CAL REFERENCE MANUAL

Butler W. Lampson

University of California, Berkeley

Document No. R - 23
Issued August 10, 1967

Contract SD-185
Office of Secretary of Defense
Advanced Research Projects Agency
Washington 25, D. C.

Sept. 21 '67

Note: ① AHI version of CAL program released this Spring.

② Not expected to change in new 1.9 TSS

③ This document is merely a better description of the current CA

This revision of the CAL Manual obsoletes documents W-13 and W-21. It describes CAL as it exists as of its date, with the exception of the DRUM feature, which is not implemented.

TABLE OF CONTENTS

1.0 Introduction 1-1

2.0 Input to CAL 2-1

2.1 Editing Input 2-2

3.0 Numbers, Variables and Expressions 3-1

4.0 Steps. 4-1

4.1 Direct-Only Steps. 4-2

4.2 Indirect-Only Steps. 4-3

5.0 Modifiers. 5-1

6.0 Forms. 6-1

6.1 Type in Form 6-1

6.2 Demand in Form 6-2

7.0 Functions. 7-1

8.0 Input-Output 8-1

9.0 Display Output 9-1

10.0 Running the Program. 10-1

CAL Summary . . . . . A

1.0 Introduction

This manual describes the CAL language and the operating features of the CAL system. It is not a primer but a reference manual. Anyone who is not an experienced programmer will have a great deal of trouble learning CAL from it.

CAL is a conversational algebraic language based on the Joss system developed by Shaw at the Rand Corporation. It is designed to facilitate the computer solution of small and medium-sized numerical problems. Since it is very much slower in execution than, say, the code produced by a Fortran compiler, it is not suitable for problems requiring a large amount of computation. It does provide nearly fool-proof operation, complete error-checking and a very powerful and convenient language.

To enter CAL, give its name to the executive:

```
@ CAL.
```

If nothing is typed within two seconds CAL will assume that you want a program of about 50 steps and no page headings. It will type out

```
>
```

which is an indication that it is expecting input.

If the assumption stated above is wrong, type a carriage return within two seconds after entering CAL. It will respond with

```
NUMBER OF STEPS NEEDED =
```

Give an estimate, followed by a carriage return. The limit is about 320. It will then request a heading:

```
HEADING, PLEASE
```

If you do not want a heading, type a carriage return. Otherwise, type the desired heading, which ends with a carriage return. This heading together with a page number will be supplied at the top of each page during the following session. A page contains 55 lines of output, together with enough separation to use 11 inches of paper.

Typing enough rubouts (usually 2 or 3) will get you from CAL back into the executive. To re-enter CAL, do

@ CONTINUE CAL.

All files will be closed.

2.0 Input to CAL

CAL accepts two kinds of input: steps and numbers. Whenever the user's program is not actually computing or typing out, CAL is waiting for one of these kinds of input. A DEMAND step in the program will cause CAL to type out the variable demanded and wait for a number to assign to it. At all other times input to CAL should be steps.

Whenever CAL is accepting a step, it will recognize the characters A^c (control A, obtained by depressing the A key and the CONTROL key simultaneously), W^c and Q^c as editing characters. A^c causes CAL to type ↑ and forget the immediately preceding character. It may be repeated to delete several characters. W^c prints \ and causes the last word typed to be deleted. More precisely, all immediately preceding blanks are deleted and then all characters up to but not including the next preceding blank. Q^c prints ← and causes the entire line typed to be deleted. That is, it deletes all characters up to the nearest preceding LINE FEED, or up to the beginning of the line. An immediately preceding line feed is also deleted. CAL accepts a number of other control characters. Their function is described at the end of this section.

Line feed is the continuation character. It causes CAL to generate a carriage return and is otherwise completely ignored, except by the Q^c operation.

A CAL program is composed of steps. Each step in the program begins with a step number, which has the form <integer>. <integer>. The decimal integer formed by the digits preceding the dot is called the part number of the step. All steps with a given number belong to that part. A linear ordering of step numbers is defined by taking them as ordinary decimal numbers. Thus

$$1 < 1.1 = 1.10 < 1.101 < 1.2 = 2.0 < 10.0$$

The ordering of steps in the program is determined by their step numbers and not by the order in which they are input. If

several steps with the same number are input, the last one will be kept and the earlier ones thrown away. There is a limit of six digits on each of the integers in a step number.

If a step is input without a step number, it will not become part of the program, but instead will be executed immediately. Such a step is called direct; steps with numbers are indirect.

Each step ends with a carriage return. Blanks are ignored, except that they may serve to delimit words and variables. Line feed continues the step onto the next line. A step may not have more than 300 characters. A ; followed by any string of characters as a comment may be appended to any step.

