

**INPUT OPTIONS**

string-var Prompt with colon and input character data  
 Example INPUT \$A  
 Delimit expressions Example A, B  
 Suppress prompting or CR LF if at end of line  
 Examples INPUT ,A  
 INPUT A,  
 # exp Allow a maximum of exp characters to be entered  
 Example INPUT # 1"Y or N"\$1  
 %exp \*Must enter exactly exp number of characters  
 Example INPUT %4 "CODE" C  
 ?<ln> \* If ln is invalid input or entry of a control character,  
 ; B is performed to the line # will be  
 -1 if there was an invalid input character,  
 SYS(0) will equal the decimal equivalent of the control  
 character  
 Example INPUT ?100,A

**OUTPUT OPTIONS**

Delimit exp s or suppress CR LF if at end of line  
 Examples: ; A, B  
 Tab to next print field Example PRINT A, B  
 Tab to exp column Example PRINT TAB (50),A  
 Print string or string-var Example PRINT "HI", \$A(0)  
 # exp \*Print exp as hexadecimal in free format  
 Example PRINT # 123  
 #.exp \*Print exp as hexadecimal in byte format  
 Example PRINT #.50  
 #:exp \*Print exp as hexadecimal in word format  
 Example PRINT #:A  
 <hex value> \*Direct output of ASCII codes Example PRINT  
 "<0D><0A>"  
 # string \*Print under specified format where  
 PRINT # "9999"1  
 9 = digit holder  
 PRINT # "000-00-0000"SS  
 0 = digit holder or force 0  
 PRINT # "\$\$\$,SS\$ 00" DLR  
 \$ = digit holder and floats \$  
 PRINT # "SS\$ 0000" \*ATN!  
 S = digit holder and floats sign  
 PRINT # " <<< 00>"1  
 < = digit holder and float on negative  
 >=number  
 PRINT # "990 99E" N  
 E = sign holder after decimal  
 PRINT # "990 99" N  
 = decimal point specifier  
 PRINT # "999,990 99" N  
 , = suppressed if before significant digit  
 PRINT # "999,990 ^ 00"1  
 ^ = translates to decimal point  
 PRINT # HI - 99"1  
 any other character is printed.

**ERROR CODES**

- 1 = SYNTAX ERROR
- 2 = UNMATCHED PARENTHESIS
- 3 = INVALID LINE NUMBER
- 4 = ILLEGAL VARIABLE NAME
- 5 = TOO MANY VARIABLES
- 6 = ILLEGAL CHARACTER
- 7 = EXPECTING OPERATOR
- 8 = ILLEGAL FUNCTION NAME
- 9 = ILLEGAL FUNCTION ARGUMENT
- 10 = STORAGE OVERFLOW
- 11 = STACK OVERFLOW
- 12 = STACK UNDERFLOW
- 13 = NO SUCH LINE NUMBER
- 14 = EXPECTING STRING VARIABLE
- 15 = INVALID SCREEN COMMAND
- 16 = EXPECTING DIMENSIONED VARIABLE
- 17 = SUBSCRIPT OUT OF RANGE
- 18 = TWO FEW SUBSCRIPTS
- 19 = TOO MANY SUBSCRIPTS
- 20 = EXPECTING SIMPLE VARIABLE
- 21 = DIGITS OUT OF RANGE (0< # of digits < 12)
- 22 = EXPECTING VARIABLE
- 23 = READ OUT OF DATA
- 24 = READ TYPE DIFFERS FROM DATA TYPE
- 25 = SQUARE ROOT OF NEGATIVE NUMBER
- 26 = LOG OF NON-POSITIVE NUMBER
- 27 = EXPRESSION TOO COMPLEX
- 28 = DIVISION BY ZERO
- 29 = FLOATING POINT OVERFLOW
- 30 = FIX ERROR
- 31 = FOR WITHOUT NEXT
- 32 = NEXT WITHOUT FOR
- 33 = EXP FUNCTION HAS INVALID ARGUMENT
- 34 = UNNORMALIZED NUMBER
- 35 = PARAMETER ERROR
- 36 = MISSING ASSIGNMENT OPERATOR
- 37 = ILLEGAL DELIMITER
- 38 = UNDEFINED FUNCTION
- 39 = UNDIMENSIONED VARIABLE
- 40 = UNDEFINED VARIABLE
- 41 = EXPANSION EPROM NOT INSTALLED
- 42 = INTERRUPT W/O TRAP
- 43 = INVALID BAUD RATE
- 44 = TAPE READ ERROR
- 45 = EPROM VERIFY ERROR
- 46 = INVALID DEVICE NUMBER



