decdatasystem

COS310 system reference manual

digital

COS-310
System Reference Manual

Order No. AA-D647A-TA

October 1978

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

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

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

digital equipment corporation · maynard. massachusetts

First Printing, October 1978

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

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

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

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

The following are trademarks of Digital Equipment Corporation:

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

CONTENTS

FIGURES

TABLES

PREFACE

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

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

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

INTRODUCTION

THE DIBOL LANGUAGE

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

OVERVIEW OF THE COS-310 SYSTEM

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

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

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

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

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

THE COS-310 FILE STRUCTURE

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

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

MANUAL NOTATION CONVENTIONS

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

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

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

COS-310 CHARACTERS

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

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

TERMS

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

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

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

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

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

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

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

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

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

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

Numeric
Refers to the characters 0-9 and combinations thereof.

CHAPTER 1

DIBOL LANGUAGE

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

1.1 SOURCE PROGRAM

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

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

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

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

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

Figure 1-1 Sample Source Program

Table 1-1
Source Program Limitations

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

1.2 STATEMENTS

There are six kinds of statements in DIBOL:

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

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

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

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

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

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

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

1.2.1 ACCEPT - Input/Output Statement

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

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

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

where:

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

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

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

Example:

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

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

Table 1-2
Terminating Characters

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

1.2.2 CALL - Control Statement

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

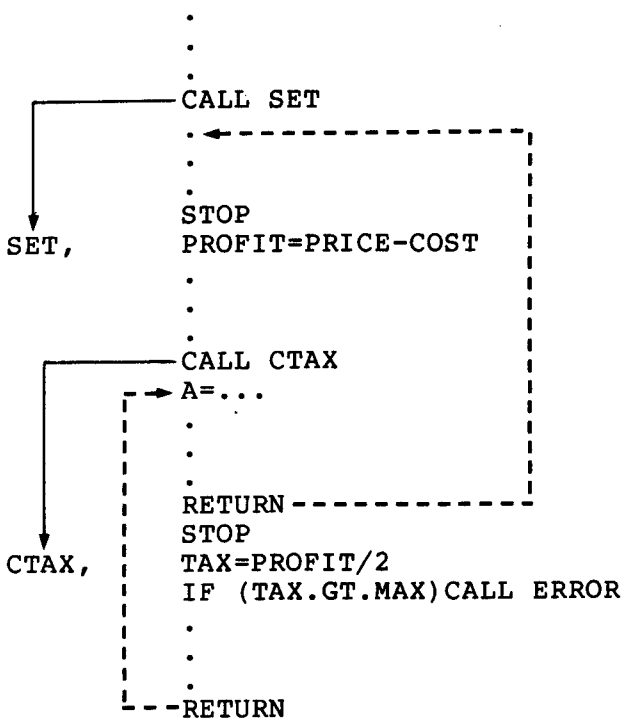
```
CALL label
```

where:

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

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

Example:



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

1.2.3 CHAIN - Control Statement

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

The form of the CHAIN statement is:

```
CHAIN n
```

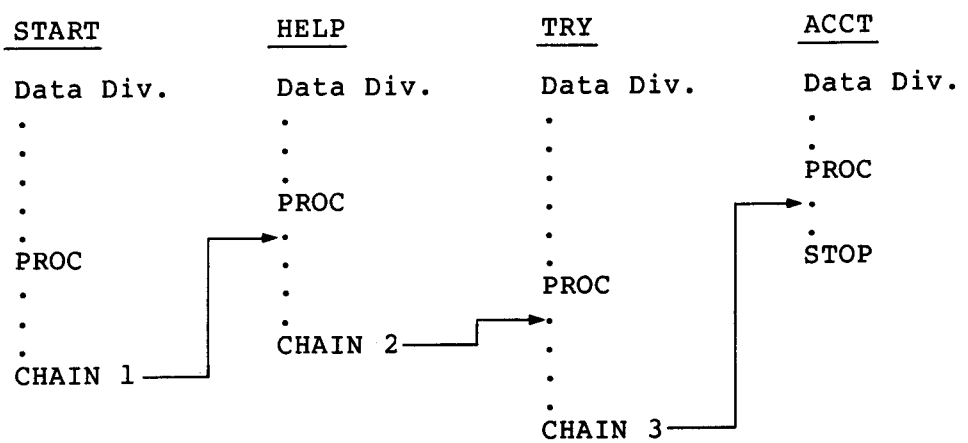
where:

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

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

Example:

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



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

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

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

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

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

1.2.4 Data Manipulation Statements

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

```
destination = source
```

where:

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

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

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

Variables:

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

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

where:

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

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

name (position m, position n)

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

Example:

```
Data Division defines A as 4D4

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

Literals:

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

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

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

Example:

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

Example:

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

Expressions:

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

1.2.4.1 Data Conversion

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

```
destination = source
```

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

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

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

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

1.2.4.2 Arithmetic Expressions

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

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

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

Order of priority:

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

These operators are used with numeric values.

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

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

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

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

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

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

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

Rounding:

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

```
destination = A#B
```

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

```
TEMP=MONEY#2

     If MONEY was 123456, TEMP becomes 1235.

     If MONEY was 123446, TEMP becomes 1234.

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

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

Division:

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

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

Multiplication, Addition, Subtraction:

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

1.2.4.3 Clearing Fields and Records

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

```
destination =
```

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

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

Examples:

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

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

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

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

A1=        ;Clears the first element in array A1.

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

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

Example:

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

1.2.4.4 Moving Alphanumeric Data

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

```
destination = source
```

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

Example:

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

1.2.4.5 Moving Numeric Data

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

```
destination = source
```

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

Example:

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

1.2.4.6 Moving Records

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

```
destination = source
```

where:

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

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

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

Example:

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

1.2.4.7 Data Formatting

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

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

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

Example:

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

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

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

Examples:

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

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

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

Table 1-3
Special Characters

Character	Explanation
X	Used in a format to arrange a numeric field for printout. Each X represents a digit and leading zeros are automatically suppressed.
Z	Used to suppress leading zeros when formatting output.
*	Used in a format string to replace leading zeros and eliminate trailing spaces on printout. If the * is anyplace except the first character in the string, digits may also be replaced.
-	Inserts an arithmetic sign in a number to be printed. The sign can be placed before or after the number. If the number is positive, a space is substituted for the minus sign. If the minus sign is placed in a position following the first significant digit but previous to the last position of a format string, it is printed like any other insertion character.
.	Inserts a decimal point in a format string and forces zeros to the right of the decimal point to be significant.
,	Used to insert a comma in a format string if there are significant digits to the left.
All other COS-310 characters are treated as unconditional insertion characters.	

1.2.5 DISPLAY - Input/Output Statement

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

The form of the DISPLAY statement is:

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

where:

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

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

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

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

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

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

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

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

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

Examples:

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

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

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

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

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

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

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

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

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

END

1.2.6 END - Compiler Statement

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

```
END[/x]
```

where:

/x is one of the following option switches.

/N suppresses the printing of a compiler source program storage map listing.

/T displays a compiler source program storage map listing on the screen.

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

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

1.2.7 FINI - Input/Output Statement

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

The FINI statement has the form:

```
FINI  (channel)
```

where:

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

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

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

Examples:

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

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

1.2.8 FORMS - Input/Output Statement

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

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

where:

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

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

Negative numbers cause unpredictable results.

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

Example:

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

Example:

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

1.2.9 GO TO - Control Statement

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

1.2.9.1 Unconditional GO TO

The form of the unconditional GO TO is:

```
GO TO label
```

where:

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

Example:

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

1.2.9.2 Computed GO TO

The computed GO TO has the form:

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

where:

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

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

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

Example:

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

1.2.10 IF - Control Statement

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

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

where:

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

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

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

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

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

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

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

Examples:

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

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

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

1.2.11 INCR

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

```
INCR variable
```

where:

variable is a numeric variable to be incremented by 1.

