UNPL 1512

Bell Laboratories Cover Sheet for Technical Memorandum

The information contained herein is for the use of employees of Bell Laboratories and is not for publication (see GEI 13.9-3)

Title: **BITE users guide.**

Date: **October 8, 1979**

Other Keywords: **BASIC**
Interpreters
BITE
Automatic Testing

TM: **79-2425-4**

Author(s)	Location	Extension	Charging Case: 20239-7048
J. P. Hawkins	**WH 8C-001**	**4610**	Filing Case: **40295-2**

ABSTRACT

BITE (BASIC Interpreter for Testing and Engineering) is a BASIC language interpreter designed for use in automated test systems controlled by PDP-11 microcomputers. The interpreter implements an extended instruction set designed for instrument control using the IEEE 488 Instrument Bus.

The major benefit of *BITE* to the user is that the development time for algorithms is decreased markedly since the user may run his/her program immediately upon making a change without recompiling.

Nearly the entire process of development and debugging of a test program requires only a knowledge of *BITE* as opposed to familiarity with compilers, assemblers, linking editors, assembly language debuggers, archiving, etc. required by present techniques. *BITE* includes self contained debugging aids.

Another significant benefit afforded by *BITE* lies in the users ability to extend the instruction set to provide control of new hardware configurations. The task of configuring Documentation for this purpose is provided in a Technical Memorandum 79-2425-5 entitled "Guide to the Internals of BITE" by R.B. Drake.

This memo describes the syntax of the language, the modes of operation of the interpreter, describes useful programming techniques, and shows several examples.

Pages Text: 13	Other: 4	Total: 17	
No. Figures: 0	No. Tables: 0	No. Refs.: 4	

DISTRIBUTION
(REFER GEI 13.9-3)

COMPLETE MEMORANDUM TO

CORRESPONDENCE FILES

OFFICIAL FILE COPY
PLUS ONE COPY FOR
EACH ADDITIONAL FILING
CASE REFERENCED

DATE FILE COPY
(FORM E-1328)

10 REFERENCE COPIES

BISHOP,J DANIEL
+BOBOS,VICTOR B
+BOURNE,STEPHEN R
CLARKE,P W
DI PIAZZA,G
DISHMAN,JOHN M
>DRAKE,RICHARD B
FICKENSCHER,H
FISHER,P D
HAMILTON,B H
KELLY,L C
+LAHTI,J N
LITWACK,B
LOMBARDI,J A
LUER,H J
MASSEY,R P
MC ELROY,J D
MENKES,H E
MICHELET,ROBERT W
MORRISON,W J
MOTTEL,SAMUEL
OSTAPIAK,R
RIDDLEBERGER,C O
SCUDERI,R
SHENNUM,R H
SMITH,DONALD H
WADLINGTON,J C
WALK,R
WISNEWSKI,STANLEY E
WITT,EUGENE F
30 NAMES

COVER SHEET ONLY TO

CORRESPONDENCE FILES

4 COPIES PLUS ONE
COPY FOR EACH FILING
CASE

AAGESEN,JOHN
ABATEMARCO,TERESA M
ABATE,JOSEPH
ACKERMAN,A FRANK

COVER SHEET ONLY TC

ACKERMAN,J T
ACKROFF,JOHN M
AHO,ALFRED V
AHRENS,RAINER R
AHUJA,SUDHIR R
ALBAGLI,V R
ALBERALLA,RICHARD J
ALBERTS,BARBARA A
ALCALAY,D
ALEXIS,A D,JR
ALKONS,FREDERICK
ALLISON,C E,JR
ALTHAMER,CATHERINE V
ALT,DOROTHY L
AMARILE,GEORGE R
AMITAY,N
AMOSS,JOHN J
AMRON,IRVING
ANDERSON,C R
ANDERSON,KATHRYN J
ANDERSON,R E
ANDERSON,R R
ANDERSON,ROBERT V
ANDREWS,W J
ANTOLICK,DAVID R
APPELBAUM,MATTHEW A
ARCHER,RUSSELL E,JR
ARMSTRONG,F O,JR
ARNOLD,GEORGE W
ARNOLD,JAMES Q
ARNOLD,PHYLLIS A
ARNOLD,THOMAS F
ARTIS,H P
ARVIDSON,W P
ASELTINE,EDWARD G
ASMUTH,RICHARD L
ASTHANA,ABHAYA
ATAL,BISHNU S
AXELSON,A L
BABU,RAJESH RATILAL
BACCASH,JEANNE M
BACH,MAURICE J
BACKUS,C P,SR
BAILEY,JAMES R
BAILY,DAVID E
BAKER,DONN
BAKER,MITCHELL B
BALASHEK,S
BALDWIN,GEORGE L
BALENSON,CHRISTINE M
BALLANCE,ROBERT A
BARBATO,ROBERT R
BARNS,R L
BAROFSKY,ALLEN
BARON,ROBERT V
BARR,DAVID L
BATISTONI,P J
BATTAGLIA,FRANCES