**POWER BASIC  
 Reference  
 Data  
 Microprocessor  
 Series™**



**REFERENCE CARD FOR DEVELOPMENT AND EVALUATION BASIC**

This card contains a summary of all POWER BASIC statements and commands for Development and Evaluation BASIC. An explanation preceded by an asterisk (\*) indicates the statement or command is not supported by Evaluation BASIC. A \* indicates the statement is supported only by the Development BASIC software enhancement package.

† Trademark of Texas Instruments

**COMMANDS**

**CONTINUE**  
 \*Execution continues from last break

**LIST**  
 LIST the user's POWER BASIC program. In LIST will list from specified line number through end of program or until ESC key is struck

**LOAD**  
 Reads a previously recorded POWER BASIC program from an auxiliary device or configures POWER BASIC to execute a program in EPROM.  
 LOAD reads program from 733ASR or cassette.  
 LOAD 1 or LOAD 2 \* reads program 1 audio cassette drive No. 1 or No 2  
 LOAD <address> \* configures POWER BASIC to execute BASIC program in EPROM at specified address

**NEW**  
 Prepare for entry of NEW POWER BASIC program or set the lower RAM memory bound after auto-sizing.  
 NEW clears pointers of POWER BASIC and prepares for entry of new program.  
 NEW <address> \* sets the lower RAM memory bound used by POWER BASIC after auto-sizing or power-up

**PROGRAM**  
 Program current POWER BASIC application program into EPROM \*

**RUN**  
 Begin program execution at the lowest line number

**SAVE n** (n is interpreted as in LOADn command)  
 Record current user program on auxiliary device

**SIZE**  
 Display current program size, variable space allocated, and available memory in bytes

**GENERAL INFORMATION**

**ARITHMETIC OPERATIONS**

A = B Assignment  
 A - B Negation or subtraction  
 A + B, \$A + \$B Addition or string concatenation  
 A \* B Multiplication  
 A / B Division  
 A ^ B Exponentiation  
 - A Unary Minus  
 + A Unary Plus

**LOGICAL OPERATORS**

LNOT A \*1's complement of integer  
 A LAND B \*Bit wise AND  
 A LOR B \*Bit wise OR  
 A LXOR B \*Bit wise exclusive OR

**RELATIONAL OPERATORS**

1 if TRUE and 0 if FALSE  
 A = B TRUE if equal, else FALSE  
 A ≈ B \*TRUE if approximately equal (1E-7), else FALSE  
 A < B TRUE if less than, else FALSE  
 A <= B TRUE if less than or equal, else FALSE  
 A > B TRUE if greater than, else FALSE  
 A >= B TRUE if greater than or equal, else FALSE  
 A <> B TRUE if not equal, else FALSE  
 NOT A \*TRUE if zero, else FALSE  
 A AND B \*TRUE if both non-zero, else FALSE  
 A OR B \*TRUE if either non-zero, else FALSE

**OPERATOR PRECEDENCE**

- 1 Expressions in parentheses
- 2 Exponentiation and negation
- 3 \*, /
- 4 +, -
- 5 <=, <>, >
- 6 >=, <
- 7 =, >
- 8 =, LXOR
- 9 NOT, LNOT
- 10 AND, LAND
- 11 OR, LOR
- 12 (=) ASSIGNMENT

