APPENDIX A

COS-310 CHARACTER SET

In both source and data files, characters (alphanumeric and numeric) are stored two characters per word in six-bit binary. Negative numbers are stored with the high-order bit of the low-order digit set to 1. For example, the number 1234- is stored as two words in the following form:

22 1	23 2	WORD 1
24 3	65 4	WORD 2 (with high-order bit on)

This number is recognized as 123T. This means that any program in which the numeric-to-alphanumeric conversion is not made might produce negative numbers with letters. Refer to Table A-1 for a list of characters representing negative numbers.

Table A-1
Characters Representing Negative Numbers

Negative Number	Equivalent Character	Decimal Code	Octal Code
-0	P	49	61
-1	Q	50	62
-2	R	51	63
-3	S	52	64
-4	T	53	65
-5	U	54	66
-6	V	55	67
-7	W	56	70
-8	X	57	71
-9	Y	58	72

Table A-2
COS-310 Character Set

Decimal Code	Octal Code	Character	Decimal Code	Octal Code	Character
00	00	Null	32	40	?
01	01	Space	33	41	@
02	02	!	34	42	A
03	03	"	35	43	B
04	04	#	36	44	C
05	05	$	37	45	D
06	06	%	38	46	E
07	07	&	39	47	F
08	10	'	40	50	G
09	11	(	41	51	H
10	12	)	42	52	I
11	13	*	43	53	J
12	14	+	44	54	K
13	15	,	45	55	L
14	16	-	46	56	M
15	17	.	47	57	N
16	20	/	48	60	O
17	21	0	49	61	P
18	22	1	50	62	Q
19	23	2	51	63	R
20	24	3	52	64	S
21	25	4	53	65	T
22	26	5	54	66	U
23	27	6	55	67	V
24	30	7	56	70	W
25	31	8	57	71	X
26	32	9	58	72	Y
27	33	:	59	73	Z
28	34	;	60	74	[
29	35	<	61	75	Tab
30	36	=	62	76	]
31	37	>	63	77	↑

APPENDIX B

COS-310 FILES

There are four types of files in the COS-310 system: source, binary, data, and system. Source, binary, and data files have similar structure. System files use standard OS/8 SAVE format.

B.1 COS-310 SOURCE FILES

Each line in a source command file or DIBOL source file must be input with a line number. This makes all source files look the same and makes them compatible with COS-310. Each input line has the following format:

word count (n)	line number	n-1 words, two COS-310 characters per word

The first word contains the word count for that line. It is computed with the following expression.

n = ((number of characters on line +1)/2)+1

The second word is the statement line number, 0000-7777 octal (0000-4095 decimal).

The third and successive words contain the text of the line packed two COS-310 characters per word. The total characters of data per line does not include the two-character (1 word) word count number.

B.2 COS-310 DATA FILES

Every block in a data file is completely devoted to the storage of data. Each logical unit holds only one data file. Labels on data files are associated with logical units by the Monitor in conjunction with DIBOL or system programs.

The format of a line in a data file is similar to the format for a line in a source file except there is no line number on a data file.

A line of text in a data file has the following format:

word count (n)	n words, two characters per word

The first word contains the word count for that line. It is computed with the following expression:

n = (number of characters in record +1)/2

The second and successive words contain the text of the line, two COS-310 characters per word.

B.3 COS-310 BINARY FILES

Although the contents of a binary file are interpreted differently than the contents of a data file, externally the two files are structured exactly alike. That is, the binary code for each line of a DIBOL source program is stored as a word count followed by the interpretive code to be used by the run-time system.

B.4 COS-310 SYSTEM FILES

All system files are stored in OS/8 SAVE format. The first block of the file is a memory control block indicating where in memory the rest of the blocks of the file are to be loaded. Each successive block is a 256-word memory image. See the OS/8 Software Support Manual for details.

B.5 SYSTEM DEVICE FORMAT

COS-310 puts a label on all devices. This label occupies the first 256 words of each device; four words are the actual label, one word is the date, and the other words may be a bootstrap.

Figure B-1 illustrates the layout of the Monitor portion of the system device. As noted in the figure, COMP should be the first file in the file area. The location of COMP is particularly important when the binary scratch area is to be expanded.

BLOCK No. (Octal)	
0	Bootstrap
1	Directory
10	Monitor
14	Editor Overlay
20	Editor
34	Run-Time System Loader
40	Edit Buffer
60	Run-Time System
70	Compiler Overlays
100	Binary Scratch Area
140	Files
END OF MEDIA	

Figure B-1 Monitor Organization

A label is automatically put on a system device. The directory of a system device is organized as follows:

Word	Contents
0	The negative number of directory entries in this block.
1	The starting block number for file storage.
2	The link word to the next directory block or empty if end. There are seven directory blocks on all multifile devices.
3	Empty (unused).
4	The negative number of auxiliary words per entry (always equals -1).
5	The first two characters of name.
6	The next two characters of name.
7	The last two characters of name.
8	A two-character extension.
9	The date.
10	Length of the file (negative).
11-255	Repeat of 5-10 for each file.

Space for other kinds of files is allocated on the disk beginning at the first free block following the COS-310 system files. On an RK05 disk, the system directory knows that the available space for file storage only extends to block 4095.

Access to Data Files

Data Files are referenced by their logical unit numbers as assigned by DFU. DFU actually sets up an internal table containing the following information for each logical unit:

- Handler address
- Drive number
- Starting segment
- Length in segments

The handler address is a pointer to the specific device handler to use for a particular logical unit. The drive number indicates which disk drive to reference. The starting segment is indicated by a 12-bit number which points to the physical device space allocated for the logical unit. The length is the number of segments reserved for this logical unit.

Example:

If logical unit 14 is assigned to a 32-block area on DK1, the fourteenth entry in the table might contain the following information:

DK handler address	drive 1	starting segment 212 (octal)	length -40 (octal)

Any references to logical unit 14 would refer to segments 212-251 (octal) of DK1. The first block in segment 212 would have a label for that logical unit.

APPENDIX C

ERROR MESSAGE INDEX

This index will refer you to the chapters where corrective and background information is located. Locate the error message you have encountered and go to the chapter referenced. If the message is listed more than once, check the message listed with the program that you are running.

Message	Program	Refer to
ACCEPT SECTION NOT FOUND	MENU	Chapter 18
ALPHA LITERAL REQUIRED	PRINT	Chapter 16
ALREADY DEFINED	PRINT	Chapter 16
BAD ALPHA VALUE	COMP	Chapter 5
BAD CHAIN	Run-Time	Chapter 2
BAD CHECKSUM	PATCH	Chapter 11
BAD COMPILATION	Monitor	Chapter 2
BAD DATE	Monitor	Chapter 2
BAD DIGIT	Run-Time	Chapter 2
BAD DIGIT IN DATA	DAFT	Chapter 15
BAD DIGIT IN NUMERIC INITIAL VALUE	SORT	Chapter 9
BAD DIRECTORY	FILEX	Chapter 10
BAD DIRECTORY	Monitor	Chapter 2
BAD DIRECTORY	PATCH	Chapter 11
BAD DIRECTORY	PIP	Chapter 8

Message	Program	Refer to
BAD LABEL	Monitor	Chapter 2
BAD NUMBER	PATCH	Chapter 11
BAD NUMERIC VALUE	COMP	Chapter 5
BAD PROC #	COMP	Chapter 5
BAD PROGRAM	Run-Time	Chapter 2
BAD RECORD SIZE	SORT	Chapter 9
BAD RELATIONAL	COMP	Chapter 5
BAD SWITCH	SYSGEN	Chapter 3
BAD SWITCH	DFU	Chapter 4
BAD WORK UNIT COUNT	SORT	Chapter 9
BLOCK TOO BIG	PATCH	Chapter 11
CANT BACKSPACE PAST BEGIN OF FILE	DAFT	Chapter 15
CANT BACKSPACE WITH SEQUENTIAL INPUT	DAFT	Chapter 15
CCP ERROR	COMP	Chapter 5
COMMA MISSING	COMP	Chapter 5
COMMAND SECTION NOT FOUND	MENU	Chapter 18
COMPARISON ERROR	PIP	Chapter 8
DATA INITIALIZATION MISSING	COMP	Chapter 5
DEVICE ERROR	FILEX	Chapter 10
dev MUST BE INCLUDED IN CONFIGURATION	SYSGEN	Chapter 3
DIBOL FILE NUMBER IN USE	Run-Time	Chapter 2
DIBOL FILE NUMBER NOT INITED	Run-Time	Chapter 2
DISPLAY SECTION NOT FOUND	MENU	Chapter 18
EDIT BUFFER FULL	Monitor	Chapter 2

