DECUS
PROGRAM LIBRARY

DECUS NO.	8-392
TITLE	VECTOR-8
AUTHOR	Richard Rothman
COMPANY	Groton School Groton, Massachusetts
DATE	April 24, 1971
SOURCE LANGUAGE	PAL-1Ø

ATTENTION

This is a USER program. Other than requiring that it conform to submittal and review standards, no quality control has been imposed upon this program by DECUS.

The DECUS Program Library is a clearing house only; it does not generate or test programs. No warranty, express or implied, is made by the contributor, Digital Equipment Computer Users Society or Digital Equipment Corporation as to the accuracy or functioning of the program or related material, and no responsibility is assumed by these parties in connection therewith.

DECUS Program Library Write-up DECUS NO. 8-392

I N T R O D U C T I O N

This document describes Vector-8, and assumes that the reader is familiar with the basic concepts of Vector-8. The article entitled "An Introduction to Vector-8, a New Programming Language," published in DECUSOPE Vol. 10, No. 1 is included and should provide the rudimentary knowledge necessary for the reading of this manual.

It is also pertinent to note that many features and some details of Vector-8 have changed since the publication of the article. Thus, the article should be read as a primer, not as a statement of the rules of syntax and other such details.

V E C T O R - 8

MISCELLANEOUS ELEMENTS

Numbers may be specified by up to 15 digits, and a decimal exponent. The value of a number in Vector-8 may vary from $.14 \times 10^{-28}$—1.7×10^{38}. Some examples of numbers:

12	155Ø	—12.98
12.2	1.5E5	—1.5E-5

Vector-8 works with 7 significant digits.

Character strings are specified by a string of characters enclosed within quotes. Any legal character, including a CR, may be part of the string. Examples (↵ stands for CR)

```
"A&!BC"
"ABC↵DE"
"DE"
```

Vector-8 recognizes 3 pairs of parenthesis: (),[],< >. They must be matched correctly, or an error results.

Vector-8 recognizes the following characters:

```
ABCDEFGHIJKLMNOPQRSTUVWXYZ
1234567890
!"#$%&'()*+,-./:;<=>?
@[\]↑ SPACE
```

The following a control characters are recognized by Vector-8 in input:

←	:	issues a CR-LF, restarts input.
RUBOUT	:	echos "\", erase char on left
ALTMODE	:	echos a CR-LF, but packs nothing. This allows lines to be continued.
LINE FEED:		only recognized in a char string. Echos and packs a CR, but does not terminate input.
CR	:	terminates input. If typed in a char string, acts as a LINE FEED.

Variables are uniquely identified by two characters, the first of which must be a letter, and the second of which may be either a letter or digit. This is analogous to FOCAL. The defining of variables is described in the section on the SET command.

Indexing is accomplished by following a variable, number string, or character strings with an expression enclosed in parenthesis. This expression must not result in a character string. All indices are truncated and must be greater than Ø. An indexed expression may itself be indexed. This may proceed to any depth. Some examples:

5 6 7 8 (1 2 4) ⇒ 5 6 8
5 6 7 8 (1 2 4) (1 2) ⇒ 5 6
5 6 7 8 (1 2 4) (2 3) ⇒ 6 8
5 6 7 8 (1 2 4) (2 3) (2) ⇒ 8
"ABCD" (4 3 2 1) ⇒ "DCBA"
"ABCD" (4 3 2 1) (3 4) ⇒ "BA"

CNTRL/C will interrupt Vector-8 during any operation and restart.

O P E R A T O R S

Vector-8 offers 15 operators. A table follows: (N-numerical mode, C-character mode).

OPERATOR	PRIORITY	MODE	OPERATION
+	3	N	addition
-	3	N	subtraction, unary minus
/	4	N	division
*	4	N	multiplication
↑	5	N	general exponentiation
:EQ:	2	N or C	relational =
:GT:	2	N or C	relational >
:LT:	2	N or C	relational <
:GE:	2	N or C	relational ≥
:LE:	2	N or C	relational ≤
:NE:	2	N or C	relational ≠
#	2	N or C	compression
&	2	N	logical and (∧)
!	2	N	inclusive or
,	1	N or C	cantenation

In an expression, exponentiation is performed first, multiplication and division second, addition and subtraction third, the relational operators, compression and logic fourth, and cantenation last.

The operator "↑" indicates general exponentiation. Thus, a ↑ b is computed $e^{b \ln a}$. Due to the discontinuities of the general expontiation function, (a ↑ b when a<o exists only for odd b), it is necessary to determine whether or not b, when a<o, is an odd fraction. This on a computer is impossible to do, thus certain comprimises have to be made. Exponentiation in Vector-8 is defined in the following manner:

When a>Ø, a ↑ b is legal for all b. Example:

```
2 ↑ 2 ⇒ 4
2 ↑ (-2) ⇒ .25
```

When a=o, a ↑ b produces Ø for all b. Example:

```
Ø ↑ .5 ⇒ Ø
Ø ↑ 2  ⇒ Ø
```

When a<Ø, a ↑ b is legal for i) all b≥Ø
ii) b<Ø if b is an integer
or 1/b is an odd integer

Example:

(-2) ↑ 2 ⇒ 4
(-2) ↑ 3 ⇒ -8
(-4) ↑ .5 ➡ error
(-8) ↑ (1/3) ⇒ -2
(-2) ↑ .2 ⇒ $\sqrt[5]{-2}$

F U N C T I O N S

Vector-8 has 32 functions. A function is specified by an "F", followed by the name. A list follows:

Numerical functions:

FSIN (ARG)	Returns sine. Arg in radians.
FCOS (ARG)	Returns cosine. Arg in radians
FTAN (ARG)	Returns tangens. Arg in radians. If cos (ARG) = co, the error "DIVISION BY ZERO" is returned.
FATN (ARG)	Returns arc tangent of Arg.
FLOG (ARG)	Returns the naperian log. If ARG ≤ Ø, the error, "LOG OF ZERO" is returned.
FEXP (ARG)	Returns e^{ARG}
FABS (ARG)	Returns |ARG|.
FSGN (ARG)	If ARG<Ø, returns -1. If ARG≥Ø, returns +1.
FNOT (ARG)	If ARG = Ø, returns a 1. If ARG = 1, returns a Ø. If ARG ≠ Ø ≠ 1, returns error message: "BAD ARG IN FUNCTION".
FINT (ARG)	Returns integer part of ARG.
FFRC (ARG)	Returns fractional part of ARG.