Rubout may be used at any time during input of steps to abort the current step. It is essentially equivalent to Y^C below.

2.1 Editing Input

This section may be skipped on an initial reading. It describes some very useful features of CAL which are unfortunately rather difficult to get the hang of.

With one exception, every input of a step into CAL may be thought of as an editing process performed on the line last typed in. Ordinary characters typed during this editing simply replace the corresponding characters in the old step, and carriage return causes the remainder of the old step to be thrown away. The user may therefore ignore the existence of the editing operation if he wishes. There are, however, a number of control characters which direct the operation of the edit. It is helpful to know about them when the step about to be entered is similar to the one last typed.

C^c copies the next character of the old line to the new one, and types out that character. C^c is especially useful in conjunction with the repeat button.

S^c skips the next character of the old line and types out %.

$Z^c\underline{C}$ copies the old line up to the next occurrence of the character $\underline{C}$. The character $\underline{C}$ is not typed until this occurrence is reached, so that the operation is exactly equivalent to a number of C^c's. If $\underline{C}$ does not appear in the line, CAL rings the bell and takes no other action.

$X^c\underline{C}$ is the same as Z^c except that it deletes the characters passed over and types %. $\underline{C}$ is never typed.

R^c (retype). CAL types line feed, then the rest of the original line, then on the next line the edited line so far. Editing can be resumed. This character is intended to permit recovery in cases where the user has become confused about the state of the edit.

T^c is the same as R^c except that the new line is correctly aligned with the remainder of the old one. It takes longer.

Y^c copies the remainder of the old line to the new one without typing it. The new line then becomes the old line for a continuation of the edit.

D^c causes the remainder of the original line to be typed out and copied to the edited line, which then replaces the original line.

F^c is the same as D^c except that the rest of the line is not typed. Among other things, it is useful for executing the same direct statement more than once.

E^c types < and switches CAL so that text typed in is inserted in the new line without replacing any characters of the old line. Another use of E^c causes > to be typed and restores replace mode.

The EDIT operation causes a specified step to be typed. This step then becomes the old line for the edit operation.

The step is not deleted or altered. If the step resulting from the edit has the same number as the original step, of course, the latter will be replaced. The following variations are possible:

```
> EDIT STEP n.n                STEP may be omitted
> EDIT FORM expression
> EDIT variable       (functions only)
```

The MODIFY operation is identical to EDIT except that the step being edited is not typed out. It may be abbreviated MOD.

3.0 Numbers, Variables and Expressions

All numbers in CAL may be thought of as being expressed in scientific notation. There are eight significant digits, and the exponent may be in the range -77 to +77. However, integers or decimals may be input and numbers will be output as integers if they have no fractional part and are smaller than 2^{23}.

There are 11*26=286 distinct named variables in CAL. A variable is named by a single letter or by a letter followed by a digit. A variable may be subscripted. Examples of variables and subscripts:

A4 A(1) S(23,65,-147.3) M3(A(1,3), B(CO(D8))*13)

A subscript may be any arbitrary expression. It is truncated to the nearest smaller integer before use, and this integer is taken modulo 2^{23}. A variable may have any number of subscripts, and is not required to have the same number each time it is referenced. A refers to a different number than A(0) or A(1).

Expressions are formed by combining operands with operators. The available operators are:

←	(replacement)
AND OR	(same as * and + except for precedence)
NOT	(changes 0 to 1, any non-zero operand to 0)
= # << = >>=	(relations)
+ -	
* / MOD	
↑	(exponentiation)

The precedence is as indicated. Thus

A AND B=C+D*E↑F ≡ A AND (B=(C+(D*(E↑F))))

The replacement operator takes the largest expression it can find on the right and sets the value of the variable on the left accordingly. The relations yield the numbers 1 or 0

depending on whether the indicated relation holds between the operands or not. The expression A MOD B is equivalent to FP(A/B)*B

Operands may be

(a) numbers, with or without decimal points. An exponent may be indicated by E followed by +, - or blank and the appropriate power of 10.

(b) variables, possibly subscripted

(c) the special functions:

ABS	
SIN	the trigonometric functions
COS	take their arguments
TAN	in radians or return
ATAN (X,Y)	the result in radians
ATAN (Z)	equivalent to ATAN (1,Z)
EXP	
LOG	
LOG10	
SQRT	
IP	(integer part, the largest integer <X. I.e. IP(-1.5)=2)
FP	(fractional part, always positive. IP(X)+FP(X)=X)

The argument of a function should follow it and should be enclosed in parentheses unless it is an expression with no operators of precedence lower than *. Thus SIN(X+1), but SIN 2*X+Z is legal and means SIN(2*X)+Z.

