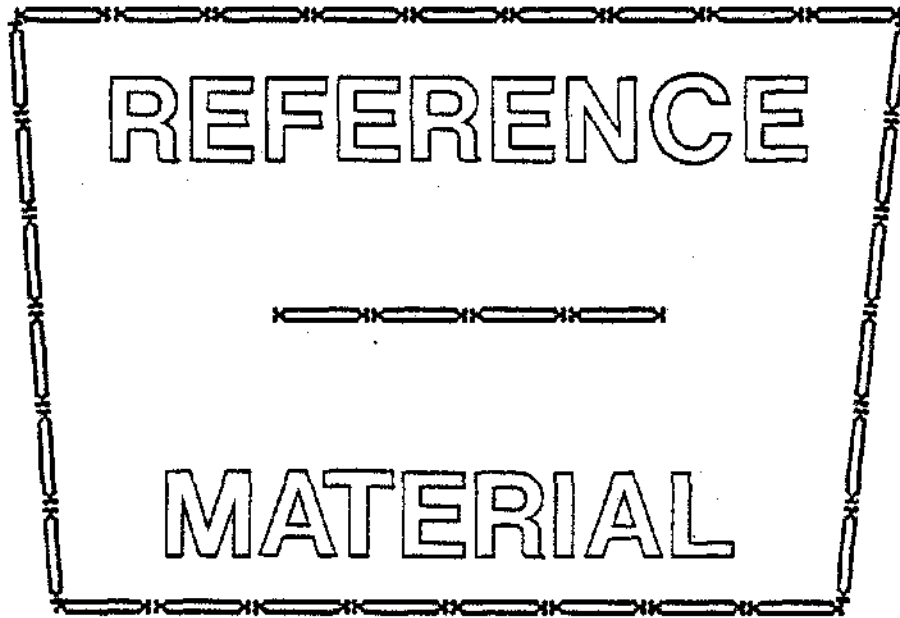


BASIC LANGUAGE



MIT'S
Creative Electronics

COMMANDS

A command is usually given after BASIC has typed OK. This is called the "Command Level". Commands may be used as program statements. Certain commands, such as LIST, NEW and CLOAD will terminate program execution when they finish.

<u>NAME</u>	<u>EXAMPLE</u>	<u>PURPOSE/USE</u>
CLEAR	*(SEE PAGE 42 FOR EXAMPLES AND EXPLANATION)	
LIST	LIST LIST 100	Lists current program optionally starting at specified line. List can be control-C'd (BASIC will finish listing the current line)
NULL	NULL 3	(Null command only in 8K version, but paragraph applicable to 4K version also) Sets the number of null (ASCII 0) characters printed after a carriage return/line feed. The number of nulls printed may be set from 0 to 71. This is a must for hardcopy terminals that require a delay after a CRLF*. It is necessary to set the number of nulls typed on CRLF to 0 before a paper tape of a program is read in from a Teletype (<i>TELETYPE is a registered trademark of the TELETYPE CORPORATION</i>). In the 8K version, use the null command to set the number of nulls to zero. In the 4K version, this is accomplished by patching location 46 octal to contain the number of nulls to be typed plus 1. (Depositing a 1 in location 46 would set the number of nulls typed to zero.) When you punch a paper tape of a program using the list command, null should be set >=3 for 10 CPS terminals, >=6 for 30 CPS terminals. When not making a tape, we recommend that you use a null setting of 0 or 1 for Teletypes, and 2 or 3 for hard copy 30 CPS terminals. A setting of 0 will work with Teletype compatible CRT's.
RUN	RUN	Starts execution of the program currently in memory at the lowest numbered statement. Run deletes all variables (does a CLEAR) and restores DATA. If you have stopped your program and wish to continue execution at some point in the program, use a direct GOTO statement to start execution of your program at the desired line. *CRLF=carriage return/line feed

RUN 200

(8K version only) optionally starting at the specified line number

NEW NEW Deletes current program and all variables

THE FOLLOWING COMMANDS ARE IN THE 8K VERSION ONLY

CONT CONT Continues program execution after a control/C is typed or a STOP statement is executed. You cannot continue after any error, after modifying your program, or before your program has been run. One of the main purposes of CONT is debugging. Suppose at some point after running your program, nothing is printed. This may be because your program is performing some time consuming calculation, but it may be because you have fallen into an "infinite loop". An infinite loop is a series of BASIC statements from which there is no escape. The ALTAIR will keep executing the series of statements over and over, until you intervene or until power to the ALTAIR is cut off. If you suspect your program is in an infinite loop, type in a control/C. In the 8K version, the line number of the statement BASIC was executing will be typed out. After BASIC has typed out OK, you can use PRINT to type out some of the values of your variables. After examining these values you may become satisfied that your program is functioning correctly. You should then type in CONT to continue executing your program where it left off, or type a direct GOTO statement to resume execution of the program at a different line. You could also use assignment (LET) statements to set some of your variables to different values. Remember, if you control/C a program and expect to continue it later, you must not get any errors or type in any new program lines. If you do, you won't be able to continue and will get a "CN" (continue not) error. It is impossible to continue a direct command. CONT always resumes execution at the next statement to be executed in your program when control/C was typed.

