is properly mounted.
6. Dial correct unit number on unit selector.
7. Set switch to REMOTE. Select either WRITE ENABLE or WRITE LOCK setting.

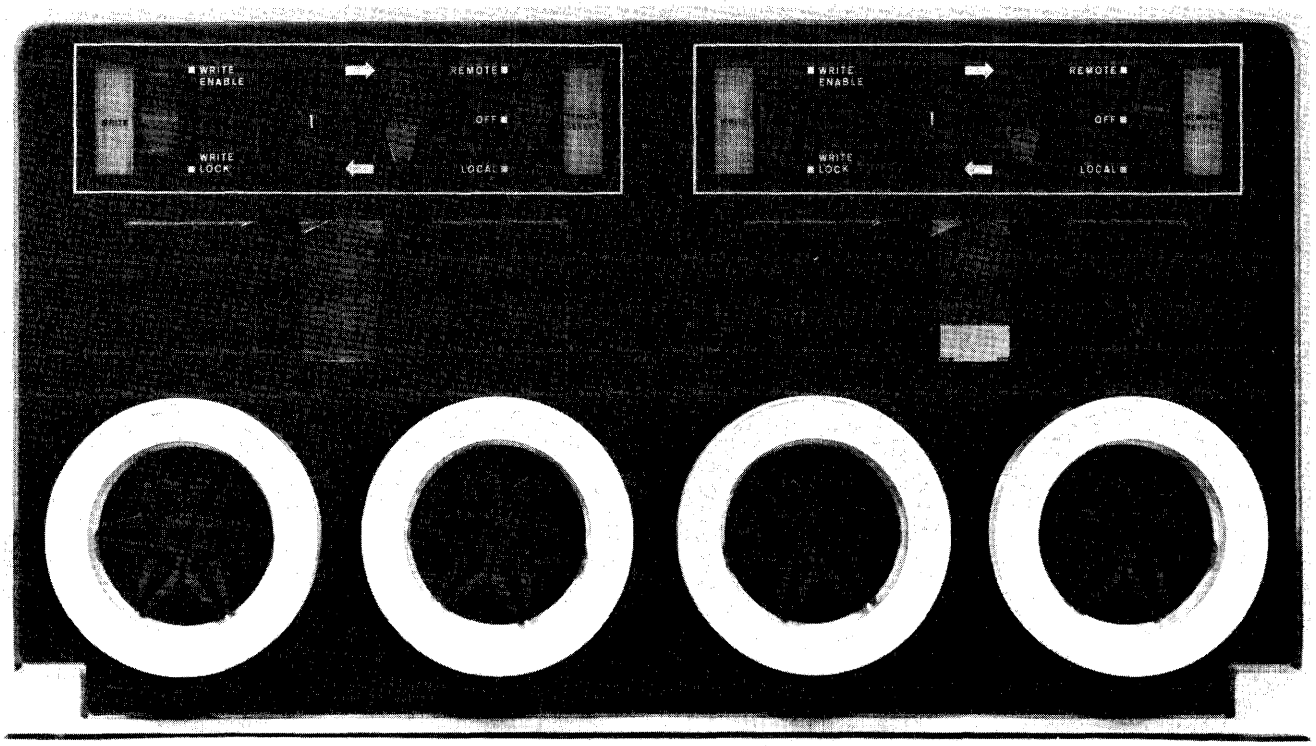


Figure F-4. DECtape Transport Unit

4.0 CARD READER

There are two types of card readers, the CR8-E which reads 12 - row, 80 column punched cards and the CM8-E which reads 12 - row, 40 column mark sense cards or 12 - row, 40 column punched data cards with timing marks.

The CR8-E card reader switches and indicators are described in Table F-6.

TABLE F-6. CARD READER SWITCHES AND INDICATORS

Switch/Indicator	Function
POWER (toggle circuit breaker and indicator)	All power to the card reader is controlled by this switch.
STOP (momentary-action pushbutton/indicator switch)	Actuation of the STOP switch immediately overrides the PICK COMMAND and negates the READY status. The card reader will stop operation after the card currently in the track is read completely. Power is not removed from the reader by this action. The red STOP indicator is illuminated as soon as the switch is depressed.
RESET (momentary-action pushbutton/indicator switch)	Actuation of the RESET switch clears all error logic and initializes all counters.
Read Check (indicator)	The READ CHECK alarm indicator denotes that the card just read may be torn on the leading or trailing edges or have punches in the 0 or 81st columns. A READ CHECK error will cause the reader to stop.
Stack Check (indicator)	The STACK CHECK alarm indicator signals that a card has failed to reach the read station after a PICK COMMAND has been received.
Hopper Check (indicator)	The HOPPER CHECK alarm indicator denotes that either the input hopper is empty or the stacker is full.

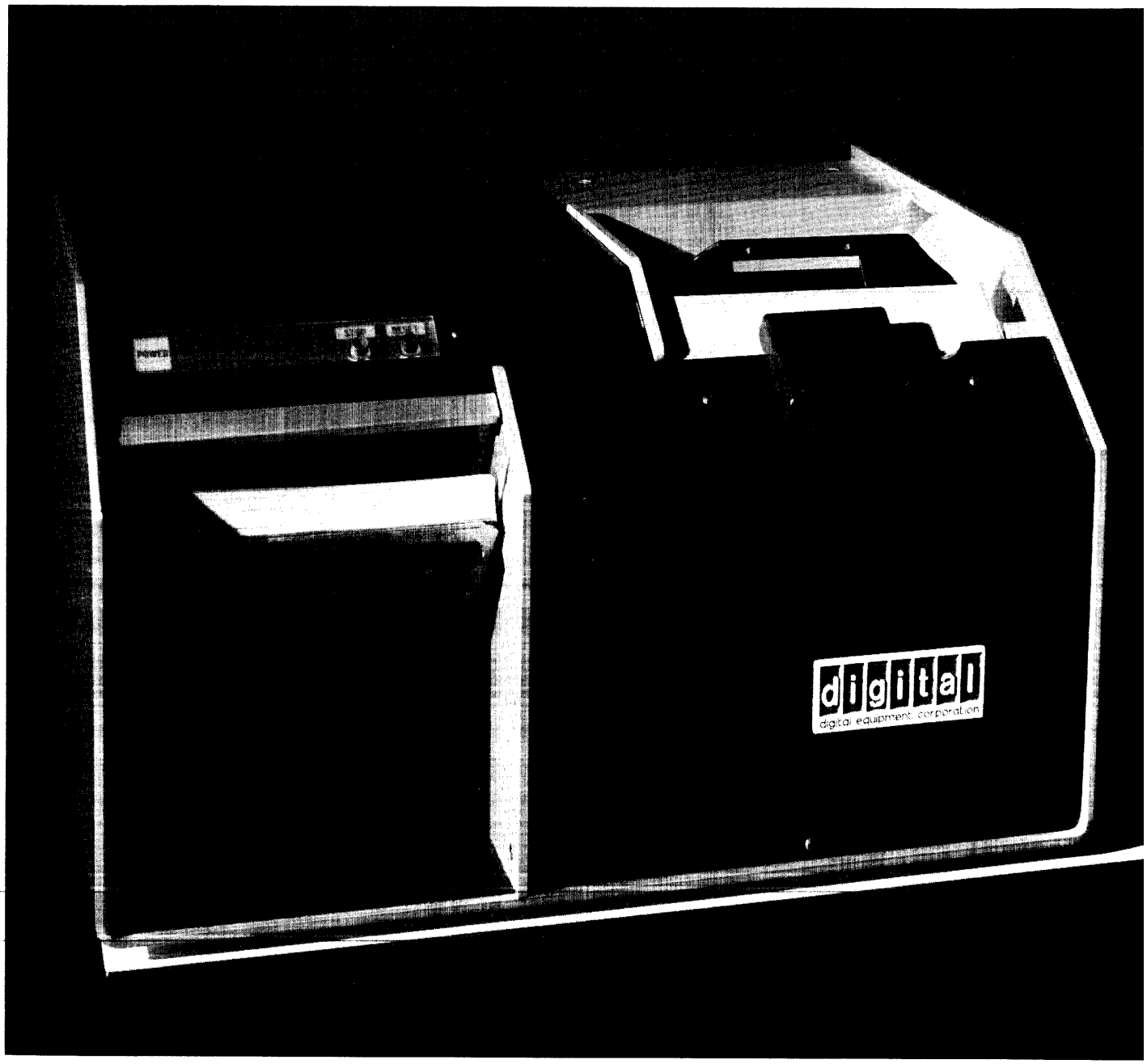


Figure F-5. Card Reader

5.0 LINE PRINTERS

5.1 LP08 Line Printer

The switches and lights described in Table F-7 are part of the LP08 line printer, Figure F-6.

TABLE F-7. LP08 SWITCHES AND INDICATOR LIGHTS

Switch/Indicator	Functions
ON LINE/OFF LINE (switch)	Push up to put printer on line with the computer.
PAPER STEP (switch)	Push up to advance paper one line; disabled in on-line mode.
TOP OF FORM (switch)	Push up to advance paper to top-of-form; disabled in on-line mode.
ON LINE (indicator)	Lights when ON LINE/OFF LINE switch is set to ON LINE.
READY (indicator)	Lights when printer is ready to receive computer output.

The ON/OFF switch controls the power to the printer.

There are two LP08 line printers: 80- and 132-column versions.

Figure F-6. LP08 80-Column Line Printer

5.2 LS8-E Line Printer

The basic character set for the LS8-E (Figure F-7) contains the 10 numeric digits, 0 through 9; 26 upper case letters, A through Z; and the 28 special characters shown below:

"	'
:	+
.	;
$	=
,	[
#	]
	?
*	&
%	-
@	/
(	>
)	\
!	^
(space)	_

The LS8-E controls and their functions are shown in Table F-8.

TABLE F-8. LS8-E CONTROLS

Control	Function
ON/OFF (switch)	Provides power to the printer and lights when in the ON position.
SELECT (switch)	Enables and halts printer and lights in the on position.
TOP OF FORM (switch)	Moves line printer paper to top of sheet or proper location of the vertical forms control tape (Refer to Centronics Manual) when installing forms.
FORMS OVERRIDE (switch)	Allows completion of form being printed even though paper out switch was activated.
HARDWARE ALARM (light)	Signals when head carrier has exceeded the right margin stop. An audio alarm also sounds. If this should happen, reset the printer and restart the job.
PAPER OUT (light)	Indicates printer is out of paper or paper handling malfunction. Correct the problem and restart the job.
Manual Controls	
Paper Advance Knob	Located on left side of carriage; provides manual forward and reverse paper positioning.

TABLE F-8. LS8-E CONTROLS (Con't)

Control	Function
Paper Feed Level	Located on rear, right top of printer; provides initial paper feed around platen to allow positioning of paper in the tractor feed mechanism.
Forms Thickness Control	Located on inside of printer, this dial with graduated markings provides adjustment of clearance between platen and face of print head. Adjustment must be made according to the thickness of the forms being used.

