DIFFERENCES BETWEEN TI EXTENDED BASIC AND ARMADILLO BASIC

DATE	CHANGE

3/22/83 Numeric variables can be of two types: REAL or INTEGER.
All numeric variables are automatically assigned the default type of REAL (floating point) unless explicitly declared otherwise in a type statement.
The ALL clause in a numeric type statement changes the default type to the declared type.
Example:
```
INTEGER ALL
```
The INTEGER ALL statement changes the default type to INTEGER.
NOTE: If the ALL specification is used, the statement should precede the first variable reference (this includes the DIM statement).

3/22/83 REAL (floating point) variables are the same as the Numeric Constants in Extended BASIC.
The formats for the REAL statement are:
```
REAL ALL
REAL variable-name    (where the variable-name is any numeric
                       variable name or array declaration or
                       a list of these things)
```

3/22/83 The INTEGER statement assigns the INTEGER type to a variable or list of variables.
An advantage to INTEGER variables is that they are processed faster and take up less space than REAL variables.
An INTEGER variable may have a value of +32767 through -32768.
Formats:
```
INTEGER ALL
INTEGER variable-name    (where the variable-name is any numeric
                          variable name or array declaration or
                          a list of these things)
```
Example:
```
INTEGER A,B,C            (variables A, B, and C are declared to
                          be type INTEGER)
```

3/22/83 There are now several ways to dimension an array.
To dimension an array A to size 10 and type REAL:
```
100 DIM A(10)
     or
100 REAL A(10)
     or
100 REAL ALL
110 DIM A(10)
```
To dimension an array A to size 20 and type INTEGER:
```
100 INTEGER A(20)
     or
100 INTEGER ALL
110 DIM A(20)
```
If you specify:
```
100 DIM A(10)
110 REAL A(10)
```
A name conflict error will result, since you have declared the array twice.

3/22/83 User-Defined Functions are 1 program line in length.
UDFs now support up to seven parameters (instead of one).

The type of the function can be specified as INTEGER, REAL, or $ (Character).
UDFs still can not be defined recursively.
Format:
DEF function-name[(parameter-list)]=expression
Examples:
```
DEF F(X,Y,Z)=expression              (defaults to a REAL function)
DEF INTEGER F(X,Y,Z)=expression      (specifies an INTEGER function)
```
The following is also legal and prints a result of 14.
```
100 DEF F(X,Y)=X+Y+4
110 PRINT F(F(1,2),3)
```

3/22/83 The type of the variables which appear in a parameter-list of a DEF or SUB statement can also be specified as INTEGER, REAL, or $ (Character).
Examples:
```
DEF F(INTEGER X, REAL Y, INTEGER A)=expression
DEF F$(A$,B$,C$)=expression
SUB F(INTEGER A, REAL X)
SUB F(A$,B$)
```

3/22/83 The VALIDATE clause in the ACCEPT statement has the following data-type additions:
ALPHA permits all uppercase and lowercase alphabetic characters.
LALPHA permits all lowercase alphabetic characters.

3/22/83 The VERSION subprogram will return a value of 200.

3/22/83 The FOR-TO-STEP statement executes much faster when a control-variable which has been declared as INTEGER type is used.

3/22/83 When a numerical expression is evaluated, INTEGERS are converted to REAL only when necessary.
This means that when a numerical expression contains only INTEGER values, all evaluation is done in INTEGER arithmetic.
Example:
```
100 INTEGER ALL
     ...
200 C=A+B
```
No conversions are necessary in the above example.
If an expression contains both INTEGER and REAL values, conversion from INTEGER to REAL is done only when necessary.
Example:
```
100 INTEGER A,B,C
     ...
200 C=A+B+3.45
```
In this example, A and B are added, the result is converted to REAL and is then added to 3.45. The final result is then rounded to the nearest integer and assigned to C.

3/22/83 There will be two or three graphics routines which will be added. Further information on these will follow.

3/22/83 The most change will occur in the Graphics Modes.
There will be six modes:
```
Graphics 1 (pattern mode)
Text
Split-1    (Graphics 2 with text on top third of screen)
Split-2    (Graphics 2 with text on bottom third of screen)
Graphics 2 (full-screen)
Multicolor
```

3/22/83 The following subprograms and statements will have some changes due to the way they respond in the various Graphics Modes.

COLOR
HCHAR
VCHAR
ACCEPT
INPUT
DISPLAY
PRINT
GCHAR

The changes will come later.