THE FOLLOWING TWO COMMANDS ARE AVAILABLE IN THE 8K CASSETTE
VERSION ONLY

CLOAD	CLOAD P	Loads the program named P from the cassette tape. A NEW command is automatically done before the CLOAD command is executed. When done, the CLOAD will type out OK as usual. The one-character program designator may be any printing character. CSAVE and CLOAD use I/O ports 6 & 7. See Appendix I for more information.
CSAVE	CSAVE P	Saves on cassette tape the current program in the ALTAIR's memory. The program in memory is left unchanged. More than one program may be stored on cassette using this command. CSAVE and CLOAD use I/O ports 6 & 7. See Appendix I for more information.

OPERATORS

<u>SYMBOL</u>	<u>SAMPLE STATEMENT</u>	<u>PURPOSE/USE</u>
=	A=100 LET Z=2.5	Assigns a value to a variable The LET is optional
-	B=-A	Negation. Note that 0-A is subtraction, while -A is negation.
+	130 PRINT X↑3 <i>(usually a shift/N)</i>	Exponentiation (8K version) (equal to X*X*X in the sample statement) 0+0=1 0 to any other power = 0 A+B, with A negative and B not an integer gives an FC error.
*	140 X=R*(B*D)	Multiplication
/	150 PRINT X/1.3	Division
+	160 Z=R+T+Q	Addition
-	170 J=100-I	Subtraction

RULES FOR EVALUATING EXPRESSIONS:

1) Operations of higher precedence are performed before operations of lower precedence. This means the multiplication and divisions are performed before additions and subtractions. As an example, 2+10/5 equals 4, not 2.4. When operations of equal precedence are found in a formula, the left hand one is executed first: 6-3+5=8, not -2.

2) The order in which operations are performed can always be specified explicitly through the use of parentheses. For instance, to add 5 to 3 and then divide that by 4, we would use $(5+3)/4$, which equals 2. If instead we had used $5+3/4$, we would get 5.75 as a result (5 plus 3/4).

The precedence of operators used in evaluating expressions is as follows, in order beginning with the highest precedence:

(Note: Operators listed on the same line have the same precedence.)

- 1) FORMULAS ENCLOSED IN PARENTHESIS ARE ALWAYS EVALUATED FIRST
- 2) + EXPONENTIATION (8K VERSION ONLY)
- 3) NEGATION -X WHERE X MAY BE A FORMULA
- 4) * / MULTIPLICATION AND DIVISION
- 5) + - ADDITION AND SUBTRACTION
- 6) RELATIONAL OPERATORS: = EQUAL
(equal precedence for all six) <> NOT EQUAL
 < LESS THAN
 > GREATER THAN
 <= LESS THAN OR EQUAL
 >= GREATER THAN OR EQUAL

(8K VERSION ONLY) (These 3 below are Logical Operators)

- 7) NOT LOGICAL AND BITWISE "NOT"
 LIKE NEGATION, NOT TAKES ONLY THE FORMULA TO ITS RIGHT AS AN ARGUMENT
- 8) AND LOGICAL AND BITWISE "AND"
- 9) OR LOGICAL AND BITWISE "OR"

In the 4K version of BASIC, relational operators can only be used once in an IF statement. However, in the 8K version a relational expression can be used as part of any expression.

Relational Operator expressions will always have a value of True (-1) or a value of False (0). Therefore, $(5=4)=0$, $(5=5)=-1$, $(4>5)=0$, $(4<5)=-1$, etc.

The THEN clause of an IF statement is executed whenever the formula after the IF is not equal to 0. That is to say, IF X THEN... is equivalent to IF $X<>0$ THEN...

<u>SYMBOL</u>	<u>SAMPLE STATEMENT</u>	<u>PURPOSE/USE</u>
=	10 IF A=15 THEN 40	Expression Equals Expression
<>	70 IF A<>0 THEN 5	Expression Does Not Equal Expression
>	30 IF B>100 THEN 8	Expression Greater Than Expression
<	160 IF B<2 THEN 10	Expression Less Than Expression
<=,=<	180 IF 100<=B+C THEN 10	Expression Less Than Or Equal To Expression
>=,=>	190 IF Q=>R THEN 50	Expression Greater Than Or Equal To Expression
AND	2 IF A<5 AND B<2 THEN 7	(8K Version only) If expression 1 (A<5) AND expression 2 (B<2) are <u>both</u> true, then branch to line 7
OR	IF A<1 OR B<2 THEN 2	(8K Version only) If <u>either</u> expression 1 (A<1) OR expression 2 (B<2) is true, then branch to line 2
NOT	IF NOT Q3 THEN 4	(8K Version only) If expression "NOT Q3" is true (because Q3 is false), then branch to line 4 Note: NOT -1=0 (NOT true=false)

AND, OR and NOT can be used for bit manipulation, and for performing boolean operations.