COVER SHEET ONLY TO

BAUER,BARBARA T
BAUER,H C
BAUER,HELEN A
BAUGH,C R
BAXTER,LESLIE A
BAYER,D L
BEBLO,WILLIAM
BECERRA,PEDRO D
BECKER,JACOB I
BECKER,RICHARD A
BECKETT,J T
BECK,R P
BEGLEY,ALOYSIUS A
BEIGHLEY,KEITH A
BENCO,DAVID S
BENISCH,JEAN
BENNETT,RAYMOND W
BENNETT,RICHARD L
BENNETT,WILLIAM C
BENOWITZ,P
BENSING,JAMES EDWARD
BERENBAUM,ALAN
BERGERON,R F,JR
BERGLAND,G D
BERKEY,M A
BERK,DONALD A
BERNHARDT,RICHARD C
BERNOSKE,BEVERLY G
BERNSTEIN,DANIELLE R
BERNSTEIN,L
BERNSTEIN,PAULA R
BERN,DAVID
BERRIN,LLOYD
BERRYMAN,R D
BERTH,R P
BERZINS,ALEXANDER H
BEYER,JEAN-DAVID
BHATIA,RAJIV
BIANCHI,M H
BICKFORD,NEIL B
BILASH,TIMOTHY D
BILOWOS,R M
BIREN,IRMA B
BISHOP,VERONICA L
BITTNER,B B
BITTRICH,MARY E
BLAKE,GARY D
BLAZIER,S D
BLECHMAN,RONALD I
BLEIER,JOSEF
BLINN,J C
BLOSSER,PATRICK A
BLUMER,THOMAS P
BLUM,MARION
BOCKUS,ROBERT J
BOCK,NANCY E
BODDIE,JAMES R
BODEN,P J

COVER SHEET ONLY TO

BOEHM,EARL W
BOEHM,KIM R
BOGART,P J
BOGART,THOMAS G
BOGINSKY,LINDA S
BOIVIE,RICHARD H
BOLINSKI,NANCY V DEVLIN
BONACHEA,R N
BONANNI,L E
BOND,F C
BOND,HOLTON C,JR
BORG,KEVIN E
BORISON,ELLEN A
BOSE,DEBASISH
BOSTON,RONALD E
BOSWELL,PAULA S
BOTHUR,R H
BOWYER,L RAY
BOYCE,W M
BOYER,PHYLLIS J
BRADFORD,EDWARD G
BRADLEY,M HELEN
BRADLEY,R H
BRANDAUER,C M
BRANDT,RICHARD B
BRAUNE,DAVID P
BRAUN,DAVID A
BREILAND,JOHN R
BRENSKI,EDWIN P
BRESLER,RENEE A
BRIGGS,GLORIA A
BRITT,WARREN D
BROAD,MARTHA M
BRONSTEIN,N
BROOKS,CATHERINE ANN
BROSS,JEFFREY D
BROVMAN,INNA
BROWNING,JASON DAVID
BROWNLOW,D L
BROWN,C W
BROWN,ELLINGTON L
BROWN,LAURENCE MC FEE
BROWN,MARK S
BROWN,STUART G
BROWN,W R
BROWN,W STANLEY
BUCHANAN,D N E
BUCK,I D
BULLEY,R M
<BURCKBUCHLER,F V
BURGESS,JOHN T,JR
BURIC,MILORAD R
BURKE,MICHAEL E
BURKE,R J
BURNETTE,W A
BURNETT,DAVID S
BURNET,ROSE M
BUROFF,STEVEN J

COVER SHEET ONLY TO

BUSCH,KENNETH J
BUTLETT,DARRELL L
BUTTON,BEVERLY
BYORICK,ROBERT S
BYRNE,EDWARD R
CAFFARRA,M G
CALL,PETER F
CALVERT,KENNETH L
CAMPBELL,JERRY H
CAMPBELL,MICHAEL R
CANADAY,BUDD H
CANDREA,RONALD D
CANNATA,PHILIP E
CAREY,J E
CARPENTER,THOMAS L,III
CARRAN,JOHN H
CARR,DAVID C
CARTER,DONALD H
CASELLA,CHARLES R
CASPERS,BARBARA E
CATO,H E
CAVINESS,JOHN D
CELLER,GEORGE K
CERMAK,I A
CHAFFEE,N F
CHAI,D T
CHAMBERS,B C
CHAMBERS,J M
CHANEY,P W
CHANG,JO-MEI
CHAO,C
CHAPMAN,LESLEY A
CHAPMAN,W P,JR
CHAPPELL,S G
CHARLES,JOYCE
CHEE,T
CHELLIS,ALICIA L
CHENG-QUISPE,ENRIQUE
CHENG,Y
CHEN,E
CHEN,ROBERT
CHEN,STEPHEN
CHEN,YUNGKANG C
CHERRY,LORINDA L
CHEUNG,ROGER C
CHE,HER-DAW
CHIANG,T C
CHILDS,CAROLYN
CHODROW,M M
CHONG,PHEE
CHOWANIEC,F P
CHRISTENSEN,S W
CHRISTENSON,DENNIS A
CHRIST,C W,JR
CHUNG,MICHAEL
CHUNG,TSUNG-JEN BEN
CICHINSKI,STEVEN
CIEMINSKI,DEBRA F

...
1434 TOTAL

+ NAMED BY AUTHOR > CITED AS REFERENCE < REQUESTED BY READER (NAMES WITHOUT PREFIX WERE SELECTED USING THE AUTHOR'S SUBJECT OR ORGANIZATIONAL SPECIFICATION AS GIVEN BELOW)