Message	Program	Refer to
END OF FILE	Run-Time	Chapter 2
END OF INPUT FILE AT RECORD nnnn	DAFT	Chapter 15
ERR IN CMD	DDT	Chapter 6
ERROR	FLOW	Chapter 17
ERROR IN COMMAND	Monitor	Chapter 2
ERROR ON dev, RETRY?	Monitor	Chapter 2
EXCESSIVE GRID SIZE	DAFT	Chapter 15
EXPECTED LABEL IS MISSING	COMP	Chapter 5
EXPRESSION NOT ALLOWED	COMP	Chapter 5
EXTRA CHARS	DAFT	Chapter 15
EXTRA CHARS AT STMNT END	COMP	Chapter 5
EXTRA CHARS AT STMNT END	SORT	Chapter 9
FIELD NUMBER MISSING OR 0	SORT	Chapter 9
FIELD TOO LARGE OR 0	COMP	Chapter 5
FILE ALREADY EXISTS	FILEX	Chapter 10
FILE NOT FOUND	Monitor	Chapter 2
FILE NOT FOUND	PATCH	Chapter 11
FULL	FILEX	Chapter 10
FULL	SYSGEN	Chapter 3
HEADER IS TOO LONG	PRINT	Chapter 16
ILLEGAL CURSOR POSITION	MENU	Chapter 18
ILLEGAL DEVICE	DFU	Chapter 4
ILLEGAL DEVICE	FILEX	Chapter 10
ILLEGAL DEVICE	Run-Time	Chapter 2
ILLEGAL DEVICE SWITCH	PIP	Chapter 8

Message	Program	Refer to
ILLEGAL OPERATOR	COMP	Chapter 5
ILLEGAL PROGRAM	Monitor	Chapter 2
ILLEGAL RECORD - CLOSING FILE	DAFT	Chapter 15
ILLEGAL RECORD #	Run-Time	Chapter 2
ILLEGAL SORT KEY	SORT	Chapter 9
ILLEGAL STATEMENT	MENU	Chapter 18
ILLEGAL STMNT	COMP	Chapter 5
ILLEGAL SUBSTRING	Run-Time	Chapter 2
ILLEGAL UNIT	Monitor	Chapter 2
ILLEGAL UNIT	SORT	Chapter 9
IMPROPER DEFINITION	PRINT	Chapter 16
IMPROPER LITERAL	PRINT	Chapter 16
IMPROPER USE OF DECIMAL PLACES	PRINT	Chapter 16
IN USE	Monitor	Chapter 2
INSUFFICIENT SPACE ON DEVICE	DFU	Chapter 4
INSUFFICIENT SPACE ON DEVICE	FILEX	Chapter 10
INITIAL ALPHA VALUE DOESN'T BEGIN WITH QUOTE	SORT	Chapter 9
INITIAL VALUE TOO BIG	SORT	Chapter 9
INITIAL VALUE TOO SMALL	SORT	Chapter 9
INITIAL VALUE WRONG SIZE	COMP	Chapter 5
INTEGER FROM 1-15 REQUIRED	PRINT	Chapter 16
INTEGER FROM 1-132 REQUIRED	PRINT	Chapter 16
INTEGER REQUIRED	PRINT	Chapter 16
INVALID OPERATION	PATCH	Chapter 13
KEY ENTIRELY PAST END OF RECORD	DAFT	Chapter 15

Message	Program	Refer to
KEY EXTENDS PAST END OF RECORD	DAFT	Chapter 15
KEY TOO BIG	DAFT	Chapter 15
LABEL NOT ALLOWED	COMP	Chapter 5
LINE TOO LONG	Monitor	Chapter 2
LINE # TOO LARGE	Monitor	Chapter 2
LINE TOO LONG	Run-Time	Chapter 2
LITERAL TOO LONG	PRINT	Chapter 16
LOCATION TOO BIG	PATCH	Chapter 11
MISSING CLOSE PAREN	COMP	Chapter 5
MISSING CLOSE QUOTE ON ALPHA INITIAL VALUE	SORT	Chapter 9
MISSING INITIAL VALUE	SORT	Chapter 9
MISSING OPEN PAREN	COMP	Chapter 5
MISSING OPERAND	COMP	Chapter 5
MISSING OR BAD MODE	COMP	Chapter 5
MISSING QUOTE	COMP	Chapter 5
MISSING RELATIONAL	COMP	Chapter 5
MOUNT filnam #nn FOR INPUT:	Monitor	Chapter 2
MOUNT filnam #01 FOR INPUT:	Monitor	Chapter 2
MOUNT filnam #nn FOR OUTPUT:	Monitor	Chapter 2
MUST BE IDENT	PRINT	Chapter 16
MUST BE NUMERIC ITEM	PRINT	Chapter 16
MUST BE S	PRINT	Chapter 16
NAME PREVIOUSLY DEFINED	COMP	Chapter 5
NEED FILE NAME	PRINT	Chapter 16
nnn IS BEING IGNORED	CREF	Chapter 7

Message	Program	Refer to
NO	BOOT	Chapter 12
NO BUFFERS LEFT	Run-Time	Chapter 2
NO CHANGE IN BLOCK	PATCH	Chapter 11
NO COMMA AFTER FIELD NAME	SORT	Chapter 9
NO DATA	DAFT	Chapter 15
NO END	FILEX	Chapter 10
NO ENDING QUOTE	PRINT	Chapter 16
NO FILE	Run-Time	Chapter 2
?NO FILE TO SAVE	Monitor	Chapter 2
NO INIT	Monitor	Chapter 2
NO INPUT	FLOW	Chapter 17
NO INPUT	SORT	Chapter 9
NO INPUT DIRECTIVE	PRINT	Chapter 16
NO INPUT FILE	DAFT	Chapter 15
NO LABEL NAME	DAFT	Chapter 15
NO LP BUFFER	Monitor	Chapter 2
NO OUTPUT FILE	DAFT	Chapter 15
NO PRINT ITEMS	PRINT	Chapter 16
NO ROOM	FILEX	Chapter 10
NO ROOM	PIP	Chapter 8
NOT A OR D	COMP	Chapter 5
NOT A OR D	SORT	Chapter 9
NOT DEFINED	PRINT	Chapter 16
NOT ENOUGH RIGHT PARENTHESES	PRINT	Chapter 16
NOT ENOUGH ROOM FOR SYSTEM AND FILES	DFU	Chapter 4

Message	**Program**	**Refer to**
NOT ENOUGH ROOM FOR SYSTEM AND FILES	FILEX	Chapter 10
NOT FOUND	FILEX	Chapter 10
NOT LABEL	COMP	Chapter 5
NOTHING AFTER FIELD NAME	SORT	Chapter 9
NUMBER REPEATED OR OUT OF ORDER	SORT	Chapter 9
# TOO LARGE	DFU	Chapter 4
NUMBER TOO LONG	Run-Time	Chapter 2
OUTPUT ERROR	SORT	Chapter 9
OUTPUT FILE ALREADY OPEN	DAFT	Chapter 15
OUTPUT FILE STILL OPEN	DAFT	Chapter 15
PICTURE TOO LONG	PRINT	Chapter 16
PROGRAM TOO BIG	COMP	Chapter 5
PROGRAM TOO BIG	Run-Time	Chapter 2
PUSHDOWN OVERFLOW	DAFT	Chapter 15
PUSHDOWN OVERFLOW	Run-Time	Chapter 2
RECORD TOO BIG	COMP	Chapter 5
REPLACE?	Monitor	Chapter 2
REPLACE filnam #nn ?	Monitor	Chapter 2
RETURN WITHOUT CALL	Run-Time	Chapter 2
STMNT TOO COMPLEX	COMP	Chapter 5
SUBSCRIPT ERROR	COMP	Chapter 5
SUBSCRIPT NOT NUMERIC	COMP	Chapter 5
SUBSCRIPT TOO BIG	Run-Time	Chapter 2
SYNTAX ERROR	DFU	Chapter 4
SYNTAX ERROR	PRINT	Chapter 16