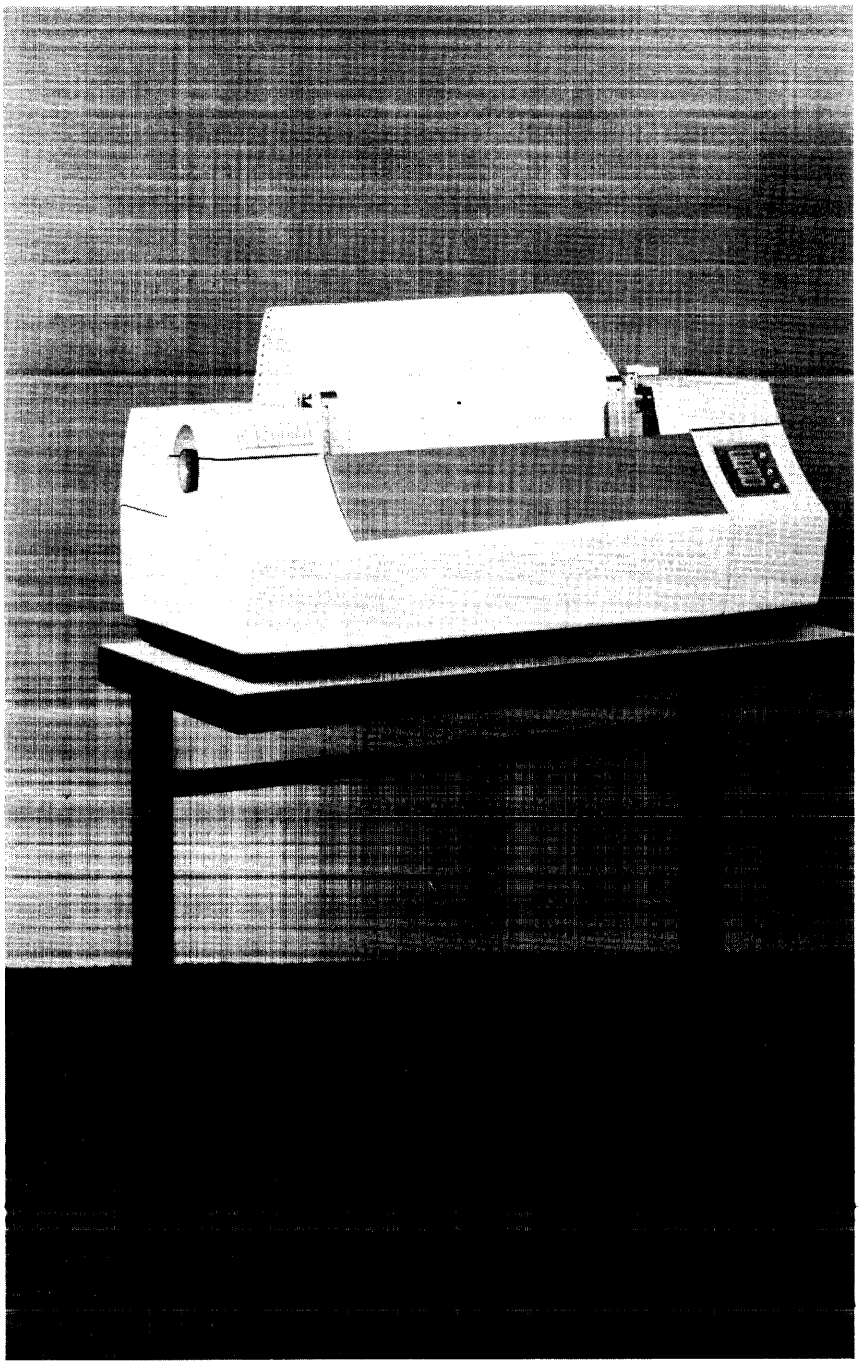


Figure F-7. LS8E Printer

6.0 RK08 DISK

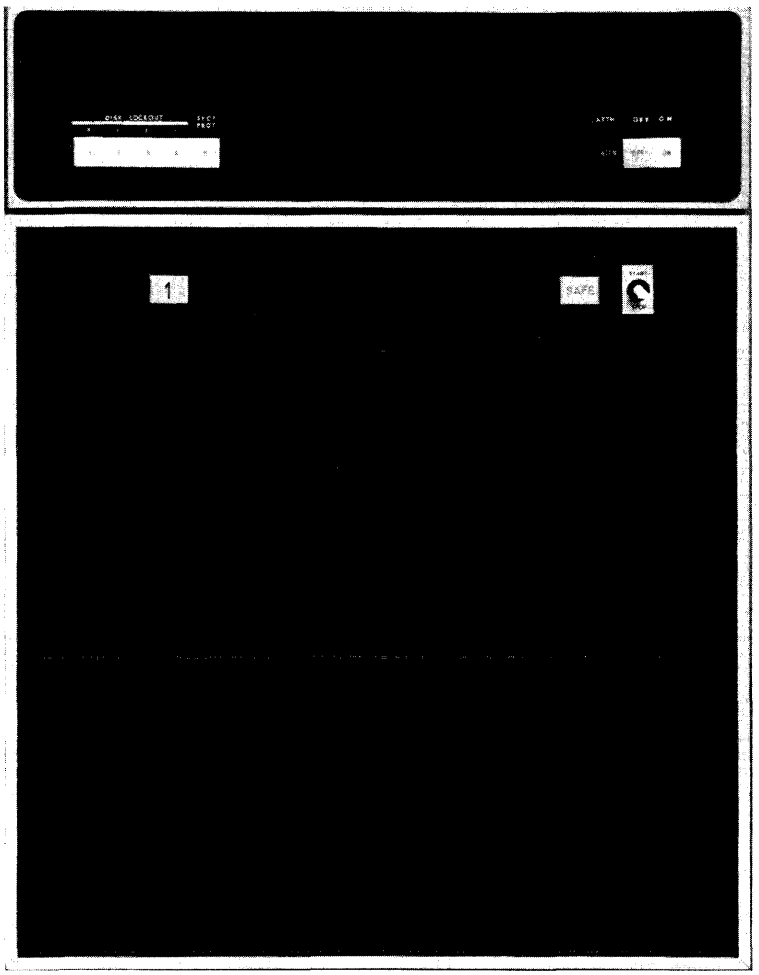


Figure F-8. RK08 Disk

The switches, buttons and lights described in Table F-9 control the operation of the RK08 Disk. (See Figure F-8.) An RK8E disk is also available.

TABLE F-9. RK08 CONTROLS AND INDICATORS

Control	Function
ON (pushbutton and indicator)	Push ON button to supply power to the disk. ON button lights when power is on.
OFF (pushbutton and indicator)	Push OFF button to halt the power supply to the disk. OFF button lights when power is off.
ATTN (pushbutton and indicator)	Indicates that the disk power supply has a malfunction. Push the STOP switch to reset the power supply. NOTE This button also lights when system power is turned off with the computer ON/OFF switch. Push this button to reset the power supply. This inter-locking action protects the data on the disk from spurious write signals while computer and disk controls are powered down.
DISK LOCKOUT 0-3 (pushbutton and indicator)	Press the appropriate button(s) (0-3) to lock out the corresponding disk so data cannot be written onto the disk platter. The button lights when pressed. The panel controls up to four disks.
SECT PROT	Prevents writing on protected sectors, as specified by bit 0 of the second header word of the disk sectors. Press to protect sectors of the disk previously coded for sector protect by setting bit 0 of the sectors 2nd header word to 1. (not implemented in COS)
START/STOP (switch)	Place in the START or up position to initiate the disk starting cycle (only if the disk cartridge is in place and the disk cartridge receiver handle is in the raised position). Place in the STOP or down position, to initiate the disk STOP sequence. Switch must be in the STOP position to mount or dismount the disk cartridge.

TABLE F-9. RK08 CONTROLS AND INDICATORS (Con't)

Control	Function
SAFE (light)	The SAFE light comes on approximately 35 seconds after a STOP sequence has been initiated. When SAFE is lit, the disk cartridge access door interlock is off, and it is safe to lower the disk cartridge receiver handle for the removal or insertion of the disk cartridge.
0,1,2 or 3 (unit #) (light)	The READY light (left-hand light) comes on approximately 90 seconds after the START sequence has been initiated. When READY is lit the disk is ready to begin communicating with the computer.

6.1 Mounting and Dismounting the RK08 Disk Cartridge

6.1.1 Mounting

Follow the steps below to initiate operation of the RK08 disk.

The DC power must always be ON before inserting a disk cartridge into the disk. If the disk cartridge is inserted without first turning the DC power on, the read/write head carriage assembly may be in the way.

1. Press the disk power supply DC ON switch. The ON light comes on.

2. When the SAFE lamp, on the front control panel of the disk, comes on and the handle lock unlatches, the disk is ready to receive the disk cartridge.

3. Pull open the disk cartridge access door panel. The cartridge receiver is now raised into position for insertion of the disk cartridge. The handle interlock will keep the cartridge receiver handle locked in its closed position whenever the SAFE lamp is turned off.

4. Slowly slide the disk cartridge into the receiver assembly and make certain there is no internal resistance to its insertion.

NOTE

If there is the slightest resistance to the insertion of the disk cartridge into the disk receiver, the cartridge should be removed and the interior of the disk receiver examined to determine the cause of resistance.

5. Raise the disk cartridge handle to the up or loaded position. Do not force the handle.

6. Close the disk cartridge access door.

NOTE

An interlock switch is activated when the disk cartridge is properly inserted. However, if the switch is not activated, the drive is not put in the ready condition. If the switch should transfer to its inoperative position during an otherwise normal operation, the disk will immediately stop and the SAFE light will light.

7. Set the START/STOP switch to the START or up position to put the disk drive system into operation. A disk handle interlock activates to ensure that the handle cannot be raised during normal operation.

8. After a 90 second delay the READY lamp comes on, indicating that it is ready for actual operation.

6.1.2 Dismounting

Follow these steps to remove the disk cartridge.

1. Make certain the DC power is ON.

2. Set the START/STOP switch to the STOP or down position. The disk decelerates and the SAFE light comes on.

3. Open the disk cartridge access door and pull the cartridge receiver handle down.

4. Remove the disk cartridge from the cartridge receiver assembly.

5. Close the disk cartridge access door.

6.2 Mounting and Dismounting the RK8E Disk Cartridge

6.2.1 Mounting

The following is the correct cartridge mounting procedure for the RK05 disk drive. Any deviation encountered during this procedure will be considered an error condition.

1. Set switch labeled RUN/LOAD to the LOAD position.

2. Verify that light labeled PWR is on.

3. Wait for light labeled LOAD to come on.

4. Verify that lights labeled RDY, ON CYL, FAULT, WT, and RD are off.

5. Open access door.

6. Insert cartridge.

7. Close access door.

8. Set switch labeled RUN/LOAD to the RUN position.

9. Wait for lights labeled RDY and ON CYL to come on.

10. Press switch labeled WR PROT and verify that the light labeled WT PROT goes on and off.

11. Press switch labeled WT PROT until light labeled WT PROT goes off.

12. Verify that lights labeled FAULT, WT, and LOAD are off.

6.2.2 Dismounting

1. Press switch labeled WT PROT and wait until light labelled WT PROT goes on.

2. Set switch labeled RUN/LOAD to the LOAD position.

3. Wait for light labeled LOAD to come on.

4. Open access door and remove cartridge.

5. Close access door.

APPENDIX G

COS FILES

There are four types of files in the COS system: source, compiler binary, system program, and data. Source, compiler binary and data files have similar structure. The fourth, system programs, uses standard OS/8 SAVE format.

1.0 COS SOURCE FILES

All source files, whether control programs (e.g., for BUILD) or DIBOL program sources must be input with the LN command or number commands. This makes all source files look the same and makes them compatible with COS. The source file format is:

word count n	line number	n-1 words, two -237 ASCII characters per word

The first word contains the word count for that line. It is computed as

$$n = \frac{(\#\text{ of characters on line}+1)}{2} + 1$$

The second word is the text line number, 0000-7777, base 8 (0000-4095, decimal).

The third and successive words contain stripped -237 ASCII characters packed two per word (e.g., AB is 4243).

2.0 COS DATA FILES

The format of data files is exactly the same as source files except there are no line numbers.

A data file looks like:

word count n	n words, two characters per word

Word 1 of a data record contains its length in words. The next and successive words contain the text of the record, two stripped ASCII characters per word.

3.0 COS DIBOL COMPILER BINARY

Although the interpretation of the contents is entirely different, externally a file of compiler binary is structured exactly like a data file. That is, for each line of DIBOL source program, the interpretive code for that line is stored as a word count followed by the interpretive code to be used by RSYS.

4.0 COS SYSTEM PROGRAMS

All system programs (e.g., PIP, COMP, BUILD, etc.) are stored in OS/8 SAVE format. The first block of the file is a core control block indicating where the rest of the blocks of the file are to be loaded. Each successive block is a 256 word core image. See the OS/8 Software Support Manual for details.

5.0 SYSTEM AND DATA TAPE FORMATS

COS puts a label on all tapes and disks. This label occupies the first 256 words of each device, of which four are the actual label, one is the date, and the rest may be a bootstrap. Files on DECtape do not use the last two blocks of storage.

Data files are completely devoted to the storage of data, beginning at block 1. Each logical unit holds one data file only. Labels on data files are put on by the Monitor in conjunction with DIBOL programs or by system programs such as SORT and BUILD.

System files have the COS Monitor, the system programs such as COMP, and a directory in OS/8 format. (There is a discrepancy of two years between directories printed by OS/8 and those printed by COS.) System files can also be used to save the sources of DIBOL programs, BUILD or SORT control specifications, or compiled DIBOL programs, but no data files. Their bootstrap starts up the COS Monitor.

Figure G-1 illustrates the layout of the Monitor portion of the system device. As noted in the figure, COMP should be the first file in the file area. The location of COMP is particularly relevant when the binary scratch area is to be expanded.

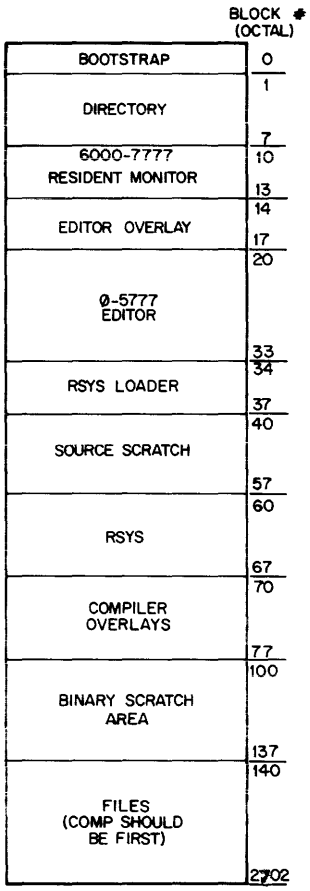


Figure G-1. Monitor Organization

The label on a system tape is unique and identifies it as a system tape. It is put on all system tapes automatically, and looks like [SYS]. The index of a system file looks like

Word	Contents
0	(-) the number of directory entries in this block.
1	The starting block number for file storage.
2	Link word to next directory block, or 0 if end. There are seven directory blocks on all multi-file devices.
3	0 (unused).
4	(-) the number of auxiliary words per entry (always equals -1).
5	First two characters of name.
6	Next two characters of name.
7	Last two characters of name.
8	Two character extension.
9	Date.
10	Length of file (negative).
11 through 255	Repeat of 5-10 for each file.

Pseudo-DECtapes are reserved areas of disk, set aside by the SYSGEN program. They are allocated beginning at the first free block following any COS system on the disk and going to the end. The system index, however, knows nothing about these pseudo-DECtapes inhabiting the upper area of the disk. All it knows is that the available space for file storage only extends to block n, where n is something less than a full disk.

Access

Data tapes and pseudo-DECtapes (which may only be used as data tapes) will be referenced by their logical unit numbers as assigned by SYSGEN. What SYSGEN will actually do is set up a table like the following:

```
Handler address      /physical device #
starting segment
length

handler address      /device #
starting segment
length               /parameters
.
.
.
```

The handler address is a pointer to the device handler to use for this logical unit (either Disk or DECtape). The physical device number is which disk or DECtape drive to reference. The starting segment gives a 16-bit starting block number, which is where on the physical device the space allocated for this logical unit begins. This allows for possible future expansion to larger disks. The length is the number of segments reserved for this logical unit.

For example, if logical unit 14 is assigned to 32-block area on DK1, the fourteenth series of entries in the table might look like:

```
RK08 handler address + 1

212             /start at segment 212(8)
-40             /go for 40(8) segments
```

So, any references to logical unit 14 would refer to blocks 5240(8)-6237(8) of DK1. Note that block 5240 would have a label for that logical unit, just as any other data tape.

For DECtapes, one of these table entries would be essentially the same, except that the starting block would be 0 and probably extend for 46 segments (one complete tape). Two logical units should not be assigned to the same DECtape, as it causes an egregious number of problems when two logical units occupy a dismountable device and a file extends to multiple units.