MERCURY SPECIFICATION

COMPLETE MEMO TO:
242-SUP

COVER SHEET TO:
2425

COPLIE = COMPUTER INTERPRETERS AND EMULATORS
COPLSP = SPECIAL-PURPOSE COMPUTER PROGRAMMING LANGUAGES AND PROCESSORS
UNOSLS = LSI RELATED DOCUMENTS ONLY
UNPL* = PROGRAMMING LANGUAGES: GENERAL OR SURVEY PAPERS ONLY
UNSA* = UNIX SPECIAL APPLICATIONS/SURVEY DOCUMENTS

NO CORRESPONDENCE FILES
NO 1A-127

TM-79-2425-4
TOTAL PAGES 17

TO GET A COMPLETE COPY:

1. BE SURE YOUR CORRECT ADDRESS IS GIVEN ON THE OTHER SIDE.
2. FOLD THIS SHEET IN HALF WITH THIS SIDE OUT AND STAPLE.
3. CIRCLE THE ADDRESS AT RIGHT. USE NO ENVELOPE.
4. INDICATE WHETHER MICROFICHE OR PAPER IS DESIRED.

PLEASE SEND A COMPLETE

( ) MICROFICHE COPY ( ) PAPER COPY

TO THE ADDRESS SHOWN ON THE OTHER SIDE.

CONTENTS

1. INTRODUCTION 1
   1.1 General Description 1
   1.2 Micro and Mini Versions 1
   1.3 Scope of This memo 2

2. Conventions 2

3. Commands 3
   3.1 Standard Commands 3
   3.2 File Commands 7
   3.3 ATS Instrument Commands (Extended Instruction Set) 7

4. Functions 8
   4.1 Standard Functions 8
   4.2 Instrument Functions (Extended Set) 8

5. Modes of Operation 9
   5.1 Editor or Idle Mode 9
   5.2 Run Mode 9
   5.3 Immediate Execution Mode 9
   5.4 Single Step Mode 9

6. Interruption of program 9

7. Programming Techniques and Tools 9
   7.1 Program Segmentation 9
      7.1.1 Chaining 9
      7.1.2 Overlaying 10
   7.2 System Shell Control 11

8. Error Messages 12
   8.1 Standard Error Messages 12
   8.2 Test Set and Instrument Error Messages 13

9. Acknowledgement 13

10. APPENDIX: A 15
   10.1 SAMPLE FILE I/O PROGRAM 15

11. APPENDIX: B 16
   11.1 SAMPLE PROGRAM TO STEP VOLTAGE ON POWER SUPPLY 16
   11.2 WAIT FOR BUTTON PRESS 16
   11.3 EXERCISE SCANNER AND RELAYS 16

Bell Laboratories

subject: **BITE users guide.**
Case: **20239-7048**
File: **40295-2**

date: **October 8, 1979**

from: **J. P. Hawkins**
WH 2425
8C-001 x4610

TM: **79-2425-4**

MEMORANDUM FOR FILE

1. INTRODUCTION

1.1 General Description

BITE is a *BASIC* language interpreter designed for use with automated test equipment. The interpreter runs on PDP-11 mini- and micro-computers using the UNIX* operating system. *BITE* is distinguished from other *BASIC* interpreters in the following ways:

- *BITE* is written in the 'C' language. It is, therefore, portable and can be installed and used on any system with a standard 'C' compiler.
- The interpreter provides an extended set of commands and functions for controlling and reading electronic instruments using the IEEE 488 Buss. The extended instruction set can be expanded by the user who is knowledgeable in 'C' programming by implementation of "custom commands" [ref Drake].
- *BITE* can be executed from a script running on the Bourne Shell of UNIX. Thus the string manipulation of the Shell and the control and computational capabilities of *BITE* are combined to provide a uniquely powerful system.
- *BITE* accepts *BASIC* language programs using the original Dartmouth syntax with little or no modification required. Those features described above which differ from standard *BASIC* are extensions of *BASIC* rather than exceptions to the syntax rules.

1.2 Micro and Mini Versions

There are two versions of *BITE*. One version is designed for the PDP-11/70 UNIX environment and the other version is a "standalone" program which runs in the PDP-11/03 and is invoked from the LSX version of UNIX.

- The PDP-11/70 version has complete interface capabilities with the Bourne Shell as described below and all math functions are available.
- The other version is designed for the PDP-11/03 micro-computer. This version contains the extended instruction set for instrument control. Differences between this and the host version include omission of verbose error messages, Shell and system call facilities, omission of some math functions and reduced user working storage. The omissions of features are due to the drastically reduced memory availability on the PDP-11/03 as compared to that of the PDP-11/70 with memory management. Since most core space is used up, the PDP-11/03 version is "standalone" (operating system not resident) with a skeleton version

* UNIX is a Trademark of Bell Laboratories.

of UNIX for file I/O. This version must be loaded with a special loader program. *BITEX* which is included with the software release package.

1.3 Scope of This memo

In the following, the syntax of *BITE* is described. When a command differs in the two versions, this fact is noted explicitly. Programming techniques, including control of *BITE* by the Shell, are described.