If DIM (ARG) = Ø, then a null vector is returned by all of numerical functions. If ARG is a character string, the error "BAD ARG IN FUNCTION" is returned.

Reductive Functions:

FMLT(ARG)	If DIM(ARG) = Ø, a 1 is returned. If DIM(ARG) = 1, ARG(1) is returned. If DIM(ARG) ≥ 2, ARG(1)X---X ARG [DIM(ARG)] is returned.
FADD(ARG)	If DIM(ARG) = Ø, a Ø is returned. If DIM(ARG) = 1, ARG(1) is returned. If DIM(ARG) ≥ 2, $\sum_{i=1}^{DIM(N)}$ ARG(i) is returned.
FSUB(ARG)	If DIM(ARG)= Ø, a Ø is returned. If DIM(ARG) = 1, ARG(1) is returned. If DIM(ARG) ≥ 2, ARG(1)-...-ABG[DIM(ARG)] is returned.
FDIV(ARG)	If DIM(ARG) = Ø, a 1 is returned. If DIM(ARG) = 1, ARG(1) is returned. If DIM(ARG) ≥ 2, ARG(1)/ARG(2)/.../ ARG[DIM(ARG)] is returned.
FMAX(ARG)	If DIM(ARG) = Ø, a null vector is returned. If DIM(ARG) = 1, ARG(1) is returned. If DIM(ARG) ≥ 2, the largest number is returned.
FMIN(ARG)	Same as FMAX, except that smallest number is returned.
FAND(ARG)	Same as FMAX, except that ARG(1)∧ARG(2)∧... ARG[DIM(ARG)] is returned.
FIOR(ARG)	Same as FMAX, except that ARG(1)∨ARG(2)∨...∨ ARG[DIM(ARG)] is returned.
FEQ(ARG)	Same FMAX, except that ARG(1):EQ:ARG(2):EQ...EQ:ARG[DIM(ARG)] is returned.

FGT(ARG)	Same as FEQ, except that :GT: is used.
FLT(ARG)	Same as FEQ, except that :LT: is used.
FGE(ARG)	Same as FEQ, except that :GE: is used.
FLE(ARG)	Same as FEQ, except that :LE: is used.
FNE(ARG)	Same as FEQ, except that :NE: is used.

The only errors possible when using the reductive functions is the error "MODE ERROR WITH OPERATOR." This occurs when ARG in FAND and FIOR is not comprised solely of 1's and Ø's.

Also note that character strings are legal in the reductive functions.

Special Purpose Functions:

FCHN(ARG)	Changes the mode of ARG, from numeric to character or vice versa.
FSER(ARG)	$1 \leq$ DIM(ARG) ≤ 3. If this is not true, error message "BAD ARG IN FUNCTION" is returned. The three forms of FSER are outlined below.
i) FSER(a)	If a = Ø, a null vector is returned. If a < Ø, "BAD ARG IN FUNCTION" is returned. If a > Ø, the series 1,2,...,a is returned.
ii) FSER(ab)	If a $\geq$ b, a null vector is returned. If a < b, the series a, a+1, a+2,...b is returned.
iii) FSER(abc)	If a $\leq$ c and b < Ø, or if b = Ø, then "BAD ARG IN FUNCTION" is returned. If a > c, a null vector is returned. If a < c and b > Ø, the series, a, a+b, a+2b,...,c is returned.
FDIM(ARG)	Returns dimension of the ARG.
FRAN(ARG)	If DIM(ARG) = Ø, the random generator is randomized. This equivalent to a BASIC RANDOMIZE command. If DIM(ARG) $\geq$ 1, ARG(1) random numbers are generated. All random numbers are between Ø and 1.

FREP(V,N)	If DIM(V,N) = Ø, or if N < Ø, or if the ARG is a character string, the error, "BAD ARG IN FUNCTION" is returned. If DIM(V,N) = 1, a null vector is returned. If DIM(V,N) ≥ 2, V is repeated N times.
FROT(V,N)	If DIM(V,N) = Ø, the error, "BAD ARG IN FUNCTION" is returned. If DIM(V,N) = 1, a null vector is returned. If DIM(V,N) ≥ 2, then V is rotated -N times left if N < Ø, or N times right if N > Ø, or no times at all if N = Ø.
FNEW(ARG)	FNEW is a function which can be defined by the uses on a machine language level. As it is defined now, it simply returns the argument.

C O M M A N D S A N D L I N E S T R U C T U R E

Vector-8 offers 22 commands. Each command is uniquely identified by 3 characters (except for the IF command).

The Vector-8 interpreter recognizes two modes of operation. One is direct mode, that is a command string is executed immediately upon input. The second is indirect mode, that is a command string is stored away, identified by a line number, for later reference. Mode one is indicated by typing a command string, and hitting a CR. Mode two is indicated by preceding the string with a line number.

A line number can be an integer greater than Ø but less than 2048. A command string is made up of a series of commands, each terminated by a "\". The entire string is terminated by a CR. This is analogous to FOCAL, except that commands in it are separated by semicolons.

A description of the commands follows:

1.	REM	Remark. This is the connect command. Any argument is ignored, and control is passed to the next command.
2.	RUN	This command runs the program in core.

3) QUIT — This causes program execution to cease. A return to immediate mode is made.

4) GOTO EXP — (EXP stands for an expression). Transfers control to line EXP(1). Possible errors:

i) "BAD ARG IN COMMAND" -DIM(EXP) = Ø or EXP is a char string.

ii) "LINE NOT FOUND" -line cannot be found.

iii) "ILLEGAL LINE NO" - The line specified by EXP(1) is less than or equal to Ø. Note, EXP(1) is fixed, so the fractional part, if it exists, is ignored.

5) GOSUB EXP — Control is transferred to a subroutine at line at EXP(1), then at line EXP(2), then at line EXP(3),..., at line EXP(DIM(EXP)). The same errors as in GOTO occur in GOSUB.

6) RETURN — Return from subroutine. If the GOSUB command has done all lines specified, then control passes to the command following the GOSUB. If not, then control passes to the next subroutine. The one possible error is:

i) "RETURN WITHOUT GOSUB" - A RETURN command is not matched with a GOSUB.