Message	**Program**	**Refer to**
TOO MANY COLUMNS IN REPORT	PRINT	Chapter 16
TOO MANY COMMANDS	MENU	Chapter 18
TOO MANY COMMANDS FOR 1 CODE	MENU	Chapter 18
TOO MANY COMPUTE STATEMENTS	PRINT	Chapter 16
TOO MANY DATA ITEMS	PRINT	Chapter 16
TOO MANY FILES	SORT	Chapter 9
TOO MANY ITEMS	COMP	Chapter 5
TOO MANY LEFT PARENTHESES	PRINT	Chapter 16
TOO MANY LIST ITEMS	PRINT	Chapter 16
TOO MANY RIGHT PARENTHESES	PRINT	Chapter 16
TOO MANY SYMBOLS!	COMP	Chapter 5
TOO MUCH DATA	COMP	Chapter 5
UNDEFINED NAME	COMP	Chapter 5
UNKNOWN DIRECTIVE	PRINT	Chapter 16
UNRECOGNIZABLE LINE	SORT	Chapter 9
WRONG DATA TYPE	COMP	Chapter 5
ZERO DIVISOR	Run-Time	Chapter 2
0 NOT ALLOWED	DAFT	Chapter 15

APPENDIX D

ADVANCED PROGRAMMING TECHNIQUES

D.1 ACCEPT AND DISPLAY

D.1.1 Background Information

XMIT statements were originally used when the terminal was a Teletype[1]. The VT52 display terminal uses newer concepts -- programmable cursor control and hardware display clear. ACCEPT and DISPLAY statments were added to the DIBOL language to use these features. The terminal can now be used in two ways:

1. As a Teletype by using XMIT statements.

2. As a powerful data entry tool by using ACCEPT and DISPLAY statements.

(Refer to the ACCEPT and DISPLAY statements in Chapter 1 before proceeding further.)

D.1.2 Interaction of ACCEPT and DISPLAY

ACCEPT and DISPLAY statements are used extensively in data entry programs. These data entry programs typically work one of two ways. The first asks (DISPLAY) questions and interprets (ACCEPT) answers. This method of operation closely simulates a Teletype. The second method displays a format or heading on the screen and moves the cursor either to the right or to a position below the question to be answered.

With the second method, the format is never cleared but data is entered and cleared continuously from the screen. This method is used in repetitive data entry and updating. Quite often the four keys up arrow, down arrow, left arrow, and right arrow have special meanings. For example, assume ten headings are displayed on the screen, indicating ten fields are to be entered or updated. The up arrow might be used to re-enter information in the first field, no matter which field is currently being entered; the down arrow might mean no more information for any of the fields; the left arrow might restart entering data into the current field; the right arrow might mean go on to the next field without changing the current field.

D.1.3 Example Using ACCEPT and DISPLAY

To enter a six-digit customer number and a 15-character customer name, the following program might be used:

```
                RECORD
TCHAR,D2
ALPHA,A15
CNO,D6
CNAME,A15
                PROC 1
                DISPLAY(1,1,1)  ;Clear screen and position cursor.
                DISPLAY(0,0,'CUSTOMER NO.   CUSTOMER NAME')
LOOP,           DISPLAY(2,1,2)  ;Clear line 2 and position cursor.
                ALPHA=          ;Clear this field.
                ACCEPT(TCHAR,ALPHA)
                ON ERROR LOOP   ;Re-enter if not numeric.
                CNO=ALPHA
                ALPHA=          ;Clear this field again.
                DISPLAY(2,16,0) ;Position cursor.
                ACCEPT(TCHAR,ALPHA)
                CNAME=ALPHA
                .
                .               ;Save data.
                .
                GO TO LOOP
```

D.1.4 Generalized ACCEPT Subroutines

D.1.4.1 Hardware Display Clear Feature - Although the previous example works properly, it lacks features which would be useful:

1. Type RUBOUT to clear the previously entered character from both the program and the display.

2. Type CTRL/U (a DIGITAL convention) to clear the entire current line from both the program and the display.

Since data acceptance is getting more sophisticated, it can best be performed by calls to a subroutine. The following two subroutines and test programs will accept data from the keyboard and use the RUBOUT key and the CTRL/U key as previously specified. The first program uses the clearing feature built into the hardware of the VT52. Unfortunately, this feature destroys data if it is on the same line and to the right of what is being accepted.

```
        START   ;Erases remainder of line for errors.
        RECORD
KBDBUF, A80     ;Storage for keyboard input.
        RECORD ,X
KBDIN,  80A1

        RECORD  ;Work area.
ROW,    D2      ;Cursor Y-coordinate on entry to subroutine VT52
                ;(needed for correction only).
COL,    D2      ;Cursor X-coordinate on entry to subroutine VT52
                ;(needed for correction only).
TCHAR,  D2      ;Terminating character in an accept statement.
CHAR,   A1      ;Input character from an accept statement.
VT52IN, D2      ;Number of characters accepted by subroutine VT52.
VTLIM,  D2      ;Number of characters to be accepted by
                ;subroutine VT52.
        PROC
BEGIN,  DISPLAY(1,1,1)  ;Clear screen.          ***********
        DISPLAY(1,40,'ERASED IN CORRECTION')    *         *
        DISPLAY(1,1,'NAME:')                    *         *
        ROW=1                                   * SAMPLE  *
        COL=6                                   *  TEST   *
        VTLIM=20        ;20 characters maximum. * PROGRAM *
        CALL VT52                               *         *
        IF (KBDBUF.EQ.'END') STOP               *         *
        GO TO BEGIN                             ***********


;       Calling sequence
;               ROW= Y coordinate
;               COL= X coordinate
;               VTLIM= Maximum number of characters to accept
;               CALL VT52

;Accept a maximum of VTLIM characters at location specified by
;ROW and COL. Return when either the maximum number of characters
;(VTLIM) has been entered, a termination character is entered,
;or a space is entered. Rubout deletes last character entered and
;CTRL/U eliminates the entire entry. RUBOUT and CTRL/U clear the
;remainder of the line faster than displaying spaces.

VT52,   VT52IN=
        KBDBUF=
VT522,  ACCEPT(TCHAR,CHAR)
        IF (TCHAR.EQ.0) GO TO VT523     ;Nonterminating character.
        IF (TCHAR.EQ.21) GO TO VT524    ;CTRL/U.
```

```
        IF (TCHAR.EQ.32) GO TO VT525       ;RUBOUT.
        RETURN                             ;Terminating character other
                                           ;than rubout or CTRL/U.
VT523,  IF(CHAR.EQ.' ') RETURN             ;Space is a terminating
                                           ;character.
                                           ;To eliminate this feature,
                                           ;remove this statement and put
                                           ;label on next statement.
        INCR VT52IN                        ;VT52IN=# of input characters.
        KBDIN(VT52IN)=CHAR
        IF (VT52IN.EQ.VTLIM) RETURN        ;The specified number of
                                           ;characters were input.
        GO TO VT522
VT524,  IF(VT52IN.EQ.0) GO TO VT52
        DISPLAY(ROW,COL,2)                 ;Clear characters entered
                                           ;to end-of-line.
        GO TO VT52
VT525,  IF(VT52IN.EQ.0) GO TO VT522
        KBDIN(VT52IN)=
        VT52IN=VT52IN-1
        DISPLAY(ROW,COL+VT52IN,2)          ;RUBOUT previous character
                                           ;to end-of-line.
        GO TO VT522
```