2. Conventions

This Memo	All things enclosed in [] are optional.
expr	Any algebraic expression which could be a constant, variable, math function or a combination of same, separated by arithmetic operators as in: a+b*3.14*(4.4+c2*sin(b+s))+a(2,2) See "variables" and "math functions" below.
Operators	+,-,*,/ or ^ for addition, subtraction, multiplication, division or exponentiation in order of lowest to highest precedence. + and - have the same precedence and * and / have the same precedence. Parenthesis () around expressions forces the contents to be higher precedence than all parts of the expression outside those parenthesis. Note also that when the - is used as a unary it maintains its low precedence, hence the expression -2^2 yields -4 instead of 4. In all cases a good rule of thumb to insure precedence is to enclose the part of high precedence in parenthesis, thereby (-2)^2 yields 4.
Relationals	<, >, =, <=, >=, <> for less than, greater than, equality, less or equal, greater than or equal and not equal.
Source Path	When reference is made to a *BITE* source file (i.e. the *old* and *load* command), two directories are searched, the first being the current directory and then /usr/lib/bites which is a "pool" where shared programs should be stored. The /usr/lib/bites directory is analogous to the /usr/bin directory in UNIX.
Source Program Name	The source program name is twelve or less characters suffixed by a .b .
Statement	A basic statement consists of a line number (integer value between 1 and 32767) followed by a command, space and operand which follows the syntax governed by the command as in: 100 print "Hello World" A statement can be typed without a line number in which case it will execute immediately. This is true for all commands, but doesn't make sense for some commands such as *for*. Immediate execution is handy for diagnostic purposes such as: print a, to find out what the value of 'a'.
Strings	Sequences of ASCII characters delimited by double quote characters at the beginning and the end.
Variables	All variable names are either a lower case alpha character (a-z) or a lower-case alpha character followed by an integer (0-9). Arrays have the same name convention as regular variables and take the form varname(expr1,expr2,expr3....expr10) where expr1-expr10 are the dimension attributes of the array and can take the form of any legal expression (including another array) as in a(b(2,2),x).

3. Commands

3.1 Standard Commands

bye or q — Exit the interpreter. Typing the control/D key will also exit the interpreter.

call name,line#

Call Overlay Subroutine. Name is the name of a file (name.b) containing a subroutine. The subroutine must be sequenced such that line# is the first line# in the file. The first line must be "line# rem name" where "name" must match the one in the *call* statement. *Call* checks the line "line#" to see if the subroutine has already been loaded. If it has, a "gosub line#" is executed. If the subroutine is not already loaded, it loads it and then does a "gosub line#".

com[mon] — Preserve variables for subsequent *run.* Issue of the *run* command otherwise de-allocates all variables.

con [line#] — Continue normal execution from single step mode. See *sing* command.

data (expr),(expr),(expr),.....

The *data* statement is a string of defined constants or expressions referred to by the "read" statement. Unlike most BASIC interpreters, the data is stored only in the form of text strings which allows the read statement to evaluate expressions as well as constants.

del[ete] lownum [, highnum]

Delete line-number specified if only lownum given. Delete all lines between lownum and highnum if both are specified. See the *undo* command.

dim variable(expr1,expr2,.....,expr10)

Allocate space and define the dimensional characteristics of subscripted variable.

end — Define logical end of program. Causes termination of current *run.*

expunge — Force all variable space, including subscripted variables to be freed. Or de-allocate used variable space.

f — Typing 'f' causes the currently referenced file (if any) to be displayed.

for - next — Cause code enclosed by this combination to be executed under the conditions specified in the *for* statement as in: for variable = expr1 to expr2 [step expr].

gosub line# — Goto subroutine, resume from following statement after *return* encountered.

goto line# — Force execution to continue starting at the line# specified.

if (expr1) relational (expr2) then line#

Redirect program flow to line# if expr1 is related to expr2 by the specified relational. The *then* in the *if* statement can be optionally replaced with *goto go to* or *gosub.* The *if* statement can also take the form:
if (expr1) relational (expr2) then var = (expr)

input [_fildes]var1[,var2,var3,....]

Prompt for input and assign inputed value to variable. If 's' is typed program is halted.

[let] variable = expr
Assign the value of expr to variable. The let is optional.

l[ist] [lownum [, highnum]]
List the text in working storage. If lownum is given then only that number is listed. if lownum and highnum are specified, then a listing is displayed between the given statement numbers.

load [program name]
Same as the *old* command, except working storage is not cleared.

mov startnum, endnum, newnum [,increm]
The mov command causes the lines beginning with *startnum* and ending with *endnum* to be moved (ie. resequenced) to the line beginning with *newnum* and incremented by *increm*. The default value for *increm* is 10. All references to the moved lines are updated. The user is responsible to see that line numbers associated with moved lines do not conflict with existing lines which will cause loss of program text. *mov* is similar to *reseq* (see below) except that only the specified lines are resequenced.

n
List the next 23 lines. Useful for paging through a listing on a CRT.

new
Clear program working storage for new program to be typed.

old [program name]
Clear user space and load program. If old is typed with no argument it will prompt the user for a program name if not defined or load the last defined program name.

on (expr) goto line#, line#,.....
Is a selective goto with multiple line number targets. The target branched to depends on the value of expr which is truncated. Control is passed to the first line# specified after goto if the value of the expression is 1. Control passes to the second line# if the value is 2, the third if 3 and so on.