These three operators convert their arguments to sixteen bit, signed two's, complement integers in the range -32768 to +32767. They then perform the specified logical operation on them and return a result within the same range. If the arguments are not in this range, an "FC" error results.

The operations are performed in bitwise fashion, this means that each bit of the result is obtained by examining the bit in the same position for each argument.

The following truth table shows the logical relationship between bits:

<u>OPERATOR</u>	<u>ARG. 1</u>	<u>ARG. 2</u>	<u>RESULT</u>
AND	1	1	1
	0	1	0
	1	0	0
	0	0	0

(cont.)

<u>OPERATOR</u>	<u>ARG. 1</u>	<u>ARG. 2</u>	<u>RESULT</u>
OR	1	1	1
	1	0	1
	0	1	1
	0	0	0
NOT	1	-	0
	0	-	1

EXAMPLES: (In all of the examples below, leading zeroes on binary numbers are not shown.)

- 63 AND 16=16 Since 63 equals binary 11111 and 16 equals binary 10000, the result of the AND is binary 10000 or 16.
- 15 AND 14=14 15 equals binary 1111 and 14 equals binary 1110, so 15 AND 14 equals binary 1110 or 14.
- 1 AND 8=8 -1 equals binary 111111111111111 and 8 equals binary 1000, so the result is binary 1000 or 8 decimal.
- 4 AND 2=0 4 equals binary 100 and 2 equals binary 10, so the result is binary 0 because none of the bits in either argument match to give a 1 bit in the result.
- 4 OR 2=6 Binary 100 OR'd with binary 10 equals binary 110, or 6 decimal.
- 10 OR 10=10 Binary 1010 OR'd with binary 1010 equals binary 1010, or 10 decimal.
- 1 OR -2=-1 Binary 111111111111111 (-1) OR'd with binary 11111111111110 (-2) equals binary 11111111111111, or -1.
- NOT 0=-1 The bit complement of binary 0 to 16 places is sixteen ones (111111111111111) or -1. Also NOT -1=0.
- NOT X NOT X is equal to -(X+1). This is because to form the sixteen bit two's complement of the number, you take the bit (one's) complement and add one.
- NOT 1=-2 The sixteen bit complement of 1 is 111111111111110, which is equal to -(1+1) or -2.

A typical use of the bitwise operators is to test bits set in the ALTAIR's inport ports which reflect the state of some external device. Bit position 7 is the most significant bit of a byte, while position 0 is the least significant.

For instance, suppose bit 1 of I/O port 5 is 0 when the door to Room X is closed, and 1 if the door is open. The following program will print "Intruder Alert" if the door is opened:

```
10 IF NOT (INP(5) AND 2) THEN 10      This line will execute over
                                     and over until bit 1 (mask-
                                     ed or selected by the 2) be-
                                     comes a 1. When that happens,
                                     we go to line 20 .
20 PRINT "INTRUDER ALERT"           Line 20 will output "INTRUDER
                                     ALERT".
```

However, we can replace statement 10 with a "WAIT" statement, which has exactly the same effect.

```
10 WAIT 5,2                          This line delays the execution of the next
                                     statement in the program until bit 1 of
                                     I/O port 5 becomes 1. The WAIT is much
                                     faster than the equivalent IF statement
                                     and also takes less bytes of program
                                     storage.
```

The ALTAIR's sense switches may also be used as an input device by the INP function. The program below prints out any changes in the sense switches.

```
10 A=300:REM SET A TO A VALUE THAT WILL FORCE PRINTING
20 J=INP(255):IF J=A THEN 20
30 PRINT J;:A=J:GOTO 20
```

The following is another useful way of using relational operators:

```
125 A--(B>C)*B-(B<=C)*C      This statement will set the variable
                               A to MAX(B,C) = the larger of the two
                               variables B and C.
```

STATEMENTS

Note: In the following description of statements, an argument of V or W denotes a numeric variable, X denotes a numeric expression, X\$ denotes a string expression and an I or J denotes an expression that is truncated to an integer before the statement is executed. Truncation means that any fractional part of the number is lost, e.g. 3.9 becomes 3, 4.01 becomes 4.

