EUROPEAN ORGANIZATION FOR NUCLEAR RESEARCH

GERN/ISR-CO/74-33
CLOSED DISTRIBUTION

DOS-BASIC: AN INTERPRETIVE LANGUAGE
FOR PDP-11 COMPUTERS

by

D. Kemp

Summary

This paper describes the interpretive language DOS-BASIC which runs on a Digital Equipment Corporation PDP-11 series machine under the Disc Operating System (DOS). As its name suggests the language is essentially single-user paper-tape BASIC as supplied by D.E.C. G. Shering modified this to run under DOS and made further additions to allow machine-code modules and the use of DOS datasets for the storage and retrieval of binary data, programs and text. For use with CAMAC (and particularly as an engineer's test facility) the language has been further extended to include octal numbers and a 32-bit integer facility. Boolean functions, shift functions and CAMAC handling functions (for the Nuclear Enterprises 9030 crate controller) have been introduced as machine-code modules.

Geneva, June 1974
CONTENTS

1. INTRODUCTION
   1.1 Method of an interpreter
   1.2 When to use an interpreter

2. INPUT TO DOS-BASIC
   2.1 Statements
   2.2 Stored statements
   2.3 Direct statements

3. HOW TO OPERATE DOS-BASIC
   3.1 Activation of the interpreter
   3.2 Keyboard editing
   3.3 Restart and termination of DOS-BASIC

4. COMMANDS
   4.1 Names
   4.2 Numbers
   4.3 Expressions
   4.4 LET command
   4.5 IF command
   4.6 GOTO command
   4.7 FOR command
   4.8 NEXT command
   4.9 PRINT command
   4.10 INPUT command
   4.11 DATA command
   4.12 READ command
   4.13 RESTORE command
   4.14 DIMension command
   4.15 RUN command
   4.16 STOP command
END command
LIST command
DELETE command
SAVE command
OLD command
REMark command
GOSUB command
RETURN command
RANDOMIZE command
DEFine command
ASSIGN, ERASE, PUT, GET commands
(see section 7)
DO command
(see section 8)

5. BUILT-IN FUNCTIONS

5.1 Introduction
5.2 Mathematical functions
5.3 User functions (see 4.26)
5.4 Machine code functions (see section 8)

6. ERROR MESSAGES

6.1 Error messages
6.2 DOS monitor messages
6.3 DOS-BASIC messages

7. THE USE OF DOS DATA SETS

7.1 Terminal input-output
7.2 SAVE, OLD, RUN
7.2.1 SAVE
7.2.2 OLD
7.2.3 RUN
7.3 PRINT, INPUT
7.4 ASSIGN command
7.5 PUT and GET commands
7.5.1 PUT command
7.5.2 GET command
7.6 Closing datasets
7.7 ERASE command
7.8 Notes of file usage

8. MACHINE CODE MODULES

8.1 Introduction
8.1.1 Acquisition variable
8.1.2 Control variable
8.1.3 DO command
8.2 Built-in machine code modules
8.2.1 Acquisition variables
8.2.2 Control variables
8.2.3 DO subroutines

9. REFERENCES

APPENDICES

1. COMMANDS AND FUNCTIONS

2. DOS-BASIC ERROR MESSAGES

3. HOW TO WRITE A MACHINE CODE MODULE

3.1 Introduction
3.2 Programming details
3.2.1 Names
3.2.2 Linking
3.2.3 Exchange of parameters with DOS-BASIC
3.2.4 DOS-BASIC internal routines
3.2.5 Error handling

Example program
Correspondence table for machine code module names
Some DOS-BASIC internal routines

4. EXAMPLES
1. INTRODUCTION

1.1 Method of an interpreter

This interpreter, based on D.E.C. BASIC with extensions by G. Shering and subsequently by D. Kemp, is a means of constructing small programs in an easy manner.

An interpreter is a program which is able to read a list of commands, decode and execute them via subroutines it contains. The interpreter never generates any machine-code as a compiler does but performs a series of calls to subroutines which immediately execute the command.

\[
\text{LET } A = B + C \quad \rightarrow \quad \text{INTERPRETER} \quad \rightarrow \\
\text{PRINT } A
\]

\[
\text{GET } B \\
\text{GET } C \\
\text{ADD } B \text{ and } C \\
\text{STORE } \text{ in } A \\
\text{PRINT } A
\]

1.2 When to use an interpreter

The big advantage of an interpreter is that program construction is carried out in an interactive way so that statements can be easily altered and new ideas can be tried out with a minimum of effort. This can be done because the interpreter remembers the whole input text and it is therefore enough just to correct, add or delete statements and run the program again without compiling the whole program.

Furthermore, one can examine and alter the value of a variable by use of its name instead of its address as in compiler generated programs. The main area for interpreter usage is therefore test programs with high alteration frequency and program prototypes which later can be written in another language and compiled.
The fact that an interpreter must decode every statement before executing the actions results in a low speed of execution. For programs which must run fast or are frequently used it is therefore not recommended to use the interpreter.
2. **INPUT TO DOS-BASIC**

2.1 **Statements**

DOS-BASIC uses lines of input. The content of a line is called a statement.

A statement may consist of several commands which must be separated by a colon (:). Commands are the orders to DOS-BASIC, e.g. PRINT, LIST, STOP etc. If a statement consists of more than one command it is called a multiple statement: otherwise it is called a simple statement.

Those commands which may be used to construct multiple statements are indicated on the list given in Appendix 1.

Examples:

Simple statement : LET A = B + C

Multiple statement consisting of two commands : LET A = B + C : PRINT A

A statement can be preceded by a number in which case it is called a stored statement: otherwise it is called a direct statement.

2.2 **Stored statements**

A stored statement is always preceded by a number in the range 1 to 8191 known as the line number.

Examples of stored statements:

```
10 LET A = B + C
20 IF ABS(B) > 0.5 GOTO 30
```

A stored statement entering the interpreter will be saved with minimal checking. When the interpreter is reading and storing statements, it is said to be in program state. When the word RUN is typed, the state is changed from program to run state. In this state the interpreter will execute the stored commands, starting with the lowest numbered statement and continue in numerical order, if the sequence is not altered by a GOTO or GOSUB command.
The interpreter continues in the run state until it encounters a \texttt{STOP} or \texttt{END} command or it has obeyed the last stored command when it will print:

\begin{center}
\texttt{STOP AT LINE XXXX}
\end{center}

and the state is changed to program state.

(Note: After \texttt{LIST, DELETE, OLD, SAVE} commands or fatal errors the interpreter also returns to program state and prints: \texttt{READY})

Alteration of a stored program is done by replacing the old statement with the new, using the same line number. Deletion of a statement is done by typing only the line number, i.e. a statement with no commands or by use of the \texttt{DELETE} command (see 4.9).

2.3 Direct statements

A direct statement is never preceded by a number and is executed immediately. The interpreter does not retain it in memory.

Example of a direct statement:

\begin{center}
\texttt{LET A = 5}
\end{center}

The direct statements operate on the same variables as the stored statements.

<table>
<thead>
<tr>
<th>LET A = 32.3</th>
<th>A direct statement</th>
</tr>
</thead>
<tbody>
<tr>
<td>10 PRINT A</td>
<td>A stored statement</td>
</tr>
<tr>
<td>RUN</td>
<td>Runs the list of stored statements i.e. 10</td>
</tr>
<tr>
<td>32.3</td>
<td>The stored statement prints out the result of the direct statement</td>
</tr>
</tbody>
</table>
3. **HOW TO OPERATE DOS-BASIC**

3.1 **Activation of the interpreter**

The interpreter is stored as an ordinary program and has the name **BASIC**. When the program is called (RUN BASIC) it answers with:

```
DOS BASIC
READY
```

as an indication that it is in program state and ready to receive direct and stored statements.

3.2 **Keyboard editing**

When typing in a line the character **rubout** will delete the last character. Repeated use of **rubout** can be used to remove up to the whole of the present input line,

```plaintext
  e.g.  ABCD rubout rubout X
        appears as:  ABCD\DC\X
        and is equivalent to:  ABX
```

The character **CTRL/U** can be used to delete the whole of the present input line.

Stored statements may be entered in any order.

3.3 **Restart and termination of DOS-BASIC**

The user may break into DOS-BASIC at any time using the **CTRL/C** character which re-establishes contact with the DOS monitor. Any DOS-BASIC printer output will be suspended but otherwise the DOS-BASIC program will continue running. Any acceptable DOS monitor command may be typed - in particular:

**KILL** which causes DOS-BASIC to terminate.

**RESTART** stops execution of the DOS-BASIC program.

**DOS-BASIC** prints

```
STOP AT LINE XXXX
```
where XXXX is the line in which execution stopped.

Program mode is then entered with the program and variables left unchanged for further examination or use.

BEGIN restarts DOS-BASIC with a 'clean slate'.
4. **Commands**

4.1 **Names**

All names of variables or arrays used in DOS-BASIC may consist of a letter or a letter followed by a digit,

e.g. A,B7

4.2 **Numbers**

All numbers are represented internally in three-word floating-point format which gives a range of approximately $10^{+1000}$ to $10^{-1000}$ with an accuracy of approximately 5 in $10^{10}$. Allowing for rounding errors it is usual to assume 1 in $10^8$.

Numbers may be input as floating-point with or without an exponent, integers or octal (preceded by # and regarded as a 32-bit integer).

e.g. 10.237
    0.461235972 E+61
    39
    #707

4.3 **Expressions**

Variables, numbers and functions are called primaries. An expression is a combination of primaries and arithmetic operators.

Valid arithmetic operators are:

+    plus
-    minus
*    multiply
/    divide
↑    power

An expression is evaluated in the following order:

1) ↑
2) */
3) +-
Examples:

\[ A+B\times C+D \text{ is evaluated as } A+(B\times(C+D)) \]
\[ A+B/C+D \text{ is evaluated as } A+(B/(C+D)) \]

Parentheses may be used to control the order of evaluation, expressions inside parentheses being computed before being used in further computations.