DIFFERENCES BETWEEN TI EXTENDED BASIC AND ARMADILLO BASIC

DATE	CHANGE

4/26/83 It is now possible to change existing line numbers.
To change a line number, enter the Edit Mode by typing a line number followed by either FCTN E (UP) or FCTN X (DOWN), and then use the FCTN S (LEFT) and FCTN D (RIGHT) to space over to the digit(s) of line number that you wish to change. Changing a line number does not delete the original line, and so can be used to duplicate a line.

4/26/83 There are now six Graphics Modes.
Format: CALL GRAPHICS(mode)
Valid modes are:

Mode	Description
1	Graphics I - 32 col x 24 row grid of pattern positions (8 x 8 pixel blocks). Color (per 8 characters) and a maximum of 32 sprites is available. Alphanumeric and special characters can be used.
2	Text - 40 col x 24 row alphanumeric grid (6 x 8 pixel blocks). Only 2 colors (foreground and background) at a time may be used. No sprites are available.
3	Split I - Split screen where the top 1/3 of the screen displays text (32 x 8 alphanumeric) and the bottom 2/3's of the screen displays high resolution graphics (256 x 128). Color is available for each character in the Graphics I portion of the screen and for each 8 dots on Graphics II portion of the screen. A maximum of 32 sprites is available.
4	Split II - Same as Split I mode, except the top 2/3's of the screen displays high resolution graphics (256 x 128) and the bottom 1/3 of the screen displays text (32 x 8 alphanumeric).
5	Graphics II - High resolution color graphics (256 x 192). The result from the DISPLAY, PRINT, INPUT, ACCEPT, and LINPUT statements is not visible in this mode. A maximum of 32 sprites is available.
6	Graphics III - Low resolution color graphics 64 x 48 blocks (4 x 4 pixel blocks). Each block is individually assigned a color. The result from the DISPLAY, PRINT, INPUT, ACCEPT, and LINPUT statements is not visible. A maximum of 32 sprites is available.

4/26/83 There is a new subprogram, CALL MARGINS, which defines the screen margins.
Format: CALL MARGINS(left,right,top,bottom)
where left,right,top, and bottom are integers which represent the number of columns or rows to indent the screen margins.
This means all 32 columns and 24 rows can be used, or smaller "windows" can be defined as desired.
There can only be one "window" defined at a time.
TI Extended BASIC has only one window which can be defined as:
CALL MARGINS(2,2,0,0)
in Armadillo BASIC.
This subprogram affects the following:
- Scrolling
- CLEAR subprogram

DISPLAY statement (The position (1,1) is the upper left-hand corner of the current window.)
ACCEPT statement (The position (1,1) is the upper left-hand corner of the current window.)

This subprogram does not affect:
HCHAR subprogram
VCHAR subprogram
GCHAR subprogram
Graphics portions of the screen in modes 3, 4, and 5.
Mode 6 is not affected at all.

4/26/83 There are four graphics subprograms which can only be used in modes 3, 4, and 5.

Format: CALL DCOLOR(foreground-color,background-color)
This specifies the color to use in the DRAW, FILL, HCHAR, and VCHAR subprograms. The default color is black on transparent.

Format: CALL DRAW(type,y1,x1,y2,x2[,y3,x3,y4,x4][,...])
where type is:
1 - For DRAW (using DCOLOR)
0 - For erase
-1 - For reverse (no color change)
where y is dot-row and x is dot-column.
This routine draws a line from the starting point to the ending point. The line type is specified in the CALL DRAW statement.

Format: CALL DRAWTO(type,y1,x1[,y2,x2][,...])
This routine draws a line from the current position to the point specified. The line type is specified in the CALL DRAWTO statement.

Format: CALL FILL(dot-row,dot-column)
This routine fills all pixels surrounding and including the pixel defined by the dot-row and dot-column given in the CALL FILL statement.
This routine fills until a pixel of the foreground color or the screen edge is encountered.
Filling may also stop if the figure is too complicated.

4/26/83 Format for the CALL COLOR statement depends on the graphics mode. The format:

CALL COLOR(#sprite-number,foreground[,...])

is valid for all modes, except mode 2.
Additional formats are:

Mode	Format
1	CALL COLOR(character-set,foreground,background[,...])
2	CALL COLOR results in an error.
3	CALL COLOR(character-code,foreground,background[,...])
4	CALL COLOR(character-code,foreground,background[,...])
5	CALL COLOR results in an error, unless for a sprite.
6	CALL COLOR(row,column,color[,...]) where row is 1 to 48 and column is 1 to 64.