7) SET VAR = EXP — The variable VAR is defined to be the result of expression EXP. Possible errors:

i) "SYNTAX ERROR" - a violation of syntax has occurred.

ii) "VAR TO LARGE" - When attempting to creat VAR, the variable storage area overflowed into the temporary stack. The user should attempt to make DIM(EXP) smaller.

SET VAR (INDEX) = EXP — Certain elements of VAR are replaced by EXP. If DIM(EXP) =1, then all elements are set to EXP, otherwise DIM(EXP) must equal DIM(INDEX). Possible errors:

i) "UNDEFINED VAR" - One cannot index a VAR which hasn't been defined.

ii) "SYNTAX ERROR" - (obvious)

iii) "BAD ARG IN COMMAND" - (DIM)EXP = Ø.

iv) "INDEX TO LARGE" - Index > DIM(VAR).

v) "INDEX = ZERO" - Indices < Ø.

vi) "INDEX A CHAR STRING" - (obvious)

8) ASK VAR — A "?" is typed on the teletype, and the user types any legal expression. It is evaluated and VAR is defined to be the result.

ASK VAR (INDEX) — A "?" is typed on the teletype. The procedure is the same as "ASK VAR", except that only certain elements of VAR are redefined.

ASK % VAR — Same as "ASK VAR", except that the input is taken from the high speed reader. Care should be taken to see that the data tape is terminated by a CR. A procedure for generating data tapes will be discussed under the FORMAT command.

ASK % VAR (INDEX) — Same as "ASK % VAR," except that only certain elements of VAR are replaced.

The possible errors are the same as occur in the SET command.

9) TYPE ARG; ARG; ...;ARG — ARG can be any of the following 5 things:

a) ! a CR-LF is typed.

b) # a CR is typed.

c) ↑ a LF is typed.

d) % output goes to the high speed punch for the rest of the TYPE command.

e) EXP If a char string, the characters are typed. If a number string, the numbers are outputed according to the format set via the FORMAT command.

There are no special error messages associated with the type command. Note, a null vector is outputted as a blank.

10) If EXP — EXP is evaluated. If EXP = 1, then the rest of the line is executed. If EXP = Ø, then control goes to the next line. Possible errors:

i) "BAD ARG IN FUNCTION" occurs if EXP 1 ≠ Ø.

11) LOOP VAR = EXP — This command initiates a loop according to the parameters in EXP. There are 2 forms of this command.

i) LOOP VAR = a b — In this case a must be ≤ b. The increment is assumed to be 1. Thus VAR varies in value from a, a+1, a+2, ...,b as the loop proceeds. The loop ends when VAR > b. All commands following the LOOP command up to the corresponding NEXT statement are executed for the limits of the loop.

ii) LOOP VAR = abc — This is exactly the same as case i, except that the increment is specified. There are 2 cases to this form of the LOOP command.

a) a < c and b > Ø — Loop proceeds till VAR > c.

b) a > c and b < Ø — Loop proceeds till VAR < c.

The main possible error is "BAD ARG IN COMMAND" which can occur if for instance EXP is null, a is a char string, or if, in case 2, a < c and b < Ø, or if b=Ø, etc.

12) NEXT VAR — This command marks the end of a particular loop initiated by a LOOP command. If the loop is not completed, it transfers control to the command following the LOOP command. If the loop is finished, the control is transferred to the command following the NEXT command.

The operation of the LOOP, NEXT commands is similiar to the operation of the BASIC FOR, NEXT commands. No check is made, though, to see if the loops are properly nested. The error in the NEXT command is "BAD ARG IN COMMAND".

13) LIB S PNAME — Saves the program in core in the library under the name, PNAME. PNAME is a name which consists of of up to 4 characters which may be a letter, digit, or period. Possible errors:

i) "DIRECTORY FULL" occurs if there is no more room or if a program is already in the library under the same name.

LIB C PNAME — Calls the program whose name is PNAME from the library into core. Possible error:

ii) "NAME NOT IN LIB" No program whose name is PNAME.

LIB D PNAME — The program whose name is PNAME is deleted. Possible errors:

ii) "NAME NOT IN LIB"

LIB L — List the directory, and the number of free blocks.

There are room for 20 programs in the library. Each block is 1024_{10} words. LIB C does not start the program once it is called.

14) CALL P1; P2 — P1 is the name of the program currently in core. P2 is called into core, executed, and control resumes at the command following the CALL command in P1. Possible errors:

i) "NAME NOT IN LIB" programs under the name of P1 or P2 are not found in the library.

In order for CALL command to operate correctly, all programs referenced by the CALL command must be in the library. This is because when P2 is called into core, P1 is necessarily erased, thus, leaving the copy of P1 in the library the only existing one. When P2 ceases execution, P1 is called back into core from the library, thus, a good copy of P1 must exist on the disk.

15) WRITE EXP — The lines specified by EXP are listed. If EXP is null, the entire program is listed. By typing, for example, "WRITE 2," the entire program is listed.

WRITE % EXP — Same as above, except output is to the high speed punch.

Possible errors:
i) "LINE NOT FOUND"
ii) "ILLEGAL LINE NO."
iii) "BAD ARG IN COMMAND" EXP is a char string

16) KILL VAR_1; VAR_2; ....; VAR_7 — The variables in the listed are deleted. If there is no list, all variables are deleted.

17) DELETE EXP — The lines specified are deleted. If EXP is null, all text is deleted.

Possible errors:
i) "ILLEGAL LINE NO"
ii) "BAD ARG IN COMMAND" - EXP is a char string

18) DUMP — No argument. Dumps out the variable storage in the following format.

N A M E M O D E (N or C) D I M E N S I O N

19) TRACE S EXP — A trace is set at the lines specified by EXP. If EXP is null, then the entire program is traced.

TRACE K EXP — A trace is removed from the lines specified by EXP. If EXP is null, all traces are removed.

The possible errors are what one would expect, "ILLEGAL LINE NO.", "LINE NOT FOUND", etc.

The trace feature works in the following way: if a line is traced, its line number is typed on the teletype as execution of the line is about to commence. Thus, this trace feature provides a picture of the flow of the program. When the program is listed, an "'" is typed following the line number to indicate that a line is traced.

19.2) PUNCH This command causes the program in core to be punched on the low speed punch. The format of the tape is as follows: Leader/Trailer, Binary dump of program, Leader/Trailer. This command should be used to save programs on paper tape, the WRITE command should not. The reason for this is that Vector-8 has a facility for the continuation of lines. The WRITE command, though, when printing a line which is continued, types CR's between the lines in order to produce readable output. But when the tape produced by the WRITE command is read back in, the CR's typed by WRITE only for legibility are interpreted as line terminators. Thus, the need for the PUNCH command.