Two operators must never be written next to one another. Hence, parentheses are essential in expressions such as \(A^*(-B)\).

4.4 **LET command**

**FORM:** LET variable = expression

A LET command is used to assign a value to a variable:

Examples:

\[ \text{LET } V1 = 39.7 \]
\[ \text{LET } V2 = V1^2 + 3 \]
\[ \text{LET } V2 = \sin(1/V1) + 0.3 \]

4.5 **IF command**

**FORM:** IF expression comparator expression THEN line number GOTO line number

The IF command is used for conditional control of the program flow.

Comparators are as follows:

- \(<\) less than
- \(<=\) less than or equal
- \(=\) equal
- \(<>\) not equal
- \(>\) greater than
- \(>=\) greater than or equal

The command following the comparison is only executed if the comparison is true. If the comparison is false execution passes to the next statement line.
Examples:

\[
\begin{align*}
\text{IF } A & > 0 \text{ THEN PRINT } "A \text{ POSITIVE}" \\
\text{IF } 2*B & \leq K \text{ GOTO 285}
\end{align*}
\]

4.6 **GOTO command**

**FORM:** GOTO line number

A GOTO command is used to change the order of execution of the program. Control is passed to the start of the statement with the given line number.

Example:

```
GOTO 10.
```

4.7 **FOR command**

**FORM:** FOR control variable = initial TO final STEP stepsize

The FOR command (in conjunction with the NEXT command) controls the repetitive execution of one or a group of statements. Statements (to the corresponding NEXT) will be executed first time with the variable equal to initial and thereafter again for the variable augmented by stepsize. This continues until the variable reaches final. If the stepsize is equal to 1, it can be omitted. The initial and final values are only calculated once - on initial entry to the loop. The value of the control variable may be modified in the loop. On exit from the loop the control variable retains the last value used within the loop. Initial, final and stepsize can be any valid expression (positive or negative, integer or fraction). The loop is not executed in the cases where initial is greater than final and stepsize is positive or when initial is less than final and stepsize is negative.

Examples:

```
150 FOR I = 0 TO 50 STEP 10
```

This will cause the statements in the loop to be executed for I = 0, 10, 20, 30, 40, 50.

```
150 FOR I = 0 TO 50
```

In this case the statements will be executed for I = 0, 1, 2, 3 ... 50.
FOR loops may be nested in the form

10 FOR I = 1 TO 50
20 FOR J = 1 TO 10
30 -----------
50 NEXT J
60 NEXT I

The inner FOR loop will be started 50 times by the outer FOR loop, i.e. statement 30 will be executed 500 times.

4.8 NEXT command

FORM: NEXT variable
Indicates the range of the FOR loop with the given variable as its control variable.

Example:

10 FOR I = 1 TO 5
50 NEXT I

4.9 PRINT command

FORM: PRINT item, item

(see also 7.3)

The output is divided into five zones of fourteen characters.
The end of a PRINT command (EOP) causes a new line. A comma (,) causes output to move to the beginning of the next fourteen character zone.
A semi-colon (;) can replace the comma as a separator and in this case there is no spacing between items. When placed at the end of a PRINT command the comma and semi-colon suppress the automatic new line.

An item may be
(a) an expression followed by ,, or EOP
   e.g. PRINT A, B, C+D*SIN(Q)

Note: Values are printed out as integer or decimal numbers when they will fit in nine spaces, i.e., seven digits plus a sign (space when positive) and a decimal point. Outside this range they are printed out in exponential or E format in a maximum of fifteen
characters, i.e.,

\[-.xxxxxxxE(-)xxxx\]

For positive numbers the sign is replaced by space.

(b) text enclosed in quotes (") followed by ;, EOP or anything (equivalent to ; anything)

e.g.

PRINT "VALUE OF A IS" A, "MMS"

(c) an octal - written OCT$(expression) followed by ;, or EOP


e.g.

PRINT OCT$(A), OCT$(AND(B,$77))

(d) an ASCII character - written CHR$(expression) followed by ;, EOP or anything (equivalent to ; anything)


e.g.

PRINT CHR$(I+1)CHR$(#101)

Note: i) The expression is masked to seven bits

  ii) When EOP follows CHR$ output the automatic new line is suppressed.

(e) a void expression can be used for layout purposes


e.g.

PRINT gives a new line
 PRINT,,A, gives 28 spaces before A and does not give a new line after A

4.10 INPUT command

FORM: INPUT variable, variable, --

(see also 7.3)

Assigns numbers to a list of variables. Upon execution of the command a question mark (?) is output and a line of data read. The numbers input will be assigned to the variables in the list, first number to the first variable and so on. The successive numbers must be separated by a comma.

Example:

10 PRINT "VALUES OF A,B ARE";
20 INPUT A,B
Note: Non-fatal errors 121, 122 which imply too little or too much data for INPUT, cause the INPUT command to be repeated.

4.11 DATA command

FORM: DATA number, number, ----

Introduces a numeric constant or a series of constants into a program for use by the READ command (see 4.12)

 e.g.  
 10 DATA 9.3, 7.9E-5, #17, 2

4.12 READ command

FORM: READ variable, variable, ----

Assigns numbers (from the DATA commands) to a list of variables. The DATA commands are accessed in line number order. Successive READ commands start at the point in the DATA commands where the previous READ finished

 e.g.  
 10 DATA I, 9.3, #13
 20 READ A, B, C, D
 30 READ E, F
 40 DATA 6, 2, 7

will have the effect of setting:

A=1, B=9.3, C=#13, D=6, E=2, F=7

4.13 RESTORE command

FORM: RESTORE

Causes the DATA commands to be rescanned from the beginning of the program

 e.g.  
 10 DATA 1, 2, 5
 12 READ A, B
 13 RESTORE
 14 READ C

will have the effect of setting:

A=1, B=2, C=1
4.14 **DIMENSION command (DIM)**

**FORM:**  
```
DIM name (integer, integer), ---
```

Defines arrays. An array is a named collection of numbers, which can be accessed in the same manner as elements of a matrix. The first integer number in the brackets is the number of elements per row and the second the number per column. The last number need not be present, in which case the array is called one-dimensional; otherwise it is called two-dimensional.

Elements are referenced by their row and column number, the latter only being relevant for two-dimensional arrays. The row and column numbers always start from 0 with a maximum value for a subscript of 255, e.g.

```
DIM X(255), Y(9,9)
```

- X is a one-dimensional array with 256 elements
- Y is a two-dimensional array with 10 rows and 10 columns

Elements from X: X(0), X(97)
Elements from Y: Y(2,7), Y(9,0)

**Note:** X≡X(0); Y≡Y(0)≡Y(0,0)

Elements from arrays can be used just as normal variables in expressions etc.

**e.g.**

```
10 LET X(I) = Y(J,I)*2 + 3
```

4.15 **RUN command**

**FORM:**  
```
RUN
```

(see also 7.2.3)

Starts a stored program from its lowest numbered statement. Starting at any given statement can be accomplished by means of a direct GOTO command.

When **RUN** is used to rerun a program previously executed all variable values are deleted from memory and RND reinitialised (see 5.2).
4.16 **STOP command**

**FORM:** STOP

Stops the execution of a program. DOS-BASIC prints:

```
STOP AT LINE XXXX
```

and the state is changed to program state. STOP indicates the logical and not the physical end of a program (see END command 4.17). It is particularly useful as an insert when debugging.

4.17 **END command**

**FORM:** END

Indicates the physical end of a program. The END command must have the highest line number. The action is as for STOP (see 4.16).

4.18 **LIST command**

**FORM:** LIST linenumber, linenumber

Lists all or part of a program. DOS-BASIC returns to program state, e.g.

- `LIST` - List the whole program
- `LIST linenumber` - List the given line
- `LIST linenumber, linenumber` - List from the first to the second linenumbers inclusive

4.19 **DELETE command**

**FORM:** DELETE linenumber, linenumber

Deletes all or part of a program. DOS-BASIC returns to program state, e.g.

- `DELETE linenumber` - Deletes the given line
- `DELETE linenumber, linenumber` - Deletes from the first to the second linenumbers inclusive
- `DELETE 1,8191` - Delete whole program

4.20 **SAVE command**

**FORM:** SAVE (see also 7.2.1)

Outputs the present stored program to the paper-tape punch.
DOS-BASIC returns to program state.

4.21 OLD command

FORM: OLD (see also 7.2.2)

Reads a stored program from the paper-tape reader. DOS-BASIC returns to program state.

4.22 REMARK command (REM)

FORM: REM text

The rest of the line is treated as comment, e.g.

    REM THIS IS A COMMENT

4.23 GOSUB command

FORM: GOSUB linenumber

The GOSUB command is the subroutine calling facility. Subroutines may be nested to any depth.

4.24 RETURN command

FORM: RETURN

This is the exit from a subroutine called by GOSUB. (Note: any one RETURN statement may act for more than one subroutine or a subroutine may have more than one RETURN)

    10 LET A = 5: LET B = 6: GOSUB 100
    20 PRINT A, B, C
    30 LET A = 6: LET B = 7: GOSUB 100
    40 PRINT A, B, C
    50 STOP
    100 LET C = A^2 + B^2
    101 RETURN
    102 END
4.25 **RANDOMIZE** command

**FORM:** RANDOMIZE

Causes the RND function to produce random rather than pseudo-random numbers. (Pseudo-random numbers always start at the same value and go through the same sequence.)

4.26 **DEFINE** command (DEF)

**FORM:** DEF FNa (variable) = expression

where a is any letter

Allows users to declare their own functions. The function must be defined before it is used. The variable is a dummy variable which may be referenced in the expression. At execution time a value is substituted in the expression.

*e.g.*

10 DEF FNA(S) = S↑2 + 2*S + 3

20 PRINT FNA(7)

result would be 7↑2 + 2*7 + 3 = 66

The function name may be used recursively.