The Color Codes have remained the same as in TI Extended BASIC.
The Character Sets for mode 1 have changed.
The new Character Sets are:

Set Number	Character Codes
29	0-7
30	8-15
31	16-23
0	24-31
1	32-39
2	40-47
3	48-55
4	56-63
5	64-71
6	72-79
7	80-87
8	88-95
9	96-103
10	104-111
11	112-119
12	120-127
13	128-135
14	136-143
15	144-151
16	152-159
17	160-167
18	168-175
19	176-183
20	184-191
21	192-199
22	200-207
23	208-215
24	216-223
25	224-231
26	232-239
27	240-247
28	248-255

4/26/83 The CALL SCREEN subprogram has been modified.
Format: CALL SCREEN(background-color[,foreground-color])
This works in all six modes, but the foreground color is only effective in mode 2 (Text Mode). If the foreground color is not specified, the foreground color is not affected.

4/26/83 The CALL HCHAR and CALL VCHAR subprograms have the same format as in TI Extended BASIC, but have different results depending on the current mode.

Mode	Rows	Columns	
1	1:24	1:32	
2	1:24	1:40	
3	1:16	1:32	Character position on graphics part of screen.
4	1:16	1:32	Character position on graphics part of screen.
5	1:24	1:32	
6	Ignored.		

CHANGES TO TI EXTENDED BASIC II

DATE	CHANGE
8/10/83	CHARSET: Only restores character codes 32-95, inclusive. CHARSET restores the default colors to all 256 characters.
8/10/83	GRAPHICS: The default for margins for all modes is 2,2,0,0.
8/10/83	NEW, END, GRAPHICS, BREAK: All 256 characters are restored to their original definition.
8/10/83	DELSPRITE: If the ALL option is specified, all sprites are deleted and can only reappear if they are defined by the SPRITE subprogram. If a sprite-number is specified, the sprite can reappear with the LOCATE subprogram, or be redefined by the SPRITE subprogram.
8/12/83	All warning messages are printed in upper case. (Error messages are still printed in lower case.)
8/12/83	Error messages in High-Resolution and Multicolor modes are printed in Pattern mode.

ADD TO DIFFERENCES

Displaying a character using the PRINT instruction in the lower-right corner of the current window will cause a scroll.
Displaying a character using the HCHAR subprogram or with the DISPLAY AT instruction in the lower-right corner of the current window will not cause a scroll.

In the OPEN instruction the SIZE option is not supported by all peripherals. See the program under the OPEN instruction for an alternate way to do this.

ADDITIONAL CHANGES

Memory at power-up with no peripherals attached: 62720

PRINT #0:print-list is valid.
INPUT #0:input-list should be valid but doesn't work currently, but may be fixed.

DISPLAY AT: Program on pg. 51 -- May need additional comments that the user does not press ENTER or any other key to input the ROW and COLUMN information. Just enter two 2-digit numbers, and the colored block appears automatically.

ERR: Program on pg. 64 -- Mike said it returned varying results. I ran it about 20 times and got the same results each time.

FILL: Mike's program needed a slight adjustment to run properly, therefore no problem with FILL. The problem was that when a certain size circle was drawn, the circle had one pixel which was not drawn therefore leaving a one pixel hole in the circle boundary. This caused the FILL routine to leave the circle through the hole and fill the entire screen. Maybe we need to comment on making sure closed figures are really closed in the manual.

KEY: Program on pg. 107 -- Needs additional comments such as which keys to press (the four arrow keys without using the FUNC key) and how to stop the sprite from moving (press any key on the left side of the keyboard except the arrow keys).

OPEN: Program on pg. 155 -- See attached program. This opens a file RELATIVE and writes to the 100th record, thereby reserving space for 100 contiguous records. This emulates having RELATIVE 100 in the OPEN instruction. The program then closes the file and reopens it SEQUENTIAL, thus showing this method can also be used to allocate space for a SEQUENTIAL file.

SPRITE: Program on pg. 216-217 -- Mike said the comments were alright about the man jumping through or passing through the barrier.

DEF: John Yantis wants to add a program using a user-defined function called MOD which does modulo arithmetic. See attached program. This program does modulo arithmetic by using the user-defined function, MOD. MOD accepts two parameters which are any two whole numbers.

LIST: Need to document the fact that LIST 100 200 works without the hyphen between the line numbers. This is the same as LIST 100-200.

LET: Program to be added (or modified), see attached sheet. This does work how we discussed yesterday.

WARNING Messages are all printed in upper-case; Error Messages are still in lower-case.

BREAK: I messed up. Everything is fine, except under BREAK errors:
If the line-number-list includes a fractional or negative line number, the message Syntax Error is displayed. If the line-number-list includes a fractional number greater than 1, the message

```
  * WARNING
    LINE NOT FOUND
```

is also displayed.
In either case, breakpoints are only set at valid line numbers preceding the line number which caused the error.