APPENDIX H

DIBOL DEBUGGING TECHNIQUE (DDT)

In addition to the TRACE feature described in Chapter 1, more complex program debugging techniques may be implemented by specifying the /D option when executing a source program (see Chapter 4) or when running a binary program (see RUN command, Chapter 2). The features of the /D option include: breakpoint, variable examination, subroutine call traceback, and iteration.

1.0 ENTERING DDT MODE

The command to run a binary program in DDT mode has the form

.RU [pronam+chain1+...+chainn,filnam1,...,filnamn]/D

There is an additional amount of space required because of the /D option. It is 768 words plus 3 words for each label in the program that appears before the PROC statement.

When the program is loaded with /D by either of the above commands, initial control is passed to DDT which prints an appropriate DDT version number followed by a "-" to indicate DDT command mode. DDT is now ready to accept commands.

Whenever a CHAIN is entered, control is passed to DDT.

2.0 DDT COMMANDS

Command	Explanation
varnam=	The current contents of variable varnam is printed. Varnam may have single or double subscripts.
varnam=v	Set varnam equal to the value v, where v is any legal alphanumeric string. If v has more characters than defined for varnam, the error message ERROR IN COMMAND is printed.
=	Prints the contents of the last variable examined.
=v	Sets the contents of the last variable examined to v.

Command	Explanation
$nnnn	Sets a breakpoint at line number nnnn. Only one breakpoint may be active at a given time. Do not set a breakpoint at line 0. The breakpoint occurs prior to executing line nnnn.
>n	The breakpoint is executed at the nth occurrence of line nnnn. For example: -$300 ->4 When the program starts to execute line 300 for the fourth time, the breakpoint is executed and control is transferred to DDT.
CTRL/Z	Starts execution of DIBOL program. If a breakpoint was set in a $nnnn command, control reverts to DDT when nnnn is reached and the following message is printed: BREAK! - The user may now type additional commands in response to the "-".
↑	The lines from which CALLs were made (push-down list) during execution of the DIBOL program is printed. This command is generally used after a breakpoint or system error has occurred to trace the execution of the program. This command is a caret (^) on the VT05 keyboard.

NOTES

1. If a DIBOL program running under DDT causes an error message such as ILLEGAL SUBSCRIPT or NUMBER TOO LONG, while a breakpoint was pending, control will be transferred to DDT and the DDT commands may then be used for program examination. If the error was fatal, the DIBOL program cannot be restarted by the CTRL/Z command.

2. Once a DIBOL program is running under DDT, DDT cannot be restarted unless a breakpoint occurs, or an error occurs with a brakpoint pending. Therefore, if the user has no requirement for a breakpoint but wishes to return to DDT for program examination if an error occurs, it is necessary to set a breakpoint to a non-existent line number.

APPENDIX I

DIBOL STATEMENT SUMMARY

Statement	Explanation
ACCEPT (terminator,receiver)	Accepts input from KBD stores it in receiver and stores terminating character as terminal code in terminator.
BLOCK	See RECORD.
CALL label	Causes control to branch to statement whose label is specified.
CHAIN decimal expression	Causes the specified CHAIN to be loaded. Execution resumes with the first PROC statement in the specified CHAIN. See Sections 1.4.3 and 2.2.6 for more detail.
DISPLAY (line,column,expr)	Display expression on screen at line and column specified, (0,0 = current location) or perform one of following operations. 0 = position cursor, 1 = clear to end of scope, 2 = clear to end of line, 25 = emit a bell or beep sound. No carriage return, line feed is generated.
END [/list-control]	An optional statement inserted at physical end of DIBOL program.
FINI(channel)	Closes file associated with channel (if output file, writes end of file, file length). Frees channel.
FORMS (channel,skip-code)	Line printer control statement. If code is 0 go to top of next form; if 1-4095 skip number of lines specified. -1 = skip to channel 2 and advance paper n lines as specified on vertical forms control tape (LS8E). -2 = print enlarged characters for next XMIT statement (LS8E).
GO TO label	Unconditional branch to label.

Statement	Explanation
GO TO (label1,...,labeln),key	If value of key is K, with K greater than 0 and less than or equal to n, branch to labelK. Otherwise execute next instruction.
IF (expression1.rel.expression2)stmnt	Execute stmnt if relationship is true .rel. is one of the following: .GT. = greater than .LT. = less than .LE. = less than or equal .GE. = greater than or equal .EQ. = equal .NE. = not equal stmnt may be one of: GO TO label CALL label STOP RETURN ON ERROR label [NO] TRACE
INCR variable	Increments value of variable by 1.
INIT (channel, device [,filnam][,unit])	Associate channel with specified device and initialize device. For mass storage devices only, specify file name. If unit is specified, system looks for file on this logical unit before issuing MOUNT message. The device is one of the following: INPUT mass storage input data file OUT mass storage output data file UPDATE mass storage data file (fixed length records) for update KBD keyboard input TTY terminal output LPT line printer output CDR card reader input PTP paper tape punch (high speed) RDR paper tape reader (high speed) SYS input source file (on system device)
[NO] TRACE	Stops/starts trace facility.
ON ERROR label	Branch to label if next statement incurs a non-fatal error.

Statement	Explanation
PROC[n][/list control][;comments]	Begins PROCcedure section and n= maximum number of mass storage logical units to be opened at one time. If omitted, n=7 is assumed. Causes top of form on compiler listing. The comments are printed on each page as headings.
READ (channel,record,rec#)	Does a direct access read of data record into an area in core.
RECORD [rname][,X] [fldnam],[x]{A / D}[n]{'alpha initial value' / ,[decimal initial value]}{,P / ,D / ,S / ,C}	Reserves areas of core where records are stored during processing and defines the fields in the record.
RETURN	Returns control to next statement after last CALL.
START [/list control][;comments]	An optional statement which may appear anywhere. Causes top of form on compiler listing.
STOP	Terminates program execution and returns control to Monitor.

rname	labels record area.
,X	overlay of another record area. record must be less than or equal to the record being overlayed.
fldnam	names a field within the record
x	repetition count (1-512) for an array
n	size of field (1-510 characters) default = 1
,P	insert initial value at execution
,D	insert system date at execution
,S	assigns fldnam to the value of the run time options (see Section 1.4.16).
,C	clears record loaded as a result of a CHAIN statement (see Section 1.4.16).

Statement	Explanation
TRAP label	Allows a DIBOL program to spool output to the line printer while executing the program. label is the name of the line printer routine (see Section 1.4.21).
WRITE (channel,record,rec#)	Does a direct access write of a data record to a specified file.
XMIT (channel,record[,eof label])	Transfers data between record area and device attached to channel. If end of file occurs for an input file, branch to label if specified.

Data manipulation statements:

variable=	clears variable to zeroes or blanks
variable=expression	assignment, conversion depends upon types.
variable=expression1,	alpha literal alpha variable formatting. Types must be: A=D,A.

variable	expression	action
D	D	Assign. If variable is too small, drops high order digits. If variable is too big, right justified.
D	A	Alpha converted to decimal then stored.
A	A	Assign. If variable is too small, drops characters on right. If too big only left-most characters are changed.
A	D	Decimal converted to alpha.
R	R	Treated as A to A.

where: D = decimal variable
A = alpha variable
R = record

APPENDIX J

ERROR MESSAGE SUMMARY

Message	Program	Explanation
ALL UNITS ASSIGNED	SYSGEN	Attempt to assign more than 15 units.
BAD ALPHA FIELD	BUILD	Illegal entry in alpha field.
BAD ALPHA VALUE	BUILD COMP	Initial value in an alpha data specification did not begin or end with a single quotation.
BAD CHAR	CONVEX	Attempted to convert an OS/8 character for which there is no COS code.
BAD CHECKDIGIT	BUILD	Checkdigit calculated by BUILD does not match the one entered.
BAD CHECKSUM	PATCH	An attempt was made to write a block which was incorrectly patched.
BAD COMPILATION	Monitor	User tried to SAVE a compiled binary that had errors.
BAD DECIMAL FIELD	BUILD	Illegal entry in decimal field.
BAD DECIMAL VALUE	COMP	The initial value for a decimal data specification was incorrectly formed.
BAD DEFAULT FIELD	BUILD	Default field does not agree with size or type of object field.
BAD DELIM	BUILD	Fields are separated by an illegal delimiter.
BAD DEPICTOR	BUILD	OUTPUT format in error.
BAD DEV	CONVEX	Output device not same type physical device as system device.
BAD DEVICE	BUILD	Not KBD, SYS, RDR or CDR.
	UPDATE	Illegal device specified on INPUT statement.

Message	Program	Explanation
BAD DIGIT AT LINE nnnn	Run Time	A character other than +, -, space, or the digits 0-9 was encountered in an alpha to decimal conversion; nnnn is the line number in which the error occurred.
BAD DIGIT IN DECIMAL INITIAL VALUE	BUILD SORT UPDATE	Alpha character in a decimal initial value.
BAD DIRECTORY	PIP Monitor PATCH CONVEX	Attempt to reference or store a file on a device with no directory (or a directory that has been destroyed). Also an attempt to get a directory of a data file.
BAD END STATEMENT	BUILD	Missing or misspelled END statement.
BAD FIELD NUMBER	BUILD	Field number specified is greater than 2047.
BAD INPUT STATEMENT	BUILD UPDATE	Statement missing, misspelled, or out of order.
BAD KEY STMNT	UPDATE	Key in KEY statement has bad syntax.
BAD KEYWORD	BUILD	Keyword entered was not described in the control program.
BAD LABEL	Monitor	Tape has no label, or its form is incorrect.
BAD NUMBER	PATCH	A number with more than 4 digits, a non-digit, or 8 or 9 was typed.
BAD OPTION	BUILD	Something other than C or + is specified in OUTPUT section.
BAD OUTPUT FORMAT #	BUILD	Format number is wrong or out of range (< 0 or 7).
BAD PROC #	COMP	The number in a PROC statement was not a digit from 0 to 7.
BAD PROGRAM AT LINE nnnn	Run Time	Attempting to run a binary program which contains a compilation error. Check compilation listing for error flags. Correct flagged errors, and recompile.

Message	Program	Explanation
BAD RANGE CONSTRUCTION	BUILD	In RANGE portion of statement an open or close parenthesis or comma is missing.
BAD RECORD SIZE	SORT	File contains records of variable length.
BAD RELATIONAL	COMP	An illegal relational occurs in an IF statement. For example, a .GX. instead of a .GT.
BAD SWITCH	SYSGEN	Not /T, /C, or /L.
BAD UPDATE COMMAND	UPDATE	Bad syntax, probably missing F, =, or comma.
BAD UPDATE STMNT	UPDATE	UPDATE statement is incorrect.
BAD VALUE	UPDATE	Value in update command too long or incorrect.
BAD WORK UNIT COUNT	SORT	# of work units not in range 3-7.
BLOCK TOO BIG	PATCH	An incorrect block number was typed. It cannot be longer than the length of the file being patched.
BLOCK NUMBER ERROR PHASE n	TDMARK	Refer to DEC-8E-EUZC-D.
C-FIELD NOT PERMITTED	BUILD	C-type fields cannot be specified in the INPUT section.
CANT BACKSPACE PAST BEGIN OF FILE	DAFT	Attempt was made to backspace past beginning of file.
CANT BACKSPACE WITH SEQUENTIAL INPUT	DAFT	Attempted to backspace with sequential input.
CCP ERROR	COMP	Matching angled bracket (< or >) missing.
CHECKSUM ERROR PHASE n	TDMARK	Refer to DEC-8E-EUZC-D.
COMMA MISSING	COMP	No comma appeared where one was expected.
CONTROL BUSY ERROR	RK8MRK	Disk IOT issued when control was busy.

Message	Program	Explanation
DATA INITIALIZATION MISSING	COMP	No data initialization followed a comma in a data specification statement.
DATA ERROR PHASE n	TDMARK	Refer to DEC-8E-EUZC-D.
DATA RATE ERROR	RK8MRK	The processor was busy and did not respond to a data break request within the 13 us required. The transfer is terminated immediately.
DATA TABLE OVERFLOW!	BUILD UPDATE SORT	Too much data. Maximum = 960 characters.
DESCRIPTOR TABLE OVERFLOW!	BUILD UPDATE SORT	Too many F, T and C fields defined. Maximum = 160.
DESTINATION FIELD NOT DECIMAL	BUILD	The destination field specified is not defined as decimal.
DIBOL FILE NUMBER IN USE AT LINE nnnn	Run Time	In INIT, the channel number is already INITed to a mass storage device.
DIBOL FILE NUMBER NOT INITED AT LINE nnnn	Run Time	An attempt was made to XMIT, READ, or WRITE with a channel number that was not INITed.
DISK DATA ERROR	RKEMRK	Refer to DEC-08-DHRKD-A-D.
DT n?	Monitor	The expected tape is not available on a DECtape drive. For example, if following a MOUNT message the operator has specified that a file should be written on unit 3, and unit 3 is not selected, or is WRITE LOCKed, this message is output. On LINCtapes there is no message. To recover, set switch to ENABLE the unit or set the dial selector to the proper number. Then type any key to continue.

Message	Program	Explanation
END OF FILE	Run Time	The last record of an input file has been read and the end of file mark encountered but no EOF label was specified in the XMIT statement.
END OF INPUT FILE AT RECORD nnnn	DAFT	Attempted to read past end of file mark on input. Not necessarily an error.
END TAPE ERROR PHASE n	DTMARK	Refer to DEC-08-EUFB-D.
ENTER "DATE MM/DD/YY"	Monitor	Typed an unrecognizable date.
ERROR IN COMMAND	Monitor	Miscellaneous.
EXEC WORD ERROR	RK8MRK	An error occurred while reading the EXEC. word of the sector.
EXCESSIVE GRID SIZE	DAFT	The grid printer width may not be greater than 130 characters.
EXPECTED LABEL IS MISSING	COMP	A required label is missing.
EXPRESSION NOT ALLOWED	COMP	An expression or bad character occurs to the left of an =.
EXTRA CHARS AT END OF DATA	BUILD	More data was entered than was defined in the control program.
EXTRA CHARS AT STMNT END	BUILD SORT UPDATE COMP	Characters not relating to statement appear on line.
FIELD NOT DECIMAL	BUILD	Field not defined as decimal.
FIELD NUMBER MISSING OR 0	BUILD SORT UPDATE UPDATE	Field number or default unit number is missing or is 0 or greater than 511.
FIELD OUT OF RANGE	BUILD	Data entered is not within the range specified in the control program.
FIELD TOO LARGE OR 0	COMP	In a data description statement, the dimension was 0 or more than 3 digits long, or the field size was 0 or larger than 511.