D.1.4.2 Clear Incorrect Data by Displaying Spaces - The following program clears incorrectly entered data by displaying spaces. This is slower than using the hardware display clear feature, but data on the same line and to the right is not cleared.

```
        START    ;Corrects only characters in error.
        RECORD
KBDBUF, A80      ;Storage for keyboard input.
        RECORD ,X
KBDIN,  80A1

        RECORD   ;Work area.
BLNK80, A80      ;80 blank characters.
ROW,    D2       ;Cursor Y coordinate on entry to subroutine VT52
                 ;(needed for correction only).
COL,    D2       ;Cursor X coordinate on entry to subroutine VT52
                 ;(needed for correction only).
TCHAR,  D2       ;Terminating character in an ACCEPT statement.
CHAR,   A1       ;Input character from an ACCEPT statement.
VT52IN, D2       ;Number of characters accepted by subroutine VT52.
VTLIM,  D2       ;Number of characters to be accepted by
                 ;subroutine VT52.
VT52XX, D2       ;Temporary storage for subroutine VT52.
        PROC
```

```
BEGIN,  DISPLAY(1,1,1)  ;Clear screen.                ***********
        DISPLAY(1,40,'NEVER ERASED')                  *         *
        DISPLAY(1,1,'NAME:')                          *         *
        ROW=1                                         * SAMPLE  *
        COL=6                                         *  TEST   *
        VTLIM=20        ;20 characters maximum.       * PROGRAM *
        CALL VT52                                     *         *
        IF (KBDBUF.EQ.'END') STOP                     *         *
        GO TO BEGIN                                   ***********

;       Calling sequence
;               ROW= Y-coordinate
;               COL= X-coordinate
;               VTLIM= Maximum number of characters to accept
;               CALL VT52
;Accept a maximum of VTLIM characters at location specified by
;ROW and COL. Return when either the maximum number of characters
;(VTLIM) has been entered, a termination character is entered,
;or a space is entered. RUBOUT deletes last character entered and
;CTRL/U eliminates the entire entry. RUBOUT and CTRL/U display
;space(s) to delete only the necessary characters (not the
;remainder of the line).

VT52,   VT52IN=
        KBDBUF=
VT522,  ACCEPT(TCHAR,CHAR)
        IF (TCHAR.EQ.0) GO TO VT523     ;Nonterminating character.
        IF (TCHAR.EQ.21) GO TO VT524    ;CTRL/U.
        IF (TCHAR.EQ.32) GO TO VT525    ;RUBOUT.
        RETURN                          ;Terminating character other
                                        ;than RUBOUT or CTRL/U.
VT523,  IF(CHAR.EQ.' ') RETURN          ;Space is a terminating
                                        ;character.
                                        ;To eliminate this feature,
                                        ;remove this statement.
        INCR VT52IN                     ;VT52IN=# of input characters.
        KBDIN(VT52IN)=CHAR
        IF (VT52IN.EQ.VTLIM) RETURN     ;The specified number of
                                        ;characters were input.
        GO TO VT522
VT524,  IF(VT52IN.EQ.0) GO TO VT52
        DISPLAY(ROW,COL,BLNK80(1,VT52IN))  ;Clear characters entered.
        DISPLAY(ROW,COL,0)              ;Reposition cursor.
        GO TO VT52
VT525,  IF(VT52IN.EQ.0) GO TO VT522
        KBDIN(VT52IN)=
        VT52IN=VT52IN-1
        VT52XX=VT52IN+COL
        DISPLAY(ROW,VT52XX,' ')         ;Rubout previous character.
        DISPLAY(ROW,VT52XX,0)           ;Reposition cursor.
        GO TO VT522
```

D.1.4.3 Other Desired Features - In addition to the features found in the previous program, the following features might also be desired:

1. Right justification of numeric fields.

2. Automatic cursor positioning.

These features are used in the following subroutine and test program:

```
        START   ;Subroutine VT52A and VT52N.
        RECORD
KBDBUF, A80     ;Storage for keyboard input.
        RECORD ,X
KBDIN,  80A1

        RECORD  ;Work area.
BLNK80, A80     ;80 blank characters.
ROW,    D2      ;Cursor Y coordinate.
COL,    D2      ;Cursor X coordinate.
TCHAR,  D2      ;Terminating character in an ACCEPT statement.
CHAR,   A1      ;Input character from an ACCEPT statement.
VT52IN, D2      ;Number of characters accepted.
VTLIM,  D2      ;Number of characters to be accepted.
VT52SW, D1      ;Cleared for alpha input, set to 1 for numeric input.
VT5215, D15     ;Contains numeric input for VT52N entry. Not changed
                ;or used in VT52A entry.
VT52XX, A16     ;Temporary storage for redisplay of numeric input.
        PROC 0
        DISPLAY(1,1,1)  ;Clear screen.                 ***********
BEGIN,  INCR ROW                                        *         *
        IF (ROW .GT. 24) STOP                           *         *
        DISPLAY (ROW,53,'NOT ERASED')                   *         *
        DISPLAY (ROW,1,'NAME:')                         *  SAMPLE *
        COL=7                                           *         *
        VTLIM=20        ;20 characters maximum.         *   TEST  *
        CALL VT52A                                      *         *
        IF (KBDBUF.EQ.'END') STOP                       * PROGRAM *
        DISPLAY(ROW,30,'NO:')                           *         *
        COL=34                                          *         *
        VTLIM=15                                        *         *
        CALL VT52N                                      *         *
        GO TO BEGIN                                     ***********


;       Calling sequence
;               ROW= Y coordinate
;               COL= X coordinate
;               VTLIM= Maximum number of characters to accept
;               CALL VT52A for alphanumeric input
;               CALL VT52N for numeric input
```

```
;Accept a maximum of VTLIM characters at location specified by ROW
;and COL. Return when VTLIM characters or a termination character
;is entered.  For numeric input, a space is a terminator.
;RUBOUT deletes last character entered and CTRL/U eliminates the
;entire entry. RUBOUT and CTRL/U display space(s) to delete only
;the necessary characters (not the remainder of the line).
;For numeric input, the entire entry is redisplayed right-justified
;with leading zeros suppressed. VT5215 contains the number
;on return to the calling program.

VT52A,  VT52SW=             ;Entry for alphanumeric input.
        GO TO VT52
VT52N,  VT52SW=1            ;Entry for numeric input.
VT52,   VT52IN=
        KBDBUF=
        DISPLAY(ROW,COL,0)       ;Position cursor.
VT522,  ACCEPT(TCHAR,CHAR)
        IF(TCHAR.EQ.0) GO TO VT523      ;Nonterminating character.
        IF(TCHAR.EQ.21) GO TO VT524     ;CTRL/U.
        IF (TCHAR.EQ.32) GO TO VT525    ;RUBOUT.
VT522X, IF (VT52IN.EQ.0) RETURN         ;No input except terminating
                                        ;character.
        IF (VT52SW.EQ.0) RETURN         ;Alphanumeric input.
VT522Y, VT5215=KBDBUF(1,VT52IN)         ;Numeric input (can't exceed
                                        ;15 digits).
        VT52XX(1,VTLIM+1)=VT5215,'XXXXXXXXXXXXXXX-' ;Allows negative
                                        ;numbers.
        DISPLAY(ROW,COL,VT52XX(1,VTLIM+1))     ;Display numeric input
                                        ;right-justified and zero
                                        ;suppressed.
        RETURN
VT523,  IF (VT52SW.NE.1) GO TO VT523X   ;Save alphanumeric input.
        IF (CHAR.EQ.' ') GO TO VT522X   ;Space as a terminating
                                        ;character for numeric input.
        IF (CHAR.EQ.'-') GO TO VT523X   ;Minus sign is acceptable.
        IF (CHAR.LT.'0') GO TO VT523B   ;Check for numeric input.
        IF (CHAR.LE.'9') GO TO VT523X
VT523B, DISPLAY (0,0,7)                 ;Sound alarm--bad input.
        GO TO VT52                      ;Start over (don't clear
                                        ;the error).
VT523X, INCR VT52IN                     ;VT52IN=# of input characters.
        KBDIN(VT52IN)=CHAR
        IF (VT52IN.EQ.VTLIM) GO TO VT526   ;The specified number of
                                        ;characters were input.
        GO TO VT522
VT524,  IF (VT52IN.EQ.0) GO TO VT52
        DISPLAY(ROW,COL,BLNK80(1,VT52IN))   ;Clear characters entered.
        GO TO VT52
VT525,  IF (VT52IN.EQ.0) GO TO VT522
        KBDIN(VT52IN)=
        VT52IN=VT52IN-1
        VT52XX=VT52IN+COL
        DISPLAY(ROW,COL+VT52IN,' ')     ;Rubout previous character.
```

```
        DISPLAY(ROW,COL+VT52IN,0)         ;Reposition cursor.
        GO TO VT522
VT526,  IF(VT52SW.EQ.1) GO TO VT522Y
        RETURN
```

D.1.4.4 Escape Code Sequences as Terminators - A command protocol is built around the Escape code (27 decimal) to implement commands needed by the VT50 and VT52, but not found in 7-bit ASCII. Upon receiving the Escape code 27, the terminal is set to Escape mode and treats the next character received as a command. Commands created in this manner are called Escape Sequences.