**SPECIAL CHARACTERS**

Separates statements typed on same line  
 ! Tail remark used for comments after program statement  
 ; Equivalent to PRINT

## EDITING

The phrase "(ctrl)" indicates that the user holds down the control key while depressing the key corresponding to the character immediately following

(CR)	Enter edited line
(ctrl)ln	*Insert n blanks
(ctrl)Dn	*Delete n characters
(ctrl)H	Backspace one character
(ctrl)F	Forward space one character
ln(ctrl)E	Display for editing source line indicated by line number (ln)
(ctrl)T	Toggle from one partition to the other partition (only in Evaluation BASIC).
(esc)	Cancel input line or break program execution
(Rubout) or (DEL)	Backspace and delete character

## STATEMENTS

InBAUD <exp 1,> <exp 2>	*sets baud rate of serial I/O port(s)
InBASE <(exp)>	Sets CRU base address for subsequent CRU operations
InCALL "Name"<subroutine address>[, <var 1> . <var 2> . <var 3> . <var 4>]	*Transfers to external subroutines. If variable is contained in parentheses, the address will be passed, otherwise, the value will be passed
InDATA {<exp> <string const>} {<exp> <string const>}	defines internal data block.
In DEF FN <x>[( <arg 1> [, <arg 2> [, <arg 3>]] = <exp>	*Defines user arithmetic function
InDIM <var (dim[, dim] . . .) : [ . . . ]	Allocates user variable space for dimensioned or array variables
InEND	Terminates program execution and returns to edit mode
In ERROR<ln>	*Specifies a subroutine that will be called via a GOSUB statement when an error occurs.
! *	*Enables or disables the escape key to interrupt program execution (always enabled in Evaluation BASIC)
InFOR <sim-var> = <exp> TD <exp> [STEP <exp>]	
InNEXT <sim-var>	Open and close program loop. Both identify the same control variable. FOR assigns starting, ending, and optionally stepping values.
InGOSUB <ln>	Transfer of control to an internal subroutine beginning at the specified line
InPOP	*Removal of most previous return address from GOSUB stack without an execution transfer
InRETURN	Return from internal subroutine
InGOTO<ln>	Transfers program execution to specified line number.
InIF <exp> THEN <statement>	
InELSE <statement>	Causes conditional execution of the statement following THEN. *ELSE statements execute when IF condition is false
InIMASK<LEVEL>	*Set interrupt mask of TMS 9900 processor to specified level
InTRAP<level> TO<ln>	*Assign interrupt level to interrupt subroutine
InIRTN	*Return from BASIC interrupt service routine
InINPUT<var> {<var> . . . {<var>}}	Accesses numeric constants and strings from the keyboard into variables in the INPUT list
In [LET] <var> = <exp>	Evaluates and assigns values to variables or array elements
InON <var> THEN GOTO ln [,ln]	
InON <var> THEN GOSUB ln [,ln]	
	*Transfers execution to the line number specified by the expression or variable
InPRINT <exp> [,exp] . . .	Print (format free) the evaluated expressions
InRANDOM [exp]	*Set the seed to the specified expression value
InREAD {<numeric var> } {<numeric var> } {<string var> } {<string var> }	Assigns values from the internal data list to variables or array elements
InREM [text]	Inserts comments
InRESTOR [exp]	Without an argument, resets pointer to beginning of data sequence; with an argument, resets pointer to line number specified
InSTOP	Terminates program execution and returns to Edit mode
InTIME	Sets, displays, or stores the 24 hour time of day clock
	lnTIME <exp> . <exp> . <exp>
	Sets and starts clock
	lnTIMF <string-var>
	E . . . storing clock time into a string variable
	ln . . . clock time as HR MN SD