An expression is a series of variables, operators, function calls and constants which after the operations and function calls are performed using the precedence rules, evaluates to a numeric or string value.

A constant is either a number (3.14) or a string literal ("FOO").

<u>NAME</u>	<u>EXAMPLE</u>	<u>PURPOSE/USE</u>
<u>DATA</u>	10 DATA 1,3,-1E3,.04	Specifies data, read from left to right. Information appears in data statements in the same order as it will be read in the program. IN THE 4K VERSION OF BASIC, DATA STATEMENTS MUST BE THE FIRST STATEMENTS ON A LINE. Expressions may also appear in the 4K version data statements.
	20 DATA " F00",Z00	(8K Version) Strings may be read from DATA statements. If you want the string to contain leading spaces (blanks), colons (:), or commas (,), you must enclose the string in double quotes. It is impossible to have a double quote within string data or a string literal. ("MITS" is illegal)
<u>DEF</u>	100 DEF FNA(V)=V/B+C	(8K Version) The user can define functions like the built-in functions (SQRT, SGN, ABS, etc.) through the use of the DEF statement. The name of the function is "FN" followed by any legal variable name, for example: FNX, FNJ7, FNKO, FNR2. User defined functions are restricted to one line. A function may be defined to be any expression, but may only have one argument. In the example B & C are variables that are used in the program. Executing the DEF statement defines the function. User defined functions can be redefined by executing another DEF statement for the same function. User defined string functions are not allowed. "V" is called the dummy variable.
	110 Z=FNA(3)	Execution of this statement following the above would cause Z to be set to 3/B+C, but the value of V would be unchanged.
<u>DIM</u>	113 DIM A(3),B(10)	Allocates space for matrices. All matrix elements are set to zero by the DIM statement.
	114 DIM R3(5,5),D*(2,2,2)	(8K Version) Matrices can have more than one dimension. Up to 255 dimensions are allowed, but due to the restriction of 72 characters per line the practical maximum is about 34 dimensions.
	115 DIM Q1(N),Z(2*1)	Matrices can be dimensioned dynamically during program execution. If a matrix is not explicitly dimensioned with a DIM statement, it is assumed to be a single dimensioned matrix of whose single subscript

117 A(8)=4

may range from 0 to 10 (eleven elements). If this statement was encountered before a DIM statement for A was found in the program, it would be as if a DIM A(10) had been executed previous to the execution of line 117. All subscripts start at zero (0), which means that DIM X(100) really allocates 101 matrix elements.

END 999 END

Terminates program execution without printing a BREAK message. (see STOP) CONT after an END statement causes execution to resume at the statement after the END statement. END can be used anywhere in the program, and is optional.

FOR 300 FOR V=1 TO 9.3 STEP .6

(see NEXT statement) V is set equal to the value of the expression following the equal sign, in this case 1. This value is called the initial value. Then the statements between FOR and NEXT are executed. The final value is the value of the expression following the TO. The step is the value of the expression following STEP. When the NEXT statement is encountered, the step is added to the variable.

310 FOR V=1 TO 9.3

If no STEP was specified, it is assumed to be one. If the step is positive and the new value of the variable is \leq the final value (9.3 in this example), or the step value is negative and the new value of the variable is \geq the final value, then the first statement following the FOR statement is executed. Otherwise, the statement following the NEXT statement is executed. All FOR loops execute the statements between the FOR and the NEXT at least once, even in cases like FOR V=1 TO 0.

315 FOR V=10*N TO 3.4/0 STEP

SQR(R) Note that expressions (formulas) may be used for the initial, final and step values in a FOR loop. The values of the expressions are computed only once, before the body of the FOR...NEXT loop is executed.

320 FOR V=9 TO 1 STEP -1 When the statement after the NEXT is executed, the loop variable is never equal to the final value, but is equal to whatever value caused the FOR...NEXT loop to terminate. The statements between the FOR and its corresponding NEXT in both examples above (310 & 320) would be executed 9 times.

330 FOR W=1 TO 10: FOR W=1 TO :NEXT W:NEXT W Error: do not use nested FOR...NEXT loops with the same index variable. FOR loop nesting is limited only by the available memory. (see Appendix D)

GOTO 50 GOTO 100 Branches to the statement specified.

GOSUB 10 GOSUB 910 Branches to the specified statement (910) until a RETURN is encountered; when a branch is then made to the statement after the GOSUB. GOSUB nesting is limited only by the available memory. (see Appendix D)

IF...GOTO 32 IF X<=Y+23.4 GOTO 92 (8K Version) Equivalent to IF...THEN, except that IF...GOTO must be followed by a line number, while IF...THEN can be followed by either a line number or another statement.