In order to use the VT50/VT52 cursor positioning keys as terminators for an ACCEPT statement, the DIBOL program must check for the Escape code (decimal 27) and then execute another ACCEPT statement into a one character alphanumeric field. The contents of this variable can be checked to determine which key was typed. The program then will erase the alpha character entered in this manner and go to the routine associated to the key that was typed.

SPECIAL ESCAPE SEQUENCES

27-A ↑ 27-B ↓ 27-C ← 27-D →	Cursor Positioning Functions
27-P 27-Q 27-R	Special function keys at top of numeric keypad (Unlabeled at present)

D.2 DIRECT ACCESS TECHNIQUES

D.2.1 Background Information

A file contains records of fixed or variable length.

Regardless of the record size, the operating system automatically writes the records into 512-character blocks. The size of a record (in characters) is two plus the number of characters in all the fields in the record. (The two added characters represent the record size in characters divided by two.) If the resulting record size is odd, add one character since only an even number of characters may be written.

Example:

> If the two fields in a record are defined as a D9 field and an A88 field, the record size is 100 (2+9+88+1).

Assuming that all of the records in this file are the same length, the operating system will pack 5 records and the first 12 characters of the sixth record into the first block; the last 88 characters from the sixth record, 4 records, and the first 24 characters from the eleventh record into the second block; and so on to the end.

When this file is later processed, either sequentially (defined as input in an INIT statement) or through direct access (defined as UPDATE in an INIT statement), the operating system will completely restore the record, even if it overlaps two blocks, before passing it to the DIBOL program.

D.2.2 The Reason for Direct Access

Many applications involve the sequential processing of data. For example, a transaction file is entered in random order, sorted and then used to update a master file sorted in the same sequence. Errors in the transaction file cannot be found until the UPDATE program is run. The errors are corrected and a new transaction file is made for the corrected items, which is then sorted and run against the master file. This process continues until no more errors exist. This type of processing evolved 20 years ago with the age of electronic data processing. Systems specialists have desired a better method of operation.

The best method is to verify that data is entered correctly. The operator keying the data file should be able to interact with the master file. For example, a program can be written in which an operator entering payroll information could type an employee number and know within a second or two whether this employee exists on the master file. This would be impossible with sequential processing because of the time involved in sequentially accessing every record. Direct access permits retrieval of any desired record without processing any other records.

D.2.3 How the Direct Access Technique Works in DIBOL

DIBOL uses a record number to access any record in a file. The program has to convert operator input into a record number recognizable by the operating system. This section on direct access will explain several methods to make this conversion.

D.2.4 Unsorted File

Assume that you have an unsorted file containing 1 to 99 records. Each record contains a KEY field as well as other fields. This key will be used for direct access. The first thing done in the following program is to fill up a table. There is a one-to-one correspondence between each element in the table and each record in the file. No I/O is necessary to determine if a specified code is in the master file since this code would not have a match in the table lookup.

```
                RECORD MASTER
KEY,            D5                     ;Could be any size field.
,               A90                    ;Remainder of file.
                RECORD                 ;Working storage.
TABLE,          100D5                  ;Table containing keys.
I,              D3                     ;Index.
LOOKUP,         D5
                PROC 1
                INIT(1,INPUT,'FILNAM')
LOAD,           XMIT(1,MASTER,EOF)
                INCR I
                TABLE(I)=KEY
                GOTO LOAD
EOF,            FINI(1)
                INCR I
                TABLE(I)=99999         ;Indicates end of table.
                INIT(1,UPDATE,'FILNAM')
                .
                .                      ;LOOKUP contains code for master
                                       ;file lookup.
                .
                I=
FINDIT,         INCR I
                IF(TABLE(I).EQ.LOOKUP) GO TO FOUND ;Match.
                IF(TABLE(I).EQ.99999) GO TO NONE ;No match.
                GOTO FINDIT
NONE            XMIT (8,'RECORD NOT FOUND')
                STOP
FOUND,          READ(1,MASTER,I)       ;Read record I.
```

D.2.5 Sorted File

Use the same circumstances as in Section D.2.4 except sort the file by key. Filling the table is the same, but table lookup is faster since the code is not compared to every element in the table. A "no match" condition is known as soon as the table element exceeds the code.

It is possible to cut down the number of comparisons in the table lookup by comparing the middle of the table to the code, checking which half of the table might contain the code, determining the middle of that half of the table, and so on until the element is found. This technique allows faster access, but programming it is much more complicated.

```
               RECORD MASTER
KEY,           D5
,              A90
               RECORD              ;Working storage.
TABLE,         100D5
I,             D3
LOOKUP,        D5
               PROC 1
               INIT(1,INPUT,'FILNAM')
LOAD,          XMIT(1,MASTER,EOF)
               INCR I
               TABLE(I)=KEY
               GOTO LOAD
EOF,           FINI(1)
               INCR I
               TABLE(I)=99999      ;Indicates end of table.
               INIT(1,UPDATE,'FILNAM')
               .
               .                   ;Lookup contains code for master
 file.
               .
               I=
FINDIT,        INCR I
               IF(TABLE(I).EQ.LOOKUP) GO TO FOUND ;Match.
               IF(TABLE(I).GT.LOOKUP) GO TO NONE  ;No match.
               GOTO FINDIT
NONE,          XMIT (8,'RECORD NOT FOUND')
               STOP
FOUND,         READ(1,MASTER,I)    ;Read record I.
               .
               .
               .
```

It is impractical to use direct access with DIBOL on an unsorted file containing many records since an exceedingly large lookup table would be needed.

D.2.6 Rough Table, No Index File

At some point, a file will contain too many records for every key to be saved in a table. When this point is reached, two solutions are available.

The first is to create a "rough" index table containing every 10th or 20th key. For lookup, the rough index will specify within 10 or 20 records on the master file which one is desired. These 10 or 20 records are then sequentially examined to find the desired record (see the following example program).

The second solution is to create a "rough" index table and a "fine" index file. In this method, the rough index table specifies to within 10 or 20 records of the file desired. The index file is then sequentially examined to find the desired key. If a match occurs, the master file is then read.

