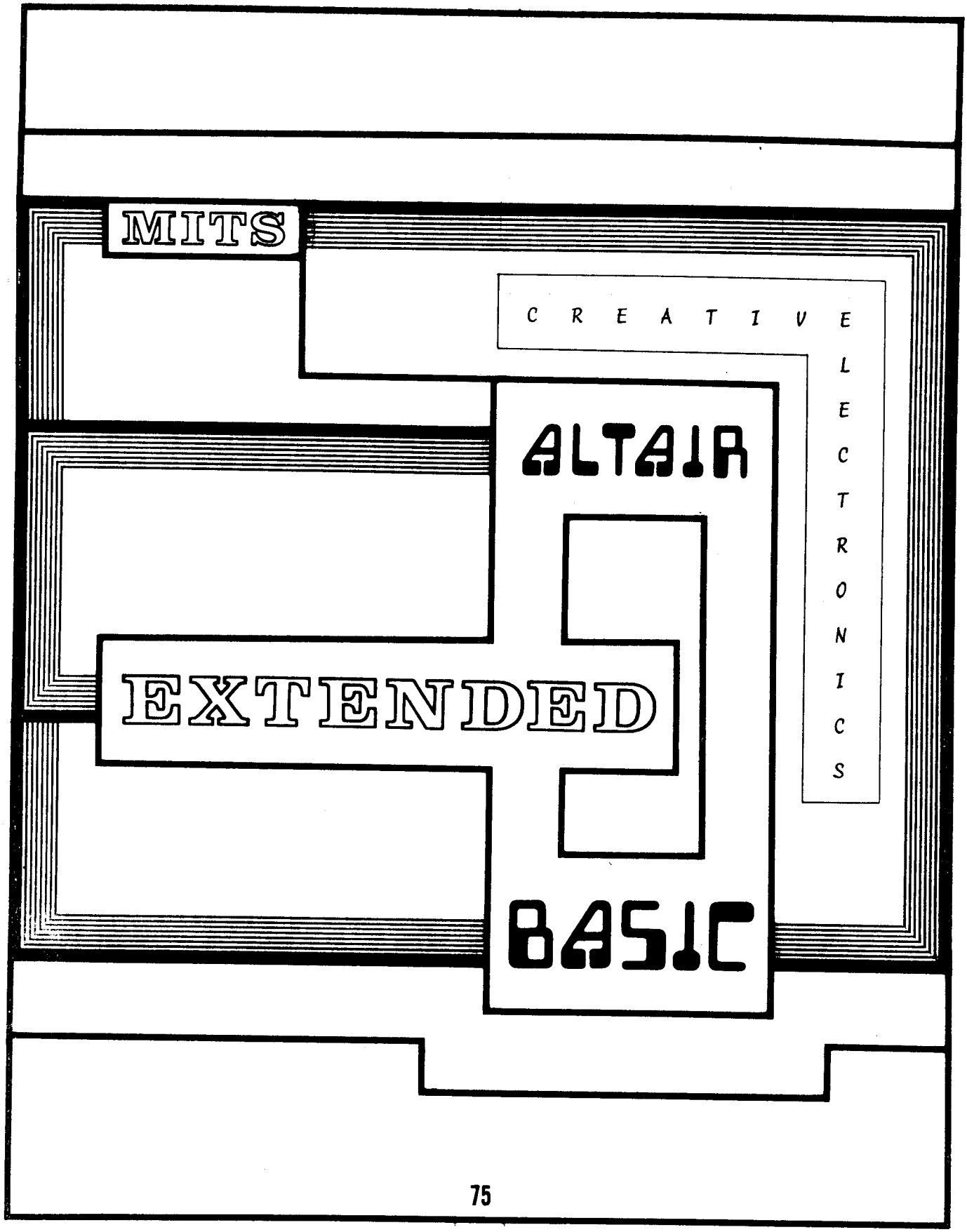

MITS
CREATIVE ELECTRONICS
ALTAIR
EXTENDED
BASIC

# ALTAIR EXTENDED BASIC

## PRELIMINARY DOCUMENTATION

THE FOLLOWING PAGES CONTAIN A CONDENSED VERSION OF THE COMPLETE "ALTAIR EXTENDED BASIC" DOCUMENTATION.

In order to get this software to our customers with a minimum of delay, it was decided to print this preliminary documentation. This will help to expedite the deliveries. The complete manual will be printed at a later date, and will be in much the same format as the previous existing BASIC documentation.

READ THESE PAGES OVER CAREFULLY. SOME OF THE INFORMATION CONTAINED HERE ALSO APPLIES TO THE 4K AND 8K VERSIONS OF BASIC.

This is meant to be an additional section to the "ALTAIR BASIC REFERENCE MANUAL", and not a separate manual in itself.

December '75

MITS

"Creative Electronics"

# ALTAIR EXTENDED BASIC

ALTAIR EXTENDED BASIC includes all of the features found in the 8K version of BASIC, with some variations. There are also a large number of additional features making this version one of the most powerful BASICs available.

The following section contains the EXTENDED BASIC features and its variations from the 8K BASIC.