(d) The iterative functions:

SUM

PROD

MAX

MIN

These are used as indicated in the example:

SUM(I=1 BY 1 TO N: I↑2) ≡ $\sum_{i=1}^{n} i^2$

The part of the argument before the parentheses is called the for clause. It is described in detail in connection with FOR modifiers (Section 5). The value of the controlled variable

(I in the example) is not altered by the iterative function; whatever value it had when the function was entered will be restored when the function is complete. Within the function, of course, its value is determined by the FOR clause.

(e) A CAL function call, of the form f[a,b+c,d(1)]. These functions are discussed in detail in Section 7.

(f) $, which has the value of the current line on the page.

(g) PI, which has the value one might expect.

Conditional expressions are also possible, and take the general form

IF e_{p1} THEN e_{r1} ELSE IF e_{p2} THEN e_{v2} ... ELSE e_{vn}

When a conditional expression is evaluated, e_{p1} is evaluated. If it is non-zero, e_{v1} is evaluated, and its value becomes the value of the expression. If e_{p1} is zero, e_{v1} is ignored and e_{p2} is evaluated. If the expression ends with ELSE e_{vn}, as indicated, then the value of e_{vn} will be the value of the expression if all the e_{pi} are 0. The expression may, however, end with

IF e_{pn} THEN e_{vn}

In this case the value of the expression is 0 if all the e_{pi} are 0.

Note that the only expressions actually evaluated are those whose values are required in determining the value of the expression. For example, the expression

IF X MOD 2=0 THEN -1 ELSE IF X>10 THEN 0 ELSE 1

has the value -1 if X is an even integer, 0 if X is an even integer and is greater than 10, and 1 otherwise.

A WHERE modifier can be appended to any expression. The form is

expression WHERE variable = expression & variable = expression

The modifier is evaluated before the expression, and its effect is to set the value of each variable to the value of the corresponding expression. Thus the expression

```
SIN(Z)/COS(Z) WHERE Z=ATAN(X,Y)
```

has the value X/Y. The expression

```
1+SUM (I=1 BY 1 WHILE T>10E-8: T WHERE
T=X↑I/F WHERE F=I*F) WHERE T=1 & F=1
```

computes e^x with accuracy better than 10^{-8}.

4.0 Steps

The following steps may be direct or indirect. Each one begins with a unique word. The variable mentioned may be subscripted.

SET variable = expression

sets the value of the variable to that of the expression The SET may be omitted.

TYPE expression, expression ...

causes the expressions to be typed out, followed by their values. Each expression is typed on a separate line. Subscripts will be replaced by their actual values. If a single unsubscripted variable name appears, any subscripted elements it may have will also be printed.

TYPE IN FORM EXPRESSION: expression, expression ...

is discussed in Section 6.

TYPE STEP step no, step no, ...
TYPE PART expression
TYPE FORM expression
TYPE "string"
TYPE ALL STEPS
TYPE ALL FORMS
TYPE ALL VALUES
TYPE ALL FUNCTIONS
TYPE ALL

do the obvious things. The "string" may contain line feeds.

TO STEP step number (STEP may be omitted)
TO PART expression

Execution of steps continues with the one specified. The first step in the program with step number $\geq$ the expression is used.

DO STEP step number (STEP may be omitted)
DO PART expression

The specified step or part is executed, and control then goes to the step following the DO. It is not legal to DO a TO step. A part being DOne may have TO steps, however. The DO is completed when the last step of any part is executed or

when a DONE step is executed. It is not completed by a transfer out of the part being DOne. The DO PART operation starts at the first step with number $\geq$ the value of the expression.

DEMAND variable, variable, ...

Each variable is typed out, and CAL awaits input of a number, which will be used to set the value of the variable. Any non-numeric characters typed before the number will be ignored. During the typing of a number, the character Q^C deletes all characters typed so far and allows the number to be retyped. The character following the number must be carriage return, space, comma or semi-colon. If it is anything else, the number is ignored and may be retyped.

DEMAND IN FORM expression: variable, variable ...

is discussed in Section 6.

PAGE

spaces to the top of the next page. This works even if there is no heading and CAL is not separating pages.

LINE

spaces one line

OPEN, CLOSE, INPUT, OUTPUT, READ, WRITE, and CALL are discussed in Section 8 on input/output.