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

Example:

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

1.2.12 INIT - Input/Output Statement

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

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

where:

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

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

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

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

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

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

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

(Continued on next page)

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

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

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

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

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

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

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

Example:

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

1.2.13 ON ERROR - Control Statement

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

```
ON ERROR label
```

where:

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

Example:

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

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

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

1.2.14 PROC - Compiler Statement

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

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

where:

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

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

/x — is one of the following option switches:

/N suppresses compiler listing of source program.

/L lists source program and errors on the printer.

/T displays source program and errors on the screen.

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

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

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

Examples:

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

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

1.2.15 READ - Input/Output Statement

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

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

where:

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

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

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

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

Examples:

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

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

1.2.16 RECORD - Data Definition Statement

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

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

where:

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

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

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

Accompanying field information is of the form:

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

where:

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

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

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

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

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

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

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

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

Example:

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

1.2.17 RETURN - Control Statement

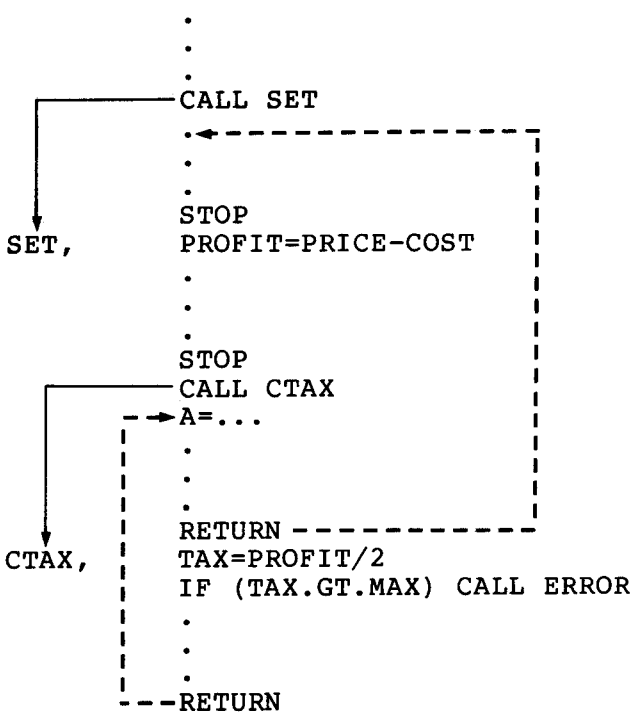
The RETURN statement is placed at the logical end of an internal subroutine. It has the form:

```
RETURN
```

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

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

Example:



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

1.2.18 START - Compiler Statement

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

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

where:

/x is one of the following option switches:

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

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

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

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

Example:

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

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

1.2.19 STOP - Control Statement

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

```
STOP
```

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

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

1.2.20 TRACE/NO TRACE - Debugging Statement

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

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

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

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

```
AT LINE xxxx
```

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

Example:

```
AT LINE 0200
000006
```

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

1.2.21 TRAP - Control Statement

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

```
TRAP label
```

where:

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

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

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

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

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

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

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

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

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

Example:

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

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

1.2.22 WRITE - Input/Output Statement

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

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

where:

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

record — is the record from which data is output.

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

Example:

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

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

1. The channel involved must refer to a logical unit on a mass storage device.
2. The file to be accessed with READ or WRITE operations must contain records of uniform size.
3. Only one volume of a multivolume file (the one currently mounted) can be accessed by a READ/WRITE statement.
4. The record which is specified from the Data Division must be the same size as the records of the file being accessed.
5. Attempting to READ or WRITE over the end-of-file mark results in an error message and program termination.
6. Reading or Writing a record after end-of-file usually results in an error message. Certain unpredictable conditions will not crash the COS-310 system but will cause garbled data (on a READ) or the loss of the output record (on a WRITE).
7. Unless a FINI statement is used before terminating a DIBOL program which has Update files, the data from the last few WRITE statements will not be output properly.

XMIT

1.2.23 XMIT - Input/Output Statement

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

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

where:

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

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

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

Examples:

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

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

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

Examples:

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

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

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

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

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

CHAPTER 2

THE MONITOR

2.1 MASTER CONTROL PROGRAM

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

- Terminal
- Mass Storage
- Printer

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

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

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

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

Table 2-1
Monitor Keyboard Commands

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

2.1.1 MOUNT Messages

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

```
MOUNT filnam #nn FOR INPUT:

MOUNT filnam #nn FOR OUTPUT:
```

where:

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

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

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

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

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

Respond to this message with a different logical unit number.