The proper use of an index file technique can cut down on the number of I/O reads. For example, a master file of 98 characters per record would take up to four I/O reads to find the desired record if the rough index could narrow within 20 records. An index file technique would take one I/O read to find the master record. This technique becomes faster as the size of the master file record increases.

```
                RECORD MASTER
KEY,            D5
,               A90
                RECORD              ;Working storage.
TABLE,          100D5               ;1st,21st,41st key,etc.
I,              D4
J,              D4
LOOKUP,         D5
                PROC 1
                INIT (1,INPUT,'FILNAM')
LOAD,           XMIT(1,MASTER,EOF)
                INCR I
                IF (I.NE.I/20*20+1) GO TO LOAD
                INCR J
                TABLE(J)=KEY        ;Save only 1st,21st,41st key, etc.
                GO TO LOAD
EOF,            FINI(1)
                INCR J
                TABLE(J)=99999      ;Indicates end of table.
                INIT(1,UPDATE,'FILNAM')
```

```
                .
                .              ;LOOKUP contains code for master file.
                .
                I=1
ROUGH,          INCR I
                IF(TABLE(I).LE.LOOKUP) GO TO ROUGH ;No rough match yet.
                I=(I-2)*20      ;Set I to beginning of rough index-1.
FINE,           INCR I
                READ(1,MASTER,I)
                IF(KEY.LT.LOOKUP) GO TO FINE ;No match yet.
                IF(KEY.EQ.LOOKUP) GO TO FOUND ;No match.
                XMIT (8,'RECORD NOT FOUND')
                STOP
FOUND,          .
                .
                .
```

D.2.7 Rough Table Plus Index File

```
                RECORD MASTER
KEY,            D5
,               A90
                RECORD                  ;Working storage.
TABLE,          100D5                   ;1st,21st,41st key, etc.
I,              D4
J,              D4
LOOKUP,         D5
                RECORD INDEX            ;Index file.
XKEY,           D5
                PROC 2
                INIT(1,INPUT,'FILNAM')
                INIT(2,OUTPUT,'XFILE')
LOAD,           XMIT(1,MASTER,EOF)
                INCR I
                XKEY=KEY
                XMIT(2,INDEX)           ;Create fine index file.
                IF(I.NE.I/20*20+1) GO TO LOAD
                INCR J
                TABLE(J)=KEY            ;Save only 1st,21st,41st key.
                GO TO LOAD
EOF,            FINI(1)
                FINI(2)
                INCR J
                TABLE(J)=99999          ;Indicates end of table.
                INIT(1,UPDATE,'FILNAM')
                INIT(2,UPDATE,'XFILE')
                .
                .              ;LOOKUP contains code for master file.
                .
                I=1
```

```
ROUGH,          INCR I
                IF(TABLE(I).LE.LOOKUP) GO TO ROUGH ;No rough match yet.
                I=(I-2)*20      ;Set to beginning of rough index-1.
FINE,           INCR I
                READ(2,INDEX,I)           ;Read index record.
                IF(XKEY.LT.LOOKUP) GO TO FINE ;No match yet.
                IF(XKEY.EQ.LOOKUP) GO TO FOUND ;No match.
                XMIT (8,'RECORD NOT FOUND')
                STOP
FOUND,          READ(1,MASTER,I)          ;Match.
                .
                .
                .
```

D.2.8 Summary

This discussion on direct access does not include information about all possible situations. In cases where the master file is between 2,000 and 40,000 records, the approach might be to have a very rough table, a rough index file, a fine index file, and a master file.

It is possible to work with a large unsorted master file by creating an index file containing two fields: the key field and the record number of the master file. Sort the index file by key. When a match is found on the key field of the index file, the program uses the record number field to read the proper record of the unsorted master file.

Creation of an index table or an index file can be done in a separate program. This separate program can save from several seconds to several minutes each time the program is run. The index file would only need to be changed when a master file is updated (perhaps on a weekly or monthly basis).

D.2.9 Record Count

To keep track of the number of records in a master file, reserve one field in the first record to contain the record count. The record count is the number of records in the file. When a record is added to this file, the record count in the first record is incremented by one and written out. This technique will work fine with a master file that is out of order.

D.3 DIRECT ACCESS NOTES

D.3.1 XMIT Statements (Extending a File)

XMIT statements can be interspersed with direct access operations on a file. An XMIT following a READ with record n is equivalent to a READ of record n+1. Successive XMIT's read records n+2, n+3, etc.

An XMIT following a WRITE of record n transmits data to record n+1.

Records n+2 to the end of the file may be changed by successive XMIT's after a WRITE. However, to change a series of records in the middle of the file, do not use a WRITE followed by several XMIT's.

The XMIT statement used after a WRITE statement has the following useful applications.

D.3.1.1 Truncating a File - To truncate a file after record N, use the following sequence:

```
          READ(channel,record,n)
          WRITE(channel,record,n)
          XMIT(channel,NULL,EOF)
          .
          .
     EOF, FINI(channel)
```

where NULL is a record with no contents defined by:

```
          RECORD NULL
          RECORD
```

D.3.1.2 Appending to a File - To append records to the end of a file with n records, use the following sequence:

```
          READ(channel,record,n)
          WRITE(channel,record,n)
          XMIT(channel,record)        ;Append records to file.
          XMIT(channel,record)
               .
               .
               .
          XMIT(channel,NULL,EOF)
     EOF, FINI(channel)
```

D.3.1.3 Rewriting A File - To rewrite a file from record n to the end of the file, use the following sequence:

```
          WRITE(channel,record,N)
          XMIT(channel,record)
          XMIT(channel,record)
               .
               .                         ;Rewrite records to end-of-file.
               .
          XMIT(channel,NULL,EOF)
     EOF, FINI(channel)
```

D.4 NUMERIC FIELD VERIFICATION

Any numeric field that is entered in a DIBOL program should be checked to determine if it contains only numeric data. The numeric field should be read as an alphanumeric field through an XMIT or ACCEPT statement. Then it is moved to a numeric field. This move is preceded by an ON ERROR statement to check for non-numeric data. For example:

```
              RECORD
          TCHAR,   D2
          DECMAL,  D5
          ALPHA,   A5
          PROC
          .
          .
          .
              ALPHA=
              ACCEPT(TCHAR,ALPHA)
              ON ERROR FIX
              DECMAL=ALPHA
```

With an alphanumeric-to-numeric move, many checks are done. The following examples illustrate most cases:

ALPHANUMERIC	NUMERIC
' 123'	00123
'123 '	00123
'00123'	00123
' -123'	0012S
' 123-'	0012S
'-123-'	00123
' 12-3'	0012S
'1-2-3'	00123
'1+2+3'	00123
'1+2-3'	0012S
'1 23 '	00123
'0012S'	illegal

The only legal characters in an alphanumeric-to-numeric move are 0 to 9, ␣, +, and -.

If a data file contains numeric fields, these fields must be read as numeric. If they contain a negative number, the least significant character contains a minus sign and is listed with its equivalent character. For example, -37 would look like 3W. If 3W were read as alphanumeric, and then converted to numeric, a run-time error would occur since any letter of the alphabet is illegal in an alphanumeric-to-numeric conversion.

D.5 CHAIN STATEMENT NOTES

D.5.1 Interaction of CHAIN and INIT (channel,SYS)

Source input files can be specified in a RUN command containing chained programs. Accessing such files must be done according to the following rules.

1. All CHAIN files must be listed in the RUN command before the source files.

   ```
   .RUN PRONAM+CHAIN1+CHAIN2,INP1,INP2
   ```

2. Any CHAIN statement which is to open the first source input file must first "skip over" the remaining chained files by issuing dummy INIT(channel,SYS) statements. In the above RUN command, in order to read file INP1, PRONAM would have to issue two dummy INIT(channel,SYS) statements. Because CHAIN2 is the last chain program, it would not have to issue any dummy statements.

If the RUN command were:

```
RUN pronam, filnam1...,filnam7
```

the source files could be processed more than once by executing a CHAIN 0 statement in pronam.

D.5.2 Transferring Variable Values

For the value of a variable to be successfully transmitted from one chained program to another, the variable in which the value appears must occupy the same location in both CHAIN programs. This may be accomplished by either of the two following methods.

1. Define all records which are to be passed between chained programs first, and make the definitions identical (except for variable names which may be different).

   Example:

   Chain1

   ```
   RECORD
   CUST, A30
   PROD, D2
   RECORD INVENT
   STOCK, D4
   .
   .
   .
   ```

   Chain2

   ```
   RECORD CPINFO
   CUST, A30
   PCODE, D2
   RECORD INVENT
   QUANTY, D4
   .
   .
   .
   ```

2. Use the compiler storage maps listing for the two CHAIN programs to verify that the desired variables occupy the same storage location.

D.5.3 Multiple CHAIN Entry Points

Sometimes it is desirable to have several entry points into a CHAIN program. However, the CHAIN statement always starts execution of the chained program at the first statement following the PROC statement. Using the technique of transferring variable values between chained programs, multiple entry points can be programmed as indicated in the following example.