4.1 Direct-Only Steps

The following steps may be used directly only:

```
DELETE STEP  step no, step no ...  (STEP may be omitted)
DELETE PART  expression
DELETE FORM  expression
DELETE  variable     DELETE ALL STEPS
DELETE ALL VALUES    DELETE ALL FORMS
DELETE ALL           DELETE ALL FUNCTIONS
```

The specified object is deleted. If it is a variable, all subscripted occurrences of the variables are deleted.

DELETE ALL starts CAL over at the very beginning. It is equivalent to leaving CAL and re-entering it from the executive with a new @CAL command.

CANCEL

Kills off the information about the current state of the running program. See Section 10.

GO

Continues execution after an interrupt or execution error. See Section 10.

DUMP

TO file.

causes all steps and forms to be written on the specified file. The format is such that the file can be listed or read in by LOAD.

LOAD

FROM file.

reads in the contents of the specified file, treating it as though it were being typed in from the teletype.

DEFINE variable [argument list] = expression
: statement

This step defines a CAL function. It is discussed in Section 7.

DRUM variable

causes the values of the indicated variable to be stored on the file which has been opened as DRUM. See Section 8. The variable may thereafter be referenced only with exactly one subscript. It may be deleted, which cancels the effect. Access to the variable may be slowed down.

STEP

causes the next step of the program to be executed. Control then returns to CAL as though a PAUSE had been executed.

4.2 Indirect-Only Steps

The following steps can be used indirectly only:

PAUSE

causes a message to be typed out and execution to stop. The program can be restarted at the next step with GO.

DONE

If a DO PART is in force, it is terminated. Otherwise the step is ignored.

RETURN expression

The expression is returned as the value of the function most recently called. If any DO or FOR started inside the function is not complete, the step is an error.

5.0 Modifiers

Any step which can be used indirectly may be followed by any number of modifiers which will govern its execution. A modifier may be preceded by a comma. The available modifiers are

IF expression

which allows the preceding step to be executed if the expression is non-zero.

UNLESS expression

which allows the preceding step to be executed if the expression is zero.

WHILE expression

which causes the preceding step to be executed repeatedly as long as the expression is non-zero.

UNTIL expression

which causes the preceding step to be executed repeatedly as long as the expression is zero.

FOR for clause

which causes the preceding step to be executed repeatedly under control of the for clause; iterative functions also contain for clauses. The form is variable = for clause section, for clause section, ... for clause section. The value of the variable is set by each for clause section and the step executed. If the variable is subscripted, the subscript is evaluated each time. For clause sections can have the form:

(a) a single expression

(b) $expression_1$ BY $expression_2$ TO $expression_3$ which sets the variable to the value of the first expression and then adds the second expression on each repetition until it passes the third expression; if the second expression is positive, repetition continues until the variable becomes larger, if it

is negative until the variable becomes smaller. If the BY clause is omitted, it is taken to be 1. Examples

1 BY 1 TO 10; 1 TO 10; 100 BY -3 TO 50

(c) expression_1 BY expression_2 $\left\{\begin{matrix}\text{WHILE}\\ \text{UNTIL}\end{matrix}\right\}$ expression_3

is exactly like (b) except that repetition continues under control of the WHILE or UNTIL, which work exactly like the modifiers described above.

The following are equivalent:

FOR X=1 TO 5

FOR X=1,2,3,4,5

FOR X=1 BY 1 WHILE X^2 <101

A useful device for initializing a vector to an arbitrary collection of values is illustrated by the following program:

1.1 SET I=I + 1 FOR X(I)=23,46,-16.5,3.14159, WHERE I=1

This also illustrates the WHERE modifier for steps; its works exactly like the modifier for expressions, described in Section 4. It is for this reason that the above step contains the comma before the WHERE; otherwise it would attach itself to the last expression of the for clause and would not be evaluated until that expression was reached. It would then not serve its intended purpose of setting I to 1 initially.

Note: WHILE, UNTIL and FOR modifiers may not be used on a TO step, for more or less obvious reasons.

6.0 Forms

6.1 Type in Form

Formatted output can be done with the TYPE IN FORM statement--the form specifies the format. The method is illustrated by the example

```
FORM 1:
SCIENTIFIC NOTATION ########     DECIMAL NOTATION %%%.%%%%
TYPE IN FORM 1: 3.1414, -10.1E-1
SCIENTIFIC NOTATION 3.14 00      DECIMAL NOTATION -1.0100
```

The characters in the form are typed literally except for fields containing #'s, which cause numbers to be printed in scientific notation, and fields containing %'s and zero or one dots, which cause numbers to be printed as decimals. The character & is not typed. If either field is too small, an error will be indicated. Note that in a # field six characters are needed for the decimal point, the exponent, the sign and at least one digit of the number. In a % field one character is needed before the decimal point for the sign. If the number is known to be positive, the position for the sign is not needed.