## COMMANDS

| NAME | EXAMPLE | PURPOSE/USE |
|---|---|---|
| DELETE | DELETE X | Deletes line in a program with the line number "X". "ILLEGAL FUNCTION CALL" error occurs if there is no line "X". |
| | DELETE -X | Deletes all lines in a program up to and including line number "X". "ILLEGAL FUNCTION CALL" if no line "X". |
| | DELETE Y-X | Deletes all lines in a program from the line number equal to or greater than "Y" up to and including the first line equal to or less than "X". "ILLEGAL FUNCTION CALL" if no line "X". |
| | If deletion is performed, all variable values are lost. Also continuing is not allowed, and all "FOR"s and "GOSUB"s are made inactive. (This is the same effect caused whenever a program is modified.) | |
| LIST | LIST X | Lists line "X" if there is one. |
| | LIST or LIST- | Lists the entire program. |
| | LIST X- | Lists all lines in a program with a line number equal to or greater than "X". |
| | LIST -X | Lists all of the lines in a program with a line number less than or equal to "X". |
| | LIST Y-X | Lists all of the lines within a program with line numbers equal to or greater than "Y", and less than or equal to "X". |

| NAME | EXAMPLE | PURPOSE/USE |
|---|---|---|
| ERASE | ERASE J%<br>ERASE X%,I#<br>ERASE A$<br>ERASE D#,NMS% | Eliminates an array. If no such array exists an "ILLEGAL FUNCTION CALL" error will occur. ERASE must refer to an array, not an array element [ERASE B(9) would be illegal]. The space the array is using is freed up and made available for other uses. The array can be dimensioned again, but the values before the ERASE are lost. |
| SWAP | SWAP I%,J%<br>SWAP B$(7),T$<br>SWAP D#(I),D#(I+1) | Exchanges the value of two variables. (If X=1 & Y=5, after SWAP X,Y the values would be switched; that is, now X=5 & Y=1.) Both, one or neither of the variables may be array elements. If a non-array variable that has not been assigned a value is referenced an "ILLEGAL FUNCTION CALL" error will occur. Both variables must be of the same type (both integers, both strings, both double precision or both single precision), otherwise a "TYPE MISMATCH" error will occur. |
| TRON | TRON | Turns on the trace flag. |
| TROFF | TROFF | Turns off the trace flag. |

TRON & TROFF can be given in either direct or indirect (program) mode. When the trace flag is on, each time a new program line is started, that line number is printed enclosed in "[]". No spaces are printed. For example:

```
TRON
OK
10 PRINT 1:  PRINT "A"
20 STOP
RUN
[10] 1
A
[20]
BREAK IN 20
```

"NEW" will also turn off the trace flag along with its other functions.

IF-THEN-ELSE

(Similar to 8K version IF-THEN statement, only with the addition of a new "ELSE" clause.)

```
IF X>Y THEN PRINT "GREATER" ELSE PRINT "NOT GREATER"
```

In the above example, first the relational condition would be tested. If it is true, the THEN clause would be executed ("GREATER" would be printed). If it is false, the ELSE clause would be executed ("NOT GREATER" would be printed).

```
10  IF A>B THEN PRINT "A>B" ELSE IF B>A THEN PRINT "B>A" ELSE PRINT "A=B"
```

The above example would indicate which of the two variables was the largest, or if they were equal. As this example indicates, IF statements may be nested to any desired level (regulated only by the maximum line length). An IF-THEN-ELSE statement may appear anywhere within a multiple-statement line; the THEN clause being always mandatory with each IF clause and the ELSE clause optional. Care must be taken to insure that IFs without ELSE clauses do not cause an ELSE to be associated with the wrong IF.

```
5  IF A=B THEN IF A=C THEN PRINT "A=C" ELSE PRINT "A<>C" ELSE PRINT "A<>B"
```

In the above example, the double under-lined portion of the line is an IF-THEN-ELSE statement which is all a part of the THEN clause of the first IF statement in the line. The second ELSE (single under-lined) is part of the first IF, and will be executed only if the first relational expression is false (A<>B). If a line does not contain the same number of ELSE and THEN clauses, the last ELSE is matched with the closest THEN.

## TYPING

Normally, numbers used in BASIC operations are stored and acted upon as single precision floating point numbers. This allows for 7 digits of accuracy.

In the extended version of BASIC greater accuracy may be obtained by typing numbers as double precision. This allows for 16 digits of accuracy. In cases where speed is critical, it is, however, slower than single precision.

The greatest advantage, in both speed and storage space can be obtained by using integer operations whenever possible. These fall within the rage <=32767 to >=-32768.

Examples:

```
(single precision)  PRINT 1/3
                    .3333333

(double precision)  PRINT 1/3D
                   .3333333333333333

(integer)           PRINT 1/3%
                      0
                    PRINT 2.76%
                      2
```

The use of these types of numbers will become clearer further on in the text.

Examples:

I%(10) uses (11 * 2) + 6 + (2 * 1) = 30

I (5,5) uses (6 * 6 * 4) + 6 + (2 * 2) = 154

## TYPING

There are four types of values used in EXTENDED BASIC programming:

| NAME | SYMBOL | # OF BYTES/VALUE |
|---|---|---|
| STRINGS (0 to 255 characters) | $ | 3 |
| INTEGERS (must be -32768 and =< 32767) | % | 2 |
| DOUBLE PRECISION (exponent: -38 to +38) 16 digits | # | 8 |
| SINGLE PRECISION (exponent: -38 to +38) 7 digits | ! | 4 |

The type a variable will be is explicitly declared by using one of the four symbols listed above. Otherwise, the first letter of the variable is used to look into the table that indicates the default type for that letter. Initially (after CLEAR, after RUN, after NEW, or after modifying a program) all letters are defaulted to SINGLE PRECISION.

The following four statements can be used to modify the DEFAULT table:

| STATEMENT | DEFAULTS VARIABLE TO |
|---|---|
| DEFINT r | INTEGER |
| DEFSTR r | STRING |
| DEFDBL r | DOUBLE PRECISION |
| DEFSNG r | SINGLE PRECISION |

r above indicates the position for the range to be given. This is to be of the following format: letter or letter 1 - letter 2. (In the second format, the "-" indicates from letter 1 through letter 2 inclusive.)