on (expr) gosub line#, line#,.....
Same action as *on-goto*, except action taken is that of *gosub*.

pause
Causes execution to be suspended until a "newline" or "return" is typed. This is useful for programs which need to be continuously in *run*, but need to allow a time for user action i.e. unit insertion.

pr[int] [_fildes](expr's, quoted strings or tab operators)
The print statement is a limited format display statement in which expressions are evaluated and displayed along with quoted literals. The tab(expr) operator causes the print head to move to the absolute column position computed by expr provided the current head position is smaller. The specifiers must be separated by one or more commas or semicolons.

printf (format string)[,expr1,expr2,.....,expr10]
This is an interpretive implementation of the UNIX 'C' library routine, printf. It is, however restricted to only the floating point format control specifiers 'f' and 'g'. Use of any of the other specifiers such as 'o', 'd' or 's' will give erroneous results. Print controls such as \b (backspace), \n (newline), \r (return) or \t can also be used. The printf format was chosen in lieu of the usual *print using* command because it was felt that printf is not only a 'C' language standard but easier to use than *print using*.
Usage Example:
100 printf "Var a=%2.2f\tVar b=%g.\n",a,b

randomize Causes *rnd* statement to start at an "unpredictable" value.

read var1,var2,var3,.....

The *read* statement causes data to be assigned to each variable in the list from the constants or expressions contained in *data* statements. The reading starts at the location of the data pointer. The data pointer points to the last data field accessed if a read was done or to the first data field in the first data statement if the *restore* statement is issued or the program is re-run.

rem — The remark statement causes no operation in *BITE* but may be followed by any string of characters for the purpose of commenting a program. Unlike compiler languages, remarks do take up program buffer space; however, they are of paramount importance in making a program readable by human beings and are therefore strongly recommended.

reseq [startnum [, increm]] — The resequence command causes the statement numbers and all references to them (such as if's goto's, gosub's, etc) to be resequenced starting at *startnum* and incremented by *increm.* If startnum and/or increm are omitted, the default values are 10 and 10 respectively.

restore — Restores the data pointer to the first field of the first *data* statement.

return — Return from subroutine called by *gosub* statement.

run [program name] — Run basic program specified. If no argument is given, *run* attempts to execute whatever is currently in working storage.

s line#/old-string/new-string[/] — Substitute in line line# the new-string for the old-string. The last delimiter is optional, unless new-string is null in which case it is desired that old-string merely be removed. See the *undo* command.

sing [line#] — Enter the single step mode starting at the line# specified or at the first line of the program if no line# is specified. In single step mode an instruction is executed and then the prompt '^' is displayed. At this time the user may enter any command (i.e. print) or hit the "return" key to execute the next instruction. See the *con* command.

size — Causes amount of storage used and remaining or free space in decimal number of bytes.

stop — Stop execution of program.

save [program name] — Save the contents of working storage in file-name specified by program name. If no program name is given, last referenced file-name is used. If no file name was referenced, the user is prompted for a name.

undo — Undo last *s* command or *single line deletion*

! — UNIX shell command invocation allows system commands to be executed from the interpreter. This command is *not included* in the LSI-11/03 versions of BITE.

3.2 File Commands

The file commands: append, openi, and openo are followed by one or more file-names separated by commas, each file-name being no more that 14 characters long. Files are assigned to designators (integer values between 1 and 4 inclusive) in the order that they are open. All commands such as *print* and *input* which refer to a file use the designator number preceded by a '_' character to refer to that file as in: 100 print _1"hello world" or 100 input _3a(x,y) .

append file1[,file2,.....,file4]
: If file exists open for output cause new data to be appended. If file does not exist, the named file is created.

openi file1[,file2,.....,file4]
: Open file for input. File must exist.

openo file1[,file2,.....,file4]
: Create named file(s) and open for output. If named files exist, the old data is destroyed.

close _fildes
: Close file associated with file designator.

closeall
: Close all files input and output.

3.3 ATS Instrument Commands (Extended Instruction Set)

These instructions are those which were implemented for the *Production Test Set.*

buspr 'busadr(text and expressions)
: Buspr is merely an extension of the print statement which allows the print string which would otherwise be displayed on the tty to be sent via the IBV-11 bus to the bus address specified by "busadr". The ' preceding busadr distinguishes the following character from anything other than a single character to be interpreted as an address. The address character can be *any ASCII character* except ones which are possibly interpreted by the system as control characters such as back-space.
Usage Example:100 buspr '6"F2R";r

cmd string
: Send character string over IBV-11 command lines.

delay
: Causes a delay of num 60ths of a second where num is an integer.
Usage Example:100 delay 120 (delay 2 minutes or 120/60ths sec)

dvminit
: Initialize Digital Voltmeter.

dvms function, range
: (for HP 3455A digital voltmeter) Digital voltmeter set command, where: function is 'ac', 'dc' or 'ohms' and range is .1, 1, 10, 100, 1k, 10k or 'aut'. i.e.
Usage Example:100 dvms dc,1k

hprintf (format string)
: (for HP 5150A Thermal Printer) Formated print to strip printer. Syntax rules are the same as *printf.* Strip printer is 20 columns wide, anything past the 20th column is truncated.