*e.g.*

10 DEF FNA(X) = X↑2

20 LET A = 2

30 PRINT FNA (2 + A*FNA(2))

If the same function name is defined more than once the first definition is used.

The function variable need not appear in the function expression.

*e.g.*

10 DEF FNA(X) = 4*RND(Ø) + 3

4.27 **ASSIGN, ERASE, PUT, GET** commands

See Section 7 on the use of DOS datasets.

4.28 **DO** command

See Section 8 on the use of Machine-code Modules.
5. **BUILT-IN FUNCTIONS**

5.1 **Introduction**

DOS-BASIC includes certain built-in functions. The result of any such function is a value, therefore they may be used in the same way as variables in expressions etc.

*e.g.*

\[ \text{LET } A = \text{SIN}(X/Z) \]

SIN is a built-in function.

5.2 **Mathematical functions**

LOG(X) natural logarithm of X

EXP(X) exponential function \( e^X \)

SQR(X) square root of X

ABS(X) absolute value of X

SIN(X) sine of X (X in radians)

COS(X) cosine of X (X in radians)

ATN(X) arctan of X (result is radians in the range \( \pm \pi/2 \))

SGN(X) -1,0,1 for X negative/zero/positive

INT(X) biggest integer less than or equal to X

*e.g.* INT(3.5) is 3

INT(-3.5) is -4

RND(X) a pseudo-random number in the range 0 to 1.

X is a dummy variable serving no function. If the RANDOMIZE command has been executed a random rather than pseudo-random number is generated.

*e.g.* To obtain a random integer uniform in the range 0-9:

\[ \text{LET } I = \text{INT}(10 \times \text{RND}(0)) \]

To generate numbers over the range A to B

\[ \text{LET } I = (B-A) \times \text{RND}(0) + A \]

5.3 **User functions**

Using the DEF command the user may define his own functions.

(see 4.26)

5.4 **Machine-code functions**

See section 8.
6. **ERROR MESSAGES**

6.1 **Error messages**

DOS-BASIC has two types of error message - those issued by DOS and those issued by DOS-BASIC. Either type may be fatal or non-fatal.

6.2 **DOS monitor messages** (see DOS version 9)

These messages may simply require action (e.g. start the card-reader) or be fatal (e.g. invalid disc call).

6.3 **DOS-BASIC messages** (see Appendix 2)

General format

ERROR XXX AT LINE YYY

If YYY is 0 the fault occurred while in program mode.

Error codes 0-64 indicate a fatal error with DOS-BASIC program execution halted (mostly bad syntax). Codes 65-127 indicate a non-fatal error - the program may continue to run (mostly arithmetic errors).
7. **THE USE OF DOS DATASETS**

7.1 **Terminal Input-Output**

The terminal Input dataset (INP) and Output dataset (OUT) are assigned to the DECWRITER. They can however be assigned to other datasets by use of the **DOS ASSIGN** command (a command of the DOS monitor and **not** of DOS-BASIC).

**e.g.**

1) ASSIGN CR:, INP
   
   RUN BASIC

2) ASSIGN DT0: DOUG.BAT, INP

Assignment of INP to a card-reader or file gives a form of Batch Processing DOS-BASIC. **(Note: DOS-BASIC terminates on receiving an End-of-file from the INP dataset)**

Assignment of INP, OUT to the keyboard and screen of a VISTA would allow DOS-BASIC to be run from this different terminal.

7.2 **SAVE, OLD, RUN**

If SAVE, OLD or RUN are followed by <CR> the action is as defined previously: however the commands may be followed by any legal dataset specifier when the actions are as follows:

7.2.1 **SAVE dataset**

**e.g.**

SAVE FRED

The current program is stored in the specified file (FRED.BAS on the system device SY:).

DOS-BASIC returns to program state.

**Note:** The default file extension is BAS.

7.2.2 **OLD dataset**

**e.g.**

OLD DT0: OLDPRG

The program stored in the specified file is loaded in place of the current program, if any.

DOS-BASIC returns to program state.
Note: The default file extension is BAS.

7.2.3 RUN dataset

e.g.    RUN FRED

The program in the specified file is loaded and run. It is the equivalent of:

OLD dataset

RUN

but when included as part of a running program causes program "chaining".

7.3 PRINT, INPUT

The PRINT and INPUT commands normally use the terminal output and input datasets (OUT, INP). If, however, AOP (Auxiliary Output) or AIP (Auxiliary Input) are assigned (see 7.4) they are used by the PRINT and INPUT commands instead of OUT and INP.

Note: If data for the INPUT command is taken from AIP then the initial output of a question mark (?) is suppressed.

7.4 ASSIGN command

FORM: ASSIGN name # dataset specifier/switches

The name may be:

AOP

AIP

PUT

GET  (see 7.5)

e.g.

10 ASSIGN  AIP#CR:

20 ASSIGN  AOP#DT0:CAMACT.TXT

The possible switches are:

/EX - the file is to be opened for extension - i.e. the new output is added to the information already stored in the file (applies only to AOP or PUT)

/TE - indicates the AOP refers to a terminal device and that output is character by character rather than
line by line (applicable only to AOP)

7.5 PUT and GET commands

7.5.1 PUT command

FORM: PUT expression, expression, ----

Stores the values of the expressions in the file assigned to PUT (note: file must be preassigned).

Values are stored in three-word floating-point format.

e.g.

    20 ASSIGN PUT#DATA
    30 PUT 2+3, A, B(3,5)

will store three numbers on the PUT-file, the numbers being 5, the value of A, and the value of array variable B(3,5).

Note: The default file extension is DAT.

7.5.2 GET command

FORM: GET variable, variable, ----

Gives the variable the next value from the GET file (note: file must be preassigned). If a simple variable, e.g. A or B, does not exist it is created by the GET command.

e.g.

    20 ASSIGN GET #DATA
    30 GET A, B, C

Note: The default file extension is DAT.

7.6 Closing datasets

A dataset can be closed using a modified form of the ASSIGN command.

FORM: ASSIGN name: CLOSE

where name is       AOP
                    AIP
                    PUT
                    GET
This is not often used however as all the above four datasets are closed automatically at the end of a run.

7.7 **ERASE command**

**FORM:** ERASE dataset

The specified dataset is deleted.

7.8 **Notes on file usage**

(a) **ERASE dataset** - does not matter whether dataset exists.

(b) **ASSIGN name: CLOSE** - does not matter whether dataset is opened or closed.

(c) **ASSIGN name # dataset** (for read) - the given dataset must exist.

(d) **ASSIGN name # dataset** (for write) - if the /EX switch is not used the dataset must not exist. This can be forced by use of

```
ERASE dataset
ASSIGN name # dataset.
```

If the /EX switch is used the dataset must exist.

(e) **OLD, RUN dataset** - the dataset must exist.

(f) **SAVE dataset** - does not matter whether the dataset exists.

However a SAVE to an existing dataset will fail if the present file is of insufficient size to accept the required save.

(g) **Note:** the commands STOP, END, RUN, LIST, DELETE, OLD, SAVE have the effect of closing datasets.
8. MACHINE CODE MODULES

8.1 Introduction

A user may write machine code modules and link them to DOS-BASIC to create a particular computer-hardware system. Three types of machine code module are catered for. (For details on how to write such modules see Appendix 3.)

8.1.1 Acquisition variables

These modules are called by DOS-BASIC's expression evaluation process and can be used in a DOS-BASIC program exactly as a normal variable, array or mathematical function. For instance one could write a routine called MAG to acquire the current in one of a series of magnets. Then the instruction

```
PRINT MAG (N)
```

would print out the current in the Nth magnet. This routine has one parameter. Any number of parameters can be handled. If no parameters are required the name is used on its own e.g.

```
PRINT SWR*2
```

would print out twice the value returned by the machine code routine SWR which has no arguments.

8.1.2 Control variables

These routines are called uniquely by DOS-BASIC's LET command. In this command they appear exactly as a normal variable or array. The example given below would have the effect of setting the Nth magnet to 450.

```
e.g.

LET MAG(N) = 450
```

Again no parameters are necessary though any number can be used.

Note: Although the same name is used in the above two examples they correspond to two different machine code routines or at least to two parts of the same routine.
8.1.3 **DO command**

FORM: DO subroutine (parameter, parameter ----)

Again any number of parameters can be used.

8.2 **Built-in machine-code modules**

Note: that all parameters may be any valid DOS-BASIC expression. The CAMAC functions relate to a NUCLEAR ENTERPRISES 9030 CRATE CONTROLLER.

8.2.1 **Acquisition variables**

- **RGT(V,N)**
  - V (considered as a 32-bit integer) is shifted right logically N places.
  - e.g. LET B = RGT(B,7)

- **LFT(V,N)**
  - V (considered as a 32-bit integer) is shifted left logically N places.
  - e.g. PRINT, OCT$(LFT(#1,1))$

- **AND(A,B)**
  - The AND of A and B (both considered as 32-bit integers)
  - e.g. LET G = AND(Z,#770)

- **ORF(A,B)**
  - The OR of A and B (both considered as 32-bit integers)
  - e.g. LET Z = ORF(Z,#1000)

- **NEQ(A,B)**
  - The exclusive OR of A and B (both considered as 32-bit integers)
  - e.g. PRINT OCT$(NEQ(Z,X))$

- **SWR**
  - The contents of the computer console switch register
  - e.g. IF AND(SWR,#10)<0 GOTO 20

- **CAM(C,N,A,F)**
  - The given CNAF is obeyed and the data returned as a 24-bit logical entity.
  - e.g. PRINT CAM(1,2,0,0)
  - Note: A branch error causes fatal error 62.