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

```
REPLACE filnam #nn ?
```

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

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

2.1.2 Operating Procedures

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

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

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

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

2.2 MONITOR COMMANDS

The following commands apply to Monitor functions.

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

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

2.2.1 BATCH

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

The form of the BATCH command is:

```
BA cmndfl
```

where:

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

Following is an example of a command file:

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

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

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

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

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

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

2.2.2 DATE

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

The form of the DATE command is:

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

where:

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

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

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

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

```
DATE?
```

If the date is entered incorrectly, the Monitor displays:

```
BAD DATE
```

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

Examples:

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

2.2.3 DELETE

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

The form of the DELETE command is:

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

where:

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

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

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

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

Examples:

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

2.2.4 DIRECTORY

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

The form of the DIRECTORY command is:

```
DI { [ dev][/T]      }
   { /logical unit # }
```

where:

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

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

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

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

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

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

Examples:

The command:

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

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

```
DIRECTORY       15-FEB-72

NAME    TYPE LN     DATE

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

The command:

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

outputs a file label similar to the one below.

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

2.2.5 PLEASE

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

The form of the PLEASE command is:

```
PLEASE text
```

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

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

Example:

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

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

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

2.2.6 RUN

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

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

The RUN command has the form:

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

where:

pronam is the name of the program to be run.

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

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

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

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

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

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

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

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

```
FILE NOT FOUND
```

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

Examples:

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

.RU JOB1            ;Runs program called JOB1.

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

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

2.2.7 SAVE

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

The form of the SAVE command is:

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

where:

pronam	is the name to be assigned to the binary program being stored.
dev	is the three-character designation for any mass storage device which has a directory. If no device is given, the system device is assumed.
/Y	is used to bypass the REPLACE? message when a duplicate name is encountered. Normally used in a batch mode to bypass operator response.

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

```
REPLACE?
```

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

2.3 EDITOR COMMANDS

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

Example:

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

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

The following commands are functions of the editor.

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

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

2.3.1 ERASE

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

The form of the ERASE command is:

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

where:

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

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

Examples:

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

.ER 5     ;Clears line 5.

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

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

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

FETCH

2.3.2 FETCH

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

The form of the FETCH command is:

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

where:

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

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

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

```
FILE NOT FOUND
```

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

Examples:

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

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

2.3.3 LIST

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

The form of the LIST command is:

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

where:

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

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

Examples:

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

.LI 5      ;Display line 5 on the screen.

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

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

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

2.3.4 Line Number

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

The form of the Line Number command is:

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

where:

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

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

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

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

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

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

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

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

Examples:

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

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

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

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

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

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

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

WRITE the edit buffer as file B.

ERASE the last half of the edit buffer.

WRITE the edit buffer as file A.

FETCH file B.

ERASE the first half of the edit buffer.

WRITE the edit buffer as file B.

2.3.5 Number Commands

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

```
nnnn [text]
```

where:

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

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

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

Examples:

Text before editing:

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

Editing commands:

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

Text after editing:

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

2.3.6 RESEQUENCE

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

The form of the RESEQUENCE command is:

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

where:

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

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

Examples:

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

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

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

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

2.3.7 WRITE

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

The form of the WRITE command is:

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

where:

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

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

```
REPLACE?
```

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

2.4 MONITOR ERROR MESSAGES

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

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

2.5 RUN-TIME ERROR MESSAGES

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

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

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

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

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

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

CHAPTER 3

SYSTEM GENERATION PROGRAM (SYSGEN)

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

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

where:

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

3.1 SYSGEN/B OPERATING PROCEDURES

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

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

```
RUN SYSGEN/B
```

After this command, SYSGEN displays the following question:

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

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