In the above four statements the default type of all of the letters within the range is changed, depending on which DEF "type" is used. Initially, DEFSNG A-Z is assumed. Care should be taken when using these statements since variables referred to without type indicators may not be the same after the statement is executed. It is recommended that these statements be used only at the start of a program, before any other statements are executed.

The following will illustrate some of the above information:

```
 10 I%=1
 20 I!=2
 30 I#=3
 40 I$="ABC"
 50 PRINT I
 60 DEFINT I
 70 PRINT I
 80 DEFSTR I
 90 PRINT I
100 DEFDBL I
110 PRINT I
```

The example on the left would print out:

```
2    at line # 50
1    at line # 70
ABC  at line # 90
3    at line # 110
```

## TYPING OF CONSTANTS

The type that a particular constant will be is determined by the following:

1) if it is more than 7 digits or "D" is used in the exponent, then it will be DOUBLE PRECISION.

2) if it is >32767 or <-32768, a decimal point (.) is used, or an "E" is used, then it is SINGLE PRECISION.

3) otherwise, it is an integer.

When a + or * operation or a comparison is performed, the operands are converted to both be of the same type as the most accurate operand. Therefore, if one or both operands are double precision, the operation is done in double precision (accurate but slow). If neither is double precision but one or more operands are single precision floating point, then the operation will be done in single precision floating point. Otherwise, both operands must be integers, and the operation is performed in integer representation.

If the result of an integer + or * is too big to be an integer, the operation will be done in single precision and the result will be single precision. Division (/) is done the same as the above operator, except it is never done at the integer level. If both operands are integers, the operation is done as a single precision divide.

The operators AND, OR, NOT, \, and MOD force both operands to be integers before the operation is done. If one of the operands is >32767 or <-32768, an overflow error will occur. The result of these operators will always be an integer. (Except -32768\-1 gives single precision.)

No matter what the operands to ↑ are, they will both be converted to single precision. The functions SIN, COS, ATN, TAN, SQR, LOG, EXP, and RND also convert their arguments to single precision and give the result as such, accurate to 6 digits.

Using a subscript >32767 and assigning an integer variable a value too large to be an integer gives an overflow error.

## TYPE CONVERSION

When a number is converted to an integer, it is truncated (rounded down). For example:

```
I%=.999                A%=-.01
PRINT I%               PRINT A%
0                      -1
```

It will perform as if the INT function was applied.

When a double precision number is converted to single precision, it is rounded off. For example:

```
D#=77777777
I!=D#
PRINT I!
7.77778E+07
```

No automatic conversion is done between strings and numbers. See the STR$, NUM, ASC, and CHR$ functions for this purpose.

## NEW FUNCTIONS

CINT — Convert the argument to an integer number

CSNG — Convert the argument to a single precision number

CDBL — Convert the argument to a double precision number

Examples:

```
CDBL(3)=3D

CINT(3.9)=3

CINT(-.01)=-1

CSNG(312456.8)=312457
```

NOTE: if X<=32767 and =>-32768 then CINT(X)=INT(X)
otherwise, CINT will give an overflow error

## NEW OPERATORS

\(backslash=shift L)
Integer Division

Examples:
```
1\3=0
7\2=3
-3\-1=3
300\7=42
-8\3=-2
-1\3=0
```

The integer division operator forces both arguments to integers and gives the integer value of the division operation. (The only exception to this is -37268\-1, which results in a value too large to be an integer.)
NOTE: A\B does not equal INT(A/B)
(if A=-1 & B=7, 0 does not equal -1)
Integer division is about eight times as fast as single precision division. Its precedence is just below that of * & /.

## NEW OPERATORS (cont.)

MOD

Examples:
```
4 MOD 7=4
13 MOD 3=1
7 MOD -11=7
-6 MOD -4=-2
```

The MOD operator forces both arguments to integers and returns a result according to the following formula:

A MOD B = A - [B * (A\B)]

If B=0 then a division by zero error will occur. MODs precedence is just below that of integer division and just above + and -.

## USER-DEFINED-FUNCTIONS

In the Extended version of BASIC, a user-defined function can be of any type and can take any number of arguments of any type.

Examples:
```
DEF FNRANDOM%=10*RND(1)+1

DEF FNTWO$(X$)=X$+X$

DEF FNA(X,Y,Z,I%)=X↑Z+I%*Y
```

The result of the function will be forced to the function type before the value is substituted into the formula with the function call.

## FOR LOOPS (Integer)

The loop variable in a FOR loop can be an integer as well as a single precision number. Attempting to use a string or double precision variable as the loop variable will cause a Type Mismatch error to occur. Integer FOR loops are about three times as fast as single precision FOR loops. If the addition of the increment to the loop variable gives a result that is too big to be an integer, an overflow error will occur. The initial loop value, increment value and the final value must all be in the legal range for integers or an overflow error will occur when the FOR is executed.