Message	Program	Explanation
FILE NOT FOUND	Monitor PATCH	The program with the name specified was not found. After, for example, FETCH FILEX.
FORMAT ERROR	RK8MRK	An error occurred while reading the header word.
FORMAT PARITY ERROR	RK8MRK	A parity error occurred in either the Header word or Exec word while reading the sector.
FULL	CONVEX	Specified an output device with -S option and source file being created was filled.
HASH FIELD MISSING	BUILD	No field specified for storage of hash total.
I/O ERROR ON xx, RETRY?	Monitor	System failed in three attempts to read from or write to a device.
I RECORD ALREADY EXISTS	UPDATE	Tried to insert a record already present.
ILLEGAL CHAIN AT LINE 0000		CHAIN argument does not match .RUN command.
ILLEGAL DEVICE	SYSGEN	Device other than DT0-DT7 or DK0-DK3.
ILLEGAL DEVICE AT LINE nnnn	Run Time	Attempt to WRITE on a file that was not INITialized for UPDATE or attempt to READ from a file that was not INITialized or INITialized for INPUT or UPDATE.
ILLEGAL DEVICE SWITCH	PIP	A switch was specified that was not /R, /C or /K for input or /P, /L or /T for output.
ILLEGAL OPERATOR	COMP	A bad character was encountered in an expression where an operator would be expected.
ILLEGAL RECORD # AT LINE nnnn	Run Time	Record number is 0, past the end of the physical unit or the length specified in the record header word does not match the length of the XMIT block (records in data file are all not the same length).
ILLEGAL SORT KEY	SORT	Bad syntax on KEY statement, KEY too complex or KEY statement missing.

Message	Program	Explanation
ILLEGAL STMNT	COMP	The statement was not a data manipulation statement (it had no =) nor did it start with a recognizable keyword.
ILLEGAL SUBSTRING AT LINE nnnn	Run Time	A DIBOL PROCedure section statement has attempted to access a data field, FI(m,n), but m=0 or m n.
?ILLEGAL UNIT	Monitor	The unit specified is not DK0-DK3 or DT0-DT7; for example, WRITE FILE,XY3 WRITE FILE, RK9
ILLEGAL UNIT	SORT	Default unit is 0 or 15.
IN USE	Monitor	The unit specified in answer to the MOUNT message is already being used.
INCREMENT TOO BIG	BUILD	Length of increment field larger than length of output field.
INSUFFICIENT SPACE ON DEVICE	SYSGEN	Attempt to allocate more segments than are available on a device.
INITIAL ALPHA VALUE DOESN'T BEGIN WITH QUOTE	BUILD SORT UPDATE	Beginning quotation mark missing for initial alpha value.
INITIAL VALUE TOO BIG	BUILD SORT UPDATE	The initial value specified is larger than the field size.
INITIAL VALUE WRONG SIZE	COMP	The initial value in a data specification statement had a length different from the field size specified.
INITIAL VALUE TOO SMALL	BUILD SORT UPDATE	The initial value specified is smaller than the field size.
KEYWORD MISSING OR TOO BIG	BUILD	
LABEL NOT ALLOWED	COMP	A symbol in an arithmetic expression was not of type alpha or decimal, or a symbol which had been redefined was used.

Message	Program	Explanation
LAST INT NOT END ZONE	DTMARK	Refer to DEC-08-EUFB-D.
LINE # TOO LARGE	Monitor	Greater than 4095.
LINE TOO LONG	Monitor	Greater than 120 characters.
LINE TOO LONG AT LINE nnnn	Run Time	An input line (record) overflowed the block into which it was read.
LOCATION TOO BIG	PATCH	A location >= 400 was typed.
MARK TRACK ERROR PHASE n	TDMARK DTMARK	The DECtape being used is bad. Try reformatting the tape.
!*!MEMORY FAIL!*!	Monitor	A hardware error has occurred. Restart the program.
MISSING CLOSE PAREN	COMP	No close parenthesis occurred where one was expected.
MISSING CLOSE QUOTE ON ALPHA INITIAL VALUE	SORT UPDATE BUILD	Quotation mark misplaced or missing.
MISSING FIELD NAME	BUILD	Field name is missing.
MISSING INITIAL VALUE	BUILD SORT UPDATE	Comma was inserted after type and size but initial value was not specified.
MISSING OPEN PAREN	COMP	No open parenthesis occurred where one was expected.
MISSING OPERAND	COMP	A binary operator occurs in an expression with no operand following it; or no expression at all occurs where one is expected.
MISSING OR BAD DEVICE	COMP	The device in an INIT statement was missing or started with an illegal character.
MISSING OR BAD OPTION	COMP	In a record statement, no option appeared after a comma or the option which appeared was not X or C.
MISSING OR OUT OF ORDER	UPDATE	On C or D, record was not there. Probably because scratch file was not sorted.
MISSING QUOTE	COMP	The statement contained an odd number of quotes (').

Message	Program	Explanation
MISSING RELATIONAL	COMP	No relational appeared in an IF statement.
MISSING UNIT	UPDATE	Unit number specified is a multiple of 16.
NAME PREVIOUSLY DEFINED	COMP	The name used was previously defined and this statement tries to redefine it.
NECESSARY FIELD MISSING	BUILD	Field specified as necessary in control program was not entered.
NO	BOOT	Device switch illegal or missing.
NO BUFFERS LEFT AT LINE nnnn	Run Time	Not enough core available for I/O buffers. An I/O buffer of some multiple of 512 characters is set up for each active mass storage file. Another possibility: too few files were specified in the PROC statement.
NO CHANGE IN BLOCK	PATCH	An attempt was made to write a block but no changes were made to it. If this was on purpose fine otherwise make the changes to the block again.
NO COMMA AFTER FIELD NAME	BUILD SORT UPDATE	No comma or a character other than comma was specified after the field name.
NO DEFINE!	UPDATE	Control file did not start with a DEFINE. Returns to Monitor immediately.
NO END	CONVEX	Entire OS/8 ASCII input file was read but no end of file (CTRL/Z) was found.
NO END STATEMENT	UPDATE	END statement missing after OUTPUT specification.
NO FILE AT LINE nnnn	Run Time	No file specified in RUN statement to satisfy INIT (SYS) command.
?NO FILE TO SAVE	Monitor	Nothing in the edit work area when WRITE command is issued.
NO INIT DONE	Monitor	Program attempted a read or write on a device not opened by the system program.

Message	Program	Explanation
NO INPUT	SORT MERGE	Input file is null or not enough input files specified for a MERGE.
NO ROOM	PIP CONVEX	Attempt to store a file on a full device.
NONCE ERROR	DAFT	Tried to use a feature not currently supported or documented.
NONEXISTENT FIELD	UPDATE	Tried to update Fn where no Fn appeared in file description section.
NOT A OR D	BUILD COMP SORT UPDATE	A character other than A or D occurred in a data specification statement where A or D was expected.
NOT BUILD FILE!	BUILD	File did not start with DEFINE statements.
NOT ENOUGH ROOM FOR SYSTEM AND FILES	SYSGEN	Device specified was too small to accommodate system program and files.
NOT FOUND	CONVEX	File with specified name not found.
NOT I,D, OR C	UPDATE	Bad first character on update command.
NOT LABEL	COMP	A symbol which was not a 'label' occurred where a label was required.
NOTHING AFTER FIELD NAME	BUILD SORT UPDATE	Field type and size are not specified after field name and comma.
NUMBER REPEATED OR OUT OF ORDER	BUILD SORT UPDATE	A field sequence number is used more than once or is out of ascending order sequence.
# TOO LARGE	SYSGEN	Number entered was greater than 4095.
NUMBER TOO LONG	Run Time	A decimal field longer than 15 digits was used in a calculation.
OUT OF ORDER	UPDATE	SORT statement missing and update commands are not in order.

Message	Program	Explanation
OUT OF ROOM!	BUILD	Too many descriptors in INPUT section. Maximum keywords = approx. 121. Approx. maximum fields for all keywords = 192. Too many fields in OUTPUT section. Approx. maximum = 128.
OUTPUT ERROR	SORT	Indicates a system malfunction.
OUTPUT RECORD TOO BIG	BUILD	Record is larger than 510 characters.
PARITY ERROR	RK8MRK	A bit in data, parity or timing has been picked up or dropped on read. Transfer will continue to the end of the sector where the error occurred. Word count, current address information can be used to identify the error.
PARITY ERROR PHASE n	DTMARK	Refer to DEC-08-EUFB-D.
PROGRAM TOO BIG	COMP	Binary output too big for the binary scratch area. Remedy: Run PIP/n with Option E to enlarge scratch area.
PROGRAM TOO BIG AT LINE nnnn	Run Time	Binary program does not fit in available core. Reduce program size.
PUSHDOWN OVERFLOW AT LINE nnnn	Run Time	Either (1) a statement is too complex or (2) subroutines are nested to a depth greater than 50, or a combination of the two.
READ STATUS ERROR	RKEMRK	Refer to DEC-08-DHRKD-A-D.
RECALIBRATE STATUS ERROR	RKEMRK	Refer to DEC-08-DHRKD-A-D.
RECORD TOO BIG	COMP	A named record exceeded 510 words in size.
REPLACE?	Monitor	Duplicate file names. Type Y to replace.
RETURN WITHOUT CALL AT LINE nnnn	Run Time	The program tried to execute a RETURN, but there was no place to go; there was no corresponding CALL statement.

Message	Program	Explanation
SECTOR NO GOOD	RK8MRK	The program attempted to read or write data on a sector whose header words indicated a bad sector. The transfer is terminated immediately.
SELECT ERROR PHASE n	DTMARK	
SETUP?	TDMARK DTMARK	Indicates an error in the DECtape setup. One of the units specified is in WRITE LOCK position, not selected, or the write flip-flop is unable to be set, or there may be a timing error.
STMNT TOO COMPLEX	COMP	The statement generated too much code and overflowed the Compiler's code buffer, or it had too much nesting and overflowed the Compiler's pushdown stack. Remedy: break up the statement into smaller parts.
SUBSCRIPT ERROR	COMP	No comma or close parenthesis occurs after a subscript.
SUBSCRIPT NOT DECIMAL	COMP	The type of a subscript was not decimal.
SUBSCRIPT TOO BIG AT LINE nnnn	Run Time	Program attempted to destroy the run-time system or itself by using a large subscript; larger than that defined in the Data Definition section. Note that the run-time system does not detect all illegal subscripts; only those which would cause the user's program or the system to be destroyed.
SWITCH NOT SET TO WTM OR SINGLE LINE FLAG FAILED TO SET SET SWITCH TO WTM.	TDMARK	Switch on the M868 module is not set to the WTM position or the timing generator for writing the mark and timing tracks is not setting the single line flag.
SYMBOLS DEFINED BUT NOT REFERENCED; xx	CREF	Symbols were defined but not referenced. Not necessarily an error.

Message	Program	Explanation
SYNTAX ERROR	SYSGEN	Missing comma, extra characters, etc.
TEXT AREA FULL	Monitor	Greater than 8,150 characters.
TIME OUT ERROR	RK8MRK	The control did not complete an operation after 32 revolutions.
TIMING ERROR PHASE n	TDMARK DTMARK	Refer to DEC-8E-EUZC-D. Refer to DEC-08-EUFB-D.
TOO BIG	BUILD	Data entered is greater than the size of the field as defined in the control program.
TOO MANY FILES	MERGE	More than 6 input files specified.
TOO MANY ITEMS	COMP	More items appeared in a data initialization than were specified by the dimension.
TOO MANY SYMBOLS!	COMP	A fatal error message 365 symbols allowed in symbol table in 8K system, and 511 symbols allowed in larger systems. The compiler stops compiling; no storage map can be produced.
TOO MUCH DATA	COMP	Program's data division exceeds 32K bytes.
TRACK ADDRESS ERROR	RK8MRK	Track, surface or sector address read from the disk did not agree with the address count registers or the disk drive electronics indicated track position 000 and the track counter did not agree. The transfer is terminated immediately.
TRACK CAPACITY EXCEEDED ERROR	RK8MRK	The program attempted to read or write beyond sector 17.
UNDEFINED FIELD	BUILD	Field referenced was not defined in field descriptor section of control program or a decimal field was being equated to an alpha field, or an alpha field was specified for a flag.
UNDEFINED FORMAT	BUILD	Format number referenced but not defined (errors given on END statement).

Message	Program	Explanation
SYNTAX ERROR	SYSGEN	Missing comma, extra characters, etc.
TEXT AREA FULL	Monitor	Greater than 8,150 characters.
TIME OUT ERROR	RK8MRK	The control did not complete an operation after 32 revolutions.
TIMING ERROR PHASE n	TDMARK DTMARK	Refer to DEC-8E-EUZC-D. Refer to DEC-08-EUFB-D.
TOO BIG	BUILD	Data entered is greater than the size of the field as defined in the control program.
TOO MANY FILES	MERGE	More than 6 input files specified.
TOO MANY ITEMS	COMP	More items appeared in a data initialization than were specified by the dimension.
TOO MANY SYMBOLS!	COMP	A fatal error message 365 symbols allowed in symbol table in 8K system, and 511 symbols allowed in larger systems. The compiler stops compiling; no storage map can be produced.
TOO MUCH DATA	COMP	Program's data division exceeds 32K bytes.
TRACK ADDRESS ERROR	RK8MRK	Track, surface or sector address read from the disk did not agree with the address count registers or the disk drive electronics indicated track position 000 and the track counter did not agree. The transfer is terminated immediately.
TRACK CAPACITY EXCEEDED ERROR	RK8MRK	The program attempted to read or write beyond sector 17.
UNDEFINED FIELD	BUILD	Field referenced was not defined in field descriptor section of control program or a decimal field was being equated to an alpha field, or an alpha field was specified for a flag.
UNDEFINED FORMAT	BUILD	Format number referenced but not defined (errors given on END statement).