19.4) PUNCH %

Same as PUNCH, except that the tape is produced on the high speed punch.

19.6) READ

A tape produced by the PUNCH command is read from the low speed reader.

19.8) READ %

Same as READ, except the high speed reader is used.

20) FORMAT ARG; ARG;... This command sets up output parameters. ARG can be one of two things:

i) NXX.YY This sets up format for the type out of number. XX is the total number of digits, and YY is the number of digits to the right of the decimal point. XX and YY must be less than 31. The current output format is retained until changed. If a number is too large for a given format, floating point format is used temporarily. If no XX.YY is specified, then floating point format is used. Format is initially set to 8.07.

ii) LEXP This sets up the format for the line. $1 \leq$ DIM(EXP) ≤ 3. Three parameters can be specified.

a) Number of numbers per line.
b) Number of spaces between numbers.
c) Number of CR-LF's between lines.
If DIM(EXP) = 1, parameter a is set
If DIM(EXP) = 2, parameters a,b are set
If DIM(EXP) = 3, parameters a,b,c are set

The numbers specified by EXP must be greater than Ø, but less than 2048.

It is possible, using the FORMAT and TYPE commands to produce a data tape on the high speed punch which then can be read in by the ASK command.

Suppose we wish to make a data tape of the numbers 1 to 500. The following command string would be used:

```
FORMAT L5ØØ\ TYPE % FSER (5ØØ) ; !
```

The FORMAT command insures that no CR's will be typed between numbers. The TYPE command types out the numbers, and terminates the tape with a CR.

It is useful to consider a more general form of the above. Suppose we wish to save a variable on paper tape. The following command string would be used:

```
FORMAT  LFDIM(VAR)\ TYPE % VAR ; !
```

It has been assumed above that we have saved a VAR of numerical mode. To save a VAR of char mode, we would have to type the string, enclose it within quotes, and terminate with a CR. The following would be used:

```
TYPE  %  FCHN (162) , VAR, FCHN (162) ; !
```

The FCHN function is necessary to type a quote because a "can't be part of a string. 162 is the decimal code of a quote.

21) SELECT ARG; ARG ... This command is used to select high or low speed input or output and to turn off and on the echo. ARG can be 6 things.

a)	% X	Select high speed for X
b)	% XX	Select high speed for X and X
c)	% XXX	Select high speed for X,X, and X
d)	X	Select low speed for X
e)	XX	Select low speed for X and X
f)	XXX	Select low speed for X, X, and X

X can be the following three things:

a)	I	Input. Keyboard or high speed reader
b)	O	Output. Teletype or high speed punch
c)	E	E - echo, % E - don't echo.

Some examples:

SELECT % IE	Read from HSR, don't echo.
SELECT IE; % O	Read from keyboard, echo on high speed punch.

If ARG, is terminated by a CR or "\", then by default low speed input, output, and echo are selected.

22) MODIFY EXP Edits line EXP (1). A CR is issued, and the command waits for the user to type the search character. The line is then typed out till the char is found. If not found, the line is not changed. Once the char is found the user has 4 options:

i) Type in as normal, i.e., use RUBOUT, "←", etc. Hitting a CR terminates the input, and terminates the MODIFY command.

ii) Hit a CNTRL/E. This will type out the rest of the line, save it in the newly modified line, and terminate the MODIFY command.

iii) Hit a FORMFEED (CNTRL/L). This will search for the next occurance of the search char.

iv) Hit a BELL (CNTRL/G). This will change the search char. The computer waits for a new search to be typed.

Note CNTRL/E, BELL, FORM FEED and the search char do not echo. The operation of the MODIFY command is very similiar to FOCAL's.

L O A D I N G V E C T O R - 8

There are 2 tapes supplied. One is the interpertor, the core resident segment. The second is the disk resident segment. The first thing to do is to load the BIN loader into field 1. This is because the interpreter uses locations 7600-7777 of field Ø.

Load tape 1, the core resident segment, into field Ø. Then place tape 2 in the high speed reader. Load address 2600_8 in field Ø. Depress START. This is a bootstrap routine which writes the tape onto the appropriate places on the disk. After tape 2 is loaded, load addr. 0200_8 in field Ø, and start. Vector - 8 will type out an error message, and a "*" to announce that it is ready for commands.

If a particular machine configuration doesn't have a high speed reader, make the following patch. It will cause tape 2 to be read in from the low speed reader.

LOCATION	FROM	TO
2605	7340	7300

D I S K S T R U C T U R E

Vector - 8 structures the DF32 in the following way:

Tracks Ø , 1	:	Disk extension of stack.
Tracks 2 , 3	:	Functions are stored here.
Tracks 4 , 5	:	Commands and other misc.
Tracks 6 , 7	:	Lib blocks 1,2,3,4.
Tracks 8 , 9	:	Lib blocks 5,6,7,8.
Tracks 10,11	:	Lib blocks 9,10,11,12.
Tracks 12,13	:	Lib blocks 13,14,15,16.
Tracks 14,15	:	Lib blocks 17,18,19,2Ø.

A map of tracks 2,3:

BLOCK					
	Ø)	SIN	14)	ADD	30) NE
	1)	COS	15)	SUB	31) ROT
	2)	TAN	16)	DIV	32) SER
	3)	ATN	17)	MAX	33) DIM + DELD
	4)	LOG	20)	MIN	34) REP
	5)	EXP	21)	AND	35) RAN
	6)	ABS + DELV	22)	IOR	36) NEW
	7)	SGN	23)	EG	37) FREE BLOCK
	10)	NOT	24)	GT	
	11)	INT	25)	LT	
	12)	FRC + SET SEG 2	26)	GE	
	13)	MLT	27)	LE	

A map of tracks 4,5:

BLOCK					
	0)	INPUT	14)	TRACE	30) FREE BLOCK
	1)	SET	15)	NEXT	31) FREE BLOCK
	2)	TYPE	16)	LOOP	32) FREE BLOCK
	3)	ERROR SEG 1	17)	DUMP	33) FREE BLOCK
	4)	DIRECTORY	20)	SELECT	34) FREE BLOCK
	5)	LIB COMMANDS	21)	MODIFY	35) FREE BLOCK
	6)	FLOAT I/O SEG 1	22)	LOG, EXP FOR ↑	36) FREE BLOCK
	7)	FLOAT I/O SEG 2	23)	ERROR SEG 2	37) FREE BLOCK
	10)	FORMAT	24)	ERROR SEG 3	
	11)	WRITE	25)	ERROR SEG 4	
	12)	DELETE	26)	ERROR SEG 5	
	13)	KILL	27)	ERROR SEG 6	