After the last number in the TYPE IN FORM statement has been printed characters continue to be printed from the form until another numeric field or the end of the form is reached. If there are more numbers in the TYPE statement than fields in the form, the form is reused as often as necessary. If the last character of a form is processed according to the above rule, a carriage return is generated, otherwise not. Several different TYPE IN FORM steps can therefore put information on the same line. If the last character of the form is &, the usual carriage return is suppressed. A carriage return is generated before CAL types its > if this has not already been done.

Example:

FORM 1:

%%%%%% %%%%%% %%%%%% %%%%%% %%%%%% %%%%%%

TYPE IN FORM 1: I↑2 FOR I = 1 TO 14 will result in

1 4 9 16 25 36

49 64 81 100 121 144

169 196

Note: The colon of a FORM must be followed by a line feed.

6.2 Demand in Form

The step

DEMAND IN FORM expression: variable, variable, ...

can be used instead of a simple demand. The characters of the form will be typed literally with the exception of # or % characters. Any contiguous group of these will cause a number to be read and assigned to one of the variables in the list. The length of the group has nothing to do with the length of the number input. This process proceeds, just like a TYPE IN FORM, until the list of variables is exhausted.

The program:

```
1.1 DEMAND IN FORM 1: N1,N2
1.2 TYPE IN FORM 2: N1+N2, N1*N2
FORM 1:
NO. 1 = #; NO. 2 = #
FORM 2:
SUM = %%%.%%; PRODUCT = #########
```

might produce a page like this:

```
>DO PART 1
NO. 1=12.6; NO. 2=40
SUM=52.60; PRODUCT = 5.040 02
```

7.0 Functions

A CAL function may be defined by the statement

DEFINE F[X,Y,Z,W] = (X↑W+Y↑W+Z↑W)↑(1/W)

This statement may be direct only. It deletes any old value which F may have and assigns the function definitions as the value of F. X,Y,Z and W are local variables of the function. This means that when the function is called, the values of these variables are saved and new values obtained from the arguments provided with the call. Thus F[3,4,5,2] will result in

X=3 Y=4 Z=5 W=2

when execution of the function is started. When the function returns, the values of the local variables are thrown away and the old values which were saved by the function call are restored.

The above DEFINE will cause the expression after the = to be evaluated and returned as the value of the function call. An alternate form permits more complicated functions to be written:

DEFINE F[X,Y,Z,W]: any statement

If the statement is not a TO, it is executed and a zero value is returned. If it is a TO, control is transferred to the specified point. The function can return by executing

RETURN expression

which causes the expression to be taken as the value of the function.

A function must not have a local variable which is the same as the name of the function itself. It is not necessary to supply values for all the local variables when the function is called. The arguments supplied will be assigned to the first few local variables, and the others will be left undefined. This permits temporary storage locations to be created for the function. Care must be taken that any DO or FOR started within the function is finished before the RETURN. It is an error to terminate a DO started before the function call within the function.

If an actual function argument is a name, its value can be changed by the function. Thus

```
DEFINE   F[A,B]:  SET  A←B↑2
```

when called with

```
SET   Y = F[I,4]
```

sets I to 16 (and Y to 0).

```
DEFINE   F[X,Y,N,I]:  SET   X(I) = Y(I) FOR I = 1 TO N
```

when called with

```
Z = F[A,B,14]
```

sets A(1)=B(1), A(2)=B(2), ..., A(14)=B(14).

A function may do anything, including calling itself. Compare two definitions of the factorial function

```
DEFINE F1[X] = IF X<=1 THEN 1 ELSE X*F1[X-1]
DEFINE F2[X, X1] = PROD (X1=1 TO X:X1)
```

Or a more complex example, which computes the exponential to one part in 10^{-7}

```
DEFINE E[X,I,J,S,T] = 1+SUM (I=1 BY 1 WHILE T/S > 1E-7
     WHERE S=S+T : T WHERE T=X↑I/PROD(J=1 TO I : J))
     WHERE S=0 & T=1
```

A somewhat more efficient form is

```
DEFINE E[X,S,T,I] = 1+SUM (I=1 BY 1 WHILE T/S> 1E-7
     WHERE S=S+T: T ← T*X/I)
     WHERE S=0 & T=1
```

8.0 Input-Output

CAL has facilities for allowing the user to read from and write on standard system files. For a discussion of file naming, consult the executive manual.