Message	Program	Explanation
UNDEFINED NAME	COMP	A name is used which was never defined in the data section.
UNEXPECTED END OF FILE	BUILD UPDATE	Missing END statement.
UNIT xx IS FREE	SORT	NOT AN ERROR - merely an informative message to aid the operator. xx is a COS unit number.
UNRECOGNIZABLE LINE	SORT	Parameter line did not start with a good keyword.
WRITE ERROR PHASE n	TDMARK	Refer to DEC-8E-EUZC-D.
WRITE LOCK ERROR	RK8MRK	The program attempted to write a section that was write protected. The write operation is terminated immediately.
WRITE STATUS ERROR	RKEMRK	Refer to DEC-08-DHRKD-A-D.
WRONG DATA TYPE	COMP	Mixed modes occurred in an expression; or an argument which was supposed to be decimal was not or one of the three arguments in a data manipulation statement was of the wrong type.
ZERO DIVISOR AT LINE nnnn	Run Time	The program attempted to divide by zero.
xxxx IS BEING IGNORED	CREF	The line specified cannot be interpreted.
xxxx SHOULD BE yyyy BLK ERROR PHASE n	DTMARK	Refer to DEC-08-EUFB-D.
xxxx SHOULD BE yyyy DATA ERROR PHASE n	DTMARK	Refer to DEC-08-EUFB-D.

APPENDIX K

COMMAND SUMMARY

The commands summarized below are described in detail in Chapter 2 of this manual.

1.0 COMMANDS FOR MONITOR FUNCTIONS

Command	Function
BAtch pronam	Executes a string of previously stored Monitor commands.
CTRL/C	Returns control to the Monitor.
CTRL/N	Turns on/off imbeddded numeric keypad at terminal (see Table 2-1).
CTRL/O	Stops/starts terminal echo of typed input. Also halts output from a LIST command and the printing of the compiler listing.
CTRL/U	Deletes the current input line.
CTRL/V	Stops/starts terminal echo of Monitor version message COS MONITOR V2.1108.
CTRL/Z	Signals end of input and returns control to the Monitor. Halts display of line numbers from an LN command.
DAte mm/dd/yy	Stores the specified date for assignment to new programs created and reports printed.
DElete pronam[,dev]/x	Removes the named source, binary or system program from the device directory.
DIrectory {[physical device] [/T] /logical unit#	Prints a list of programs stored on a physical device or a label of a file stored on a logical unit.
PLease text string	Displays the specified text string when used in a BATCH program.
RUBOUT	Erases the last character typed and echoes the deleted character.

Command	Function
Run [pronam][+chainl+...+chainn],[filnaml,...,filnamn][/xx]	Loads and executes the named system or binary program using the named file.
SAve pronam[,dev][/Y]	Stores the binary program from the Compiler scratch area on the named device.

2.0 COMMANDS FOR EDITING FUNCTIONS

Command	Function
ERase [n1][,n2]	Erases specified lines of text from the edit buffer.
FEtch pronam[,dev]	Loads the named source file into core from the specified device.
LIst [n1][,n2][/x]	Outputs the specified lines to the high-speed punch, line printer or terminal.
LN [n][,inc][/x]	Automatically displays line numbers on the terminal for program creation.
nnnn text (number command)	Inserts the specified line number and text into the edit buffer. If this line number already exists, the new text replaces the old.
REsequence [n][,inc]	Renumbers the lines of the program in the edit buffer.
WRite pronam[,dev][/Y]	Stores the source program located in the edit buffer on the specified device.

APPENDIX L

ADVANCED
PROGRAMMING
TECHNIQUES

1.0 ACCEPT AND DISPLAY

1.1 Background

XMIT statements were originally used when the terminal was a Teletype. The arrival of the VT05 display terminal introduced new concepts --programmable cursor control and hardware display clear. ACCEPT and DISPLAY statments were added to the DIBOL language to use these features. The terminal may now be used in two ways:

1. As a Teletype by using XMIT statements.

2. As a powerful data entry tool using ACCEPT and DISPLAY statements.

(Refer to the ACCEPT and DISPLAY statements in Chapter 1 before proceeding further.)

1.2 Interaction of ACCEPT and DISPLAY Statements

ACCEPT and DISPLAY statements are used extensively in data entry programs. These programs typically work one of two ways. The first asks (DISPLAYs) questions and interprets (ACCEPTs) answers. This mode of operation could be easily simulated by using a Teletype. The second method displays a format or heading on the screen and moves the cursor to the right or just below the question to be answered.

The format is never cleared, but data is being entered and cleared continuously from the screen. This mode is used in repetitive data entry and updating. Quite often the four keys; up arrow, down arrow, left arrow, and right arrow have special meanings. For example, assume ten headings are displayed on the screen, indicating ten fields are to be entered or updated. Up arrow might be used to re-enter information in the first field, no matter which field is currently being entered; down arrow might mean no more information for any of these fields; left arrow might mean restart entering the current field; right arrow might mean go on to the next field without changing the current field.

1.3 Simple Example Using ACCEPT and DISPLAY

In order to enter a six digit customer number and a 15-character customer name, the program might be as follows:

```
        RECORD
TCHAR,D2
ALPHA,A15
CNO,D6
CNAME,A15
.
.
.
        PROC 1
.
.
.
        DISPLAY(1,1,1)   ;CLEAR SCREEN AND POSITION CURSOR
        DISPLAY(0,0,'CUSTOMER NO.   CUSTOMER NAME')
LOOP,   DISPLAY(2,1,2)   ;CLEAR LINE 2 AND POSITION CURSOR
        ALPHA=           ;CLEAR THIS FIELD
        ACCEPT(TCHAR,ALPHA)
        ON ERROR LOOP    ;RE-ENTER IF NOT NUMERIC
        CNO=ALPHA
        ALPHA=           ;CLEAR THIS FIELD AGAIN
        DISPLAY(2,16,0)  ;POSITION CURSOR
        ACCEPT(TCHAR,ALPHA)
        CNAME=ALPHA
        .
        .                ;SAVE DATA
        .
        GO TO LOOP
        .
        .
        .
```

1.4 Generalized ACCEPT Subroutines

1.4.1

Although the previous example works properly, it lacks several features which would be quite practical to have:

1. Type RUBOUT to clear the previous character that was entered in the current field from both the program and the display.

2. Type CTRL/U (a DEC convention) or left arrow to clear the entire current field from both the program and the display.

Since our data acceptance is getting more sophisticated, it can best be performed by calls to a subroutine. The following two subroutines and test programs will accept data from the terminal and use the RUBOUT key and the CTRL/U (or left arrow) key as previously specified. The first program uses the hardware clearing features built into the VT05. Unfortunately, this destroys data if it is on the same line and to the right of what is being ACCEPT'ed.

```
        START   ;SUBROUTINE VT05---CORRECTS REMAINDER OF LINE
                ;FOR ERRORS
        RECORD
KBDBUF, A72     ;STORAGE FOR KEYBOARD INPUT
        RECORD ,X
KBDIN,  72A1

        RECORD          ;WORK AREA
ROW,    D2      ;CURSOR Y-COORDINATE ON ENTRY TO SUBROUTINE VT05
                ;(NEEDED FOR CORRECTION ONLY)
COL,    D2      ;CURSOR X-COORDINATE ON ENTRY TO SUBROUTINE VT05
                ;(NEEDED FOR CORRECTION ONLY)
TCHAR,  D2      ;TERMINATING CHARACTER IN AN ACCEPT STATEMENT
CHAR,   A1      ;INPUT CHARACTER FROM AN ACCEPT STATEMENT
VT05IN, D2      ;NUMBER OF CHARACTERS ACCEPTED BY SUBROUTINE VT05
VTLIM,  D2      ;NUMBER OF CHARACTERS TO BE ACCEPTED BY
                ;SUBROUTINE VT05
        PROC
BEGIN,  DISPLAY(1,1,1)  ;CLEAR SCREEN           ;***********
        DISPLAY(1,40,'ERASED IN CORRECTION')    ;*          *
        DISPLAY(1,1,'NAME:')                    ;*          *
        ROW=1                                   ;* SAMPLE   *
        COL=6                                   ;*  TEST    *
        VTLIM=20        ;20 CHARS MAX.          ;* PROGRAM  *
        CALL VT05                               ;*          *
        IF (KBDBUF.EQ.'END') STOP               ;*          *
        GO TO BEGIN                             ;***********



;       CALLING SEQUENCE
;               ROW=            ;SET Y-COORDINATE
;               COL=            ;SET X-COORDINATE
;               VTLIM=          ;SPECIFY MAXIMUM NUMBER OF CHARS
;                               ;TO ACCEPT
;               CALL VT05

;ACCEPT A MAXIMUM OF VTLIM CHARACTERS AT LOCATION SPECIFIED BY
;ROW AND COL. RETURN WHEN VTLIM CHARACTERS, A TERMINATION CHARACTER,
;OR A SPACE IS ENTERED. RUBOUT DELETES LAST CHARACTER ENTERED AND
;CTRL/U ELIMINATES THE ENTIRE ENTRY RUBOUT AND CTRL/U CLEAR THE
;REMAINDER OF THE LINE (FASTER THAN DISPLAYING SPACES)

VT05,   VT05IN=
        KBDBUF=
VT052,  ACCEPT(TCHAR,CHAR)
        IF (TCHAR.EQ.0) GO TO VT053     ;NON-TERMINATING CHARACTER
        IF (TCHAR.EQ.21) GO TO VT054    ;CTRL/U (TO USE LEFT ARROW ON
                                        ;VT05 KEYBOARD, CHANGE 21 TO
                                        ;8)
        IF (TCHAR.EQ.32) GO TO VT055    ;RUBOUT
        RETURN                          ;TERMINATING CHARACTER OTHER
                                        ;THAN RUBOUT OR CTRL/U
VT053,  IF(CHAR.EQ.' ') RETURN          ;SPACE IS A TERMINATING CHAR
                                        ;(TO ELIMINATE THIS FEATURE,
                                        ;REMOVE THIS STATEMENT AND PUT
                                        ;LABEL ON NEXT STATEMENT.)
        INCR VT05IN                     ;VT05IN=# OF INPUT CHARACTERS
```

```
        KBDIN(VT05IN)=CHAR
        IF (VT05IN.EQ.VTLIM) RETURN      ;THE SPECIFIED NUMBER OF
                                         ;CHARACTERS WERE INPUT
        GO TO VT052
VT054,  IF(VT05IN.EQ.0) GO TO VT05
        DISPLAY(ROW,COL,2)               ;CLEAR CHARACTERS ENTERED
                                         ;TO END OF LINE
        GO TO VT05
VT055,  IF(VT05IN.EQ.0) GO TO VT052
        KBDIN(VT05IN)=
        VT05IN=VT05IN-1
        DISPLAY(ROW,COL+VT05IN,2)        ;RUBOUT PREVIOUS CHARACTER
                                         ;TO END OF LINE
        GO TO VT052
```

1.4.2

The next program clears incorrectly entered data by displaying spaces. This is obviously much slower than using the hardware display clear feature, but data on the same line and to the right is not cleared.

```
        START   ;SUBROUTINE VT05---CORRECTS ONLY CHARACTERS IN ERROR
        RECORD
KBDBUF, A72     ;STORAGE FOR KEYBOARD INPUT
        RECORD ,X
KBDIN,  72A1

        RECORD          ;WORK AREA
BLNK72, A72     ;72 BLANK CHARACTERS
ROW,    D2      ;CURSOR Y-COORDINATE ON ENTRY TO SUBROUTINE VT05
                ;(NEEDED FOR CORRECTION ONLY)
COL,    D2      ;CURSOR X-COORDINATE ON ENTRY TO SUBROUTINE VT05
                ;(NEEDED FOR CORRECTION ONLY)
TCHAR,  D2      ;TERMINATING CHARACTER IN AN ACCEPT STATEMENT
CHAR,   A1      ;INPUT CHARACTER FROM AN ACCEPT STATEMENT
VT05IN, D2      ;NUMBER OF CHARACTERS ACCEPTED BY SUBROUTINE VT05
VTLIM,  D2      ;NUMBER OF CHARACTERS TO BE ACCEPTED BY
                ;SUBROUTINE VT05
VT05XX, D2      ;TEMPORARY STORAGE FOR SUBROUTINE VT05
        PROC
BEGIN,  DISPLAY(1,1,1)  ;CLEAR SCREEN           ;***********
        DISPLAY(1,40,'NEVER ERASED')            ;*          *
        DISPLAY(1,1,'NAME:')                    ;*          *
        ROW=1                                   ;* SAMPLE   *
        COL=6                                   ;*  TEST    *
        VTLIM=20        ;20 CHARS MAX.          ;* PROGRAM  *
        CALL VT05                               ;*          *
        IF (KBDBUF.EQ.'END') STOP               ;*          *
        GO TO BEGIN                             ;***********


;       CALLING SEQUENCE
;               ROW=            ;SET Y-COORDINATE
;               COL=            ;SET X-COORDINATE
;               VTLIM=          ;SPECIFY MAXIMUM NUMBER OF CHARS
```

```
;                                   ;TO ACCEPT
;               CALL VT05

;ACCEPT A MAXIMUM OF VTLIM CHARACTERS AT LOCATION SPECIFIED BY
;ROW AND COL. RETURN WHEN VTLIM CHARACTERS, A TERMINATION CHARACTER,
;OR A SPACE IS ENTERED. RUBOUT DELETES LAST CHARACTER ENTERED AND
;CTRL/U ELIMINATES THE ENTIRE ENTRY RUBOUT AND CTRL/U DISPLAY
;SPACE(S) TO DELETE ONLY THE NECESSARY CHARACTERS(NOT THE
;REMAINDER OF THE LINE)

VT05,   VT05IN=
        KBDBUF=
VT052,  ACCEPT(TCHAR,CHAR)
        IF (TCHAR.EQ.0) GO TO VT053     ;NON-TERMINATING CHARACTER
        IF (TCHAR.EQ.21) GO TO VT054    ;CTRL/U (TO USE LEFT ARROW
                                        ;ON VT05 KEYBOARD,
                                        ;CHANGE 21 TO 8)
        IF (TCHAR.EQ.32) GO TO VT055    ;RUBOUT
        RETURN                          ;TERMINATING CHARACTER OTHER
                                        ;THAN RUBOUT OR CTRL/U
VT053,  IF(CHAR.EQ.' ') RETURN          ;SPACE IS A TERMINATING CHAR
                                        ;(TO ELIMINATE THIS FEATURE,
                                        ;REMOVE THIS STATEMENT)
        INCR VT05IN                     ;VT05IN=# OF INPUT CHARACTERS
        KBDIN(VT05IN)=CHAR
        IF (VT05IN.EQ.VTLIM) RETURN     ;THE SPECIFIED NUMBER OF
                                        ;CHARACTERS WERE INPUT
        GO TO VT052
VT054,  IF(VT05IN.EQ.0) GO TO VT05
        DISPLAY(ROW,COL,BLNK72(1,VT05IN))    ;CLEAR CHARACTERS ENTERED
        DISPLAY(ROW,COL,0)              ;REPOSITION CURSOR
        GO TO VT05
VT055,  IF(VT05IN.EQ.0) GO TO VT052
        KBDIN(VT05IN)=
        VT05IN=VT05IN-1
        VT05XX=VT05IN+COL
        DISPLAY(ROW,VT05XX,' ')         ;RUBOUT PREVIOUS CHARACTER
        DISPLAY(ROW,VT05XX,0)           ;REPOSITION CURSOR
        GO TO VT052
```

1.4.3

In addition to the features found in the previous program, the following features might also be desired:

1. Programmable numeric keypad (not to be confused with the CTRL/N feature of COS 300 which is described in Chaper 2 and is not programmable).