LOADING USER FUNCTIONS AND COMMANDS

First some comments: There are two ways user routines can be interfaced with Vector-8. One is to load into free blocks, the other is to replace already defined blocks. Care must be taken with the second method, for some blocks cannot be replaced. For example, in the function tracks (2,3), blocks 6, 12, and 33 cannot readily be replaced. Block 6 because it has in addition to ABS, the routine which deletes variables, which is necessary to SET. Block 12, because it has part of the SET command, and block 33 because it contains the routine to delete lines, which is needed if one wishes to be able to retype stored lines.

There is room to define two new functions, in blocks 36 and 37 respectively. For block 37, though, a name must be defined. This name is then entered in FUNLST (see listing FOIL.LST). If one wishes to replace a defined function such as FEQ, one simply replaces the slot FEQ holds in FUNLST with the new name and loads up the new function into the proper block on the disk.

In the command tracks (4,5) it is not recommended that commands be replaced, for there are 7 free blocks. One simply codes the command anme, only up to the first 3 letters, enters the name in NCORE, (see FOIL.LST), and its block number in PTAB1 (see FOIL. LST).

A command or function name is coded in base 4. An example:

$$
\begin{aligned}
\text{ABS} &= 4^2 \text{ X A} + 4 \text{ X B} + 1 \text{ X S} \\
&= 16 \text{ X } 1 + 4 \text{ X } 2 + 1 \text{ X } 19 \\
&= 16 + 8 + 19 \\
&= 24 + 19 \\
&= 43
\end{aligned}
$$

One uses the 6 bit code of the characters:

$$
\begin{aligned}
\text{SET} &= 16 \text{ X } 19 + 4 \text{ X } 5 + 20 \\
&= 304 + 20 + 20 \\
&= 344
\end{aligned}
$$

The user command or function must be assembled for locations 72ØØ - 7377. It is then loaded into those locations with the BIN loader. Then a patch should be toggled into locations 74ØØ-7412 to write the functionor command into the proper block of the disk. There are 2 patches, one for the function tracks, the other for the command tracks.

PATCH 1, for writing onto function tracks:

LOCATION	OCTAL	SYMBOLIC	
74ØØ	6ØØ1	ION	
74Ø1	76Ø4	LAS	/ Read in block n
74Ø2	4562	MULT	
74Ø3	Ø2ØØ	2ØØ	
74Ø4	3211	DCA .+5	
74Ø5	4532	WRITE	
74Ø6	76ØØ	-2ØØ	
74Ø7	7177	FNCBUF-1	
741Ø	Ø1ØØ	T2UØDØ	
7411	ØØØØ	Ø	
7412	74Ø2	HLT	

PATCH 2, for writing onto command tracks is the same except for location 741Ø which should be Ø2ØØ.

Example: Suppose I wish to define a function, using the name FNEW, which would read in N ADC values from channel Ø. The following would do it:

```
        *72ØØ
FNEW,   Ø
        TAD (2Ø2)               / Assume No Errors.
        DCA PNTRTS
        POPFT                   / Get # of Values.
        FLTTMP
        JMS FIXIDX              / FIX IT.
        DCA DIMRI               / SET DIM OF RESULT.
        TAD DIMRI
        CIA
        DCA FPCNTR              / SET CNTR.
        INIT                    / INIT TEMP. STACK.
TLOUP,  ADCV
        SKAD
        JMP .-1
        RADC
        FLOAT                   / FLOAT VALUE.
        PUSHFT                  / PUSH RESULT.
        FLTTMP
        IST FPCNTR              / DONE?
        JMP TLOUP               / NO.
        JMP I FNEW              / YES.
```

This patch would be assembled, with the appropriate definitions, for such symbols as FPCNTR, INIT, etc. exist in the Vector-8 source, but not in the patch source. The binary would be loaded into locations 72ØØ-7377. The patch would be toggled in. Location 74ØØ would be load addressed, 36, the block number of FNEW, would be put in the SR, and START would be depressed. The LAS instruction would read in the block number, and locations 72ØØ-7377 would be written onto block 36.

ERROR LIST

An error in Vector-8 is reported in the following format,

MESSAGE ; LINE XXXX COMMAND YYYY.

Where MESSAGE is the error message, XXXX is the line at which the error occured, and YYYY specifies which command in the string caused the error. So if an error occured while executing the third command in a command string, YYYY would be 3.

A list of the errors:

1) ILLEGAL COMMAND
2) RETURN WITHOUT GOSUB
3) BAD COMMAND TERMINATOR - A command was not terminated by a CR or " ".
4) LINE NOT FOUND
5) BAD ARG IN COMMAND
6) ILLEGAL LINE NO.
7) SYNTAX ERROR
8) INPUT BUFFER OVERFLOW - The input buffer has overflowed into text area. Either cut down size of text or cut down the input string.
9) INDEX = ZERO
10) INDEX TO LARGE
11) STACK OVERFLOW - An expression was too complex. Simplify.
12) STACK UNDERFLOW - This should not occur. If it does, there was probably a bomb out.
13) MODE ERROR WITH OPERATOR - An attempt was made to add char strings, or to do logical and with char strings, etc. In other words, an operator has bad arguments.
14) ILLEGAL CHARACTER
15) TEMP STACK OVERFLOW - Temporary stack has overflowed into variable storage. Either cut down VAR storage or simplify the expression.
16) ILLEGAL FUNCTION
17) MIS-MATCHED PARENS