Example:

```
1 FOR I%=20000 TO 30000 STEP 20000
2 PRINT I%
3 NEXT I%
RUN
20000
OVERFLOW IN 3
OK
```

## NEW ERROR MESSAGES

These messages replace the old error messages listed in APPENDIX C (p. 53) of the BASIC manual.

```
NEXT WITHOUT FOR
SYNTAX ERROR
RETURN WITHOUT GOSUB
OUT OF DATA
ILLEGAL FUNCTION CALL
OVERFLOW
OUT OF MEMORY
UNDEFINED STATEMENT
SUBSCRIPT OUT OF RANGE
REDIMENSIONED ARRAY
DIVISION BY ZERO
ILLEGAL DIRECT
TYPE MISMATCH
OUT OF STRING SPACE
STRING TOO LONG
STRING FORMULA TOO COMPLEX
CAN'T CONTINUE
UNDEFINED USER FUNCTION
```

Examples:

```
10 GOTO 50
RUN
UNDEFINED STATEMENT IN 50
OK
PRINT 1/0
DIVISION BY ZERO
OK
```

## ADDITIONAL NOTES ON EXTENDED BASIC

**PEEK & POKE**

In the 8K version of BASIC you can't PEEK at or POKE into memory locations above 32767. In the Extended version this can be done by using a negative argument. If the address to be PEEKed or POKEd is greater than 32767, subtract 65536 from it to give the proper argument.

Examples:

```
to PEEK at 65535      PEEK(-1)

to POKE at 32768      POKE -32768,I%
```

**INT**

The INT function will work on numbers both single & double precision which are too large to be integers. Double precision numbers maintain full accuracy. (see CINT)

Examples:

```
INT(1E38)=1E38

INT(123456789.6)=123456789
```

ADDITIONAL NOTES (cont.) (miscellaneous)

Extended BASIC uses 10.2K of memory to reside.

String space is defaulted to 100 in the Extended version.

A comma before the THEN in an IF statement is allowed.

USR pass routine [4,5] passes in [H,L] not [D,E], and the pass back routine [6,7] receives in [H,L] not [A,B].

Files CSAVEd in 8K BASIC cannot be CLOADed in EXTENDED BASIC, nor the opposite.

UPDATE TO EXISTING MATERIAL

In cassette BASICs (both 8K* and Extended), CLOAD? some character file name, reads the specified file and checks it against the file in core. If the files do not match, the message "NO GOOD" is printed. If they do match, BASIC returns to command level and prints "OK".

In the Extended version of BASIC, active FOR loops (integer or single precision) require 17 bytes.

Each non-array

| | | |
|---|---|---|
| string | variable uses | 6 bytes. |
| integer | | 5 |
| double precision | | 11 |
| single precision | | 7 |

This is because it takes 3 bytes to store the name of a variable.

Each array uses: (# of elements)* [INT=2 / DBL=8 / STR=3 / SNG=4] +6+2*(# of dimensions).

Examples:

I%(10) uses (11*2)+6+(2*1)=30 bytes

I(5,5) uses (6*6*4)+6+(2*2)=154 bytes

Stored programs take exactly the same amount of space as in the 8K version of BASIC, except the reserved word ELSE takes 2 bytes instead of 1 byte as with the other reserved words.

UPDATE TO EXISTING MATERIAL
(Applies to 8K versions 3.2 and later.)

In both Extended & 8K* BASIC, if a number is between >=1E-2 and <1E-1, the number will be printed as:

.OXXXXXX (trailing zeros suppressed)
instead of X.XXXXXXE-2

An 8K BASIC program should run exactly the same under Extended BASIC. No conversion should be necessary.

USRLOC in extended is:

101 octal=65 decimal,
still 111 in 8K and 4K to load.

EXTENDed:

(Non-disk) location 002 in the BOOT should be 57 (8K=37, 4K=17)

UPDATE TO EXISTING MATERIAL
(Applies to page 57 of version 3.2 and later.)

Each active GOSUB takes 5 bytes.

Each active FOR loop takes 16 bytes.

## EDIT COMMAND

The EDIT command is for the purpose of allowing modifications and additions to be made to existing program lines without having to retype the entire line each time.

Commands typed in the EDIT mode are, as a rule, not echoed. Most commands may be preceded by an optional numeric repetition factor which may be used to repeat the command a number of times. This repetition factor should be in the range 0 to 255 (0 is equivalent to 1). If the repetition factor is omitted, it is assumed to be 1. In the following examples a lower case "n" before the command stands for the repetition factor.

In the following description of the EDIT commands, the "cursor" refers to a pointer which is positioned at a character in the line being edited.

To EDIT a line, type EDIT followed by the number of the line and hit the carriage return. The line number of the line being EDITed will be printed, followed by a space. The cursor will now be positioned to the left of the first character in the line.

NOTE: The best way of getting the "feel" of the EDIT command is to try EDITing a few lines yourself. Commands not recognized as part of the EDIT commands will be ignored.

### MOVING THE CURSOR

A space typed in will move the cursor to the right and cause the character passed over to be printed out. A number preceding the space (nS) will cause the cursor to pass over and print out the number (n) of characters chosen.

### INSERTING CHARACTERS

I Inserts new characters into the line being edited. After the I is typed, each character typed in will be inserted at the current cursor position and typed on the terminal. To stop inserting characters, type "escape" (or Alt←mode on some terminals).

If an attempt is made to insert a character that will make the line longer than the maximum allowed (72 characters), a bell will be typed (control G) on the terminal and the character will not be inserted.

WARNING: It is possible using EDIT to create a line which, when listed with its line number, is longer than 72 characters. Punched paper tapes containing such lines will not be read in properly. However, such lines may be CSAVEd and CLOADed without error.

INSERTING CHARACTERS (cont.)

← (or _) A backarrow (or underline) typed during an insert command will delete the character to the left of the cursor. Characters up to the beginning of the line may be deleted in this manner, and a backarrow will be echoed for each character deleted. However, if no cnaracters exist to the left of the cursor, a bell is echoed instead of a backarrow.

If a carriage return is typed during an insert command, it will be as if an escape and then carriage return was typed. That is, all characters to the right of the cursor will be printed and the EDITed line will replace the original line.