2. Right justification of numeric fields.

3. Automatic cursor positioning.

These features can be found in the following subroutine and test program:

```
        START   ;SUBROUTINE VT05A AND VT05N
        RECORD
KBDBUF, A72     ;STORAGE FOR KEYBOARD INPUT
        RECORD ,X
KBDIN,  72A1

        RECORD          ;WORK AREA
BLNK72, A72     ;72 BLANK CHARACTERS
ROW,    D2      ;CURSOR Y-COORDINATE
COL,    D2      ;CURSOR X-COORDINATE
TCHAR,  D2      ;TERMINATING CHARACTER IN AN ACCEPT STATEMENT
CHAR,   A1      ;INPUT CHARACTER FROM AN ACCEPT STATEMENT
VT05IN, D2      ;NUMBER OF CHARACTERS ACCEPTED
VTLIM,  D2      ;NUMBER OF CHARACTERS TO BE ACCEPTED
VT05I,  D1      ;INDEX FOR VT05TA
VT05TA, 7A1,'M','J','K','L','U','I','O' ;NUMERIC KEYPAD OVERLAY KEYS
VT05SW, D1      ;CLEARED FOR ALPHA INPUT, SET TO 1 FOR NUMERIC INPUT
VT0515, D15     ;CONTAINS NUMERIC INPUT FOR VT05N ENTRY (NOT CHANGED
                ;OR USED IN VT05A ENTRY)
VT05XX, A16     ;TEMPORARY STORAGE FOR REDISPLAY OF NUMERIC INPUT
        PROC 0
        DISPLAY(1,1,1)  ;CLEAR SCREEN           ;***********
BEGIN,  INCR ROW                                ;*          *
        IF (ROW .GT. 20) STOP                   ;*          *
        DISPLAY (ROW,53,'NOT ERASED')           ;*          *
        DISPLAY (ROW,1,'NAME:')                 ;*  SAMPLE  *
        COL=7                                   ;*          *
        VTLIM=20        ;20 CHARS. MAX          ;*   TEST   *
        CALL VT05A                              ;*          *
        IF (KBDBUF.EQ.'END') STOP               ;* PROGRAM *
        DISPLAY(ROW,30,'NO:')                   ;*          *
        COL=34                                  ;*          *
        VTLIM=15                                ;*          *
        CALL VT05N                              ;*          *
        GO TO BEGIN                             ;***********



;       CALLING SEQUENCE
;               ROW=            ;SET Y-COORDINATE
;               COL=            ;SET X-COORDINATE
;               VTLIM=          ;SPECIFY MAXIMUM NUMBER OF CHARACTERS
;                               ;TO ACCEPT
;               CALL VT05A (OR VT05N)   ;VT05A IS FOR ALPHANUMERIC
;                               ;INPUT; VT05N IS FOR NUMERIC INPUT

;ACCEPT A MAXIMUM OF VTLIM CHARACTERS AT LOCATION SPECIFIED BY ROW
;AND COL. RETURN WHEN VTLIM CHARACTERS OR A TERMINATION CHARACTER
;IS ENTERED.  FOR NUMERIC INPUT, A SPACE IS A TERMINATOR.
;RUBOUT DELETES LAST CHARACTER ENTERED AND CTRL/U ELIMINATES THE
;ENTIRE ENTRY. RUBOUT AND CTRL/U DISPLAY SPACE(S) TO DELETE ONLY
;THE NECESSARY CHARACTERS (NOT THE REMAINDER OF THE LINE).
;FOR NUMERIC INPUT, THE ENTIRE ENTRY IS REDISPLAYED RIGHT-JUSTIFIED,
;WITH LEADING ZEROS SUPPRESSED. (VT0515 CONTAINS THE NUMBER
;ON RETURN TO THE CALLING PROGRAM).

VT05A,  VT05SW=         ;ENTRY FOR ALPHANUMERIC INPUT
        GO TO VT05
```

```
VT05N,  VT05SW=1        ;ENTRY FOR NUMERIC INPUT
VT05,   VT05IN=
        KBDBUF=
        DISPLAY(ROW,COL,0)      ;POSITION CURSOR
VT052,  ACCEPT(TCHAR,CHAR)
        IF(TCHAR.EQ.0) GO TO VT053      ;NON-TERMINATING CHARACTER
        IF(TCHAR.EQ.21) GO TO VT054     ;CTRL/U (TO USE LEFT ARROW
                                        ;ON VT05 KEYBOARD,
                                        ;CHANGE 21 TO 8)
        IF (TCHAR.EQ.32) GO TO VT055    ;RUBOUT
VT052X, IF (VT05IN.EQ.0) RETURN         ;NO INPUT EXCEPT TERMINATING
                                        ;CHARACTER
        IF (VT05SW.EQ.0) RETURN         ;ALPHANUMERIC INPUT
VT052Y, VT0515=KBDBUF(1,VT05IN)         ;NUMERIC INPUT (CAN'T EXCEED
                                        ;15 DIGITS)
        VT05XX(1,VTLIM+1)=VT0515,'XXXXXXXXXXXXXXX-' ;ALLOWS NEGATIVE
                                        ;NUMBERS
        DISPLAY(ROW,COL,VT05XX(1,VTLIM+1))      ;DISPLAY NUMERIC INPUT
                                        ;RIGHT-JUSTIFIED AND ZERO
                                        ;SUPPRESSED
        RETURN
VT053,  IF (VT05SW.NE.1) GO TO VT053X   ;SAVE ALPHANUMERIC INPUT
        IF (CHAR.EQ.' ') GO TO VT052X   ;SPACE AS A TERMINATING
                                        ;CHARACTER FOR NUMERIC INPUT
        IF (CHAR.EQ.'-') GO TO VT053X   ;MINUS SIGN IS OK
        VT05I=
        IF (CHAR.LT.'0') GO TO VT053B   ;CHECK FOR NUMERIC KEYPAD
        IF (CHAR.LE.'9') GO TO VT053X   ;0-9 INPUT
VT053B, INCR VT05I
        IF (CHAR.EQ.VT05TA(VT05I)) GO TO VT053C ;MATCH
        IF (VT05I.NE.7) GO TO VT053B    ;NO MATCH, TRY AGAIN
        DISPLAY (0,0,25)                ;SOUND BELL--NOT NUMERIC INPUT
        GO TO VT05                      ;START OVER (DON'T CLEAR
                                        ;THE ERROR)
VT053C, CHAR=VT05I-1                    ;SAVE PROPER VALUE
        DISPLAY(ROW,COL+VT05IN,CHAR)    ;REDISPLAY NUMERIC VALUE
VT053X, INCR VT05IN                     ;VT05IN=# OF INPUT CHARACTERS
        KBDIN(VT05IN)=CHAR
        IF (VT05IN.EQ.VTLIM) GO TO VT056    ;THE SPECIFIED NUMBER OF
                                        ;CHARACTERS WERE INPUT
        GO TO VT052
VT054,  IF (VT05IN.EQ.0) GO TO VT05
        DISPLAY(ROW,COL,BLNK72(1,VT05IN))    ;CLEAR CHARACTERS ENTERED
        GO TO VT05
VT055,  IF (VT05IN.EQ.0) GO TO VT052
        KBDIN(VT05IN)=
        VT05IN=VT05IN-1
        VT05XX=VT05IN+COL
        DISPLAY(ROW,COL+VT05IN,' ')     ;RUBOUT PREVIOUS CHARACTER
        DISPLAY(ROW,COL+VT05IN,0)       ;REPOSITION CURSOR
        GO TO VT052
VT056,  IF(VT055W.EQ.1) GO TO VT052Y
        RETURN
```

1.4.4

For those who would like to have a subroutine that uses special function keys, such as up arrow and down arrow, or a new feature, such as alpha override on a numeric field, there is a simple solution--understand the previous subroutine and modify it to your needs.

2.0 DIRECT ACCESS TECHNIQUES

2.1 Background

A file may contain records of fixed or variable length. With COS 300 direct access cannot be performed on a file containing variable length records.

Records are written in 512 character blocks. Regardless of the record size, the operating system automatically blocks the records into 512 character blocks. The size of a record (in characters) is two plus the size of all the fields in the record. (The first two characters are the record size in characters divided by two.) If the resulting size is odd, add one character since only an even number of characters may be written. For example, if there are two fields in a record which are defined as a D9 field and an A88 field, the record size is 100 (2+9+88+1). The operating system will pack five records and twelve characters into the first block, 88 characters, four records and 24 characters into the second block, etc.

When this file is later processed, either sequentially (defined as input in an INIT statement) or thru direct access (defined as UPDATE in an INIT statement), the operating system will completely restore the record even if it overlaps two blocks, before passing it to the DIBOL program.

2.2 The Reason for Direct Access

Many applications involve the processing of data sequentially. For example, a transaction file is entered in random order, sorted and then used to update a master file sorted in the same sequence. Errors in the transaction file cannot be found until the UPDATE program is run. The errors are corrected and a new transaction file is made for the corrected items, which is then sorted and run against the master file. This process continues until no more errors exist. This type of processing evolved 20 years ago with the age of electronic data processing. Systems specialists have desired a better method of operation. The best way is to verify when data is entered that it is correct. The operator keying the data file should be able to interact with the master file. This is now possible with COS 300. For example, a program can be written in which an operator entering payroll information could type an employee number and know within 1/10th of a second whether this employee exists on the master file. This would be impossible with sequential processing of the master file, since the operator would spend 90% of her time waiting for the

employee record to be found. Direct access permits retrieval of any desired record without processing any other records.

2.3 How It Works In DIBOL

DIBOL permits a program to access any record in a file by specifying the desired record number. Since the operator is specifying some code (which is usually not a record number) the program will have to convert this code to the record number containing this code. The remainder of this section on direct access will explain several methods to do this.

2.4 Unsorted File

Assume that you have an unsorted file containing 1 to 99 records. Each record contains a "key" field as well as other fields. This key will be used for direct access. Look at the following program. Notice that the first thing done is to fill up a table. There is a one to one correspondence between each element in the table and each record in the file. Also note that no I/O is necessary to determine if a specified code is in the master file since this code would not have a match in the table lookup.

```
        RECORD MASTER
KEY,    D5              ;COULD BE ANY SIZE DECIMAL OR ALPHA FIELD
,       A90             ;REMAINDER OF FILE
        RECORD          ;WORKING STORAGE
TABLE,  100D5           ;TABLE CONTAINING KEYS
I,      D3              ;INDEX
LOOKUP, D5
        PROC 1
        INIT(1,INPUT,'FILNAM')
LOAD,   XMIT(1,MASTER,EOF)
        INCR I
        TABLE(I)=KEY
        GOTO LOAD
EOF,    FINI(1)
        INCR I
        TABLE(I)=99999  ;INDICATES END OF TABLE
        INIT(1,UPDATE,'FILNAM')
        .
        .               ;LOOKUP CONTAINS CODE FOR MASTER FILE LOOKUP
        .
        I=
FINDIT, INCR I
        IF(TABLE(I).EQ.LOOKUP) GO TO FOUND      ;MATCH
        IF(TABLE(I).EQ.99999) GO TO NONE        ;NO MATCH
        GOTO FINDIT
FOUND,  READ(1,MASTER,I)                ;READ THE ITH RECORD
```

2.5 Sorted File

Take the same circumstances as in section 2.4 except that the file is sorted by key. Filling the table is the same, but table lookup is faster since the code is not compared to every element in the table (as in example in section 2.4). Thus, a "no match" condition is known as soon as the table element exceeds the code.

It would also be possible to cut down the number of comparisons in the table lookup by comparing the middle of the table to the code, determine which half of the table might contain the code, check the middle of that half of the table, and so on until the element were found. This techique would be faster, but obviously, the programming would be much more complicated.

```
        RECORD MASTER
KEY,    D5
,       A90
        RECORD          ;WORKING STORAGE
TABLE,  100D5
I,      D3
LOOKUP, D5
        PROC 1
        INIT(1,INPUT,'FILNAM')
LOAD,   XMIT(1,MASTER,EOF)
        INCR I
        TABLE(I)=KEY
        GOTO LOAD
EOF,    FINI(1)
        INCR I
        TABLE(I)=99999          ;INDICATES END OF TABLE
        INIT(1,UPDATE,'FILNAM')
        .
        .                       ;LOOKUP CONTAINS CODE FOR MASTER FILE
        .
        I=
FINDIT, INCR I
        IF(TABLE(I).EQ.LOOKUP) GO TO FOUND      ;MATCH
        IF(TABLE(I).GT.LOOKUP) GO TO NONE       ;NO MATCH
        GOTO FINDIT
FOUND,  READ(1,MASTER,I)                ;READ THE ITH RECORD
```

It is impractical to do direct access with DIBOL on an unsorted file containing many records, since an exceedingly large table would be needed. This will become apparent in Section 2.6.