18) EXPONENT OVERFLOW - A calculation caused a number to be too large.
19) EXPONENT UNDERFLOW - A calculation caused a number to be too small.
20) DIVISION BY Ø - This error will also occur when FTAN is called is COS(ARG)=Ø.
21) ERROR IN ↑
22) TOO MANY DIGITS IN NUMBER
23) NULL OR CHAR INDEX - An index is a null or is a char string.
24) DIM ERROR WITH OPERATOR - The dimensions of an argument of an operator are not right.
25) DISK I/O ERROR - Something is wrong with the disk.
26) MANUAL RESTART - Vector-8 has been restarted via a CNTRL/C or via the SR.
27) LOG OF ZERO - An attempt was made to take the log of a number $\leq$ Ø.
28) BAD ARG IN FUNCTION
29) MODE ERROR IN COMMAND - This will occur for example, when trying to assign a character string to particular elements of a numerical variable. There is an illegal mixing of modes.
30) VAR TOO LARGE - While defining a VAR, it overflowed into the Temp Stack. Simplify the expression or cut down the size of the VAR.
31) UNDEFINED VAR - A Var was referenced that was not defined.
32) NAME NOT FOUND IN LIB
33) ILLEGAL OP
34) DIRECTORY FULL

AN INTRODUCTION TO VECTOR-8
A NEW PROGRAMMING LANGUAGE

Richard Rothman
Groton School
Groton, Massachusetts

Vector-8, a new interpretive language, is designed to provide a PDP-8 family computer sporting 8K and a DF32 disk with power previously unavailable to such a mac' ne. The reasons for this are manifold. First, Vector-8 is not a language written for u basic 4K machine and slightly modified to accommodate extra hardware, but has rather been developed with the knowledge that 8K and a 32K disk, if properly utilized, can add considerable versatility to a language.

Secondly, Vector-8 applies a powerful concept to the manipulation of vectors. A vector is considered to be a unit rather than a collection of individual elements that have in common only the name of the vector. For example, in such languages as BASIC or FORTRAN, it is necessary, when applying an operation to an entire vector, or part of a vector, to do so element by element. In Vector-8 though, an operation is applied to an entire vector. There is no need to loop, and this removes much of the drudgery from programming. In addition, the ability to do in a single expression what would take several commands in other languages, allows for a flexible and powerful syntax. Vector-8, above all, provides one with the means of manipulating, with ease, in an on line environment, large amounts of data.

Vector-8 offers 15 operators (5 arithmetic, 6 relational, 3 logical, and cantenation), 32 functions, powerful commands, and a complete character string facility. User commands and functions can be easily interfaced to the Vector-8 interpreter, which is also equipped to work with high speed paper tape, if present. Now for a more detailed explanation of the language.

A vector can be specified in four ways:

1. A number string
2. A character string
3. By creating a variable
4. By cantenating together separate vectors specified as per 1, 2, 3.

1. A number string consists of a series of numbers separated by spaces. The dimension of the vector specified is the number of numbers. Some examples:

> 1 2 3 4 specifies a vector of dimension whose first element is 1, whose second element is 2, whose third element is 3, and whose fourth element is 4.
>
> 5 4 1 specifies a vector of dimension 3 whose first element is 5, whose second element is 4, and whose third element is 1.

2. A character string consists of a series of characters contained within quotes. Carriage returns may be imbedded by typing line feeds. Line feeds will only be recognized within quotes. The dimension of the character string is equal to the number of characters in the string. Some examples:

"ABCD" specifies a vector of dimension 4 whose first element is "A," whose second element is "B," whose third element is "C," and whose fourth element is "D."

"CDX" specifies a vector of dimension 3 whose first element if "C," whose second element is "D," and whose third element is "X."

3. A variable is a symbol representing a vector. It is specified via a SET command. Some examples:

SET A=1 2 3 4 specifies A as a vector of 4 whose first element is 1, whose second element is 2, whose third element is 3, and whose fourth element is 4.

A variable consists of at most two recognized characters. The first must be a letter, but not an "F." The second can be any letter or digit. Any subsequent letters or digits are ignored.

4. By cantenating together specified vectors, new vectors can be specified. Some examples:

1 2 3, 3 2 1 ==> [1] 1 2 3 3 2 1

"ABC","DEF" ==> "ABCDEF"

SET A=1 2
A, 1 2 3 4 5 ==> 1 2 1 2 3 4 5

All the vectors specified above have been of dimension 1 or greater. It is also possible to specify a vector of dimension 0. For example:

" " specifies a null character string

The ways of specifying null number strings will be shown later.

Specified vectors may be indexed by following the vector with an expression enclosed in parenthesis. Some examples:

1 2 5 7 (1 3 4) > 1 5 7

1 4 8 9 [1] > 1

"ABCDEF" (6 3 2) > "ECB"

SET A=1 2 3

A<2 3> > 2 3

Indexed vectors may themselves be indexed. Some examples:

1 46 8 (1 3 2) [1 2] > 1 6

"ABCDEFG"(1 3 2) <1 2> ==> "AC"

Indexing may go to any depth, and indices do not have to be integers, for they are truncated anyway.

1. reads "produces"

Vector-8, as mentioned before, has 15 operators, with which vectors are combined in various ways. All operators, with the exception of cantenation, can have their arguments in three forms. In the ensuing discussion, let :o: represent any operation except for cantenation. Let also d_n stand for a vector of dimension n. The three forms are:

1) d_1 :o: $d'_{n>1}$

2) $d_{n>1}$:o: d'_1

3) $d_{n>0}$:o: $d_{n>0}$

In case 1, the one element on the left is applied to the n elements on the right, to produce, in most cases, a result of dimension n. This process can be described by:

for i=1,n apply d_1 to d'_i and set the result r_i

In case 2, the n elements on the left are applied to the one element on the right, to produce, in most cases, a result dimension n. This process can be described by:

for i=1,n apply d_i to d'_1 and set the result r_i

In case 3, each element on the left is applied to its corresponding element on the right. This process can be described by:

for i=1,n apply d_i to d'_i and set the result r_i

It should be noted that the :o: operations can not be used with null arguments.

Cantenation takes the following form:

$d_{n\geq 0}$, $d_{n\geq 0}$

The following are the Vector-8 operations:

	symbol	priority	description
1) Arithmetic:	+	3	addition
	-	3	subtraction (or unary minus)
	*	4	multiplication
	/	4	division
	↑	5	exponentiation. a^b computed $e^{b \ln a}$.

	symbol	priority	description
2) Relational:	:EQ:	2	relational =
	:GT:	2	relational >
	:LT:	2	relational <
	:GE:	2	relational $\geq$
	:LE:	2	relational $\leq$
	:NE:	2	relational $\neq$
3) Logical:	&	2	logical and ($\wedge$)
	!	2	inclusive or ($\vee$)
	#	2	compression
4) Cantenation:	,	1	cantenation

Note: The higher the priority number, the higher is the priority of the operation. So in an expression, exponentiation is performed first, followed by multiplication and division, addition and subiraction, the relational and logical operators, and finally cantenation.