lodset lodnum,mode,value
: (for POWER DESIGNS X-510 & TRANSISTOR DEVICES DLP 50-60-1000 electronic loads) Set load. Where lodnum an integer describing which load referred to, mode is the manner in which the load is set and value is an expression describing the current or resistance the load was set to depending on the mode. Mode is a single character 'r', 'R', 'i' or

'I' where 'r' is resistance mode (value in ohms) and 'i' is current mode (value in amperes). Small letter causes a hunt for the value and capital causes set on first try.

relay function, relnum1[, relnum2, relnum3,...]
(for HP 6940B MULTIPROGRAMMER) Set multiprogrammer relays. Function is m (make), b (break) or c (clear). Function is followed by all relay numbers to be acted upon which may be expressions or variables. The clear function when not followed by anything, simply means open all relays. When followed by relay numbers, clear means all relays are open EXCEPT the ones specified.
Usage Example:100 relay m,10,20,21,a,b,rnd(10)

ps psno,voltage,current limit,overvoltage
(for KEPCO ATE 75-15M, ATE 150-7M, ATE 55-1M & JQE Power supplies) Set power supply parameters. Psno is an integer representing the power supply number describing which power supply is to be used, voltage, current limit and overvoltage are self explanatory. Each one of the parameters may be a legal algebraic expression so that they may be controlled by the program.
Usage Example:100 dvms 1,10,1,11 or 100 dvms n,v1,i1,v1+1

scan scanner-channel
(for HP 3495A SCANNER) Set 3495A Scanner channel to number specified.

4. Functions

4.1 Standard Functions

abs(expr)	Absolute value.
atn(expr)	Arc-tangent.
cos(expr)	Cosine.
exp(expr)	Natural exponential.
int(expr)	Integerize or truncate fractional part of result of expr.
log(expr)	Natural log.
rnd(expr)	Return random number between 0 and evaluated expr.
sin(expr)	Sine.
sqr(expr)	Square root.

4.2 Instrument Functions (Extended Set)

btn(expr)	Button function returns non-zero if control button number (expr) is depressed.
dvmr()	Return digital voltmeter reading.
error()	Return 1 if last instrument command caused instrument error, otherwise return 0.

5. Modes of Operation

5.1 Editor or Idle Mode

When the *BITE* interpreter is invoked with no argument, a prompt '*' appears meaning that the interpreter is waiting for the user to enter something from the keyboard. *BITE* is then said to be in the *Editor or Idle mode.*

Editing is accomplished as it is in any BASIC language interpreter in that lines are entered by typing a line-number followed by the statement and removed or deleted by merely typing the line-number. Listing is accomplished with the *list* command (explained under "Standard Commands"). In addition to the above, it is possible to list single lines by typing the return key in which case the program is listed one line-at-a-time, starting at the first. When the last one is reached, the sequence starts at the first line again. At any time it is also possible to type the '-' symbol to "backup" a line-at-a-time. Other editing facilities are *s, delete,* and *reseq* also explained under "Standard Commands".

5.2 Run Mode

If the *run* command is typed and a program is currently in user storage, the program begins execution, starting with the first line of the program, then executing each line in order of line numbered sequence. The sequence of execution is altered by program flow control statements like *if, for-next* or any statement containing a *goto.*

5.3 Immediate Execution Mode

Immediate execution is accomplished by typing a command without preceding it with a line number. Although this is possible with all commands, it doesn't always make sense. For example, using commands that control program flow in immediate mode is unlikely and often disastrous.

Immediate mode is designed so that the user may get immediate action as in the command *run* or *print a.* Some commands are almost always used in immediate mode such as *q, delete, expunge, load, list, old, reseq, save,* etc.

5.4 Single Step Mode

Single step mode is entered with the *sing* command and exited with the *con* command. During this mode, one may find "BUGS" in the program by observing the program flow or sequence or examining the values of variables at given points in the program to see if they have the expected values. See *sing* or *con* under the "Standard Commands" section of this paper.

6. Interruption of program

At times it becomes necessary to escape from an endless loop or abort an action such as *list* before it completes. To cause such an interruption, the (DEL) or (RUB) key is typed.

7. Programming Techniques and Tools

7.1 Program Segmentation

In situations of limited memory space as in the case of the LSI-11/03 it becomes impossible to fit large programs in storage at any one time time. It then becomes necessary to write the program in pieces or segments each of which must be loaded separately as needed. Segmented are handled by "chaining" or "overlaying".

7.1.1 Chaining Chaining is accomplished by insertion of the *run* command in the program text. If a program is to be split into say, program1 and program2 then by simply inserting the line "run program2" as the last executed statement of program1, program2 is now chained to program1. Upon completion of program2, if it is desired to reload program1, the last executed statement should be "old program1" which will clear user space and reload program1.

It must be noted that execution of a *run* causes variables to be wiped out. To preserve variables from one program to the next, the statement *common* must be executed prior to the chaining *run* statement.

The chaining process may go on indefinitely, the only expense being some time delay for each program load.

The chaining point must sometimes be strategically chosen so that it will not occur during an instrument action or some time critical part of the program.