2.6 Rough Table, No Index File

At some point a file will contain too many records for each key to be saved in a table. When this point is reached, two solutions are available. The first is to create a "rough" index table containing every 10th or 20th key. For lookup, the rough index will specify within 10 or 20 records on the master file which one is desired. These 10 or 20 records are then sequentially examined to find the desired record (see the following program example). The second solution is to create a "rough" index table and a "fine" index file.

In this method, the rough index table specifies within 10 or 20 records on the index file. The index file is then sequentially examined to find the desired key. If a match occurs, the master file is then read.

Why go through the extra step of an index file instead of searching through 10 or 20 records on the master file? To cut down on the number of I/O reads. For example, a master file of 98 characters per record would take up to four I/O reads to find the desired record if the rough index could narrow within 20 records (refer to Section 2.1--record size and records per block). An index file technique would only take one I/O read to find the master record. This technique becomes faster as the size of the master file record increases. Refer to Section 2.7 for an example using index file method.

```
        RECORD MASTER
KEY,    D5
,       A90
        RECORD          ;WORKING STORAGE
TABLE,  100D5           ;1ST,21ST,41ST KEY,ETC.
I,      D4
J,      D4
LOOKUP, D5
        PROC 1
        INIT (1,INPUT,'FILNAM')
LOAD,   XMIT(1,MASTER,EOF)
        INCR I
        IF (I.NE.I/20*20+1) GO TO LOAD
        INCR J
        TABLE(J)=KEY            ;SAVE ONLY 1ST,21ST,41ST KEYS ETC.
        GO TO LOAD
EOF,    FINI(1)
        INCR J
        TABLE(J)=99999          ;INDICATES END OF TABLE
        INIT(1,UPDATE,'FILNAM')
        .
        .               ;LOOKUP CONTAINS CODE FOR MASTER FILE
        .
        I=1
ROUGH,  INCR I
        IF(TABLE(I).LE.LOOKUP) GO TO ROUGH      ;NO ROUGH MATCH YET
        I=(I-2)*20      ;SET I TO BEGINNING OF ROUGH INDEX MINUS 1
FINE,   INCR I
        READ(1,MASTER,I)
        IF(KEY.LT.LOOKUP) GO TO FINE    ;NO MATCH YET
        IF(KEY.GT.LOOKUP) GO TO NONE    ;NO MATCH
        .
        .               ;MATCH
        .
```

2.7 Rough Table Plus Index File

```
        RECORD MASTER
KEY,    D5
,       A90
        RECORD          ;WORKING STORAGE
TABLE,  100D5           ;1ST,21ST,41ST KEY ETC.
I,      D4
J,      D4
LOOKUP, D5
        RECORD INDEX    ;INDEX FILE
XKEY,   D5
        PROC 2
        INIT(1,INPUT,'FILNAM')
        INIT(2,OUTPUT,'XFILE')
LOAD,   XMIT(1,MASTER,EOF)
        INCR I
        XKEY=KEY
        XMIT(2,INDEX)   ;CREATE FINE INDEX FILE
        IF(I.NE.I/20*20+1) GO TO LOAD
        INCR J
        TABLE(J)=KEY            ;SAVE ONLY 1ST,21ST,41ST KEYS ETC.
        GO TO LOAD
EOF,    FINI(1)
        FINI(2)
        INCR J
        TABLE(J)=99999          ;INDICATES END OF TABLE
        INIT(1,UPDATE,'FILNAM')
        INIT(2,UPDATE,'XFILE')
        .
        .       ;LOOKUP CONTAINS CODE FOR MASTER FILE
        .
        I=1
ROUGH,  INCR I
        IF(TABLE(I).LE.LOOKUP) GO TO ROUGH      ;NO ROUGH MATCH YET
        I=(I-2)*20      ;SET TO BEGINNING OF ROUGH INDEX MINUS 1
FINE,   INCR I
        READ(2,INDEX,I)         ;READ INDEX RECORD
        IF(XKEY.LT.LOOKUP) GO TO FINE   ;NO MATCH YET
        IF(XKEY.GT.LOOKUP) GO TO NONE   ;NO MATCH
        READ(1,MASTER,I)        ;MATCH
```

2.8 Summary

This discussion on direct access has not described all situations. In cases where the master file is between 2,000 and 40,000 records, the approach might be to have a very rough table, a rough index file, a fine index file and a master file. Several other techniques have been developed for direct access, but since they are more difficult to use, a discussion of them will be omitted at the present time.

It would also be possible to work with a large unsorted master file. An index file is created containing two fields: the key field and the record number of the master file. The index file is then sorted by key. When a match is found on the key field of the index file, the program would use the record number field to read the proper record of the unsorted master file.

Creation of an index table or an index file may be done in a separate program. This could save from several seconds to several minutes each time the program is run. Thus when a master file is updated (perhaps on a weekly or monthly basis), the index file would then be created only once during this time period.

3.0 DIRECT ACCESS NOTES

3.1 XMIT Statements Used With Direct Access

XMIT statements may be interspersed with direct access operations on a file. An XMIT following a READ with record number N is equivalent to a READ of record number N+1. Successive XMIT's read records N+2, N+3, etc.

An XMIT following a WRITE of record number N transmits data to record number N+1.

CAUTION

> Records N+2 to the end of the file may be changed by successive XMIT's after a WRITE. Therefore if the user wants to change a series of records in the middle of the file, he should not use a WRITE followed by several XMIT's.

The XMIT statement used after a WRITE statement has the following useful applications.

3.1.1 Truncating a File

To truncate a file after record N use the following sequence:

```
          READ(channel,record,N)
          WRITE(channel,record,N)
          XMIT(channel,NULL,DUMEOF)
          .
          .
DUMEOF,   FINI(channel)
```

where NULL is a record with no contents defined by:

```
RECORD NULL
RECORD
```

3.1.2 Appending to a File

To append records to the end of a file with N records, use the following sequence:

```
                READ(channel,record,N)
                WRITE(channel,record,N)
                XMIT(channel,record)        ;APPEND RECORDS TO FILE
                XMIT(channel,record)
                     .
                     .
                     .
                XMIT(channel,NULL,DUMEOF)
                     .
                     .
                     .
        DUMEOF, FINI(channel)
```

3.1.3 Rewriting A File

To rewrite a file from record N to the end of the file, use the following sequence:

```
                WRITE(channel,record,N)
                XMIT(channel,record)
                XMIT(channel,record)
                     .
                     .
                     .
                XMIT(channel,NULL,DUMEOF)
                     .
                     .
                     .
        DUMEOF, FINI(channel)
```

4.0 NUMERIC FIELD VERIFICATION

Any numeric field that is first entered in a DIBOL program should be checked to determine if it contains only decimal data. The numeric field should be read as an alpha field through an XMIT or ACCEPT statement. Then it is moved to a decimal field. This move is preceded by an ON ERROR statement to check for non-numeric data. For example:

```
    RECORD                          RECORD A
TCHAR,   D2                     ALPHA,   A5
DECMAL,  D5                     DECMAL,  D5
ALPHA,   A5
PROC                            PROC 1
.                               .
.                               .
.                               .
    ALPHA=                          XMIT(n,A)
    ACCEPT(TCHAR,ALPHA)             ON ERROR FIX
    ON ERROR FIX                    DECMAL=ALPHA
    DECMAL=ALPHA                .
.                               .
.                               .
.
```

With an alpha-to-decimal move, many checks are done. Some of them include right justification, + and - sign conversion and ignoring spaces. The following examples should cover all cases:

ALPHA	DECIMAL
' 123'	00123
'123 '	00123
'00123'	00123
' -123'	0012S
' 123-'	0012S
'-123-'	00123
' 12-3'	0012S
'1-2-3'	00123
'1+2+3'	00123
'1+2-3'	0012S
'1 23 '	00123
'0012S'	ILLEGAL

The only legal characters in an alpha-to-decimal move are 0 to 9, space, +, and -.

If a data file is written which contains numeric fields, these fields must be read as numeric. If they contain a negative number, the least significant character contains a minus sign. For example, -37 would look like 3W. If this field were read as alpha, and then converted to numeric, a run time error would occur since any letter of the alphabet is illegal in a alpha-to-decimal conversion.

5.0 CHAIN STATEMENT NOTES

5.1 Interaction of CHAIN and INIT (n,SYS)

SYS input files may be specified in a RUN statement which also specifies CHAIN programs. However, the method of accessing such files must adhere to the following rules.

1. All CHAIN files must be listed in the RUN command before the SYS FILES. FOR EXAMPLE:

```
.RUN PROG+CHAIN1+CHAIN2,INP1,INP2
```

2. Any CHAIN which is to open the first SYS input file must first "skip over" the remaining CHAIN files by issuing dummy INIT(n,SYS) statements. In the above RUN command, for example, in order to read file INP1, the program PROG would have to issue two dummy INIT(n,SYS) statements. CHAIN2 would not have to issue any dummy statements since it is the last CHAIN program.

If the RUN command were:

```
. RUN PROG, source1,...,source7
```

the source files could be processed more than once by executing a CHAIN0 statement in PROG.

5.2 Communication Between CHAINs

5.2.1 File Status

All file status information is destroyed between CHAINed programs. Therefore, all OUTPUT and UPDATE files should be FINI'd before executing a CHAIN to prevent the loss of information.

5.2.2 Clearing CHAINed Records

Any DIBOL record in a program loaded into core directly by the .RUN command is automatically cleared. If the record is loaded in a program by the CHAIN statement in the PROC section of another program, it will retain whatever contents it may have had in the previous program unless the clear option (,C) is specified for the record.

5.2.3 Tranferring Variable Values

For the value of a variable to be successfully transmitted from one CHAIN'ed program to another, the variable in which the value appears must occupy the same location in both CHAIN programs. This may be accomplished by either of two methods:

1. Define all records which are to be passed between CHAIN programs before other records, and make the declarations identical (except for variable names which may be different).

   Example:

```
CHAIN1                      CHAIN2

RECORD                      RECORD CPINFO
CUST, A30                   CUST, A30
PROD, D2                    PCODE, D2
RECORD INVENT               RECORD INVENT
STOCK, D4                   QUANTY, D4
.                           .
.                           .
.                           .
```

2. Use the Storage Maps, produced by the compiler, for the two CHAIN programs to verify that the desired variables occupy the same storage location.

5.3 Multiple CHAIN Entry Points

Sometimes it is desirable to have several entry points into a CHAIN program. However, the CHAIN statement always starts execution of the CHAINed program at the first statement following the PROC statement. Using the technique of transferring variable values between CHAINed programs (discussed in Section 5.2.3), multiple entry points can be programmed as indicated in the following example.

```
        CHAIN1                                  CHAIN2

RECORD                                  RECORD
WHERE,D2,01                             WHERE,D2
RETURN,D2                               RETURN,D2
.                                       .
.                                       .
.                                       .
PROC                                    PROC
GO TO(L1,L2,L3,L4),WHERE                GO TO(E1,E2,E3),WHERE
L1,RETURN=2
.                                       .
.                                       .
.                                       .
CHAIN 2                                 E1,...
L2,...                                  .
.                                       .
.                                       .
.                                       WHERE=RETURN
                                        CHAIN 1
```

5.4 Miscellaneous CHAIN Facts

1. Both the TRACE and TRAP features are turned off when a CHAIN statement is executed. They may be turned on again in the CHAINed program via an appropriate TRACE or TRAP instruction. Control returns to DDT when a CHAIN statement is executed.

2. The size of the data area in a program may be larger than, smaller than, or equal to the size of the data area in any program that it chains to.

3. Issuing a CHAIN statement with an argument greater than 7 (e.g., CHAIN 8) results in the error message

   ILLEGAL CHAIN

4. Issuing a CHAIN statement which does not correspond to a valid DIBOL binary file in the .RUN statement results in the error message:

   ILLEGAL CHAIN
   AT LINE 0000

6.0 DIBOL PROGRAMMING OF SYS FILES

6.1 Operating Procedures

Up to seven source files are available to a DIBOL program. They are specified at run time by:

```
. RUN PROG, source1,...,source7
```

The Editor scratch area is not available to a DIBOL program.

6.2 Data Section

The RECORD description would be as follows:

```
RECORD recnam
LINENO, A2
CHAR, A120
```

LINENO contains a two character line number in binary. Most programs would ignore the line number. However, it can be converted to decimal by the statement:

```
varnam = #LINENO*64+#LINENO(2,2)
```

Varnam must be a four digit field.

CHAR contains the characters of one line created by the Editor. The DIBOL program may want to examine CHAR for trailing spaces to determine the last character in CHAR.

There is no tabbing within CHAR. The tabbing seen by output from the Monitor command LIST or LIST/L is done by the operating system. Tabs are internally stored as characters with a decimal equivalent of 61. Any character may be checked for a tab by the statement:

```
IF(#CHAR(n,n).EQ.61) GO TO LABEL
```

6.3 PROCedure Section

The first source file specified in the RUN command is opened by the statement:

```
INIT(n,SYS)
```

Each record is accessed by the statement:

```
XMIT(n, recnam, EOF label)
```

When an end-of-line condition occurs, the program transfers to the EOF label of the XMIT statement. At that EOF label, a FINI n statement must be executed prior to an INIT(n,SYS) to open a second source file. To handle a variable number of source files, precede the INIT(n,SYS) statement by an ON ERROR label statement. The program would transfer to the ON ERROR label statement when an INIT(n,SYS) statement was executed and there were no more source files.