IF...THEN IF X<10 THEN 5 Branches to specified statement if the relation is True.
 20 IF X<0 THEN PRINT "X LESS THAN 0" Executes all of the statements on the remainder of the line after the THEN if the relation is True.
 25 IF X=5 THEN 50:Z=A WARNING. The "Z=A" will never be executed because if the relation is true, BASIC will branch to line 50. If the relation is false Basic will proceed to the line after line 25.
 26 IF X<0 THEN PRINT "ERROR, X NEGATIVE": GOTO 350 In this example, if X is less than 0, the PRINT statement will be executed and then the GOTO statement will branch to line 350. If the X was 0 or positive, BASIC will proceed to execute the lines after line 26.

INPUT 3 INPUT V,W,W2

Requests data from the terminal (to be typed in). Each value must be separated from the preceding value by a comma (,). The last value typed should be followed by a carriage return. A "?" is typed as a prompt character. In the 4K version, a value typed in as a response to an INPUT statement may be a formula, such as $2*\text{SIN}(.16)-3$. However, in the 8K version, only constants may be typed in as a response to an INPUT statement, such as $4.5\text{E}-3$ or "CAT". If more data was requested in an INPUT statement than was typed in, a "??" is printed and the rest of the data should be typed in. If more data was typed in than was requested, the extra data will be ignored. The 8K version will print the warning "EXTRA IGNORED" when this happens. The 4K version will not print a warning message. (8K Version) Strings must be input in the same format as they are specified in DATA statements.

5 INPUT "VALUE";V

(8K Version) Optionally types a prompt string ("VALUE") before requesting data from the terminal. If carriage return is typed to an input statement, BASIC returns to command mode. Typing CONT after an INPUT command has been interrupted will cause execution to resume at the INPUT statement.

LET 300 LET W=X
310 V=5-1

Assigns a value to a variable. "LET" is optional.

NEXT 340 NEXT V
345 NEXT
350 NEXT V,W

Marks the end of a FOR loop. (8K Version) If no variable is given, matches the most recent FOR loop. (8K Version) A single NEXT may be used to match multiple FOR statements. Equivalent to NEXT V:NEXT W.

ON...GOTO

100 ON I GOTO 10,20,30,40 (8K Version) Branches to the line indicated by the I'th number after the GOTO. That is:
IF I=1, THEN GOTO LINE 10
IF I=2, THEN GOTO LINE 20
IF I=3, THEN GOTO LINE 30
IF I=4, THEN GOTO LINE 40.

If I=0 or I attempts to select a non-existent line (≥ 5 in this case), the statement after the ON statement is executed. However, if I is >255 or <0 , an FC error message will result. As many line numbers as will fit on a line can follow an ON...GOTO.

105 ON SGN(X)+2 GOTO 40,50,60

This statement will branch to line 40 if the expression X is less than zero, to line 50 if it equals zero, and to line 60 if it is greater than zero.

ON...GOSUB

110 ON I GOSUB 50,60

(8K Version) Identical to "ON...GOTO", except that a subroutine call (GOSUB) is executed instead of a GOTO. RETURN from the GOSUB branches to the statement after the ON...GOSUB.

OUT

355 OUT I,J

(8K Version) Sends the byte J to the output port I. Both I & J must be ≥ 0 and ≤ 255 .

POKE

357 POKE I,J

(8K Version) The POKE statement stores the byte specified by its second argument (J) into the location given by its first argument (I). The byte to be stored must be ≥ 0 and ≤ 255 , or an FC error will occur. The address (I) must be ≥ 0 and ≤ 32767 , or an FC error will result. Careless use of the POKE statement will probably cause you to "poke" BASIC to death; that is, the machine will hang, and you will have to reload BASIC and will lose any program you had typed in. A POKE to a non-existent memory location is harmless. One of the main uses of POKE is to pass arguments to machine language subroutines. (see Appendix J) You could also use PEEK and POKE to write a memory diagnostic or an assembler in BASIC.

PRINT

360 PRINT X,Y;Z

370 PRINT

380 PRINT X,Y;

390 PRINT "VALUE IS";A

400 PRINT A2,B,

Prints the value of expressions on the terminal. If the list of values to be printed out does not end with a comma (,) or a semicolon (;), then a carriage return/line feed is executed after all the values have been printed. Strings enclosed in quotes (") may also be printed. If a semicolon separates two expressions in the list, their values are printed next to each other. If a comma appears after an

expression in the list, and the print head is at print position 56 or more, then a carriage return/line feed is executed. If the print head is before print position 56, then spaces are printed until the carriage is at the beginning of the next 14 column field (until the carriage is at column 14, 28, 42 or 56...). If there is no list of expressions to be printed, as in line 370 of the examples, then a carriage return/line feed is executed.