- **CSR(B)**
  - A one or zero is returned depending on the state of the specified bit (B) in the
CAMAC status register (BIT 0 is the least significant bit).

e.g. IF CSR(12) = 0 THEN PRINT "Q=0"

Note: BIT MEANS
11 BUSY (=1)
12 Q
13 X-LINE FAULT (=1)
14 BRANCH DEMAND (=1)
15 BRANCH ERROR (=1)

8.2.2 Control variables

CAM(C,N,A,F) - Outputs the 24 bits of data and then obeys the given CNAF

 e.g. LET CAM(C,N(I),A,16) = X(I,A)

Note: A branch error causes fatal error 62.

8.2.3 DO subroutines

CAM(C,N,A,F) - Obeys the given CNAF

 e.g. DO CAM(C,N,I+1,8)

Note: A branch error causes fatal error 62.
9. REFERENCES

1. "BASIC programming manual - paper tape software"  
   by D.E.C.

2. "DOS/BATCH user guide"  
   by D.E.C.

3. "DOS/BATCH programmers manual"  
   by D.E.C.

4. "DOS BASIC" by G. Shering  
   LAB II-CO/Int/Comp. Note 72-3

5. "Writing machine code modules for DOS BASIC" by G. Shering  
   LAB II-CO/Int/Comp. Note 72-5

6. "Storage of results and data from DOS-BASIC" by G. Shering  
   LAB II-CO/Int/Comp. Note 72-7
## APPENDIX I

### COMMANDS AND FUNCTIONS

<table>
<thead>
<tr>
<th>COMMAND</th>
<th>FORM</th>
<th>M</th>
<th>S/D</th>
<th>REFERENCE</th>
</tr>
</thead>
<tbody>
<tr>
<td>LET</td>
<td>LET var = exp</td>
<td>Y</td>
<td>Y</td>
<td>4.4</td>
</tr>
<tr>
<td>IF</td>
<td>IF exp comp exp THEN command THEN linemumber</td>
<td>Y*</td>
<td>Y</td>
<td>4.5</td>
</tr>
<tr>
<td></td>
<td></td>
<td>L</td>
<td>Y</td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td>L</td>
<td>Y</td>
<td></td>
</tr>
<tr>
<td>GOTO</td>
<td>GOTO linemumber</td>
<td>L</td>
<td>Y</td>
<td>4.6</td>
</tr>
<tr>
<td>FOR</td>
<td>FOR var=exp to exp STEP exp</td>
<td>F</td>
<td>S</td>
<td>4.7</td>
</tr>
<tr>
<td>NEXT</td>
<td>NEXT var</td>
<td>L</td>
<td>S</td>
<td>4.8</td>
</tr>
<tr>
<td>PRINT</td>
<td>PRINT item, item</td>
<td>Y</td>
<td>Y</td>
<td>4.9/7.3</td>
</tr>
<tr>
<td>INPUT</td>
<td>INPUT var, var</td>
<td>Y</td>
<td>Y</td>
<td>4.10/7.3</td>
</tr>
<tr>
<td>DATA</td>
<td>DATA numb, numb</td>
<td>N</td>
<td>S</td>
<td>4.11</td>
</tr>
<tr>
<td>READ</td>
<td>READ var, var</td>
<td>Y</td>
<td>Y</td>
<td>4.12</td>
</tr>
<tr>
<td>RESTORE</td>
<td>RESTORE</td>
<td>Y</td>
<td>Y</td>
<td>4.13</td>
</tr>
<tr>
<td>DIM</td>
<td>DIM name (integer,integer)</td>
<td>Y</td>
<td>Y</td>
<td>4.14</td>
</tr>
<tr>
<td>RUN</td>
<td>RUN dataset</td>
<td>L</td>
<td>Y</td>
<td>4.15/7.2.3</td>
</tr>
<tr>
<td>STOP</td>
<td>STOP</td>
<td>L</td>
<td>S</td>
<td>4.16</td>
</tr>
<tr>
<td>END</td>
<td>END</td>
<td>L</td>
<td>S</td>
<td>4.17</td>
</tr>
<tr>
<td>LIST</td>
<td>LIST lineno., lineno.</td>
<td>L</td>
<td>Y</td>
<td>4.18</td>
</tr>
<tr>
<td>DELETE</td>
<td>DELETE lineno., lineno.</td>
<td>L</td>
<td>Y</td>
<td>4.19</td>
</tr>
<tr>
<td>SAVE</td>
<td>SAVE dataset</td>
<td>L</td>
<td>Y</td>
<td>4.20/7.2.1</td>
</tr>
<tr>
<td>OLD</td>
<td>OLD dataset</td>
<td>L</td>
<td>Y</td>
<td>4.21/7.2.2</td>
</tr>
<tr>
<td>REM</td>
<td>REM text</td>
<td>L</td>
<td>S</td>
<td>4.22</td>
</tr>
<tr>
<td>GOSUB</td>
<td>GOSUB lineno</td>
<td>L</td>
<td>Y</td>
<td>4.23</td>
</tr>
<tr>
<td>RETURN</td>
<td>RETURN</td>
<td>L</td>
<td>Y</td>
<td>4.24</td>
</tr>
<tr>
<td>RANDOMIZE</td>
<td>RANDOMIZE</td>
<td>Y</td>
<td>Y</td>
<td>4.25</td>
</tr>
<tr>
<td>DEF</td>
<td>DEF FNa(var) = exp</td>
<td>Y</td>
<td>S</td>
<td>4.26</td>
</tr>
<tr>
<td>ASSIGN (open)</td>
<td>ASSIGN name dataset</td>
<td>L</td>
<td>Y</td>
<td>7.4</td>
</tr>
<tr>
<td>ASSIGN (close)</td>
<td>ASSIGN name : CLOSE</td>
<td>L</td>
<td>Y</td>
<td>7.6</td>
</tr>
</tbody>
</table>
Appendix I (contd.)

<table>
<thead>
<tr>
<th>COMMAND</th>
<th>FORM</th>
<th>M</th>
<th>S/D</th>
<th>REFERENCE</th>
</tr>
</thead>
<tbody>
<tr>
<td>ERASE</td>
<td>ERASE dataset</td>
<td>L</td>
<td>Y</td>
<td>7.7</td>
</tr>
<tr>
<td>PUT</td>
<td>PUT exp, exp----</td>
<td>Y</td>
<td>Y</td>
<td>7.5.1</td>
</tr>
<tr>
<td>GET</td>
<td>GET var, var----</td>
<td>Y</td>
<td>Y</td>
<td>7.5.2</td>
</tr>
<tr>
<td>DO</td>
<td>DO subr(par,par--)</td>
<td>Y</td>
<td>Y</td>
<td>8.1.3</td>
</tr>
</tbody>
</table>

**Legend:**

1) The column M refers to the **multiple command statements**
   
   Y = Yes - anywhere
   
   N = No
   
   F = First of multiple
   
   L = Last of multiple

2) The column S/D refers to **stored and direct statements**
   
   Y = Both
   
   S = Stored only
   
   D = Direct only

3) **REFERENCE** is where to find further details in this manual

* depends on following command
Appendix I (contd.)

<table>
<thead>
<tr>
<th>FUNCTION</th>
<th>TYPE</th>
<th>EXAMPLE</th>
</tr>
</thead>
<tbody>
<tr>
<td>LOG(X)</td>
<td>A</td>
<td>PRINT LOG(A+0.5)</td>
</tr>
<tr>
<td>EXP(X)</td>
<td>A</td>
<td>LET Y = A*EXP(B-2)</td>
</tr>
<tr>
<td>SQR(X)</td>
<td>A</td>
<td>LET P = SQR(X<strong>2+Y</strong>2)</td>
</tr>
<tr>
<td>ABS(X)</td>
<td>A</td>
<td>IF ABS(F) &gt; 3 GOTO 5</td>
</tr>
<tr>
<td>SIN(X)</td>
<td>A</td>
<td>LET J = SIN(A/B)</td>
</tr>
<tr>
<td>COS(X)</td>
<td>A</td>
<td>PRINT COS(P-Q)</td>
</tr>
<tr>
<td>ATN(X)</td>
<td>A</td>
<td>IF ATN(F) &gt; 0 GOTO 20</td>
</tr>
<tr>
<td>SGN(X)</td>
<td>A</td>
<td>LET Q = SGN(I)*ABS(F)</td>
</tr>
<tr>
<td>INT(X)</td>
<td>A</td>
<td>PRINT INT(P*100)/100</td>
</tr>
<tr>
<td>RND(X)</td>
<td>A</td>
<td>LET A(INT(10*RND(0))) = I</td>
</tr>
<tr>
<td>RGT(V,N)</td>
<td>A</td>
<td>PRINT OCT$(RGT(D,I))</td>
</tr>
<tr>
<td>LFT(V,N)</td>
<td>A</td>
<td>LET A = LFT(1,I)</td>
</tr>
<tr>
<td>AND(A,B)</td>
<td>A</td>
<td>LET B = AND(RGT(C,3),#17)</td>
</tr>
<tr>
<td>ORF(A,B)</td>
<td>A</td>
<td>PRINT ORF(Z,#1000)</td>
</tr>
<tr>
<td>NEQ(A,B)</td>
<td>A</td>
<td>LET B = NEQ(C,#7777)</td>
</tr>
<tr>
<td>SWR</td>
<td>A</td>
<td>IF AND(SWR,#10) = 0 GOTO 100</td>
</tr>
<tr>
<td>CAM(C,N,A,F)</td>
<td>A</td>
<td>LET B = CAM(2,3,0,1)</td>
</tr>
<tr>
<td>CSR(B)</td>
<td>A</td>
<td>IF CSR(13) = 1 GOTO 200</td>
</tr>
<tr>
<td>CAM(C,N,A,F)</td>
<td>C</td>
<td>LET CAM(3,I+1,A,16) = X</td>
</tr>
<tr>
<td>CAM(C,N,A,F)</td>
<td>D</td>
<td>DO CAM(C,4,0,8)</td>
</tr>
</tbody>
</table>

**Legend:**
- **A** - Acquisition (gives a value)
- **C** - Control (uses a value)
- **D** - DO (obeys a request)
APPENDIX 2

DOS-BASIC ERROR MESSAGES

Codes 0-64 inclusive indicate a fatal error — program execution will halt. Codes 65-127 inclusive indicate a non-fatal error — the program may continue to run after the error message is printed.