lnUNIT <exp>

\*Designates device(s) to receive all printed output

## FUNCTIONS

ABS . (exp)	'Absolute value of expression
ASC <(string var)>	*Returns decimal ASCII code for first character of string variable
ATN <(exp)>	Arctangent of expression in radians
BIT <(var, exp)>	*Reads or modifies any bit within a variable.
BIT <(var, exp 1)> = <exp 2>	Returns a 1 if bit is set and 0 if not set Selected bit is set to 1 if assigned value is non-zero and to zero if the assigned value is zero
COS >(exp)>	Cosine of the expression in radians
CRB <(exp)>	Reads CRU bit as selected by CRU base + exp. Exp is valid for -127 thru 126
CRB <(exp 1)> = <(exp 2)>	Sets or resets CRU bit as selected by CRU base + exp 1. If exp 2 is non-zero, the bit will be set, else reset. Exp 1 is valid for -127 thru 126
CRF <(exp)>	Reads n CRU bits as selected by CRU base where exp evaluates to n. Exp is valid for 0 thru 15. If exp = 0, 16 bits will be read
CRF <(exp 1)> = <(exp 2)>	Stores exp 1 bits of exp 2 to CRU lines as selected by CRU BASE. Exp 1 is valid for 0 thru 15. If exp 1 = 0, 16 bits will be stored
EXP <(exp)>	*Raise the constant e to the power of the evaluated expression
INP <(exp)>	Returns the signed integer portion of the expression
LOG <(exp)>	*Returns natural logarithm of the expression
MEM <(exp)>	Reads byte from user memory at address specified by exp. Exp must be in the integer range, (0 to 65535)
MEM <(exp 1)> = <(exp 2)>	Stores byte exp 2 into user memory specified by exp 1. Exp 1 and exp 2 must be in the integer range
MCH <(string 1), (string 2)>	*Returns the number of characters to which the two strings agree
NYK <(exp)>	Conditionally samples the keyboard in run time mode. If exp <> 0, return decimal value of last key struck and clear key register. (0 if no key struck) If exp = 0, return a 1 if the last key struck has the same decimal value as the EXPRESSION Clear key register if TRUE, else return 0 if FALSE
RND	Returns a random number between 0 and 1
SIN <(exp)>	Sine of the expression in radians
SQR <(exp)>	Square root of expression
SRH <(string 1), (string 2)>	*Return the position of string 1 in string 2, 0 if not found
SYS <(exp)>	*Obtains system parameters generated during program execution. Example: SYS(0) = INPUT control character, SYS(1) = Error code number, SYS(2) = error line number
TIC <(exp)>	Returns the number of time tics less the expression value. One TIC equals 40 milliseconds (1/25 second).

## STRINGS

ASCII Character	ASC (string-var)
Conversion Function	*Convert first character of string to ASCII numeric representation
Assignment	<string-var> = {<string-var>} {<string-const>}
	Store string into string-var ending with a null.
Character Match Function	MCH <(string 1), <string 2>>
	*Return the number of characters to which the 2 strings agree
Character Search Function	SRH (<string 1>, <string 2>>)
	*Return the position of string 1 in string 2. Zero is returned if not found
Concatenate	<string-var> = {<string-var>} + {<string-var>} + {<string-const>} + {<string-const>} + {<string-const>} + {<string-const>}
Convert to ASCII	<string-var> = <exp> <string-var> = # <string> . <exp>
	*Convert exp to ASCII characters ending with a null. # string specifies a formatted conversion <var 1> = <string> . <var 2>
Convert to Binary	*Convert string into binary equivalent. Var 2 receives the delimiting non-numeric character in first byte
Delete	<String-var> = / <exp>
	*Delete exp characters from string-var
Insertion	<string-var> = / <string>
	*Pick byte into string-var
Pick	<string-var> = {<string-var>} {<string-const>} {<exp>}
	Pick number of characters specified by exp into string-var ending with a null
Replace	<string-var> = {<string-var>} {<string-const>} {<exp>}
	Replace number of characters specified by exp of string-var with string.
String Length Function	<var> = LEN <(string-var) . <var> = LEN "string"
	*Return the length of string