Group 1, the arithmetic group, does not operate on character strings. A few examples:

1 + 1 2 3 ==> 2 3 4

1 2 3*2 ==> 2 4 6

4/2 ==> 2

1 2 3+4 5 6 ==> 5 7 9

The second group, the relational group, compares the two arguments in the same manner as :o:. If the condition is true, a 1 is placed in the result. If the condition is false, a 0 is placed in the result. Thus, the two logical quantities, true and false, are defined to be 1 and 0 respectively.

Character strings may be compared with each other, but not with number strings. The two modes cannot be mixed. When character strings are used with the relational operation, their character codes are compared, thus, "A" is less than "B." Some examples:

1:EQ:1 2 3 ==> 1 0 0

1 2 3:GT:4 1 2 ==> 0 1 1

"ABC":NE:"DEC" ==> 1 1 0

The logical operators must have for arguments only 1's and 0's. Some examples:

1 1 0&0 1 0 ==> 0 1 0

1 0 1 0 ==> 1 1 1

The operator # has logical quantities for its left argument, and a number or character string for its right. It functions in the following way: Where there is a 1 in the left arg, the corresponding element in the right arg is retained in the result. Where there is a 0, the corresponding element is not retained in the result. Some examples:

1#1 0 5 8 ==> 1 0 5 8

1 0 1 1 1#8 ==> 8 8 8 8

1 0 1#1 2 3 ==> 1 3

1 2 3:EQ:1 2 4#"ABC" ==> "AB"

0#1 ==> null vector of numerical mode

SET A=2 2 3 2 4 5 6 2 2

2:EQ:A#A ==> 2 2 2 2 2

2:EQ:A#1 2 3 4 5 6 7 8 9 ==> 1 2 4 8 9

These are the Vector-8 operators. Vector-8 also offers 32 functions. A function is represented by an "F" followed by the name. A list follows:

Numerical Functions:

FSIN($d_{n>0}$)	sine. argument in radians.
FCOS($d_{n>0}$)	cosine. argument in radians.
FTAN($d_{n>0}$)	tangent. argument in radians.
FATN($d_{n>0}$)	arc-tangent.
FLOG($d_{n>0}$)	natural logarithm.
FEXP($d_{n>0}$)	exponential. e^x
FABS($d_{n>0}$)	absolute value.
FSGN($d_{n>0}$)	sign function.
FNOT($d_{n>0}$)	logical not function. Argument must contain only 1's and 0's.
FINT($d_{n>0}$)	integer function.
FFRC($d_{n>0}$)	return fractional part of argument.

Reductive Functions:

FMLT($d_{n>1}$)	reductive multiplication.
FADD($d_{n>1}$)	reductive addition.
FSUB($d_{n>1}$)	reductive subtraction.
FDIV($d_{n>1}$)	reductive division.
FMAX($d_{n>1}$)	returns largest number.
FMIN($d_{n>1}$)	returns smallest number.
FAND($d_{n>1}$)	reductive logical and ($\wedge$).
FIOR($d_{n>1}$)	reductive inclusive or ($\vee$).
FEQ($d_{n>1}$)	reductive :EQ: .
FGT($d_{n>1}$)	reductive :GT: .
FLT($d_{n>1}$)	reductive :LT: .
FGE($d_{n>1}$)	reductive :GE: .
FLE($d_{n>1}$)	reductive :LE: .
FNE($d_{n>1}$)	reductive :NE: .

Other Functions:

FCHN($d_{n>0}$)	change mode of argument.
FROT($d_{n>2}$)	rotates the argument.
FSER($d_{1\leq n\leq 3}$)	generates a series.
FREP($d_{n>0}$)	repeats the argument.
FDIM($d_{n\geq 0}$)	returns dimension of argument.
FRAN($d_{n>0}$)	generates n random numbers.
FNEW( )	user defined function.

Numerical functions have for arguments vectors whose dimensions are greater than 0. The reason being that it doesn't make sense to, for instance, take the sine of a null vector. Each function applies its particular operation to each element of its vector argument, and places in the element operated on the result of the operation. So for example:

FNOT(1 0 0 1 1) ==> 0 1 1 0 0

FSIN(1 2 3) ==> sin 1 sin 2 sin 3

Each reduction function reduces its vector argument, whose dimension must be 2 or greater, to a single number by applying its operation to the elements of the vector. For example, FMLT applies its operation, multiplication, to produce a result which is equal to all the elements of the argument multiplied together. This process can be visualized by the following: Let $a_1, a_2, a_3, a_4, \ldots a_n$ be the arguments of the reductive function whose operation is :q:. Then,

a_1 :q: a_2 :q: a_3 :q: a_4 :q: . . . a_n ==> result.

The other functions must be gone into further:

1. FCHN changes the mode of the argument. If the argument is a character string, then the result is a number string whose elements are the codes of the characters. If the argument is a number string, then the result is a character string.

Some examples:

FCHN("ABC)" ==> 193 194 195

FCHN(193 194 195) ==> "ABC"

This function is designed so that functions such as FROT and FREP can be used with character strings.

2. FROT rotates a vector. The last element of the vector argument is used to indicate the direction and number of times that the vector is to be rotated. Call the last element N.

If N=0 the vector is not rotated.

If N>0 the vector is rotated N times right.

If N<0 the vector is rotated -N times left.

Because it is not reasonable to rotate a vector of dimension 1, the dimension of FROT's argument must be 3 or greater. (N plus the vector to rotate.) Some examples:

FROT(1 2 3 1) >312 -rotate 1 2 3 once right.

FCHN(FROT(FCHN("ABC"),1)) > "CAB" - change "ABC." rotate once right. change back.

3. FREP repeats a vector. The last element of the argument, as in FROT, is used to indicate how many times.

If N=0 a null vector is returned.

If N>0 the vector is repeated N times.

If N<0 an error results.

If the dimension of the argument is 1, then a null vector is returned.

Some examples:

FREP(1) ⟹ null vector of numerical mode.

FREP(1 4) ⟹ 1 1 1 1 -repeat 1 four times.

FREP(1 2 2) ⟹ 1 2 1 2 -repeat 1 2 two times.