<table>
<thead>
<tr>
<th>Error Code</th>
<th>Meaning</th>
</tr>
</thead>
<tbody>
<tr>
<td>FATAL:</td>
<td>0</td>
</tr>
<tr>
<td></td>
<td>User storage overflow</td>
</tr>
<tr>
<td></td>
<td>1 Recognizable statement</td>
</tr>
<tr>
<td></td>
<td>Illegal GOTO or GOSUB</td>
</tr>
<tr>
<td></td>
<td>3 Illegal character terminating a statement</td>
</tr>
<tr>
<td></td>
<td>4 Return without corresponding GOSUB</td>
</tr>
<tr>
<td></td>
<td>5 Badly formed subscript</td>
</tr>
<tr>
<td></td>
<td>6 Subscript not in range 0 to 255 or exceeds maximum set by program</td>
</tr>
<tr>
<td></td>
<td>7 Mismatched parentheses</td>
</tr>
<tr>
<td></td>
<td>8 Illegal LET</td>
</tr>
<tr>
<td></td>
<td>9 Illegal relational operator in IF</td>
</tr>
<tr>
<td></td>
<td>10 Illegal IF</td>
</tr>
<tr>
<td></td>
<td>11 Illegal PRINT</td>
</tr>
<tr>
<td></td>
<td>12 Input line too long (&gt; 80 characters)</td>
</tr>
<tr>
<td></td>
<td>13 Bad dimension in DIM statement</td>
</tr>
<tr>
<td></td>
<td>14 Not enough storage for the array</td>
</tr>
<tr>
<td></td>
<td>15 Badly formed DEF statement</td>
</tr>
<tr>
<td></td>
<td>16 Illegal line number or dimension value</td>
</tr>
<tr>
<td></td>
<td>17 DIM of previously declared or used item</td>
</tr>
<tr>
<td></td>
<td>18 Bad variable in INPUT list</td>
</tr>
<tr>
<td></td>
<td>19 Bad variable in READ list</td>
</tr>
<tr>
<td></td>
<td>20 Out of DATA in READ list</td>
</tr>
<tr>
<td></td>
<td>21 Bad DATA statement format</td>
</tr>
<tr>
<td></td>
<td>22 Illegal FOR statement</td>
</tr>
<tr>
<td></td>
<td>23 No NEXT matching FOR</td>
</tr>
<tr>
<td></td>
<td>24 NEXT without FOR</td>
</tr>
<tr>
<td></td>
<td>25 Unmatched quotes in statement</td>
</tr>
<tr>
<td></td>
<td>26 EXP function - not applicable for DOS</td>
</tr>
<tr>
<td></td>
<td>27 Ill-formed expression</td>
</tr>
<tr>
<td></td>
<td>28 Argument error</td>
</tr>
<tr>
<td></td>
<td>29 Undefined Acquisition function</td>
</tr>
<tr>
<td></td>
<td>30 Command String error</td>
</tr>
<tr>
<td></td>
<td>31 Error in DO command</td>
</tr>
<tr>
<td></td>
<td>32 ASSIGN syntax</td>
</tr>
<tr>
<td></td>
<td>33 GET syntax</td>
</tr>
<tr>
<td></td>
<td>34 Out of data for GET</td>
</tr>
<tr>
<td></td>
<td>35 PUT syntax</td>
</tr>
<tr>
<td></td>
<td>36 File error</td>
</tr>
</tbody>
</table>
Appendix 2 (contd.)

62

63

NON-FATAL: 120
121
122
123
124
125
126
127

Hardware error - Branch Error for CAMAC
Invalid parameter

Illegal characters on input
Not enough data typed to INPUT
Too much data typed to INPUT
Non-existent variable
Number too large to fix
Div.-Mul. Overflow or Underflow
SQR of negative number
LOG of negative or zero number
APPENDIX 3

HOW TO WRITE A MACHINE-CODE MODULE

1. Introduction

As mentioned in section 8 there are three types of Machine-code Module:

(a) Acquisition variable - returns a value
(b) Control variable - uses a value
(c) Do subroutine - obeys a request

2. Programming Details

2.1 Names

These must consist of three characters, two letters, followed by a letter or number. The names are represented internally by a one-word code calculated as follows

\[ \text{Name code} = ((C_1 \times 36) + C_2) \times 36 + C_3 \]

where \(C_1\), \(C_2\), \(C_3\) are the character codes for the individual characters. Letters A-Z are represented by the codes 1-26, and the numbers 0-9 by the codes 48-57 (see Table 1), e.g. for name NEW

\[ N = 14, \ E = 5, \ W = 23 \]

Hence code = \(((14 \times 36) + 5) \times 36 + 23\)

= 18347 decimal

= 43653 octal

2.2 Linking

Linking Machine-code Modules to the DOS-BASIC modules is accomplished by means of globals. For each type of routine there are two tables, the name table and the corresponding entry-point table. The entries in these tables are declared global. In the Machine-code Module the names must be equated to the code defined above and the corresponding label used as entry-point, both of course being declared global. Ten entries
in each table have been provided initially (can be easily changed):

(a) Acquisition variables

Name table        FNM1 - - FNM10
Entry-point table FAD1 - - FAD10

(b) Control variables

Name table        LFN1 - - LFN10
Entry-point table LAD1 - - LAD10

(c) DO subroutines

Name table        DNM1 - - DNM10
Entry-point table DAD1 - - DAD10

DOS-BASIC itself consists of four object modules: TRPHD1, BASFP1, MAIN21, MAIN11. To these four are added the two Machine-Code Modules LOGOPS for Boolean and shift functions and BASCAM for CAMAC functions.

(Note:  BASCAM uses  FNM1, FAD1 - CAM
           FNM2, FAD2 - CSR
           LFN1, LAD1 - CAM
           DNM1, DAD1 - CAM

LOGOPS uses  FNM3, FAD3 - AND
             FNM4, FAD4 - ORF
             FNM5, FAD5 - NEQ
             FNM6, FAD6 - LFT
             FNM7, FAD7 - RGT )

MAIN11 must always be the highest module linked as the DOS-BASIC user space is between the top of MAIN11, as defined by the top switch (obligatory) and top of core. An example LINK command string for the user module FUNDEM might be:

BASIC<DT1:FUNDEM,LOGOPS,BASCAM,TRPHD1,BASFP1,MAIN21,MAIN11/T:50000/E

2.3  Exchange of Parameters with DOS-BASIC

(a) Acquisition variable

e.g.  FUN(ARG1, --- ARGN)

Entry is made to the Machine-Code Module with the values of the arguments on the stack in three-word floating-point format. R1 contains
the number of arguments. ARGN is on the top of the stack with the others in descending order below it. The user must remove the parameters from the stack, push the result on to the stack and then return with the instruction:

MOV 6(SP), PC

DOS-BASIC evaluates the arguments before calling the routine so the routine can be used recursively without the code being re-entrant. An example of a "NEW" function is given at the end of the appendix.

(b) Control variable

  e.g. LET CON(ARG1, -- ARGN) = Expression
       The value of the expression is contained in R2, R3, R4.

The arguments are on the stack as in (a) above, the number of arguments N above being contained in R1. The user must remove the arguments, then return with the instruction

RTS PC

(c) DO subroutines

These subroutines can be written in two ways:

  i) with evaluated arguments, e.g.

      DO SUB(ARG1, -- ARGN)

      The arguments are evaluated and put on the stack as before
      with the number of arguments in R1. The user must remove the
      arguments from the stack and return with

      RTS PC

This type of subroutine is called if the name is equated to
DNM1 to DNM5.

(ii) free format subroutines, e.g.

      DO SUB __________________ [ ; ]
      [ or ]
      [ LF ]

At entry to the Machine-Code Module, the text pointer R1 points to the character after the subroutine name. At exit it must point to a colon or line feed statement terminator. The user can use any DOS-BASIC internal routine. Return is
made via RTS PC.

Note: for these free format subroutines R5 must be preserved. In all other cases R5 is saved and restored by DOS-BASIC so that the user can employ freely all general purpose registers.

2.4 DOS-BASIC internal routines

A programmer, if sufficiently familiar with the DOS-BASIC internal structure, may use any of DOS-BASIC's internal routines (except GTOPR, GETOP which are used by EVAL and which, in turn, use EVAL in a recursive fashion). The routines generally available are:

(calling sequences etc. given in Table 2)

- ATOF - Convert ASCII string to floating-point number
- ATOI - Convert ASCII string to integer number
- ITOA - Convert integer to ASCII string
- IMUL - Integer Multiply
- ADDFX - Floating-point addition
- SUBFX - Floating-point subtraction
- NEGFX - Floating-point negation
- DIVFX - Floating-point divide
- MULFX - Floating-point multiply
- CMPFX - Floating-point compare
- FIX - Convert floating-point number to integer (rounded)
- FLT - Convert integer to floating-point number
- FIXD - Convert floating-point number to 32-bit integer (rounded)
- FLTID - Convert 32-bit integer to floating-point number
- EVAL - Evaluate argument in ASCII parameter list

The EVAL function (for use with free format DO subroutines) may be used to evaluate arguments in a parameter list under the following rules:

i) The parameter to be evaluated must follow all the rules for a normal DOS-BASIC expression

ii) EVAL must be called once for each parameter to be evaluated.

iii) A parameter must be followed by a comma or a right parenthesis.

Upon entry to EVAL, R1 must point to the start of the character string to be evaluated and R5 must have, or be restored to, the same value it had when the Machine-Code Module was entered. On return R2,3,4 will contain the value of the expression. R1 will contain the address where the scan failed. (A comma causes a scan failure and is
therefore an effective delimiter.)