X X is the same as I, except that all characters to the right of the cursor are printed, and the cursor moves to the end of the line. At this point it will automatically enter the insert mode (see I command).

X is very useful when you wish to add a new statement to the end of an existing line. For example:

```
Typed by User        EDIT 50 (carriage return)
Typed by ALTAIR      50 X=X+1:Y=Y+1
Typed by User           X    :Y=Y+1 (carriage return)
```

In the above example, the original line #50 was:

```
50     X=X+1
```

The new EDITed line #50 will now read:

```
50     X=X+1:Y=Y+1
```

H H is the same as I, except that all characters to the right of the cursor are deleted (they will not be typed). The insert mode (see I command) will then automatically be entered.

H is most useful when you wish to replace the last statements on a line with new ones.

DELETING CHARACTERS

D nD deletes n number of characters to the right of the cursor. If less than n characters exist to the right of the cursor, only that many characters will be deleted. The cursor is positive to the right of the last character deleted. The characters deleted are enclosed in backslashes (\). For example:

```
Typed by User        20 X=X+1:REM JUST INCREMENT X
Typed by User        EDIT 20  (carriage return)
Typed by ALTAIR      20   \X=X+1:\ REM JUST INCREMENT X
Typed by User           6D (carriage return)
```

The new line #20 will no longer contain the characters which are enclosed by the backslashes.

SEARCHING

S — The nSy command searches for the $n^{th}$ occurance of the character y in the line. The search begins at the character one to the right of the cursor. All characters passed over during the search are printed. If the character is not found, the cursor will be at the end of the line. If it is found, the cursor will stop at that point and all of the characters to its left will have been printed.

For example:

```
Typed by User          50 REM INCREMENT X
Typed by User          EDIT 50
Typed by ALTAIR        50 REM INCR
Typed by User          2SE
```

K — nKy is equivalent to S, except that all of the characters passed over during the search are deleted. The deleted characters are enclosed in backslashes. For example:

```
Typed by User          10 TEST LINE
Typed by User          EDIT 10
Typed by ALTAIR        10  \TEST\
Typed by User             KL
```

TEXT REPLACEMENT

C — A character in a line may be changed by the use of the C command. Cy, where y is some character, will change the character to the right of the cursor to y. The y will be typed on the terminal and the cursor will be advanced one position. nCy may be used to change n number of characters in a line as they are typed in from the terminal. (See example below.)

If an attempt is made to change a character which does not exist, the change mode will be exited.

Example:

```
Typed by User          10 FOR I=1 TO 100
Typed by User          EDIT 10
Typed by ALTAIR        10 FOR I=1 TO 256
Typed by User          2S1      3C256
```

ENDING AND RESTARTING

Carriage Return — Tells the computer to finish editing and print the remainder of the line. The edited line replaces the original line.

E — E is the same as a carriage return, except the remainder of the line is not printed.

Q — Quit. Changes to a line do not take effect until an E or carriage return is typed. Q allows the user to restore the original line without any changes which may have been made, if an E or carriage return has not yet been typed. "OK" will be typed and BASIC will await further commands.

L — Causes the remainder of the line to be printed, and then prints the line number and restarts EDITing at the beginning of the line. The cursor will be positioned to the left of the first character in the line.

L is most useful when you wish to see how the changes in a line look so that you can decide if further EDITs are necessary.

Example:

```
Typed by User        EDIT 50
Typed by ALTAIR      50 REM  INCREMENT X
Typed by User        2SM          L
Typed by ALTAIR      50
```

A — Causes the original copy of the line to be restored, and EDITing to be restarted at the beginning of the line. For example:

```
Typed by User        10 TEST LINE
Typed by User        EDIT 10
Typed by ALTAIR      10 \TEST  LINE\
Typed by User        10D            A
Typed by ALTAIR      10
```

In the above example, the user made a mistake when he deleted TEST LINE. Suppose that he wants to type "1D" instead of "10D" By using A command, the original line 10 is reentered and is ready for further EDITing.

IMPORTANT

Whenever a SYNTAX ERROR is discovered during the execution of a source program, BASIC will automatically begin EDITing the line that caused the error as if an EDIT command had been typed. For Example:

```
10 APPLE
RUN
SYNTAX ERROR IN 10
10
```

Complete editing of a line causes the line edited to be re-inserted. Re-inserting a line causes all variable values to be deleted, therefore you may want to exit the EDIT command without correcting the line so that you can examine the variable values.

This can be easily accomplished by typing the Q command while in the EDIT mode. If this is done, BASIC will type OK and all variable values will be preserved.

## PRINT USING

The PRINT USING statement can be employed in situations where a specific output format is desired. This situation might be encountered in such applications as printing payroll checks or an accounting report. Other uses for this statement will become more apparent as you go through the text.

The general format for the PRINT USING statement is as follows:

(line number) PRINT USING <string>; <value list>

The "string" may be either a string variable, string expression or a string constant which is a precise copy of the line to be printed. All of the characters in the string will be printed just as they appear, with the exception of the formatting characters. The "value list" is a list of the items to be printed. The string will be repeatedly scanned until: 1) the string ends and there are no values in the value list 2) a field is scanned in the string, but the value list is exhausted.

The string should be constructed according to the following rules:

### STRING FIELDS

| | |
|---|---|
| ! | Specifies a single character string field. (The string itself is specified in the value list.) |
| \n spaces\ | Specifies a string field consisting of 2+n characters. Backslashes with no spaces between them would indicate a field of 2 characters width, one space between them would indicate a field 3 characters in width, etc. |

In both cases above, if the string has more characters than the field width, the extra characters will be ignored. If the string has less characters than the field width, extra spaces will be printed to fill out the entire field.