Chain1

```
RECORD
WHERE,D2,01
RETURN,D2
.
.
.
PROC
GO TO(L1,L2,L3,L4),WHERE
L1,RETURN=2
.
.
.
CHAIN 2
L2,...
.
.
.
```

Chain2

```
RECORD
WHERE,D2
RETURN,D2
.
.
.
PROC
GO TO(E1,E2,E3),WHERE
.
.
.
E1,...
.
.
.
WHERE=RETURN
CHAIN 1
```

D.6 DIBOL PROGRAMMING OF SOURCE FILES

D.6.1 Operating Procedures

Up to seven source files can be used in a DIBOL program. They are specified at run time by:

```
RUN pronam, filnaml...,filnam7
```

The edit buffer is not available to a DIBOL program.

D.6.2 Data Division

The RECORD description would be as follows:

```
RECORD recnam
LINENO, A2
CHAR, A120
```

LINENO contains a two-character line number in binary. Most programs ignore the line number. However, it can be converted to decimal by the statement:

```
varnam = #LINENO*64+#LINENO(2,2)
```

Varnam must be a four-digit field.

CHAR contains the characters of one line created by the editor. The DIBOL program may want to determine the number of characters in the record. This can be done by preceding the RECORD statement with the lines:

```
RECORD
TRICK, 2A1
```

and adding the following line in the Procedure Division:

```
varnam = (4096-64*#TRICK(3)-#TRICK(4)*2
```

Varnam must be a three-digit field.

There is no tabbing within CHAR. The tabbing seen by output from the Monitor command LIST or LIST/L is done by the operating system. Tabs are internally stored as characters with a decimal equivalent of 61. Any character may be checked for a tab by the statement:

```
IF(#CHAR(n,n).EQ.61) GO TO TAB
```

D.6.3 Procedure Division

The first source file specified in the RUN command is opened by the following statement:

```
INIT(channel,SYS)
```

Each record is accessed by the following statement:

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

When an end-of-file condition occurs, the program transfers to the EOF label of the XMIT statement. At that EOF label, a FINI channel statement must be executed prior to an INIT(channel,SYS) to open a second source file. To handle a variable number of source files, precede the INIT(channel,SYS) statement by an ON ERROR statement. The program will transfer to the ON ERROR statement when an INIT(channel,SYS) statement is executed and there are no more source files.

The only way to process a source file more than once is to execute a CHAIN n statement which resets the operating system pointers to the source files.

This example combines up to seven source files into a single data file. The resulting data file can be converted to a source file using FILEX.

Example:

```
        RECORD
SIZE,   D3
        RECORD
TRICK,  2A1
        RECORD LINE            ;Line from source file.
,       A122
        PROC 2
        ON ERROR NOFILE
        INIT(1,SYS)            ;Open system file.
        INIT(2,O,'$FILE',1)    ;Open output file on logical unit 1.
NEXT,   XMIT(1,LINE,EOF)       ;Read a line from source file.
        SIZE=(4096-64*#TRICK(3)-#TRICK(4))*2 ;Get size of line.
        XMIT(2,LINE(3,SIZE+2)) ;Output line without line number.
        GO TO NEXT
EOF,    FINI(1)                ;Close system file.
        ON ERROR DONE
        INIT(1,SYS)            ;Open next system file.
        GO TO NEXT
DONE,   FINI(2)                ;Close output file.
NOFILE, STOP
```

D.7 CHECKDIGIT FORMULA

In most applications involving identification numbers, each number may be verified for accuracy by a checkdigit, a redundant digit added to the normal number. The checkdigit is determined by performing an arithmetic operation on the number in such a way that the usual errors encountered in transcribing a number are detected. The checkdigit is determined as follows:

Step 1 Start with a number....5764.

Step 2 Multiply the first digit and every other digit by 2 (left to right). 5 * 2 = 10 6 * 2 = 12

Step 3 Add the digits in the resulting numbers and the digits not multiplied. 1+0+7+1+2+4 =15

Step 4 Subtract the sum from Step 3 from the next higher number ending in zero. 20-15=5

Step 5 Add the checkdigit to the end of the original number. 57645 (This is the correct checkdigit if the number is entered in a D4 field.)

Note that a checkdigit procedure is not completely error proof. In the example given above, 5764 or 5673 give the same checkdigit. It is unlikely, however, that transpositions of this sort will occur. The checkdigit does not guard against the possible assignment of an incorrect but valid code, such as the assignment of a wrong valid identification code to a customer.

If the number entered for a checkdigit calculation is shorter than the field, the rightmost digit is used as the checkdigit and the remainder of the number is right-justified and padded with zeros on the left. The zeros are considered when the checkdigit formula is calculated.

D.8 VT50/VT52 ESCAPE SEQUENCES

A command protocol is built around the escape code (027) to implement those commands needed by the VT50/VT52 but not found in 7-bit ASCII. Upon receiving the escape code 027, the terminal is set to escape mode and treats the next character received as a command. Commands created in this manner are called Escape Sequences. The VT50/VT52 recognizes the following Escape Sequences:

Code	Character	Action Taken
27	ESC	The first 027 changes the mode, the second 027 changes it back.
65	A	Moves cursor up one line.
66	B	Moves cursor down one line.
67	C	Moves cursor right one position.
68	D	Moves cursor left one position.
72	H	Moves cursor to the home position.
74	J	Erases from cursor position to the end-of-screen.
75	K	Erases line from cursor to right margin.
90	Z	Requests the terminal to identify itself. The VT50 terminal will respond with 027 047 072; VT52 terminal will respond with 027 047 075. Other terminals will respond in different ways.

[1]Teletype is a registered trademark of the Teletype Corporation.

GLOSSARY

alphanumeric
A character set that contains letters, digits, and other characters such as punctuation marks. The COS-310 alphanumeric character set includes the uppercase letters A-Z, the digits 0-9, and most of the special characters on the terminal keyboard. Two of these characters, back slash (\) and back arrow (←) (shown on some terminals as an underscore), are illegal.

array
A DIBOL technique for specifying more than one field of the same length and type. The array 5D3 reserves space for five numeric fields, each to be three digits long. The array 2A10 describes two alphanumeric fields, each to be ten characters long.

ASCII
American Standard Code for Information Interchange. This is one method of coding alphanumeric characters.

batch file
A file containing a sequence of commands. A command to execute the file will cause the commands within the file to be executed sequentially.

batch processing
The technique of automatically executing a group of previously stored Monitor commands.

binary operator
An operator, such as + or -, which acts upon two or more constants or variables (e.g., A=B-C).

binary program
The kind of program which is output by the compiler.

binary scratch area
The area in memory where the binary program is stored during execution.

bit
A binary digit (0 or 1).

block
The basic COS-310 unit of mass storage capacity. A block consists of up to 512 characters.

bootstrap
A short routine loaded at system start-up time which enables the system software to be read into machine memory.

branch
A change in the sequence of execution of COS-310 program statements.

buffer
A temporary storage area usually used for input or output data transfers.

bug
An error or malfunction in a program or machine.

byte
A group of bits considered as a unit. A byte is the smallest unit of information that can be addressed in a DIBOL program.

channel
A number between 1 and 15 used to associate an input/output statement with a specified device.

character
A letter, digit, or other symbol used to control or to represent data.

character string
A connected linear sequence of characters.

clear
Setting an alphanumeric field to spaces or a numeric field to zeros.

command
An operator request for Monitor services; usually to be executed following a RETURN key.

comments
Notes for people to read; they are ignored by the compiler. Comments are optional and follow a semicolon on a statement line.

concatenated
Strung together without intervening space.

conversational program
A program that prompts responses from an operator and reacts depending upon the response from the operator.

cursor
The flashing light indicator which appears at the point on the screen where the next character will be displayed.

data
A representation of information in a manner suitable for communication, interpretation, or processing by either people or machines. In COS-310 systems, data is represented by characters.

data entry
The process of collecting and inputting data into the computer data files. Data entry is key to disk.

data management
The planning, development, and operation of a system like COS-310 by an organization to mechanize its information flow and make available the data needed by the organization.

debug
To detect, locate, and remove errors or malfunctions from a program or machine.