If the scan failed on any character other than a right parenthesis, R1 will point to the character where the failure occurred. If the scan ended on a right parenthesis, R1 will point to the character following the parenthesis and the V (overflow) bit in the status register will be set. If the V-bit is set on any parameter other than the last or is cleared on the last parameter, a mismatched parenthesis error has occurred.

Errors may occur in the evaluation as follows:

i) If a parenthesis (other than the special case mentioned above) is missing or improperly placed in an expression to be evaluated, a fatal error call is made from EVAL. The user routine will not regain control.

ii) If a storage overflow occurs due to an evaluation, a fatal error call is made. The user routine will not regain control.

iii) If a non-existent variable is referenced, the value of the non-existent variable is assumed to be zero for the evaluation, and the user routine will regain control normally.

Note: The EVAL routine is fully recursive and uses all registers.

2.5 Error Handling

Fatal error codes should be used. Error code 28 is declared global as ARGERR and can be used to indicate the wrong number of arguments. Other error codes should be used as 63 downwards (DOS-BASIC currently uses 0-36). Note that 63 is used to indicate invalid parameter value and 62 is used for CAMAC branch error.
TITLE FUNDEN

; DEMONSTRATION PROGRAM FOR "NEW" FUNCTION
; CALLED AS "NEW(EXPRESSION 1, EXPRESSION 2)
; MUST HAVE TWO ARGUMENTS, OTHERWISE GIVES ERROR 28
; "FIX" ES EACH ARGUMENT, THEN DOES INTEGER ADD
; GIVES ERROR CODE 63 IF ADDITION OVERFLOWS
; RETURNS FLOATING POINT CONVERSION OF INTEGER SUM

; GLOBL ARGERR, FIX, FLT
; GLOBL FNMI, FADI

; NAME ASSIGNMENT

043653 FNMI = 43653 ; CODE FOR "NEW"

; REGISTER ASSIGNMENTS

000020 R0 = 20 ; FREE
000021 R1 = 21 ; CONTAINS NO OF ARGUMENTS
000022 R2 = 22 ; FREE
000023 R3 = 23 ; FREE
000024 R4 = 24 ; FREE
000025 R5 = 25 ; FREE
000026 SP = 26 ; STACK POINT
000027 PC = 27 ; PROGRAM COUNTER

000000 022127 FADI: CMP R1,#2 ; SHOULD BE TWO ARGS
000002
000004 011022 BNE NEW01 ; ERROR IF NOT TWO
000006 312622 MOV (SP)+,R2 ; SECOND
000010 312633 MOV (SP)+,R3 ; ARGUMENT
000014 022329' FIX
000016 312621 MOV R0,R1 ; SAVE TEMPORARILY
000020 312602 MOV SP,R0 ; STACK NOT OK FOR RETURN
000024 032322 MOV (R0)+,R2 ; GET
000028 012033 MOV (R0)+,R3 ; FIRST
000030 312024 MOV (R0)+,R4 ; ARGUMENT
000032 032329' FIX
000036 362031 ADD R0,R1 ; DO ADDITION
000034 102405 BUS NEW02 ; ERROR IF OVERFLOW
000036 313622 MOV SP,R0 ; READY FOR FLOAT
000038 032243' FLT
000042 316627 MOV 6(SP),PC ; FLOAT R1 ONTO STACK
000026
000046 230822'NEW01: ARGERR
000052 124577 NEW02: .WORD 104577 ;ERROR IN ARGUMENT LIST
000056 .END ;ERROR CODE 63

PAGE 001

000000 ERRORS
Appendix 3 (contd.)

TABLE 1

CORRESPONDENCE TABLE FOR MACHINE-CODE MODULE NAMES

| A  | 1  | 0  | 48 |
| B  | 2  | 1  | 49 |
| C  | 3  | 2  | 50 |
| D  | 4  | 3  | 51 |
| E  | 5  | 4  | 52 |
| F  | 6  | 5  | 53 |
| G  | 7  | 6  | 54 |
| H  | 8  | 7  | 55 |
| I  | 9  | 8  | 56 |
| J  | 10 | 9  | 57 |
| K  | 11 |
| L  | 12 |
| M  | 13 |
| N  | 14 |
| O  | 15 |
| P  | 16 |
| Q  | 17 |
| R  | 18 |
| S  | 19 |
| T  | 20 |
| U  | 21 |
| V  | 22 |
| W  | 23 |
| X  | 24 |
| Y  | 25 |
| Z  | 26 |
### Table 2: Some DOS-Basic Internal Routines

<table>
<thead>
<tr>
<th>Name</th>
<th>Trap or Call</th>
<th>Input</th>
<th>Output</th>
<th>Regs. Used</th>
</tr>
</thead>
<tbody>
<tr>
<td>ATOF</td>
<td>ATOF=104406</td>
<td>Pointer to ASCII string in R1</td>
<td>3 word F.P. Number stored where RO pointed at entry. RI points to 1st illegal character.</td>
<td>ALL</td>
</tr>
<tr>
<td>ATOI</td>
<td>ATOI=104410</td>
<td>Pointer to ASCII in R1</td>
<td>Number in RO. RI points to 1st illegal character.</td>
<td>ALL</td>
</tr>
<tr>
<td>ITOA</td>
<td>ITOA=104412</td>
<td>Number in R1</td>
<td>Pointer to output area in R0</td>
<td>ALL</td>
</tr>
<tr>
<td>IMUL</td>
<td>IMUL=104416</td>
<td>Numbers in RO,R1</td>
<td>High order in RO. Low order in R1</td>
<td>ALL</td>
</tr>
<tr>
<td>ADDFX</td>
<td>ADDFX=104420</td>
<td>Pointers to numbers in RO,R1</td>
<td>Result where RO pointed at entry</td>
<td>ALL</td>
</tr>
<tr>
<td>SUBFX</td>
<td>SUBFX=104422</td>
<td>&quot;</td>
<td>&quot;</td>
<td>&quot;</td>
</tr>
<tr>
<td>NEGFX</td>
<td>NEGFX=104424</td>
<td>Pointer to number in R1</td>
<td>&quot;</td>
<td>&quot;</td>
</tr>
<tr>
<td>DIVFX</td>
<td>DIVFX=104426</td>
<td>Pointers to numbers in RO,R1</td>
<td>&quot;</td>
<td>&quot;</td>
</tr>
<tr>
<td>MULFX</td>
<td>MULFX=104430</td>
<td>&quot;</td>
<td>Same condition codes as CMP instruction</td>
<td>&quot;</td>
</tr>
<tr>
<td>CMPFX</td>
<td>CMPFX=104434</td>
<td>&quot;</td>
<td>&quot;</td>
<td>&quot;</td>
</tr>
<tr>
<td>FIX</td>
<td>FIX=104440</td>
<td>Number in R2,R3,R4</td>
<td>Result in RO</td>
<td>R0,2,3,4</td>
</tr>
<tr>
<td>FLT</td>
<td>FLT=104436</td>
<td>Number in R1</td>
<td>Result where RO pointed at entry</td>
<td>RO-R4</td>
</tr>
<tr>
<td>FIXD</td>
<td>JSR PC,FIXDØ Ø</td>
<td>Number in R2, R3,R4(R3 = m.s.)</td>
<td>Result in R3,R2</td>
<td>R0,2,3,4</td>
</tr>
<tr>
<td>FLTD</td>
<td>JSR PC,FLTDØ Ø</td>
<td>Number in R3,R2 (R3 = m.s.)</td>
<td>Result where RO pointed at entry</td>
<td>R0-R4</td>
</tr>
<tr>
<td>EVAL</td>
<td>EVAL=104536</td>
<td>Pointer to ASCII string in R1. R5 as for DOS-BASIC</td>
<td>Result in R2,3,4. RI points to 1st illegal character</td>
<td>ALL</td>
</tr>
</tbody>
</table>
Example 1  Illustrates General Features

1. The program exists on disc as the file DFKI.BAS.
   This is loaded to core by OLD DFKI.

2. Line no.
   110    Use of multiple command statement
   130    User function FNA will be DEFINed later
   260    N is a rounded integer
   270    The terminal comma suppresses the new line and moves
           printing to the beginning of the next fourteen
           character zone.
   280    The semicolon suppresses new line and spacing
   290    PRINT gives a new line

3. Line no.
   10     The function FNA is DEFINed

4. The program is RUN
RUN BASIC
DOS BASIC
READY
OLD DFK1
READY

LIST

80 REM --- PROGRAM TO ANALYSE RANDOM DISTRIBUTIONS ---
99 REM ANALYSE INTO 20 SLOTS
100 DIM A(19)
105 REM SET SLOTS TO ZERO
110 FOR I=0 TO 19 : LET A(I)=0 : NEXT I
119 REM HISTOGRAM INTO SLOTS
120 FOR I=1 TO 1000
129 LET J=INT(20*RND(0))
130 LET A(J)=A(J)+1
150 NEXT I
199 REM FIND LARGEST SLOT VALUE
200 LET M=0
210 FOR I= 0 TO 19 : IF A(I)>M THEN LET M=A(I)
220 NEXT I
249 REM CONVERT TO MAX OF 56 (IF NECESSARY)
258 FOR I=0 TO 19 : IF M>56 THEN LET A(I)=(A(I)/M)*56
268 LET N=INT(A(I)+0.5)
279 PRINT A(I),
288 FOR J=1 TO N : PRINT "*"; : NEXT J
298 PRINT
399 NEXT I
999 END
READY

10 DEF FNA(X)=RND(0)

RUN
38.35616
93.12329
41.42466
39.89041
56
38.35616
33.75342
36.85479
40.65753
39.89041
37.58904
31.45205
32.21918
42.9589
36.85479
36.82192
31.45205
42.19178
32.9863
39.89041
STOP AT LINE 999
READY
Example 2  Use of DATA, READ, RESTORE

1. Note that the DATA statements may be scattered at will through the program.

2. No END or STOP statement is needed. An END is forced when one runs out of statements.

3. Line no.  
   14 Use of octal data  
   40 READ takes items sequentially from the list  
   70 In the PRINT command note the use of semicolon for space suppression and comma to position printing.