Trying to print a number in a string field will cause a TYPE MISMATCH error to occur.

Example:

```
10 A$="ABCDE":B$="FGH"
20 PRINT USING "!";A$,B$
30 PRINT USING "\  \";B$,A$
```

(the above would print out)

```
AF
FGH ABCD
```

Note that where the "!" was used only the first letter of each string was printed. Where the backslashes were enclosed by two spaces, four letters from each string were printed (an extra space was printed for B$ which has only three characters). The extra characters in the first case and for A$ in the second case were ignored.

## NUMERIC FIELDS

With the PRINT USING statement, numeric prin-outs may be altered to suit almost any applications which may be found necessary. This should be done according to the following rules:

\# Numeric fields are specified by the # sign, each of which will represent a digit position. These digit positions are always filled. The numeric field will be right justified; that is, if the number printed is too small to fill all of the digit positions specified, leading spaces will be printed as necessary to fill the entire field.

. The decimal point position may be specified in any particular arrangement as desired; rounding is performed as necessary. If the field format specifies a digit is to precede the decimal point, that digit will always be printed (as 0 if necessary).

The following program will help illustrate these rules:

```
10 INPUT A$,A
20 PRINT USING A$;A
30 GOTO 10
RUN
? ##,12
12
? ###,12
 12
? #####,12
   12
?##.##,12
12.00
? ###.,12
 12.
? #.###,.02
0.020
? ##.#,2.36
 2.4
```

\+ This sign may be used at either the beginning or end of the numeric field, and will force the + sign to be printed at either end of the field as specified, if the number is positive. If it is used at the end of the field, and the number is negative, a -sign will be forced at the end of the number.

\- The - sign when used at the end of the numeric field designation will force the sign to be printed trailing the number, if it is negative. If the number is positive, a space is printed.

NOTE: There are cases where forcing the sign of a number to be printed on the trailing side will free an extra space for leading digits. (See exponential format.)

**  The ** placed at the beginning of a numeric field designation will cause any unused spaces in the leading portion of the number printed out to be filled with asterisks. The ** also specifies positions for 2 more digits. (Termed "asterisk fill")

$$  When the $$ is used at the beginning of a numeric field designation, a $ sign will be printed in the space immediately preceding the number printed. Note that the $$ also specifies positions for two more digits, but the $ itself takes up one of these spaces. Exponential format cannot be used leading $ signs, nor can negative numbers be output unless the sign is forced to be trailing.

**$  The **$ used at the beginning of a numeric field designation causes both of the above (** & $$) to be performed on the number being printed out. All of the previous conditions apply, except that **$ allows for 3 additional digit positions, one of which is the $ sign.

,  A comma appearing to the left of the decimal point in a numeric field designation will cause a comma to be printed every three digits to the left of the decimal point in the number being printed out. The comma also specifies another digit position. A comma to the right of the decimal point in a numeric field designation is considered a part of the string itself and will be treated as a printing character.

↑↑↑↑  Exponential Format. If the exponential format of a number is desired in the print out, the numeric field designation should be followed by ↑↑↑↑ (allows space for E±XX). As with the other formats, any decimal point arrangement is allowed. In this case, the significant digits are left justified and the exponent is adjusted.

%  If the number to be printed out is larger than the specified numeric field, a % character will be printed and then the number itself in its standard format. (The user will see the entire number.) If rounding a number causes it to exceed the specified field, the % character will be printed followed by the rounded number. (Such as field= .##, and the number is .999 will print % 1.00.)

If the number of digits specified exceeds 24, a FUNCTION CALL error will occur.

Try going through the following examples to help illustrate the preceding rules. A single program such as follows is the easiest method for learning PRINT USING.

Examples: Type the short program into your machine as it is listed below. This program will keep looping and allow you to experiment with PRINT USING as you go along.

Program:

```
10 INPUT A$,A
20 PRINT USING A$;A
30 GOTO 10
RUN
```

The computer will start by typing a ?. Fill in the numeric field designator and value list as desired, or follow along below.

```
? +#,9
+9
? +#,10
%+10
? ##,-2
-2
? +#,-2
-2
? #,-2
%-2
? +.###,.02
+.020
? ####.#,100
 100.0
? ##+,2
 2+
? THIS IS A NUMBER ##,2
THIS IS A NUMBER  2
? BEFORE ## AFTER,12
BEFORE 12 AFTER
? ####,44444
%44444
? **##,1
***1
? **##,12
**12
? **##,123
*123
? **##,1234
1234
? **##,12345
%12345
? **,1
*1
? **,22
22
? **.##,12
12.00
? **####,1
*****1
```

```
(note: not floating $)    ? $####.##,12.34
                          $  12.34
(note: floating $)        ? $$####.##,12.56
                             $12.56
                          ? $$.##,1.23
                          $1.23
                          ? $$.##,12.34
                          %$12.34
                          ? $$###,0.23
                             $0
                          ? $$####.##,0
                              $0.00
                          ? **$###.##,1.23
                          ****$1.23
                          ? **$.##,1.23
                          *$1.23
                          ? **$###,1
                          ****$1
                          ? #,6.9
                          7
                          ? #.#,6.99
                          7.0
                          ? ##-,2
                           2
                          ? ##-,-2
                           2-
                          ? ##+,2
                           2+
                          ? ##+,-2
                           2-
                          ? ##↑↑↑↑,2
                           2E+00
                          ? ##↑↑↑↑,12
                           1E+01
                          ? #####.###↑↑↑↑,2.45678
                           2456.780E-03
                          ? #.###↑↑↑↑,123
                          0.123E+03
                          ? #.##↑↑↑↑,-123
                          -.12E+03
                          ? #####,###.#,1234567.89
                          1,234,567.9
```

## APPENDIX A SUPPLEMENT

## HOW TO LOAD BASIC

For BASIC versions 3.2 and later, the load procedure has been updated to allow the use of the new I/O boards (2SIO & 4PIO), the old 88-PIO board, and more general channel assignments.