The only way of processing a source file more than once is to execute a CHAIN n statement which resets the operating system pointers to the source file(s). Refer to Section 5.1 of this appendix for an explanation of CHAINing and source files.

alphanumeric	A character set that contains letters, digits, and other characters such as punctuation marks. The COS alphanumeric character set includes the upper case 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. 5D3 reserves space for five decimal fields, each to be three digits long. 2A10 describes two alphanumeric fields, each to be ten characters long.
ASCII	American National Standard Code for Information Interchange. This is one method of coding alphanumeric characters.
batch processing	The technique of automatically executing a group of monitor commands such that each is completed before the next is started. In COS the BATCH command executes a group of previously stored commands, allowing "unattended" system operation.
binary operator	An operator such as +, -, etc., which acts upon two or more constants or variables (e.g., A=B-C).
binary program	The form of user's program which is output by the compiler and runs with RSYS on the computer.
bit	A binary digit (0 or 1).
blank	A part of a medium in which no characters are recorded.
block	The basic COS unit of physical data transfer used primarily to determine storage capacity. A block consists of up to 510 characters. A DECtape reel contains 737 blocks; an RK08 disk cartridge contains 3248; an RK8E disk cartridge contains 6496; an RF08 disk platter contains 1024 blocks.

bootstrap	A short routine loaded at system startup time to be able to read in system software.
branch	A change in the sequence of execution of COS program statements.
buffer	A temporary storage area usually used for input or output data transfers.
bug	A program error, or a hardware malfunction.
character	A letter, digit, or other symbol used to control or to represent data. See switch character.
character string	A linear sequence of characters.
clear	Setting an alphanumeric field to space characters, or a decimal field to zeros. In the Data Definition section of a DIBOL program C initially clears a RECORD storage area.
collating sequence	An ordering assigned to a group of records based on a key item or field within the records. One possible ascending sequence is 0-9, A-Z.. Then the descending sequence is Z-A, 9-0.
command	An operator request for Monitor services; usually to be executed immediately.
command string	The characters that make up a complete command.
COMP	The DIBOL compiler program which translates from source programs written in DIBOL language to binary programs which run on the computer.
comments	Notes for people to read; ignored by the compiler. Optional, following a semicolon on any statement line.
code	(1) the representation of information, as in ASCII code. (2) A set of instructions or statements called "a piece of code". (3) "To code" means to write a program.
data	A representation of information in a manner suitable for communication, interpretation, or processing by either people or machines. In COS systems, data is represented by characters.

data base	The entire set of data files available for processing by a COS data management system.
data definition	The specification of record formats in both format programs and source programs. Gives the length of each field, states whether it is alphanumeric or decimal, and may give a field name and initial entry. Data definitions are stored on the systems device, and may be referenced by any other COS program.
data entry	The process of collecting and inputting data into the computer data files. Data entry is either key-to-tape or key-to-disk. The systems utility program, BUILD, checks the incoming data for type and length, and writes the records on DECtape or disk.
debug	To detect, locate, and remove errors or malfunctions from a program or machine.
DEC	Acronym for Digital Equipment Corporation.
DECtape reel	Each 4-inch reel contains 260 feet of 3/4-inch wide magnetic tape. Each reel is a logical file of up to 737 blocks of 512 characters each. A large file may consist of up to 63 reels.
data management	The planning, development, and operation of a system like COS by an organization to mechanize its information flows and make available the data needed by the organization.
detail file	Same as transaction file.
device independence	COS system design permits data files and programs to be stored on either DECtape or disk. At run-time, the operator chooses the most suitable, or available, input and output devices.
device names	3-character abbreviations are used to name the COS I/O devices.

```
DT0-DT7 DECtapes 0-7
TTY     Terminal printer
KBD     Terminal keyboard
RDR     Paper tape reader
PTP     Paper tape punch
CDR     Card reader
LPT     Line printer
DK0-DK3 Disk drives
```

DIBOL	DIgital's Business Oriented Language. The source language of the COS 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 DECtapes and disks in COS 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	See Systems Directory.
disk	A standard mass storage device in COS systems giving very fast access to data files and programs.
dump	To copy the contents of all or part of storage, usually from core memory to external storage.
end of tape mark	Control characters which mark the physical end of a multi-reel (multi-disk) file. For both input and output files, Monitor will detect this EOT mark and type a message for the operator asking that the next reel in this file be mounted.
end of file mark	Identifies the end of the logical file.
error message	An indication that an error has been detected.
field	A specified area in a data record used for one item of alphanumeric or decimal data, which cannot exceed the specified character length.
file	A collection of records, treated as a logical unit.
fixed-length records	When each record in a data file is the same length. Fixed-length records are the only type handled by COS utility programs, and the only type on which direct access to data files is allowed. See also variable-length records.
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 flows.

illegal character	A character that is not valid according to the COS design rules. DIBOL will not accept back slash (\) and back arrow (←) (backarrow is replaced on some terminals with underscore) in alphanumeric strings.
input	Data flowing into the computer to be processed by a binary program is input data. When the processed data flows out of the computer, it is output data.
input/output	Either input or output, or both. I/O
instruction	A program statement that specifies an executable computer operation.
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 decimal 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.
label	One or more characters used to identify a statement in a COS source program up to a maximum of 6 characters.
leader	The blank section of tape at the beginning of a DECtape or papertape.
library	A collection of related files. For example, the collection of inventory control files may form a library, and the libraries used by an organization are known as its data bank.
library routine	A proven routine that is maintained in a program library.
line printer	A high-speed output device that prints all the characters of a line as a unit. Speed of the LP08 Line Printer ranges from 245 to 1110 lines per minute. LS8E

	has a speed of 165 characters per second.
linkage	Coding that connects two separately coded routines.
load	To enter data or programs into main core storage.
load-and-go	An operating technique in which there are no stops between the loading and execution phases of a program.
location	Any place where data may be stored.
logical file	A collection of logical records independent of their physical environment. Portions of the same logical record may be located in different physical blocks.
logical units	A section of mass storage. Up to 15 logical units may be assigned at system startup by the SYSGEN program. These areas and their assigned sizes are listed in the logical units table printed by SYSGEN.
loop	A sequence of instructions that is executed repeatedly until a terminal condition prevails. A commonly used programming technique in processing data records.
magnetic core	The very fast, direct access, storage media used in the PDP-8E's main internal memory. Contains 2 COS characters per 12-bit word. An 8K core stores over 16,000 characters.
main memory	Or main storage. The computer's primary internal storage.
mass storage device	A device having large storage capacity, such as DECtapes and disks.
master file	A data file that is either relatively permanent, or that is treated as an authority in a particular job.
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 TTY, RDR and DT4.
Monitor	COS's top system program that loads and runs other programs and performs many other useful tasks.
name	A name identifies the place where a file, a field, or a statement is stored. Names must start with a letter and may contain up to 6 characters, not including embedded spaces.
nest	To embed subroutines or loops or data in other subroutines or programs.
object program	A compiled program in binary form ready to be loaded and executed.
off line	Equipment or devices that are not under control by the computer.
on line	Equipment or services under control 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.
parametric programming	COS control programs are parametric programs. The applications programmer lists the description of his data files as parameters in the Data Definition, and the processing needed is listed in another section as another parameter list. Each parameter list is written by answering a series of questions. The DIBOL data language is used for writing procedural programs.
pass	One cycle of processing a body of data.
patch	To modify a routine or program in a rough or expedient way.
peripheral equipment	Data processing equipment which is distinct from the computer. DECtapes, disks and card readers are examples.
position	In a string, any location that may be occupied by a character.
program	See source program, binary program, object program, format program.

program library	An organized collection of computer programs, off line storage media, and related documentation.
programmers	People who design, write, and test computer programs.
pseudo-random numbers	A sequence of numbers, computed by an arithmetic process, that is satisfactorily random for a given purpose. Such a sequence may approximate a statistical distribution such as uniform, normal, or Gaussian.
push-down list	A list of items where the last item entered is the first item in the list, and the relative position of the other items is pushed back one.
random access	Similar to direct access.
range	The difference between the highest and lowest values that a quantity or function may assume. For example, the range of decimal numbers that COS systems can process is: -999,999,999,999,999 to +999,999,999,999,999.
real time	Use of a computer to guide, control, or acquire data from a related physical process, during the actual time that the physical process transpires.
record	A collection of related data fields, and the basic logical unit in COS data files. A RECORD statement reserves core storage areas for DIBOL data language programs. See also fixed-length and variable-length records. Maximum record size is 510 characters.
segment	To divide a program or file into parts such that the program can be executed without the entire program being in internal storage at any one time.
sequential operation	Performance of operations, such as record processing, 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. Most magnetic tapes are serially accessed, but DECtapes have fixed addresses and COS programs have fast, direct access to their DECtape records.
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 for a specified purpose, especially a digit that must be kept to preserve a certain accuracy or precision. Leading zeros are not significant.
simulate	A computer program that represents the behavior of another system. An example would be a program which simulates the behavior of a market when a new product is introduced.
source program	A program written in COS data language, DIBOL.
space fill	To fill the remaining character positions in an alphanumeric field with space characters.
special character	A graphic character that is neither a letter, nor a digit, nor a space character.
spool	To provide for continuous operation of a peripheral device.
statement	An instruction in a source program.
string	A linear sequence of characters.
switch character	A single letter specified in a command following a slash (/).
syntax	The rules governing the structure of a language.
system device	A mass storage device reserved for Monitor, RSYS and other system and user programs. This is always logical unit 0.

systems directory	A list of programs on the systems device with lengths, dates of creation and other useful information.
tape drive	A device that moves tape past a head. Synonymous with tape transport.
terminal	A point in the system at which data can either enter or leave.
transaction file	A file containing relatively transient data to be processed in combination with a master file. For example, in a payroll application, a transaction file indicating hours worked might be processed with a master file containing employee name and rate of pay. Synonymous with detail file.
unary operator	An operator such as +, -, etc., which acts upon only one variable or constant (e.g., A=-C).
utility program	A group of system programs which perform common services and require format programs. Examples are BUILD, SORT and UPDATE.
variable	A quantity that can assume any of a given set of values.
variable-length record	A file in which the data records are not uniform in length. Specified by V in an INIT statement. Variable length records may be created by DIBOL source programs only, but cannot be processed by utility programs, and direct access to such records by system programs is impossible.
verify	To determine whether a transcription of data has been accomplished accurately.
word	A string of 12 binary bits, representing two COS characters.
zero fill	To fill the remaining character positions in a decimal field with zeros.

HOW TO OBTAIN SOFTWARE INFORMATION

Announcements for new and revised software, as well as programming notes, software problems, and documentation corrections are published by Software Information Service in the following newsletters.

Digital Software News for the PDP-8 and PDP-12
Digital Software News for the PDP-11
Digital Software News for 18-bit Computers

These newsletters contain information applicable to software available from Digital's Software Distribution Center. Articles in Digital Software News update the cumulative Software Performance Summary which is contained in each basic kit of system software for new computers. To assure that the monthly Digital Software News is sent to the appropriate software contract at your installation, please check with the Software Specialist or Sales Engineer at your nearest Digital office.

Questions or problems concerning Digital's software should be reported to the Software Specialist. In cases where no Software Specialist is available, please send a Software Performance Report form with details of the problems to:

Digital Equipment Corporation
Software Information Service
Programming Department
Maynard, Massachusetts 01754

These forms, which are provided in the software kit, should be fully filled out and accompanied by Teletype output as well as listings or tapes of the user program to facilitate a complete investigation. An answer will be sent to the individual and appropriate topics of general interest will be printed in the newsletter.

Orders for new and revised software manuals, additional Software Performance Report forms, and software price lists should be directed to the nearest Digital Field office or representative. USA customers may order directly from the Software Distribution Center in Maynard. When ordering, include the code number and a brief description of the software requested.

Digital Equipment Computer Users Society (DECUS) maintains a user library and publishes a catalog of programs as well as the DECUSCOPE magazine for its members and non-members who request it. For further information, please write to:

Digital Equipment Corporation
DECUS
Programming Department
Maynard, Massachusetts 01754

COS 300 System
Reference Manual
DEC-08-OCOSA-E-D

READER'S COMMENTS

Digital Equipment Corporation maintains a continuous effort to improve the quality and usefulness of its publications. To do this effectively we need user feedback -- your critical evaluation of this manual.

Please comment on this manual's completeness, accuracy, organization, usability, and readability.

Did you find errors in this manual? If so, specify by page.

How can this manual be improved?

Other comments?

Please state your position. _____ Date: _____

Name: _____ Organization: _____

Street: _____ Department: _____

City: _____ State: _____ Zip or Country _____

— — — — — — — — — — — — — — — — — — 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

Digital Equipment Corporation
Software Information Services
Programming Department
Maynard, Massachusetts 01754

COS 300 System
Reference Manual
DEC-08-OCOSA-E-D

READER'S COMMENTS

Digital Equipment Corporation maintains a continuous effort to improve the quality and usefulness of its publications. To do this effectively we need user feedback -- your critical evaluation of this manual.

Please comment on this manual's completeness, accuracy, organization, usability, and readability.

Did you find errors in this manual? If so, specify by page.

How can this manual be improved?

Other comments?

Please state your position. _____ Date: _____

Name: _____ Organization: _____

Street: _____ Department: _____

City: _____ State: _____ Zip or Country _____

— — — — — — — — — — — — — — — — — — 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

Digital Equipment Corporation
Software Information Services
Programming Department
Maynard, Massachusetts 01754

COS 300 System
Reference Manual
DEC-08-OCOSA-E-D

READER'S COMMENTS

Digital Equipment Corporation maintains a continuous effort to improve the quality and usefulness of its publications. To do this effectively we need user feedback -- your critical evaluation of this manual.

Please comment on this manual's completeness, accuracy, organization, usability, and readability.

Did you find errors in this manual? If so, specify by page.

How can this manual be improved?

Other comments?

Please state your position. _____ Date: _____

Name: _____ Organization: _____

Street: _____ Department: _____

City: _____ State: _____ Zip or Country _____

— — — — — — — — — — — — — — — — — — 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

Digital Equipment Corporation
Software Information Services
Programming Department
Maynard, Massachusetts 01754

digital