4. The program is RUN

5. Line no.  
   45 The RESTORE command is added. This forces the DATA command to be read from the beginning.

6. The program is RUN
RUN BASIC

DOS BASIC
READY
OLD DFK6
READY

LIST

10 REM --- PROGRAM TO ILLUSTRATE DATA, READ, RESTORE ---
14 DATA #77, 4, 3998765
20 DIM A(1, 4)
30 FOR I = 0 TO 1
34 DATA 1.987E+176, -8.765E-37, 0.1237648652224E-9
40 FOR J = 0 TO 4 : READ A(I, J) : NEXT J
50 NEXT I
60 FOR I = 0 TO 1
70 FOR J = 0 TO 4 : PRINT I, J, A(I, J) : NEXT J
74 DATA 31.98236597, 0.8, 0.1234567
80 NEXT I
104 DATA 4, 3, 2
READY

RUN
0 0 63
0 1 4
0 2 3998765
0 3 .1987E 177
0 4 -.8765E-36
1 0 .1237649E-9
1 1 31.98237
1 2 .8
1 3 .1234567
1 4 4

STOP AT LINE 104
READY

45 RESTORE

RUN
0 0 63
0 1 4
0 2 3998765
0 3 .1987E 177
0 4 -.8765E-36
1 0 63
1 1 4
1 2 3998765
1 3 .1987E 177
1 4 -.8765E-36

STOP AT LINE 104
READY
<table>
<thead>
<tr>
<th>Line no.</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>10</td>
<td>Use of octal expression</td>
</tr>
<tr>
<td>20</td>
<td>Use of octal output</td>
</tr>
</tbody>
</table>
LIST

5 REM --- PROGRAM TO ILLUSTRATE BOOLEAN FUNCTIONS ---
10 LET P=#52525252
20 PRINT OCT$(P), OCT$(NER(P, #77777777))
30 PRINT OCT$(AND(P, #77777780)), OCT$(ORF(P, #7777780))
40 PRINT
45 REM --- AND SHIFT FUNCTIONS ---
46 LET B0=#2000000000
50 FOR I=0 TO 32
60 PRINT OCT$(LFT(I, I)), OCT$(RGT(B0, I))
70 NEXT I
80 END

READY

RUN
#52525252      #25252525
#5252800       #52777752

#1            #2000000000
#2            #1000000000
#4            #4000000000
#10           #2000000000
#20           #1000000000
#40           #4000000000
#100          #2000000000
#200          #1000000000
#400          #4000000000
#1000         #2000000000
#2000         #1000000000
#4000         #4000000000
#10000        #2000000000
#20000        #1000000000
#40000        #4000000000
#100000       #2000000000
#200000       #1000000000
#400000       #4000000000
#1000000      #2000000000
#2000000      #1000000000
#4000000      #4000000000
#10000000     #2000000000
#20000000     #1000000000
#40000000     #4000000000
#100000000    #2000000000
#200000000    #1000000000
#400000000    #4000000000
#1000000000   #2000000000
#2000000000   #1000000000
#4000000000   #4000000000
#10000000000  #2000000000
#20000000000  #1000000000
#40000000000  #4000000000
#100000000000 #2000000000
#200000000000 #1000000000
#400000000000 #4000000000
#1000000000000 #2000000000
#2000000000000 #1000000000
#4000000000000 #4000000000
#10000000000000 #2000000000
#20000000000000 #1000000000
#40000000000000 #4000000000
#100000000000000 #2000000000
#200000000000000 #1000000000
#400000000000000 #4000000000
#1000000000000000 #2000000000
#2000000000000000 #1000000000
#4000000000000000 #4000000000
#10000000000000000 #2000000000
#20000000000000000 #1000000000
#40000000000000000 #4000000000
#100000000000000000 #2000000000
#2000000000000000000 #1000000000
#4000000000000000000 #4000000000
#10000000000000000000 #2000000000
#200000000000000000000 #1000000000
#4000000000000000000000 #4000000000
#10000000000000000000000 #2000000000
#200000000000000000000000 #1000000000
#4000000000000000000000000 #4000000000
#10000000000000000000000000 #20
#200000000000000000000000000 #10
#4000000000000000000000000000 #4
#10000000000000000000000000000 #2
#200000000000000000000000000000 #1
#0               #0

STOP AT LINE 80
Example 4

CAMAC functions

1. Line no.
   15 Generation of bit moving up the register
   20 CAMAC control variable
   30 CAMAC acquisition variable

RUN BASIC

DOS BASIC
READY
OLD DFK?
READY
LIST

5 REM --- A TEST FOR A CAMAC REGISTER ---
6 REM TEST IS DONE BIT BY BIT
10 FOR I=0 TO 23
15 LET A=LFT(I,1)
20 LET CAM(0,2,0.16)=A
30 LET B=CM(0,2,0,0)
40 IF A=B THEN GOTO 50
41 REM OUTPUT IF FAILURE HAS OCCURED
45 PRINT I, OCT$(A), OCT$(B)
50 NEXT I
READY

RUN

STOP AT LINE 50
READY
Example 5

Subroutines, CAMAC status register

1. **Line no.**
   
   20  final semicolon to suppress new line
   30  INPUT of variables
   40  CAMAC DO subroutine
   50  Use of GOSUB to call output of CSR.
   1010  Output in binary of contents of CSR.
   1030  Negative STEP in FOR command

2. The program is RUN. Note the question mark used to prompt INPUT.
RUN BASIC

DOS BASIC
READY
OLD DFK9
READY

LIST

10 REM --- PROGRAM TO TEST CAMAC STATUS RETURN ---
20 PRINT "CRATE, MODULE, SUBADDRESS";
30 INPUT C, N, A
40 DO CAM(C, N, A, 8)
50 GOSUB 1000
60 LET CAM(C, N, A, 16)="#52525252525252"
70 GOSUB 1000
80 LET B=CAM(C, N, A, 0)
90 GOSUB 1000
100 STOP
1000 PRINT "CSR(BINARY)=";
1010 FOR Z=15 TO 0 STEP -1 : PRINT CHR$(#60+CSR(Z)) : NEXT Z
1020 PRINT
1030 RETURN
1040 END
READY

RUN
CRATE, MODULE, SUBADDRESS?0, 2, 0
CSR(BINARY)=0000000010000000
CSR(BINARY)=0001000010000000
CSR(BINARY)=0001000010000000

STOP AT LINE 100
READY
Example 6

1. **Line no.**
   
   20   ERASE file - to ensure it does not exist
   30   ASSIGN file for PUT data
   50   PUT command
   60   PUT command in multiple command
   70   Close file assigned to PUT before it can be assigned to GET
   110  GET command
   120  GET command in multiple command

2. The file will automatically be closed at the end of the program and is therefore not closed after GET.

```
LIST

10 REM --- PROGRAM TO ILLUSTRATE PUT AND GET ---
20 ERASE PWR5.DAT
30 ASSIGN PUT # PWR5
40 LET S0=0 : LET S1=9
50 PUT S0,S1
60 FOR I=S0 TO S1 : PUT SIN(I/S1) : NEXT I
70 ASSIGN PUT:CLOSE
100 ASSIGN GET # PWR5
110 GET X1,X2
120 FOR Q=X1 TO X2 : GET D :PRINT D :NEXT Q
```

```
RUN

0
.1108826
.2203977
.3271947
.4299564
.5274154
.6183698
.7016979
.7763711
.8414707

STOP AT LINE 120
READY
```
### Example 7

<table>
<thead>
<tr>
<th>Line no.</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>100</td>
<td>ERASE in case already exists</td>
</tr>
<tr>
<td>110</td>
<td>ASSIGN printing (AOP) to this file</td>
</tr>
<tr>
<td>140</td>
<td>Use of non-integer values in FOR command</td>
</tr>
</tbody>
</table>

2. The program is run

3. **DOS-BASIC is KILLED** and PIP run

4. PIP is used to output the file of text
LIST

50 REM --- PROGRAM TO SHOW PRINT-OUT TO DATASET ---
60 REM --- PRINT-OUT IS OBTAINED BY USE OF PIP ---
100 ERASE MYTEXT.TXT
110 ASSIGN AOP # MYTEXT.TXT
120 PRINT " TEST OF MATHS FUNCTIONS "
130 PRINT "VALUE", "LOG", "EXP", "ATN", "COS"
140 FOR I=0.1 TO 0.9 STEP 0.05
150 PRINT I, LOG(I), EXP(I), ATN(I), COS(I)
160 NEXT I
READY