410 PRINT MID\$(A#,2); (8K Version) String expressions may be printed.

READ

490 READ V-W

Reads data into specified variables from a DATA statement. The first piece of data read will be the first piece of data listed in the first DATA statement of the program. The second piece of data read will be the second piece listed in the first DATA statement, and so on. When all of the data have been read from the first DATA statement, the next piece of data to be read will be the first piece listed in the second DATA statement of the program. Attempting to read more data than there is in all the DATA statements in a program will cause an OD (out of data) error. In the 4K version, an SN error from a READ statement can mean the data it was attempting to read from a DATA statement was improperly formatted. In the 8K version, the line number given in the SN error will refer to the line number where the error actually is located.

REM

500 REM NOW SET V=0

Allows the programmer to put comments in his program. REM statements are not executed, but can be branched to. A REM statement is terminated by end of line, but not by a ":".

505 REM SET V=0: V=0

In this case the V=0 will never be executed by BASIC.

506 V=0: REM SET V=0

In this case V=0 will be executed

RESTORE

510 RESTORE

Allows the re-reading of DATA statements. After a RESTORE, the next piece of data read will be the first piece listed in the first DATA statement of the program. The second piece of data read will be the second piece listed in the first DATA statement, and so on as in a normal READ operation.

RETURN	50 RETURN	Causes a subroutine to return to the statement after the most recently executed GOSUB.
STOP	9000 STOP	Causes a program to stop execution and to enter command mode. (8K Version) Prints BREAK IN LINE 9000. (as per this example) CONT after a STOP branches to the statement following the STOP.
WAIT	805 WAIT I,J,K 806 WAIT I,J	(8K Version) This statement reads the status of input port I, exclusive OR's K with the status, and then AND's the result with J until a non-zero result is obtained. Execution of the program continues at the statement following the WAIT statement. If the WAIT statement only has two arguments, K is assumed to be zero. If you are waiting for a bit to become zero, there should be a one in the corresponding position of K. I, J and K must be ≥ 0 and ≤ 255 .

4K INTRINSIC FUNCTIONS

ABS(X)	120 PRINT ABS(X)	Gives the absolute value of the expression X. ABS returns X if $X \geq 0$, -X otherwise.
INT(X)	140 PRINT INT(X)	Returns the largest integer less than or equal to its argument X. For example: INT(.23)=0, INT(7)=7, INT(-.1)=-1, INT(-2)=-2, INT(1.1)=1. The following would round X to D decimal places: $\text{INT}(X*10^D+.5)/10^D$
RND(X)	170 PRINT RND(X)	Generates a random number between 0 and 1. The argument X controls the generation of random numbers as follows: X<0 starts a new sequence of random numbers using X. Calling RND with the same X starts the same random number sequence. X=0 gives the last random number generated. Repeated calls to RND(0) will always return the same random number. X>0 generates a new random number between 0 and 1. Note that $(B-A)*\text{RND}(1)+A$ will generate a random number between A & B.

SGN(X)	230 PRINT SGN(X)	Gives 1 if X>0, 0 if X=0, and -1 if X<0.
SIN(X)	190 PRINT SIN(X)	Gives the sine of the expression X. X is interpreted as being in radians. Note: COS (X)=SIN(X+3.14159/2) and that 1 Radian =180/PI degrees=57.2958 degrees; so that the sine of X degrees= SIN(X/57.2958).
SQR(X)	180 PRINT SQR(X)	Gives the square root of the argument X. An FC error will occur if X is less than zero.
TAB(I)	240 PRINT TAB(I)	Spaces to the specified print position (column) on the terminal. May be used only in PRINT statements. Zero is the leftmost column on the terminal, 71 the rightmost. If the carriage is beyond position I, then no printing is done. I must be =>0 and <=255.
USR(I)	200 PRINT USR(I)	Calls the user's machine language subroutine with the argument I. See POKE, PEEK and Appendix J.

8K FUNCTIONS *(Includes all those listed under 4K INTRINSIC FUNCTIONS plus the following in addition.)*

ATN(X)	210 PRINT ATN(X)	Gives the arctangent of the argument X. The result is returned in radians and ranges from -PI/2 to PI/2. (PI/2=1.5708)
COS(X)	200 PRINT COS(X)	Gives the cosine of the expression X. X is interpreted as being in radians.
EXP(X)	150 PRINT EXP(X)	Gives the constant "E" (2.71828) raised to the power X. (E^X) The maximum argument that can be passed to EXP without overflow occurring is 87.3365.
FRE(X)	270 PRINT FRE(0)	Gives the number of memory bytes currently unused by BASIC. Memory allocated for STRING space is not included in the count returned by FRE. To find the number of free bytes in STRING space, call FRE with a STRING argument. (see FRE under STRING FUNCTIONS)
INP(I)	265 PRINT INP(I)	Gives the status of (reads a byte from) input port I. Result is =>0 and <=255.