Location 001 of the bootstrap loaders listed in APPENDIX A must be changed from 175 to 256 to load BASIC versions 3.2 and later. For the older versions of BASIC, the location should be left at 175.

For EXTENDED BASIC, location 002 (set at 017 for 4K & 037 for 8K) should be set at 057.

The checksum loader has a new error message "M" which indicates that the data that was loaded into memory did not read back properly (see step 22 on page 50 of APPENDIX A). Loading into non-existent, protected or malfunctioning memory can cause this to occur. The new error message will also be sent repeatedly, instead of only once. The message is sent on channels 1, 21 and 23; so, if no terminal device is on one of these three channels, the panel lights must be examined to see if a checksum error has occured.

### Error Detection

The new checksum loader (BASIC versions 3.2 & later) will display X7647 on the address lights when running properly. (X above will be 0 for 4K BASIC, 1 for 8K or 2 for EXTENDED.)

When an error occurs (checksum "C"-bad tape data, memory "M"-data won't store properly, overlay "O"-attempt to load over top of the checksum loader) the address lights will then display X7637. The ASCII error code will be stored in the accumulator (A).

More simply, A5 should be on with A4 & A3 off during proper loading. When an error occurs, A5 will turn off and A4 & A3 will turn on.

### Load Options

| LOAD DEVICE | SWITCHES UP | OCTAL CHANNELS | STATUS BITS ACTIVE | OCTAL MASKS |
|---|---|---|---|---|
| SIOA,B,C (not REV 0) | none | 0,1 | low | 1/200 |
| ACR | A15 (and terminal opts.) | 6,7 | low | 1/200 |
| SIOA,B,C (REV 0) | A14 | 0,1 | high | 40/2 |
| 88-PIO | A13 | 0,1 | high | 2/1 |
| 4PIO | A12 | 20,21 | high | 100/100 |
| 2SIO | A11 (and A10 up=1stop bit down=2 stop bits) | 20,21 | high | 1/2 |

0 TO 10 LEADS
10 TO 206

There are six different bootstrap loaders, one for each of the six types of I/O boards listed in the Load Option chart. Be sure that you use the correct one for your particular board.

If the load device is an ACR, the Terminal Options (see second chart) can be set in the switches (along with A15) before the loading is done. If A15 is set, the checksum loader will ignore all of the other switches and BASIC will ignore A15.

If the load device and the terminal device are not the same, and the load device is not an ACR, then only the load options should be set before the loading. When the load completes, BASIC will start-up and try to send a message to the load device. STOP BASIC, EXAMINE LOCATION 0, SET THE TERMINAL OPTION SWITCHES, AND THEN DEPRESS RUN.

If the initialization dialog hasn't completed, everytime BASIC is restarted at zero, it will examine the sense switches and reconfigure the terminal input/output options. Once the initialization dialog is complete, the sense switches are no longer examined and the I/O configuration is fixed until BASIC is reloaded.

Terminal Options

| TERMINAL DEVICE | SWITCHES UP | OCTAL CHANNEL DEFAULT |
|---|---|---|
| SIOA,B,C (not REV 0) | none | 0,1 |
| SIOA,B,C (REV 0) | A14 | 0,1 |
| 88-PIO | A13 | 0,1 |
| 4PIO | A12 | 20,21 (INPUT)<br>22,23 (OUTPUT) |
| 2SIO | A11 | 20,21 (A10 up=1 stop bit<br>down=2 stop bits) |

The default channels listed above may be changed as desired by raising A8 and storing the lowest channel number (Input flag channel) in one of the following locations:

7777 (octal) for 4K BASIC
17777 (octal) for 8K BASIC
27777 (octal) for EXTENDED BASIC (non-disk version)

NOTE: The "Input flag channel" may also be refered to as the "control channel" in other ALTAIR documentation.

The above information is useful only when the load device and terminal device are not the same. During the load procedure A8 will be ignored; therefore, the board from which BASIC is loaded must be strapped for the channels listed in the Load Option chart.

The following page contains three new bootstrap loaders for the 88-PIO, 4PIO and 2SIO boards. The conditions for using the other loaders listed in APPENDIX A are given at the beginning of this supplement.

88-PIO (for versions 3.2 & later only)

| OCTAL ADDRESS | OCTAL CODE | |
|---|---|---|
| 000 | 041 | |
| 001 | 256 | |
| 002 | 017 | (for 4K, 037 for 8K, 057 for EXTENDED) |
| 003 | 061 | |
| 004 | 023 | |
| 005 | 000 | |
| 006 | 333 | |
| 007 | 000 | |
| 010 | 346 | NOTE: Switch A13 should be up; |
| 011 | 040 | 88-PIO should be strapped |
| 012 | 310 | for channels 0,1. |
| 013 | 333 | |
| 014 | 001 | |
| 015 | 275 | |
| 016 | 310 | |
| 017 | 055 | |
| 020 | 167 | |
| 021 | 300 | |
| 022 | 351 | |
| 023 | 003 | |
| 024 | 000 | |