RUN

STOP AT LINE 160
READY

°C
.KILL

$RUN PIP
PIP V08-09
#KB:<MYTEXT.TXT

<table>
<thead>
<tr>
<th>VALUE</th>
<th>LOG</th>
<th>EXP</th>
<th>ATN</th>
<th>COS</th>
</tr>
</thead>
<tbody>
<tr>
<td>0.1</td>
<td>-2.302585</td>
<td>1.105171</td>
<td>.9966865E-1</td>
<td>.9950042</td>
</tr>
<tr>
<td>0.15</td>
<td>-1.89712</td>
<td>1.161834</td>
<td>.1488899</td>
<td>.9887711</td>
</tr>
<tr>
<td>0.2</td>
<td>-1.609438</td>
<td>1.221403</td>
<td>.1973956</td>
<td>.9800666</td>
</tr>
<tr>
<td>0.25</td>
<td>-1.386294</td>
<td>1.284025</td>
<td>.2449787</td>
<td>.9689124</td>
</tr>
<tr>
<td>0.3</td>
<td>-1.203973</td>
<td>1.349859</td>
<td>.2914568</td>
<td>.9553365</td>
</tr>
<tr>
<td>0.35</td>
<td>-1.049822</td>
<td>1.419068</td>
<td>.3366748</td>
<td>.9393727</td>
</tr>
<tr>
<td>0.4</td>
<td>-0.916207</td>
<td>1.491825</td>
<td>.3805064</td>
<td>.921061</td>
</tr>
<tr>
<td>0.45</td>
<td>-0.7985077</td>
<td>1.568312</td>
<td>.4228539</td>
<td>.9004471</td>
</tr>
<tr>
<td>0.5</td>
<td>-0.6931472</td>
<td>1.648721</td>
<td>.4636476</td>
<td>.8775826</td>
</tr>
<tr>
<td>0.55</td>
<td>-0.597837</td>
<td>1.733253</td>
<td>.5028432</td>
<td>.8525245</td>
</tr>
<tr>
<td>0.6</td>
<td>-0.5108256</td>
<td>1.822119</td>
<td>.5404195</td>
<td>.8253356</td>
</tr>
<tr>
<td>0.65</td>
<td>-0.4307829</td>
<td>1.915541</td>
<td>.5763752</td>
<td>.7960838</td>
</tr>
<tr>
<td>0.7</td>
<td>-0.3566749</td>
<td>2.013753</td>
<td>.610726</td>
<td>.7648422</td>
</tr>
<tr>
<td>0.75</td>
<td>-0.2876821</td>
<td>2.117</td>
<td>.6435011</td>
<td>.7316889</td>
</tr>
<tr>
<td>0.8</td>
<td>-0.2231436</td>
<td>2.225541</td>
<td>.6747409</td>
<td>.6967067</td>
</tr>
<tr>
<td>0.85</td>
<td>-0.1625189</td>
<td>2.339647</td>
<td>.7044941</td>
<td>.6599831</td>
</tr>
<tr>
<td>0.9</td>
<td>-0.1053605</td>
<td>2.459603</td>
<td>.7328151</td>
<td>.62161</td>
</tr>
</tbody>
</table>
Example 8

INPUT from a dataset

1. Line no.
   100 take INPUT(AIP) from the card-reader
   130 Read a card and assign the data to the given variables

LIST

10 REM --- PROGRAM TO SHOW INPUT FROM DATASET ---
100 ASSIGN AIP # CR:
110 DIM A(2,4)
120 FOR I=0 TO 2
130 INPUT A(I,0),A(I,1),A(I,2),A(I,3),A(I,4)
140 NEXT I
200 FOR I=0 TO 2
210 FOR J=0 TO 4
220 PRINT I,J,A(I,J)
230 NEXT J
240 PRINT
250 NEXT I
READY

RUN
0 0       63
0 1       54
0 2       45
0 3       36
0 4       33
1 0       135647E 533
1 1       7.6
1 2       342E-451
1 3       2.6
1 4       5
2 0       0
2 1       2
2 2       3
2 3       6
2 4       47

STOP AT LINE 250
READY
Example 9  Program "Chaining"

1. Using one of the previous examples we can cause program chaining by inserting the stored command RUN DFK5.

```
10 REM --- PROGRAM TO ILLUSTRATE DATA, READ, RESTORE ---
14 DATA #77, 4, 3998765
20 DIM A(1,4)
30 FOR I=0 TO 1
34 DATA 1.987E+176, -8.765E-37, 0.1237649652224E-9
40 FOR J=0 TO 4 : READ A(I,J) : NEXT J
50 NEXT I
60 FOR I=0 TO 1
70 FOR J=0 TO 4 : PRINT I;J,A(I,J) : NEXT J
74 DATA 31.98236597, 0.8, 0.1234567
80 NEXT I
104 DATA 4,3,2
READY

90 PRINT "DFK5 CALLED"
100 RUN DFK5
```

```
RUN
  0  0       63
  0  1       4
  0  2       3998765
  0  3       1987E 177
  0  4       8765E-36
  1  0       1237649E-9
  1  1       31.98237
  1  2       8
  1  3       1234567
  1  4       4
DFK5 CALLED
  0
 .1108826
 .2203977
 .3271947
 .4299564
 .5274154
 .6183698
 .7016979
 .7763711
 .8414787
STOP AT LINE 120
READY
```
Example 10  SAVE and OLD

1. The program listed can be stored on the file FRED.BAS by use of
   SAVE FRED

2. Program DFK6 can be loaded by use of
   OLD DFK6

   Note that the program could have been loaded and RUN by
   RUN DFK6

3. Reload the program previously saved as FRED.BAS.
LIST

10 REM --- PROGRAM TO TEST CAMAC STATUS RETURN ---
20 PRINT "CRATE, MODULE, SUBADDRESS";
30 INPUT C, N, A
40 DO CAM(C, N, A, 8)
50 GOSUB 1000
60 LET CAM(C, N, A, 16)#52525252
70 GOSUB 1000
80 LET B=CAM(C, N, A, 0)
90 GOSUB 1000
100 STOP
1000 PRINT "CSR(BINARY)=";
1010 FOR Z=15 TO 0 STEP -1 : PRINT CHR$(#60+CSR(Z)) : NEXT Z
1020 PRINT
1030 RETURN
1040 END

SAVE FRED
READY

OLD DFK6
READY

LIST

10 REM --- PROGRAM TO ILLUSTRATE DATA, READ, RESTORE ---
14 DATA #77, 4, 3998765
20 DIM A(1, 4)
30 FOR I=0 TO 1
34 DATA 1.987E+176, -8.765E-37, 0.1237648652224E-9
40 FOR J=0 TO 4 : READ A(I, J) : NEXT J
50 NEXT I
60 FOR I=0 TO 1
70 FOR J=0 TO 4 : PRINT I, J, A(I, J) : NEXT J
74 DATA 31.98236597, 0.8, 0.1234567
80 NEXT I
104 DATA 4, 3, 2
READY

OLD FRED
READY

LIST

10 REM --- PROGRAM TO TEST CAMAC STATUS RETURN ---
20 PRINT "CRATE, MODULE, SUBADDRESS";
30 INPUT C, N, A
40 DO CAM(C, N, A, 8)
50 GOSUB 1000
60 LET CAM(C, N, A, 16)#52525252
70 GOSUB 1000
80 LET B=CAM(C, N, A, 0)
90 GOSUB 1000
100 STOP
1000 PRINT "CSR(BINARY)=";
1010 FOR Z=15 TO 0 STEP -1 : PRINT CHR$(#60+CSR(Z)) : NEXT Z
1020 PRINT
1030 RETURN
1040 END

READY
MODIFICATIONS TO DOS-BASIC

1. INTRODUCTION

BASIC has been added to or modified to include:

a) TIME
b) DATE
c) STRINGS
d) COMPUTED GOTO
e) COMPUTED GOSUB
f) BETTER "SAVE" COMMAND
g) MACHINE INDEPENDENCE

2. TIME AND DATE

These are read by means of the standard function CLK. If no parameters are specified, or if one parameter of value zero is specified, then the time is read. If one parameter of value 1 is specified, then the date is read.

The time is returned as a single integer representing the number of 20 ms "ticks" since midnight. Whenever the internal clock reached midnight, this integer is automatically reset to zero.

The date is returned as a single integer whose value is calculated as follows:

\[ \text{DATE} = ((\text{YEAR} - 1970) \times 1000) + (\text{DAYS SINCE JAN. 1st}) \]

Whenever the internal clock passes midnight this number is automatically incremented by 1.

(This is the format used by D.E.C. and on Geneva bus tickets!)

Examples:

100 LET A = CLK will read the time
110 LET B = CLK(0) " " " "
120 LET C = CLK(1) will read the date

or to wait for \( W \) secs (neglecting midnight)

1 LET A = CLK
2 IF CLK-A=\( W \times 50 \) GOTO 2
3. CHARACTER STRING HANDLING

Facilities now exist for the input, comparison and output of character strings.

(a) Input

Character strings can now be typed instead of numbers in response to an INPUT request. A mixture of strings and numbers can be typed on the same line; each variable must be separated by a comma.

Examples:

(1) 100 INPUT A
    will cause a question mark to be output. In response, the user may type: FRED (carriage return)
    in which case the string "FRED" will be stored in the variable A.

(2) 100 INPUT A, B, C
    with a user response of
    ANN, 5, JILL (carriage return)
    will cause "ANN" to be stored in A, the number 5 in B, and "JILL" in C.

(b) Comparison of strings

A string can be used as a "primary" in a BASIC program by enclosing it in quotes ("). Thus comparisons can be performed.

Examples:

(1) 100 LET A = "FINISH"
    110 INPUT B
    120 IF A = B GOTO 500
    would cause control to pass to line 500 if the user typed FINISH.

(2) 100 INPUT A
    110 IF A = "FINISH" GOTO 500
    would have exactly the same effect.

(c) Printing of strings

This can be done by using STR$(variable) as an item in the PRINT statement.

Example:

100 LET A = "TEST"
110 PRINT STR$(A)
    would print TEST.
(d) Notes

(1) A string must start with an alphabetic character
(2) Strings must not contain commas, quotes, or colons
(3) Only the first six characters of a string are stored internally
(4) Strings can be introduced into a program by means of the READ and
DATA statements in exactly the same way as for INPUT.

4. COMPUTED GOTO AND GOSUB

i.e. GOTO expression
     GOSUB expression

e.g. GOTO J+10*I
     GOSUB F+Q

5. BETTER SAVE COMMAND

The command

SAVE dataset

will delete the dataset if it already exists. This gets over the problem of
fault messages associated with overflowing an already existing file.

6. NOTES ON INCLUSION OF USER MACHINE-CODE MODULES

(FOR SPECIALISTS ONLY - Ref: Page 33 of DOS-BASIC)

(a) The module TIMDAT uses FNM10, FAD10 for CLK
(b) The link sequence should include the module TIMDAT between BASCAM and
TRPHDL. It is usual to concatenate the various .OBJ modules using PIP

Example:

A.OBJ<LOGOPS.OBJ,BASCAM.OBJ,TIMDAT.OBJ,TRPHDL.OBJ,.....

and then link with

BASIC,KB:/CR<FUNDEM,A/CC/T:50000/E