To pen a file and assign it a number, the step

OPEN "name" FOR {INPUT / OUTPUT} AS FILE expression

The value of the expression should be used to reference the file after it is open.

When the file is no longer being read or written, the step

CLOSE expression

will close it. The information is not affected; the file simply becomes unavailable for input/output until it is reopened.

The step

WRITE ON expression:

is exactly like TYPE, and anything which is legal after TYPE may follow the colon, with one exception: The format

WRITE ON expression IN FORM expression:

should be used if a form is desired.

To read from a symbolic file, the step

READ FROM expression:

may be used. It is exactly like DEMAND. The convention that preceding non-numeric characters are ignored is convenient. READ IN FORM makes little sense and is not available.

Binary floating point numbers may be written on a file, two words per number, with the step

OUTPUT ON expression: expression, expression ...

Such a file may be read with

INPUT FROM expression: variable, variable ...

A file can also be opened with

```
OPEN "name" AS DRUM
```

Thereafter it is available for storage of singly-subscripted variables. See Section 5.

If an input operation encounters an end of file, it normally aborts. To prevent this from happening, the step

```
CALL function ON END OF FILE
```

exists.

If an end of file occurs on input, the specified function will be called with a single argument whose value is the number of the file responsible. The value of the function will be taken to be the number which the program was trying to read from the file. Example: If numbers are to be read first from file V and then from file W, the following program will do the job:

```
1.1  OPEN "V" FOR INPUT AS FILE 1
1.2  CALL EO ON END OF FILE
1.3  READ FROM 1:  X(I) FOR I=1 TO 1000
DEFINE EO[A,B]:  10 STEP 2.1
2.1  CLOSE A
2.2  OPEN "W" FOR INPUT AS FILE A
2.3  TYPE "FILE W OPENED" IF TO
2.4  READ FROM A:B
2.5  RETURN B
```

It will also type a message when it switches files if TO≠0.

9.0 Display Output

CAL contains a number of features which permit figures and text to be presented on one of the two Project Genie displays attached to the 940 through PDP-5's.

To begin using the display through CAL, the following procedure should be followed:

1) Start the link loader running in the PDP-5. This is normally done by putting 7740 in the switches and doing LOAD ADDRESS and START. If the memory address register shows a fairly tight loop in the vicinity of 7740, the loader can be assumed to be running. If this is not the case, instructions attached to the machine should be followed to get the loader into core.
2) Execute the CAL step

   OPEN DISPLAY e

   Where e is an expression with the value 0 or 1. An 0 value denotes the Burroughs display, a 1 the IDI display. Other values of e will give rise to error comments. This step may be executed directly or indirectly.

As a result of these operations, a program (the PDP-5 part of the object package) will be sent to the PDP-5, and the character D will appear in the lower left corner of the screen. If the D does not appear, push the GO button on the display. If it does not appear, something is wrong. Try turning up the intensity on the display. If this does not work, close the display in CAL (see below), read in a fresh copy of the link loader, and try again. If this does not work, give up and complain.

Once the display has been successfully opened, it will remain attached to CAL and ready for output until it is closed with the step.

CLOSE DISPLAY

Return to the exec will close the link files, but they will be reopened by a CONTINUE CAL command.

Do not attempt to open the display unless the link loader is running. If you do, it will probably be necessary to restart CAL from scratch, as it will be very confused.

Two types of material may be displayed through CAL: "curves" and text. Locations of both are specified by internal display coordinates, which are initialized to the range -1 to +1 on both X and Y axes. An internal coordinate of -1 corresponds to the bottom or left side of the display, one of +1 to the top or right side. The range of internal coordinates can be changed by the step

SCALE X AXIS FROM e1 TO e2

and the corresponding operation for Y. The word AXIS may be omitted. This step makes e1 correspond to the bottom (or left side) of the display and e2 to the top (or right side). It is not necessary that e1 < e2, but e1 = e2 is an error. Addressing points on the display outside the range specified by a SCALE is not forbidden, and a linear extrapolation is performed.

The portion of the scope face addressed by the current SCALEs can be altered with the step

DISPLAY X AXIS FROM e1 TO e2

and the corresponding operation for Y. The word AXIS may be deleted. The expressions here specify absolute display coordinates, which range from 0 (bottom or left side) to 1023 (top or right side), and the values of the expressions are restricted to this range. CAL is initialized to address the whole scope face. The initial state may thus be reestablished by the four steps

SCALE X AXIS FROM -1 TO 1
SCALE Y AXIS FROM -1 TO 1
DISPLAY X AXIS FROM 0 TO 1023
DISPLAY Y AXIS FROM 0 TO 1023

As with SCALE, it is possible to display on the scope outside the area defined by DISPLAY; the same linear extrapolation is performed. After an internal display coordinate has been converted to an absolute one, it is taken modulo 1024.