4. FDIM returns the dimension of the argument. Some examples:

FDIM(1 2) ⟹ 2

FDIM(FREP(1 100)) ⟹ 100

FDIM(" ") ⟹ 0

FDIM(0#4) ⟹ 0

FDIM("ABCDEFR") ⟹ 7

5. FSER takes three forms. One with the argument of dimension 1, another with the argument of dimension 2, and the third with the argument of dimension 3.

a. FSER(d_1). Let N be the one element. If N≤0 an error results. Otherwise the series 1, 2, 3, 4, . . . N is generated. Some examples:

FSER(1) ⟹ null vector of numerical mode.

FSER(5) ⟹ 1 2 3 4 5

FSER(10) ⟹ 1 2 3 4 5 6 7 8 9 10

b. FSER(d_2). Let M be the first, and N be the second element. A series is generated, in increments of 1, from M to N. If N<M then a null vector is returned. Some examples:

FSER(1 5) ⟹ 1 2 3 4 5

FSER(3 6) ⟹ 3 4 5 6

FSER(2 1) ⟹ null vector

FSER(1.2 4.2) ⟹ 1.2 2.2 3.2 4.2

FSER(2 2) ⟹ 2

c. FSER(d_3). Let N, M, P be the first, second and third elements respectively. A series is produced from M to P in increments of N. If M>P then a null vector is returned. If N=0 then an error results. If P>M and N<0 then an error results, for it would be impossible for M to reach P with a negative increment. Some examples:

FSER(1 3 12) ==> 1 4 7 10

FSER(1,(-2),10) ==> error

FSER(1 0 5) ==> error

FSER(.2 .01 .3) ==> .20 .21 .22 .23 .24 .25 .26 .27 .28 .29 .30

d. FRAN generates as many random numbers as the dimension of the argument. A null argument is not legal. Some examples:

FRAN(FSER(2)) ==> 2 random numbers.

FRAN(FSER(1000)) ==> 1000 random numbers.

We have now seen the functions and operators that make up a Vector-8 expression. Now for the commands that make up a Vector-8 program.

1. TYPE EXP;!;#; ↑ ;% (EXP stands for a Vector-8 expression)

!	CR-LF
#	CR
↑	LF
%	Changes output to the high speed punch (HSP) till the TYPE command is terminated

If EXP is a character string, then the characters are outputted. If EXP is a number string, then it is outputted according to the parameters specified in the FORMAT command.

2. SET VAR=EXP -create a new variable and set to EXP

   SET VAR(INDEX)= EXP -replace certain elements of VAR with EXP.

3. ASK VAR -create a new variable, VAR. Get values from the TTY.

   ASK VAR(INDEX) -replace certain elements of VAR. Get values from the TTY.

   ASK %VAR -create a new var, VAR. Get values from the HSR.

   ASK % VAR (INDEX) -replace certain elements of VAR. Get values from the HSR.

Note: HSR stands for High Speed Reader.

4. GOTO EXP — -EXP must result in a number string. Control is transferred to line EXP(1).

5. GOSUB EXP — -EXP must result in a numbering string. Control is transferred to a subroutine at line EXP(1).

6. RETURN — -This command is used to exit a subroutine. Control is transferred to the line following the GOSUB that called the subroutine.

7. CALL PNAME — -The program whose name is PNAME is called down from the disk and executed. When the program is done, control is returned to the line following the CALL command in the original program.

8. WHEN EXP; COMMAND — -If the result of EXP equals 1, the COMMAND is executed.

-If the result of EXP equals 0, control is transferred to the next line

-If the result of EXP does not equal 1 or 0, then an error results.

9. REM — -This command transfers control immediately to the next line and ignores any argument. It is the comment command.

10. RUN — -This command runs the program in core.

11. DELETE EXP — -EXP must result in a number string. The lines specified by the elements of the number string are deleted.

12. KILL VARNAM — -The variable, VARNAM, is deleted.

13. EDIT EXP — -EXP must result in a number string. The line EXP(1) is edited in a manner similar to FOCAL's MODIFY command.

14. WRITE EXP — -EXP must result in a number string. The lines specified by the result are listed on the teletype. If EXP results in a null vector, the entire program is listed.

WRITE % EXP — -Same as above, except that the listings are produced on the HSP.

15. DUMP — -This command dumps the symbol table in the following format:

NAME DIMENSION MODE (char or number string)

16. SELECT I;O;%O;%1 — -This command selects I/O devices.

I - low speed input (teletype)

O - low speed output (teletype)

%I - high speed input (HSR)

%O - high speed output (HSR)

17. FORMAT EXP -EXP must result in a number string of dimension 3. This command supplies format parameters concerning the output of numerical vectors to the TYPE command. Let e_1, e_2, e_3, be the three elements:

e_1 specifies the number of spaces between elements.

e_2 specifies the number of elements per line.

e_3 specifies the number of CR-LF's between lines.

This command comes in useful when, for instance, a vector of dimension 200 is being outputted. That many elements cannot possibly fit on a line, so the option is left open to the user as to formatting the output.

18. LIB S PNAME -Saves program in core in the library under the name PNAME.

LIB D PNAME - Deletes the program in the library whose name is PNAME.

These are the Vector-8 commands as they are currently defined.

LIB C PNAME -Calls the program whose name is PNAME from the library into core.

LIB L -Produces a listing of all current entries in the library.

19. LOAD ONAM; NNAM -A command or function of the name ONAM is replaced by NNAM, and the binary for the new command or function is read in from the low speed reader and written up onto the appropriate block on the disk.

LOAD ONAM; NNAM; -Same as above, except that the binary is read in from the HSR.

20 QUIT -Program execution is halted, and a return to direct mode is made.

These are the Vector-8 commands as they are currently defined. Programs are created in much the same way that they are in FOCAL. If a line of inputted text begins with a line number, the text is stored away for later reference. If the line doesn't begin with a line number, the text is assumed to be a command, and is executed in direct mode. Thus, Vector-8 can operate as a calculator, as well as execute stored programs.

A line number is a whole number from 0 to 2047 inclusive. Line number 0 may not be deleted, written, or operated on in any way except by a GOTO command. A GOTO line number 0 causes the program to halt, just as if a QUIT command had been executed. This is in effect a computed halt.

Vector-8 operates with 7 significant digits. Floating point output is formatted, but the format cannot be set by the user, for there is not enough core for the lengthy routine that would make this possible.

This article should serve as an introduction to Vector-8.

Correction Notice

Blocks 30 and 31 of the command tracks are occupied by the PUNCH and READ commands.