```
ENTER NUMBER CORRESPONDING TO DESIRED CONFIGURATION

1     DK        RK05 CARTRIDGE DISK DRIVES

2     RX        RX01 FLOPPY DISK DRIVES

3     DY        RX02 FLOPPY DISK DRIVES

4     DK & RX   RK05 AND RX01 DISK DRIVES

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

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

```
PLEASE TYPE NUMBER OF PRINTER MODEL ON SYSTEM

1     LA8A      DECPRINTER I USING DKC8-AA INTERFACE

2     LA35      DECWRITER II

3     LA36RO    DECWRITER II

4     LQP       LETTER-QUALITY PRINTER

5     LP05      300 LPM PRINTER

6     LA8       DECPRINTER I

7     LA120     DECWRITER III

8     NONE      NO PRINTER
```

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

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

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

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

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

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

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

    DO YOU WANT TO TRANSFER YOUR FILES?

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

SYSGEN then asks:

    IS EVERYTHING CORRECT?

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

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

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

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

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

3.2 SYSGEN/C OPERATING PROCEDURES

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

    RUN SYSGEN/C

SYSGEN displays the following statement:

```
ENTER NUMBER CORRESPONDING TO DESIRED CONFIGURATION

1    DK         RK05 CARTRIDGE DISK DRIVES

2    RX         RX01 FLOPPY DISK DRIVES

3    DY         RX02 FLOPPY DISK DRIVES

4    DK & RX    RK05 AND RX01 DISK DRIVES

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

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

```
PLEASE TYPE NUMBER OF PRINTER MODEL ON SYSTEM

1    LA8A       DECPRINTER I USING DKC8-AA INTERFACE

2    LA35       DECWRITER II

3    LA36RO     DECWRITER II

4    LQP        LETTER-QUALITY PRINTER

5    LP05       300 LPM PRINTER

6    LA8        DECPRINTER I

7    LA120      DECWRITER III

8    NONE       NO PRINTER
```

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

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

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

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

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

After you enter YES or NO, SYSGEN asks:

```
IS EVERYTHING CORRECT?
```

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

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

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

3.3 SYSGEN ERROR MESSAGES

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

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

CHAPTER 4

DATA FILE UTILITY PROGRAM (DFU)

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

4.1 DFU OPERATING PROCEDURES

To execute DFU, type:

```
        ⎧ ,filnam ⎫
RUN DFU ⎨         ⎬
        ⎩ /xx     ⎭
```

where:

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

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

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

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

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

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

4.1.1 DFU,filnam Operating Procedures

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

```
RUN DFU,filnam
```

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

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

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

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

4.1.2 DFU/B Operating Procedures

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

```
RUN DFU/B
```

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

4.1.3 DFU/K Operating Procedures

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

```
RUN DFU/K
```

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

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

4.1.4 DFU/D Operating Procedures

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

```
RUN DFU/D
```

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

```
UNIT DEV.  SEGS.

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

where:

UNIT is the logical unit number.

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

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

4.1.5 DFU/DL Operating Procedures

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

```
RUN DFU/DL
```

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

```
UNIT DEV.  SEGS.

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

where:

UNIT is the logical unit number.

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

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

4.1.6 DFU/E Operating Procedures

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

```
RUN DFU/E
```

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

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

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

where:

UNIT — is the logical unit number.

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

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

NAME — is the name assigned to the data file.

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

DATE — is the creation date of the data file.

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

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

4.1.7 DFU/EL Operating Procedures

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