The display may be treated as a teletype with the step

WRITE ON DISPLAY: output

where the output may be anything which could appear after TYPE. Another version, also modeled after TYPE, is

WRITE ON DISPLAY IN FORM e: output

If no other action is taken, output will appear after the initial D in the lower left corner. The location of the next character output by a WRITE ON DISPLAY can be fixed, however, by adding after DISPLAY the modifier

AT e,e

where the two expressions are internal display coordinates of a point on the scope. Characters will then be written on the horizontal line specified by the Y coordinate and starting at the position specified by the X coordinate. A carriage return or overflow off the right side of the screen will cause the next character to be written at the same X position as the first one, but one line further down. I.e., an AT modifier defines the left margin.

The display can also be used for presenting graphical material. This is done with the PLOT step, which has the form

PLOT ON CURVE e: e1,e2

The e1 and e2 are internal scope coordinates specifying a point on the scope face. A line is drawn from the last point plotted on curve e to this one. If no points have been plotted on curve e, no display will be generated by this step; it will simply serve to establish the origin for the line which will be drawn by the next plot on this curve.

Curve numbers must range between 1 and 20. The phrase ON CURVE e may be omitted, in which case the last curve mentioned will be assumed. Curve 1 will be assumed initially.

Any of the modifiers BLANK, DOTS (or POINTS), DASHED or SOLID may appear after the curve specification. These modifiers

determine the kind of line to be drawn. They are self-explanatory except perhaps for DOTS, which simply causes the endpoint of the line to be displayed. It is useful for plotting isolated points.

All curves are initialized to SOLID. A modifier remains in force until another one is used for the same curve. Examples:

```
PLOT ON CURVE 5, DASHED:  .5,.2
PLOT ON CURVE 6+I, SOLID:  X, SIN(X+I*PI) FOR X=0 BY
  .1 TO 2*PI FOR I=0,1
```

will plot sin x and sin(x+n) between 0 and 2n.

```
PLOT DOTS:  I,J FOR I=1 BY .1 TO 1 FOR J=-1 BY .1 TO 1
```

will plot a matrix of 121 dots evenly spaced over the scope face (assuming initial scale and range).

```
PLOT ON CURVE 10, BLANK:  -1,0
PLOT SOLID:  1,0
PLOT BLANK:  0,-1
PLOT SOLID:  0,1
```

will put x and y axes on the screen.

Material to be displayed is buffered in the 930 and <u>will not appear on the screen</u> until the step

```
DISPLAY
```

is executed, except that if this buffer overflows, its contents will be sent to the display automatically. Too frequent use of DISPLAY will slow down the execution of the program, however, because of the manner in which the system handles the output. A DISPLAY is supplied automatically whenever CAL type its > sign.

There is no way to remove material which has been put on the display with PLOT or WRITE except by execution of

```
CLEAR DISPLAY
```

which reinitializes the display to its original state. Scale and range are reset and all material except the initial D is removed from the screen.

It is possible to overflow the PDP-5 memory available for buffering the display. If this happens, output is terminated

and an error message generated. Execution of CLEAR DISPLAY will permit new material to be displayed. Experience indicates, however, that flicker will reach objectionable levels before this overflow occurs. Flicker can be alleviated by adjustment of the intensity, and by turning off the lights in the display half of the room. A cord which controls these lights hangs down from the ceiling near the IDI display.

The amount of space taken up by a display picture depends to a considerable extent on how it is generated. Space requirements are minimal for a display in which all the points for one curve are plotted before any plotting is done for any other curve, and in which all text is written without any interspersed PLOT steps and with a minimum number of AT modifiers. Two words are used for each point, one for each character, one for each occurrence of DOTS, DASHED or SOLID. Two extra words are used each time a point is plotted on a curve different from the one on which the last point is plotted. There are about 3400 words available.

Occasionally the display will start to show something funny. Pushing the GO button will usually correct this. Do not, however, push the GO button while your program in transmitting to the display. If things become fouled up, clear the display and try again. If the PDP-5 program becomes clobbered, close the display, restart the link loader, and re-open it.