LOG(X)	160 PRINT LOG(X)	Gives the natural (Base E) logarithm of its argument X. To obtain the Base Y logarithm of X use the formula LOG(X)/LOG(Y). Example: The base 10 (common) log of 7 = LOG(7)/ LOG(10).
PEEK	356 PRINT PEEK(I)	The PEEK function returns the contents of memory address I. The value returned will be =>0 and <=255. If I is >32767 or <0, an FC error will occur. An attempt to read a non-existent memory address will return 255. (see POKE statement)
POS(I)	260 PRINT POS(I)	Gives the current position of the terminal print head (or cursor on CRT's). The leftmost character position on the terminal is position zero and the rightmost is 71.
SPC(I)	250 PRINT SPC(I)	Prints I space (or blank) characters on the terminal. May be used only in a PRINT statement. X must be =>0 and <=255 or an FC error will result.
TAN(X)	200 PRINT TAN(X)	Gives the tangent of the expression X. X is interpreted as being in radians.

STRINGS (8K Version Only)

- 1) A string may be from 0 to 255 characters in length. All string variables end in a dollar sign (\$); for example, A\$, B9\$, K\$, HELLO\$.
- 2) String matrices may be dimensioned exactly like numeric matrices. For instance, DIM A\$(10,10) creates a string matrix of 121 elements, eleven rows by eleven columns (rows 0 to 10 and columns 0 to 10). Each string matrix element is a complete string, which can be up to 255 characters in length.
- 3) The total number of characters in use in strings at any time during program execution cannot exceed the amount of string space, or an OS error will result. At initialization, you should set up string space so that it can contain the maximum number of characters which can be used by strings at any one time during program execution.

<u>NAME</u>	<u>EXAMPLE</u>	<u>PURPOSE/USE</u>
DIM	25 DIM A\$(10,10)	Allocates space for a pointer and length for each element of a string matrix. No string space is allocated. See Appendix D.

LET	27 LET A#="FOO"+V#	Assigns the value of a string expression to a string variable. LET is optional.
=		String comparison operators. Comparison is made on the basis of ASCII codes, a character at a time until a difference is found. If during the comparison of two strings, the end of one is reached, the shorter string is considered smaller. Note that "A " is greater than "A" since trailing spaces are significant.
>		
<		
<=		
>=		
<>		
+	30 LET Z#=R#+Q#	String concatenation. The resulting string must be less than 256 characters in length or an LS error will occur.
INPUT	40 INPUT X#	Reads a string from the user's terminal. String does not have to be quoted; but if not, leading blanks will be ignored and the string will be terminated on a ",", " or ":" character.
READ	50 READ X#	Reads a string from DATA statements within the program. Strings do not have to be quoted; but if they are not, they are terminated on a ",", " or ":" character or end of line and leading spaces are ignored. See DATA for the format of string data.
PRINT	60 PRINT X# 70 PRINT "FOO"+A#	Prints the string expression on the user's terminal.

STRING FUNCTIONS (8K Version Only)

ASC(X#)	300 PRINT ASC(X#)	Returns the ASCII numeric value of the first character of the string expression X\$. See Appendix K for an ASCII/number conversion table. An FC error will occur if X\$ is the null string.
CHR#(I)	275 PRINT CHR#(I)	Returns a one character string whose single character is the ASCII equivalent of the value of the argument (I) which must be =>0 and <=255. See Appendix K.
FRE(X#)	272 PRINT FRE(" ")	When called with a string argument, FRE gives the number of free bytes in string space.
LEFT#(X#,I)	310 PRINT LEFT#(X#,I)	Gives the leftmost I characters of the string expression X\$. If I<=0 or >255 an FC error occurs.

LEN(X#) 220 PRINT LEN(X#) Gives the length of the string expression X\$ in characters (bytes). Non-printing characters and blanks are counted as part of the length.

MID\$(X#,I) 330 PRINT MID\$(X#,I) MID\$ called with two arguments returns characters from the string expression X\$ starting at character position I. If I>LEN(I\$), then MID\$ returns a null (zero length) string. If I<=0 or >255, an FC error occurs.

MID\$(X#,I,J) 340 PRINT MID\$(X#,I,J) MID\$ called with three arguments returns a string expression composed of the characters of the string expression X\$ starting at the Ith character for J characters. If I>LEN(X\$), MID\$ returns a null string. If I or J <=0 or >255, an FC error occurs. If J specifies more characters than are left in the string, all characters from the Ith on are returned.

RIGHT\$(X#,I) 320 PRINT RIGHT\$(X#,I) Gives the rightmost I characters of the string expression X\$. When I<=0 or >255 an FC error will occur. If I>=LEN(X\$) then RIGHT\$ returns all of X\$.