2SIO (for versions 3.2 & later only)

| OCTAL ADDRESS | OCTAL CODE | | OCTAL ADDRESS | OCTAL CODE |
|---|---|---|---|---|
| 000 | 076 | | 030 | 300 |
| 001 | 003 | | 031 | 351 |
| 002 | 323 | | 032 | 013 |
| 003 | 020 | | 033 | 000 |
| 004 | 076 | | | |
| 005 | 021 | (=2 stop bits, | | |
| 006 | 323 | 025=1 stop bit) | | |
| 007 | 020 | | | |
| 010 | 041 | | | |
| 011 | 256 | | | |
| 012 | 017 | (for 4K, 037 for 8K, 057 for EXTENDED) | | |
| 013 | 061 | | | |
| 014 | 032 | | | |
| 015 | 000 | NOTE: Switch A11 should be up; | | |
| 016 | 333 | If the 2SIO also is the | | |
| 017 | 020 | terminal device, set A10 | | |
| 020 | 017 | up for 1 stop bit or down | | |
| 021 | 320 | for 2 stop bits. The 2SIO | | |
| 022 | 333 | should be strapped for | | |
| 023 | 021 | channels 20,21. | | |
| 024 | 275 | | | |
| 025 | 310 | | | |
| 026 | 055 | | | |
| 027 | 167 | | | |

4PIO (for versions 3.2 & later only)

| OCTAL ADDRESS | OCTAL CODE | |
|---|---|---|
| 000 | 257 | |
| 001 | 323 | |
| 002 | 020 | |
| 003 | 000 | |
| 004 | 323 | |
| 005 | 021 | |
| 006 | 076 | |
| 007 | 004 | |
| 010 | 323 | |
| 011 | 020 | |
| 012 | 041 | |
| 013 | 256 | |
| 014 | 017 | (for 4K, 037 for 8K, 057 for EXTENDED) |
| 015 | 061 | |
| 016 | 035 | |
| 017 | 000 | |
| 020 | 333 | NOTE: Switch A12 should be up. |
| 021 | 020 | |
| 022 | 346 | |
| 023 | 100 | |
| 024 | 310 | |
| 025 | 333 | |
| 026 | 021 | |
| 027 | 275 | |
| 030 | 310 | |
| 031 | 055 | |
| 032 | 167 | |
| 033 | 300 | |
| 034 | 351 | |
| 035 | 015 | |
| 036 | 000 | |

The following three programs are echo programs for the 88-PIO, the 4PIO and the 2SIO boards.

If you wish to test a device that does Input only, dump the echoed characters on a faster device or store them in memory for examination.

For an Output only device, send the data in the sense switches or some constant for the test character. Make sure to check the ready-to-receive bit before doing output.

If the echo program works, but BASIC doesn't; make sure the load device's UART is strapped for 8 data bits and that the ready-to-receive flag gets set properly on the terminal device.

ECHO PROGRAMS:

88-PIO

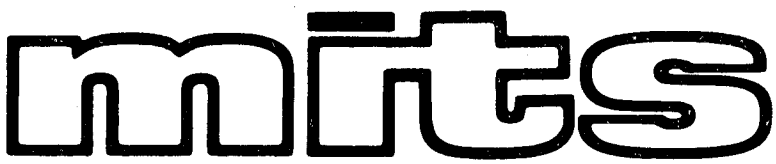
| OCTAL ADDRESS | OCTAL CODE | OCTAL ADDRESS | OCTAL CODE |
|---|---|---|---|
| 000 | 333 | 007 | 333 |
| 001 | 000 | 010 | 001 |
| 002 | 346 | 011 | 323 |
| 003 | 040 | 012 | 001 |
| 004 | 312 | 013 | 303 |
| 005 | 000 | 014 | 000 |
| 006 | 000 | 015 | 000 |

2SIO

| OCTAL ADDRESS | OCTAL CODE | OCTAL ADDRESS | OCTAL CODE |
|---|---|---|---|
| 000 | 076 | 013 | 322 |
| 001 | 003 | 014 | 010 |
| 002 | 323 | 015 | 000 |
| 003 | 020 (flag ch.) | 016 | 333 |
| 004 | 076 | 017 | 021 (data ch.) |
| 005 | 021 (1 st. bt., | 020 | 323 |
| 006 | 323 025 for 2) | 021 | 021 |
| 007 | 020 | 022 | 303 |
| 010 | 333 | 023 | 010 |
| 011 | 020 | 024 | 000 |
| 012 | 017 | | |

4PIO

| OCTAL ADDRESS | OCTAL CODE | OCTAL ADDRESS | OCTAL CODE |
|---|---|---|---|
| 000 | 257 | 024 | 312 |
| 001 | 323 | 025 | 020 |
| 002 | 020 | 026 | 000 |
| 003 | 323 | 027 | 333 |
| 004 | 021 | 030 | 022 |
| 005 | 323 | 031 | 346 |
| 006 | 022 | 032 | 100 |
| 007 | 057 | 033 | 312 |
| 010 | 323 | 034 | 027 |
| 011 | 023 | 035 | 000 |
| 012 | 076 | 036 | 333 |
| 013 | 004 | 037 | 021 |
| 014 | 323 | 040 | 323 |
| 015 | 020 | 041 | 023 |
| 016 | 323 | 042 | 303 |
| 017 | 022 | 043 | 020 |
| 020 | 333 | 044 | 000 |
| 021 | 020 | | |
| 022 | 346 | | |
| 023 | 100 | | |



2450 Alamo SE
Albuquerque, NM 87106

# 2-SIO BOARD ERRATA

Refer to Special Note, page 8 of Theory of Operation:

Note that if the 2-SIO boot loader is used, first start the program (push STOP/RUN switch to RUN), then start the reader.