10.0 Running the Program

A program can be started by typing a direct TO or DO step. Execution will continue until

1) The DO is complete or, in the case of a TO, the last step of the program has been executed.
2) A PAUSE step is executed, CAL types out
   PAUSE IN STEP n.n:
   and awaits instructions. Steps can be added to the program and any steps except DO and TO may be used directly. The program can then be restarted with a GO step. If a TO or DO has been executed the state of the program at the time of the PAUSE is lost. This also happens if any directly executed step causes an error, or if any step, function or form being executed is deleted or changed. If a complex DO or function call structure exists, there may be many such steps.
3) The rubout key on the teletype is pushed. CAL completes execution of the current step, or of the current loop of a FOR, and then types
   INTERRUPTED IN STEP n.n:
   The situation is then identical to that produced by a PAUSE.
4) An error occurs in the running program. A suitable message will be typed out, and the situation is then identical to that produced by a PAUSE.

CAL SUMMARY

Indirect (with step number) or direct (execute immediately):

SET v = e (SET may be omitted)

TYPE e1, e2, e3 ...
TYPE IN FORM e: e1, e2 ...
TYPE(STEP)n.n, n.n, ...
TYPE PART e
TYPE FORM e
TYPE "string"
TYPE ALL STEPS
TYPE ALL FORMS
TYPE ALL VALUES
TYPE ALL FUNCTIONS
TYPE ALL

DEMAND v1, v2 ...
DEMAND IN FORM e: v1, v2 ...

OPEN "name FOR INPUT / OUTPUT AS FILE e

OPEN "name AS DRUM
CLOSE e
WRITE ON e: <as for TYPE>
READ FROM e: v1, v2 ...
OUTPUT ON e: e1, e2 ...
INPUT FROM e: v1, v2 ...

PAGE
LINE

TO(STEP)n.n
TO PART e

Direct only:

DELETE v
DELETE(same as for TYPE).

DUMP
LOAD

GO
STEP
CANCEL
DEFINE v[v,v,v...] =e
:statement

EDIT(or MODIFY or MOD) (STEP) n.n
EDIT FORM e
EDIT v (function only)

Indirect only:

DRUM v

RETURN e

PAUSE (equivalent to rubout)

DONE (stops a DO PART)

FORM n: <line feed>
NO 1: ####### NO 2: %%%%%%.%%%

Modifiers may be attached to any step. Several modifiers may appear on one step.

IF e

UNLESS e

FOR v = e BY e { TO / WHILE / UNTIL } e (BY may be omitted)

e
any string of these separated by commas

WHILE e
UNTIL e
WHERE $v_1 = e_1$ & $v_2 = e_2$

Expressions (represented by "e" elsewhere) are made up of operands and the operators.

← (replacement)

AND OR (same as *, + except for precedence)
NOT (changes 0 to 1, anything not 0 to 0)

=# > >= < <= (yielding 0 or 1)

+ -
* /
↑ (exponentiation)

with the precedence levels indicated by the order. Parentheses may be used freely.
THEN e_{v1} ELSE IF e_{p2} THEN e_{v2} ... ELSE e_{vr}
WHERE may be appended to expressions.

Operands may be

variables
numbers
$, with the value of the current line
PI
Special functions
Iterative functions
CAL functions

Variables (represented by v elsewhere) are single letters followed by 0 or 1 digits, possibly with any number of subscripts. Thus

X1S(1) X(Y(Z5),W9↑2)

No restrictions on subscripts, which are taken to the nearest integer mod 2^{23}.

Numbers may be written with decimal digits, a decimal point, and an exponent field indicated by E. Thus

10 = 1E1 = .001E4 = 10000E-03

Special Functions are

SIN, COS, TAN with arg in rads

ATAN (X,Y) with result in rads

LOG, LOG10, EXP, SQRT, ABS

IP (integer part); FP (franctional part)

The argument need not be enclosed in parentheses unless it has an operator of lower precedence than *.

Iterative Functions are

SUM, PROD, MAX and MIN.

Example:

SUM (I=0 BY 1 WHILE I<10:

X↑I/PROD(J=1 TO I:J))

$$\sum_{i=0}^{10} x^i/i!$$

CAL Functions

A CAL function is named by a variable, which cannot then have numeric values. It is called thus:

F[A, 16.3*W(3)].

Input: You are editing the statement last typed in. Characters typed replace old ones. Control characters are

- A^c print ↑ and delete preceding character
- W^c print \ and delete preceding word
- Q^c print ← and delete preceding line
- cr throw away the rest of the old line. Done.
- C^c copy a character
- S^c skip a character and print %.
- Z^cC copy up to character C, inclusive
- X^cC skip up to character C, inclusive
- R^c retype
- T^c pretty retype
- Y^c copy rest of old line without typing and start over
- D^c copy and type out rest of old line. Done.
- F^c copy rest of old line. Done.
- E^c type < and switch to inserting character typed. Second E^c means type > and switch back to replacing.