7.1.2 Overlaying Overlaying is best accomplished with the *call* command. This technique is particularly useful where core space is minimal and lends itself to keeping programs modular. The *call* command has a built-in feature which prevents loading a module which is already resident. In a line numbered language, overlay segments are delimited by line number boundaries rather than address boundaries as is true in machine level programming. The following is an example of a simple implementation of an overlay:

```
100 rem THIS IS THE ROOT SEGMENT OF THE PROGRAM
110 rem The "root" segment remains resident and usually contains
120 rem all the commonly called subroutines
130 call pscheck,2000          call in power supply check overlay
140 call loadck,2000           call in load check overlay
150 call loadck,2000           call load check again
150 rem THE REST OF THE "ROOT" SEGMENT
                |
                |
                |
900 stop
2000 rem oldstuff
2010 rem THIS PART OF THE PROG IS DESTROYED WHEN OVERLAYS ARE CALLED
2020 return

2000 rem pscheck
2010 rem This is the power supply check routine which is called in
2020 rem the root segment
2030 rem the first line (2000) must appear in the program as shown
                |
                |
3000 return

2000 rem loadck
2010 rem This is the load check overlay. This occupies the same
2020 rem line number space as pscheck and will therefor replace it
                |
                |
3000 return
```

In the above example, two overlays (pscheck and loadck) are called. If those routines are found on disk they will be loaded and replace the old program text starting at line 2000. Note that loadck is called twice in succession. The second call will not cause a load since loadck is already resident. The interpreter believes this to be true by virtue of the *rem* statement with the name of the overlay at line 2000.

7.2 System Shell Control

Invoked by the Bourne shell, *BITE* can be a powerful tool which can add mathematical capabilities to the shell. The following shell-script invokes *BITE* without running it, loading the program add.b, entering a data statement with two numbers to be added, issuing the *run* command and putting the result in file "result".

```
:
: shell script to add two numbers
:
bite - add <<! > result
115 data $1,$2
run
!
```

The following is the add.b program invoked by the above shell script.

```
90 rem BASIC PROGRAM TO BE RUN BY SHELL SCRIPT
91 rem add numbers in data statement and output to standard output
100 read a,b                    get the values of the 2 numbers in the data
110 print a+b                   output the result
115 rem This line is replaced by a shell script line data statement
120 bye                         exit
```

BITE gives the Bourne shell the complete ability to EXECUTE BASIC INSTRUCTIONS! This is accomplished by putting all the command lines in the shell-script and using the <<! device pass the program to BITE.

Below is a shell script which adds two numbers passed to it as arguments. Note that the statements need not be in numerical order, since *BITE* will order them as they are entered. Type this program in, give it a name, say add and make it executable via chmod. Then try it by typing "add 3.14 2.22".

```
bite <<!
100 read a,b
110 print a+b
120 bye
115 data $1,$2
run
!
```

An example of a shell-script using the immediate mode of *BITE* to get the sine of an angle in degrees is:

```
bite <<!
print sin(((2*3.1415926)/360)*$1)
bye
!
```

By naming the above shell-script "sin", one can type "sin 45" from the system level to get an immediate answer.

Strings can be manipulated by the Bourne shell and system utilities then passed to a *BITE program*. The following is a simple example:

```
a="Now is the time"
b="for all good men"
bite <<!
print "$a $b to come to the aid of their country."
q
!
```

8. Error Messages

Diagnostic error messages are issued by the interpreter which indicate syntax errors , system failure, illegal commands or expressions,etc. The *LSI-11/03* Version of *BITE* does not issue explicit error messages, but displays an error number in which the meanings are listed below. This is done to regain approximately 2 kilobytes memory in an already tight *LSI-11/03* memory.

8.1 Standard Error Messages

NUMBER	MESSAGE TEXT
0	REFERS TO A NON-EXISTING LINE NUMBER
1	UNRECOGNIZABLE OPERATION
2	CANNOT OPEN FILE
3	ILLEGAL VARIABLE NAME
4	BAD FILENAME
5	WORKING STORAGE AREA EMPTY
6	RUNS NESTED TOO DEEPLY
7	UNASSIGNED VARIABLE
8	EXPRESSION SYNTAX
9	BAD KEYWORD IN STATEMENT
10	IMPROPER OR NO RELATIONAL OPERATOR
11	UNBALANCED QUOTES
12	FILE EDITING NOT PERMITTED IN SINGLE STEP MODE
13	MISSING OR ILLEGAL DELIMITER
14	GOSUB WITH NO RETURN
15	IS FATAL
16	UNBALANCED PARENTHESIS
17	UNKNOWN MATH FUNCTION
18	NEXT WITH NO OR WRONG FOR IN PROGRESS
19	CANNOT PROCESS IMAGINARY NUMBER
20	WHAT ?
21	BAD DIMENSION SYNTAX
22	TOO MANY DIMENSIONS
23	REDUNDANT DIM STATEMENT
24	NOT ENOUGH WORKING STORAGE SPACE
25	VARIABLE NOT DIMENSIONED
26	WRONG NUM OF DIMS
27	ONE OR MORE DIMS LARGER THAN ASSIGNED
28	NEG. OR ZERO DIMENSION ILLEGAL
29	DIVIDE BY ZERO
30	BAD TAB SPEC. IN PRINT
31	SYS CALL FAILED
32	BAD FILE DECLARE SYNTAX
33	OUT OF DATA
34	FILE-NAME TOO LONG
35	FILE DES. USED UP