DEC
Acronym for Digital Equipment Corporation.

decimal
Refers to a base ten number.

delimiting
The bounds (beginning and end) of a series or string.

device designation
A three-character designation for a mass storage device. The first two characters designate the type of device; the third character designates the number of the drive on which the device is mounted.

device independence
COS-310 system design permits data files and programs to be stored on either diskettes or disks. At run time, the operator chooses the most suitable or most available input and output devices.

device designations
A three-character abbreviation used to name the COS-310 I/O devices.

TTY	= Screen
KBD	= Keyboard
LPT	= Printer
DK0-DK3	= Disk drives
RX0-RX3	= Disk drives
DY0-DY3	= Disk drives

DIBOL
Digital's Business Oriented Language is a COBOL-like language used to write business application programs. The source language of the COS-310 system.

direct access
The process of obtaining data from, or placing data into, a storage device where the availability of the data requested is independent of the location of the data most recently obtained or placed in storage. Direct access is available to users of COS-310 systems by writing the position number of any record in a data file. For example, you can request the 5th, 35th, and 711th records in a file.

directory
A place for listing information for reference. Displayed or printed with the DI command.

dump
To copy the contents of all or part of storage, usually from memory to external storage.

edit buffer
The work area in memory where source files are created and edited.

end-of-file mark
A control character which marks the physical end of a multivolume file. For both input and output files, the Monitor detects this EOF mark and types a message for the operator asking that the next volume in the file be mounted.

fatal error
An error which terminates program execution.

field
A specified area in a data record used for alphanumeric or numeric data; cannot exceed the specified character length.

file
A collection of records, treated as a logical unit.

fixed-length records
Each record in a data file is the same length. Fixed-length records are the only type handled by COS-310 utility programs and the only type on which direct access to data files is allowed.

flowchart
A pictorial technique for analysis and solution of data flow and data processing problems. Symbols represent operations, and connecting flowlines show the direction of data flow.

handlers
A specialized software function which interfaces between the system and peripheral devices.

illegal character
A character that is not valid according to the COS-310 design rules. DIBOL will not accept back slash (\) and back arrow (←) (back arrow is replaced on some terminals with underscore) in alphanumeric strings.

initialization
Putting a device into the correct format or position where it can successfully function in a configuration.

input
Data flowing into the computer.

input/output
Either input or output, or both. I/O.

jump
A departure from the normal sequence of executing instructions in a computer.

justify
The process of positioning data in a field whose size is larger than the data. In alphanumeric fields, the data is left-justified and any remaining positions are space-filled; in numeric fields the digits are right-justified and any remaining positions to the left are zero-filled.

key
One or more fields within a record used to match or sort a file. If a file is to be arranged by customer name, then the field that contains the customers' names is the key field. In a sort operation, the key fields of two records are compared and the records are resequenced when necessary.

load
To enter data or programs into main memory.

load-and-go
An operating technique in which there are no stops between the loading and executing phases of a program.

location
Any place where data may be stored.

logical unit number
A number (1-15) which identifies an entry in a logical unit table. The table references the number to a location on a mass storage device.

logical units
An area of storage on a mass storage device. Up to 15 logical units may be assigned at system start-up by the data file utility program (DFU). These areas and their assigned sizes are listed in the logical unit table printed by DFU.

loop
A sequence of instructions that is executed repeatedly until a terminal condition prevails. A commonly used programming technique in processing data records.

machine-level programming
Programming using a sequence of binary instructions in a form executable by the computer.

mass storage device
A device having large storage capacity.

master file
A data file that is either relatively permanent or that is treated as an authority in a particular job.

memory
The computer's primary internal storage.

merge
To combine records from two or more similarly ordered strings into another string that is arranged in the same order. The latter phases of a sort operation.

mnemonic
Brief identifiers which are easy to remember. Examples are KBD, LPT, and TTY.

mode
A designation used in INIT statements to indicate the purpose for which a file was opened or to indicate the input/output device being used.

modulo
A condition where the specified number exceeds the maximum condition in a variable. The maximum allowable number is then subtracted from the specified number and the remainder is used by the processor. In modulo 16, if 17 were specified (maximum is 15), 16 would be subtracted from 17 and the processor would use 1 as the variable.

Monitor
A COS-310 system program that loads and runs programs and performs other useful tasks.

nest
To embed subroutines, loops, or data in other subroutines or programs.

nonfatal error
An error which will not completely terminate program execution.

nonsystem device
A device that does not contain the operating system and the Monitor. A device used exclusively for data storage.

option switch
A one- or two-character designation indicating a special function in conjunction with a command. Usually preceded by a slash (/) in COS-310.

output
Data flowing out of the computer.

overlay
The technique of specifying several different record formats for the same data. Special rules apply.

parameter
A variable that is given a constant value for a specific purpose or process.

peripheral equipment
Data processing equipment which is distinct from the computer.

pushdown stack
A list of items where the last item entered becomes the first item in the list and where the relative position of the other items is pushed back one.

random access
Similar to direct access.

RECORD
A statement that reserves memory for DIBOL data language programs.

segment
Sixteen blocks of storage. A block is 512 bytes long.

sequential operation
Operations performed, one after the other.

serial access
The process of getting data from, or putting data into, storage where the access time is dependent upon the location of the data most recently obtained or placed in storage.

screen line number
The number which indicates the order of the horizontal lines on the screen.

sign
Indicates whether a number is negative or positive. Positive numbers do not require a sign, but negative numbers are prefixed with the minus sign (-).

significant digit
A digit that is needed or recognized for a specified purpose.

source program
A program written in COS-310 DIBOL language.

statement
An instruction in a source program.

string
A connected linear sequence of characters.

subscript
A designation which clarifies the particular parts (characters, values, records) within a larger grouping or array.

switch character
A single letter specified in a command following a slash (/).

syntax
The rules governing the structure of a language.

system configuration
The combination of hardware and software that make up a usable computer system.

system device
A mass storage device reserved for Monitor, Run-Time System, and other system and source programs.

systems directory
A list of programs on the systems device with lengths, dates of creation, and other useful information.

system handlers
The specialized software which interfaces between the system and peripheral devices.

terminal alarm
A signal emitted from the terminal.

unary operator
An operator, such as + or -, which acts upon only one variable or constant (e.g., A=-C).

utility program
A system program which performs common services and requires format programs. Examples are SORT and PRINT.

variable
A quantity that can assume any one of a set of values.

variable-length record
A file in which the data records are not uniform in length. Direct access to such records is not possible.

verify
To determine if a transcription of data has been accomplished accurately.

word
A string of 12 binary bits representing two COS-310 characters.

zero fill
To fill the remaining character positions in a numeric field with zeros.

A

B

C

E

F

G

H

I

O

P

R

T

U

V

W

X

Z

READER'S COMMENTS

NOTE: This form is for document comments only. DIGITAL will use comments submitted on this form at the company's discretion. Problems with software should be reported on a Software Performance Report (SPR) form. If you require a written reply and are eligible to receive one under SPR service, submit your comments on an SPR form.

Did you find errors in this manual? If so, specify by page.

Did you find this manual understandable, usable, and well-organized? Please make suggestions for improvement.

Is there sufficient documentation on associated system programs required for use of the software described in this manual? If not, what material is missing and where should it be placed?

Please indicate the type of user/reader that you most nearly represent.

- ☐ Assembly language programmer
- ☐ Higher-level language programmer
- ☐ Occasional programmer (experienced)
- ☐ User with little programming experience
- ☐ Student programmer
- ☐ Non-programmer interested in computer concepts and capabilities

Name ______________________ Date ______________

Organization ______________________

Street ______________________

City ______________ State ______________ Zip Code or Country ______________

Please cut along this line.

Fold Here

Do Not Tear - Fold Here and Staple

FIRST CLASS
PERMIT NO. 33
MAYNARD, MASS.

BUSINESS REPLY MAIL
NO POSTAGE STAMP NECESSARY IF MAILED IN THE UNITED STATES

Postage will be paid by:

digital

Business Products
Software Development Group
MK1-2/H32
Continental Blvd.
Merrimack, New Hampshire 03054