```
RUN DFU/EL
```

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

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

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

where:

UNIT — is the logical unit number.

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

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

NAME — is the name assigned to the data file.

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

DATE — is the creation date of the data file.

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

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

4.2 LOGICAL UNIT ASSIGNMENTS ON THE COS-310 SYSTEM

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

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

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

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

4.2.1 Determining Logical Unit Size

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

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

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

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

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

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

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

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

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

4.2.2 How Logical Units are Assigned by DFU

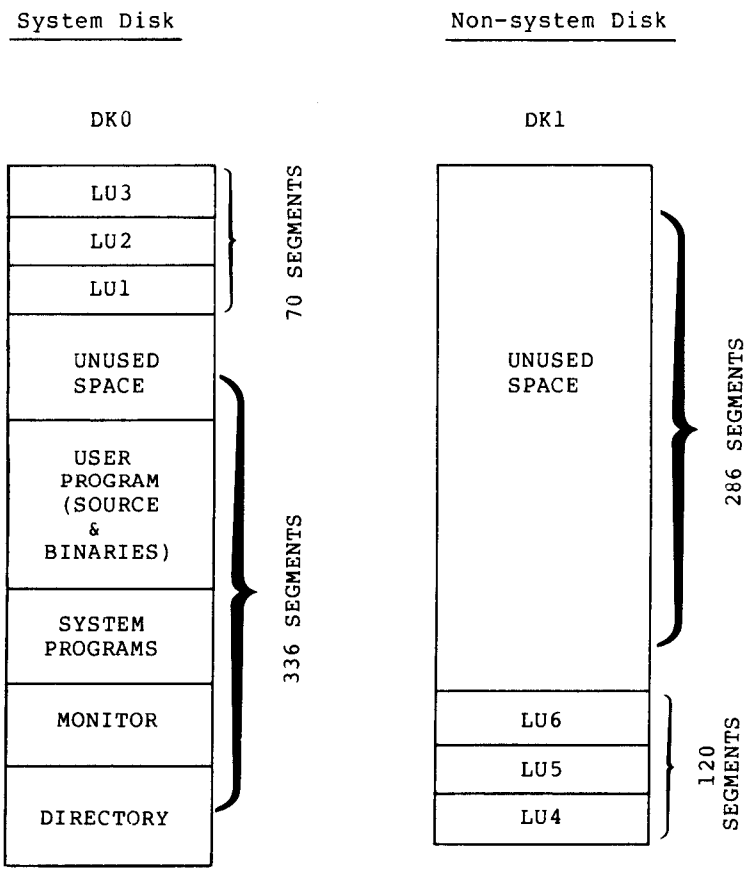
On the system device, logical units are assigned in a pushdown order beginning, at the end of the device. For example, a disk with three logical units would be arranged, starting from the beginning, as: Directory; Monitor; System programs; User programs; Unused space to which new user programs may be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

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

Example:

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

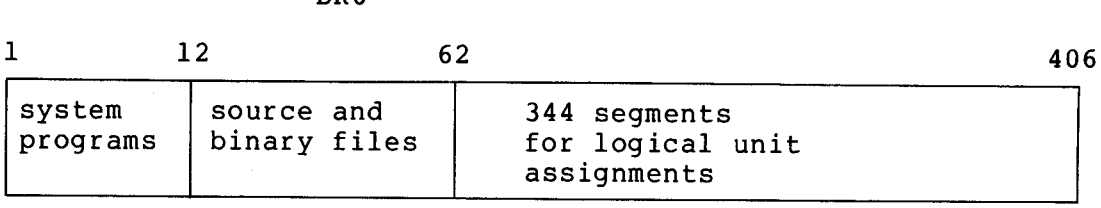
The preceding logical unit assignments would cause an RK05 system disk (DK0) and nonsystem disk (DK1) to be organized as follows:



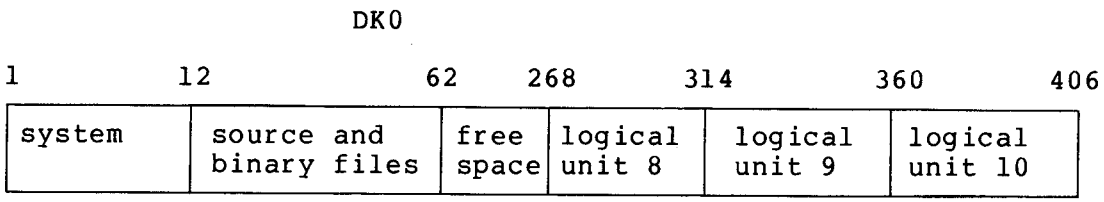
It is advisable to create logical units only slightly larger than the actual data file size since a short file in a large logical unit wastes storage.

4.3 DISK USERS

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



A sample logical unit assignment might be:



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

4.4 DFU ERROR MESSAGES

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