36 FILE NOT OPEN FOR OUTPUT
37 FILE NOT OPEN FOR INPUT
38 EXPRESSION YIELDS AN IMPOSSIBLE VALUE
39 PRINTF: ARG COUNT MISMATCH
40 PRINTF: MORE THAN 10 ARGS
41 LINE TOO LONG FOR STRIP PRINTER
42 MOV REQUIRES 3 LINE #'s, SPACING IS OPTIONAL
43 BAD NAME OR LINE NUMBER AT BEGINNING OF SUBROUTINE

8.2 Test Set and Instrument Error Messages

100 MISSING ' DELIMITER BEFORE BUS ADDR
101 PS: VOLTAGE OUT OF RANGE
102 PS: CURRENT OUT OF RANGE
103 PS: OVERVOLTAGE OUT OF RANGE
104 RELAY ERR
105 RELAY: INVALID FUNC.
106 RELAY: INVALID NUMBER
107 DVM: INVALID MODE
108 DVM: INVALID RANGE
109 LODSET: IMPROPER NUMBER OF ARGUMENTS
110 LODSET: IMPROPER MODE
111 LODSET: UNABLE TO SET LOAD

9. Acknowledgement

BITE was written in the 'C' programming language by R. B. Drake and myself. This memorandum was reviewed for accuracy and clarity by R. B. Drake and J. D. McElroy. The device drivers used for the extended instruction set are a result of the efforts of R. B. Drake, R. E. Ellenbogen, L. W. Schaper and myself. Testing of the initial versions of *BITE* were made by N. P. Episcopo and D. J. Jackowski. D. J. Jackowski performed extensive testing of *BITE*, made many helpful suggestions and showed us how *BITE* could be used with the *Bourne Shell*. N. P. Episcopo made some benchmark numerical comparisons between *BITE* and *BASIC* on *TSS*. Many helpful suggestions were contributed by R. Scuderi.

James P. Hawkins

WH-2425-JPH-UNIX

J. P. Hawkins

Att:
References
Appendix A and B

REFERENCES

[1] R. B. Drake "Guide to the Internals of BITE", Bell Laboratories TM 79-2425-5

[2] DEC, "introduction to BASIC", Digital Equipment Corporation

[3] C. Joseph Sass, "BASIC Programming and Applications", Allyn and Bacon, Inc.

[4] S. R. Bourne, "An Introduction to the UNIX Shell" Bell Laboratories TM 78-1274-4

[5] R. E. Ellenbogen & John Tardy, "Automated Testing of Power Supplies" Bell Laboratories TM 79-2425-1

10. APPENDIX: A

10.1 SAMPLE FILE I/O PROGRAM

The following is a sample of a program which opens a file for output, writes values out to it, closes the file, re-opens the file for input, reads the values from the file into an array and tabulates the values.

The *comments* to the right of the statements are there as an aid in this document. It is improper syntax to insert comments or remarks in any other manner than by using the *rem* statement.

```
100 dim d(100)                    dimension a 100 variable array
110 openo junk                    open file junk for output
120 for a = 1 to 100              for values of 'a' from 1 to 100
130 pr _1a*10                     output 10*a to file junk
140 next a                        loop
150 closeall                      close file junk
160 openi junk                    open file junk for input
170 for x = 1 to 100              for 100 points points in array d(x)
180 input _1d(x)                  input next value to d(x)
190 next x                        loop
200 closeall                      close file junk
210 for x = 1 to 100              tabulate values of elements in d(100)
220 printf "d(%4.1f) =%4.2f\n",x,d(x)
230 next x
```

The output of the above program looks as follows:

```
d( 1.0) =10.00
d( 2.0) =20.00
d( 3.0) =30.00
d( 4.0) =40.00
d( 5.0) =50.00
d( 6.0) =60.00
d( 7.0) =70.00
d( 8.0) =80.00
d( 9.0) =90.00
d(10.0) =100.00
        |
        |
d(100.0) =1000.00
```

11. APPENDIX: B

11.1 SAMPLE PROGRAM TO STEP VOLTAGE ON POWER SUPPLY

```
100 for v = 1 to 30 step .5     for voltages of 1 - 30 in steps of .5 volts
110 ps 1,v,1,30                 set power supply number 1 to 'v' volts
120 next v                      loop again
130 ps 0,0,0,0                  shut down power supply
```

11.2 WAIT FOR BUTTON PRESS

```
100 if btn(1) = 0 then 100      wait for button 1 to be pressed
110 n = 2                       set a button number
120 if btn(n) = 0 then 120      wait for button 2
130 n = n + 1                   next button
140 if btn(n) = 0 then 140      wait for button 3
```

11.3 EXERCISE SCANNER AND RELAYS

```
100 for s = 1 to 30             for 30 scanner channels
120 scan s                      set scanner channel s
130 delay 30                    wait 1/2 second
140 next s                      loop
150 rem
160 rem exercise
170 rem
180 relay c                     open all relays
190 relay m,10,20,30            close relay 10,20 and 30
200 relay b,10,20               open only relays 10 and 20
210 relay c                     open all relays
220 for r = 1 to 60             for relays 1 through 60
230 relay m,r                   close cumulatively
240 next r                      loop
250 rem now 60 relays are closed
260 relay c,1                   open all but relay number 1
```