STR\$(X) 290 PRINT STR\$(X) Gives a string which is the character representation of the numeric expression X. For instance, STR\$(3.1)=" 3.1".

VAL(X#) 280 PRINT VAL(X#) Returns the string expression X\$ converted to a number. For instance, VAL("3.1")=3.1. If the first non-space character of the string is not a plus (+) or minus (-) sign, a digit or a decimal point (.) then zero will be returned.

SPECIAL CHARACTERS

<u>CHARACTER</u>	<u>USE</u>
@	Erases current line being typed, and types a carriage return/line feed. An "@" is usually a shift/P.
←	<i>(backarrow or underline)</i> Erases last character typed. If no more characters are left on the line, types a carriage return/line feed. "←" is usually a shift/O.

CARRIAGE RETURN A carriage return must end every line typed in. Returns print head or CRT cursor to the first position (leftmost) on line. A line feed is always executed after a carriage return.

CONTROL/C Interrupts execution of a program or a list command. Control/C has effect when a statement finishes execution, or in the case of interrupting a LIST command, when a complete line has finished printing. In both cases a return is made to BASIC's command level and OK is typed.
(8K Version) Prints "BREAK IN LINE XXXX" , where XXXX is the line number of the next statement to be executed.

: (colon) A colon is used to separate statements on a line. Colons may be used in direct and indirect statements. The only limit on the number of statements per line is the line length. It is not possible to GOTO or GOSUB to the middle of a line.

(8K Version Only)

CONTROL/O Typing a Control/O once causes BASIC to suppress all output until a return is made to command level, an input statement is encountered, another control/O is typed, or an error occurs.

? Question marks are equivalent to PRINT. For instance, ? 2+2 is equivalent to PRINT 2+2. Question marks can also be used in indirect statements. 10 ? X, when listed will be typed as 10 PRINT X.

MISCELLANEOUS

- 1) To read in a paper tape with a program on it (8K Version), type a control/O and feed in tape. There will be no printing as the tape is read in. Type control/O again when the tape is through. Alternatively, set nulls=0 and feed in the paper tape, and when done reset nulls to the appropriate setting for your terminal. Each line must be followed by two rubouts, or any other non-printing character. If there are lines without line numbers (direct commands) the ALTAIR will fall behind the input coming from paper tape, so this is not recommending.

Using null in this fashion will produce a listing of your tape in the 8K version (use control/O method if you don't want a listing). The null method is the only way to read in a tape in the 4K version.

To read in a paper tape of a program in the 4K version, set the number of nulls typed on carriage return/line feed to zero by patching location 46 (octal) to be a 1. Feed in the paper tape. When

the tape has finished reading, stop the CPU and repatch location 46 to be the appropriate number of null characters (usually 0, so deposit a 1). When the tape is finished, BASIC will print SN ERROR because of the "OK" at the end of the tape.

- 2) To punch a paper tape of a program, set the number of nulls to 3 for 110 BAUD terminals (Teletypes) and 6 for 300 BAUD terminals. Then, type LIST; but, do not type a carriage return. Now, turn on the terminal's paper tape punch. Put the terminal on local and hold down the Repeat, Control, Shift and P keys at the same time. Stop after you have punched about a 6 to 8 inch leader of nulls. These nulls will be ignored by BASIC when the paper tape is read in. Put the terminal back on line. Now hit carriage return. After the program has finished punching, put some trailer on the paper tape by holding down the same four keys as before, with the terminal on local. After you have punched about a six inch trailer, tear off the paper tape and save for later use as desired.
- 3) Restarting BASIC at location zero (by toggling STOP, Examine location 0, and RUN) will cause BASIC to return to command level and type "OK". However, typing Control/C is preferred because Control/C is guaranteed not to leave garbage on the stack and in variables, and a Control C'd program may be continued. (see CONT command)
- 4) The maximum line length is 72 characters** If you attempt to type too many characters into a line, a bell (ASCII 7) is executed, and the character you typed in will not be echoed. At this point you can either type backarrow to delete part of the line, or at-sign to delete the whole line. The character you typed which caused BASIC to type the bell is not inserted in the line as it occupies the character position one beyond the end of the line.

*CLEAR	CLEAR	Deletes all variables.
	CLEAR X	(8K Version) Deletes all variables. When used with an argument "X", sets the amount of space to be allocated for use by string variables to the number indicated by its argument "X".
	LD CLEAR SD	(8K Version) Same as above; but, may be used at the beginning of a program to set the exact amount of string space needed, leaving a maximum amount of memory for the program itself.

NOTE: If no argument is given, the string space is set at 200 by default. An OM error will occur if an attempt is made to allocate more string space than there is available memory.

**For inputting only.