Example (to explain to you in case this is not clear):

```
BREAK 150, 1.5, 160
```

prints messages WARNING LINE NOT FOUND and Syntax Error
and assigns a breakpoint to line 150, but not to line 160, even though both lines are in the program.

UNBREAK: Works the same as BREAK; including the above explanation.

GOSUB and ON GOSUB: The descriptions should include the "Subroutines may be recursive ... " paragraph. This needs to be added to ON GOSUB.

SUB is NOT recursive.
DEF is NOT recursive.

KEY: David Parnell thinks KEY should cross-reference ACCEPT, INPUT, and LINPUT.

ADDITIONAL CHANGES 9/1/83

LOAD: First example pg. 46 -- Change example to:

CALL LOAD(VALHEX("849E"),x,0) where x is 0,1,2,3,4,5,6,7,8, or 9.

Setting x=0 specifies full speed.
Setting x=2 slows the execution of assembly language programs down to approximately TI Home Computer 99/4A speed.
Experiment with the value of x to slow down the execution speed of BASIC programs to your desired speed.
The greater the value of x, the slower the processing speed.

KEY: Pg. 105 -- We really need to modify this description. The 99/4A User's Reference Guide has a fairly good description.

Delete: The KEY subprogram remains in effect until another KEY subprogram is called.

Add: Specifying 3, 4, or 5 as key-unit maps the keyboard to a particular mode of operation. The keyboard mode you specify determines the character codes returned by certain keys.
The keyboard mode remains in effect until the program ends (either normally or because of an error), stops at a breakpoint, or another keyboard mode is specified.
Specifying 1 or 2 as key-unit only remains in effect for that CALL KEY and does not effect the keyboard mode.

ADDITIONAL ADDITIONS 9/8/83

The maximum program size is 63,230 bytes. This only includes program text, not any variables, strings, or array space.

The maximum string length is 4090 bytes.

A string array cannot have more than 16383 elements.
An INTEGER array cannot have more than 32767 elements.
A REAL array cannnot have more than 8191 elements.

The maximum program line length is 255 characters, including the line number and spaces. However, after a program line is "crunched" it has a maximum length of 160 characters. (Refer to LINES in TI EXTENDED BASIC II CONVENTIONS for a description of "crunched" program lines.)

Variable names are a maximum of 15 characters in length.

APPENDIX: Differences between previous BASICs and Extended BASIC II

Since TI Extended BASIC II executes more rapidly than previous BASICs, delay loops do not delay a program as long as expected. Similarly, if a negative duration is used with the SOUND subprogram, the sounds are executed approximately three times as fast as expected.

The INIT subprogram with no parameter allocates 8K bytes of memory, if a parameter is specified, up to 24336 bytes (approx. 24K) may be allocated.
TI Extended BASIC allocated approximately 6K bytes of memory.
The Editor/Assembler with Console BASIC allocated 32K bytes of memory.

The HCHAR and VCHAR subprograms will print outside the margins, but will not scroll. This could cause unexpected results when first entering TI Extended BASIC II since characters displayed with the HCHAR or VCHAR subprograms could appear between the screen edge and the margins (columns 1, 2, 31, and 32). These characters will not scroll, but may be erased with the CLEAR subprogram.

The CHARSET subprogram restores more character codes to their default values than previous BASICs. This includes the lower-case character set.

If a PROTECTed program stops executing, it is now necessary to reload the program in order to run it again.

A CALL to routines in a plug-in module will not work. Also, OPEN #1:"SPEECH",OUTPUT for the Terminal Emulator II will not work.

The INTEGER variable and function type may cause some unexpected results. INTEGERs are written to an INTERNAL file as a 2-byte field; REALs are written as an 8-byte field. This means that INTEGER data type values in a TI Extended BASIC II data file will have to be read as string values with a 99/4 BASIC. Also, INTEGER type functions, return an INTEGER value rather than a floating point value. This would be important if numeric expressions were using INTEGER functions, such as: A = LEN(B$) / LEN(C$), since this will do integer division, rather than floating point division.

Assembly language programs which use absolute addressing with BASIC variables or with program loading, will probably not execute. Relocatable assembly language programs will execute correctly. However, NUMREF may return an INTEGER value rather than a floating point. This is denoted by a >6B in FAC+2. In addition, a string reference error may occur with STRREF since it only allows strings up to 255 characters in length, while BASIC allows strings up to 4090 characters in length.

Also, some assembly language programs worked with errors caused by referring to addresses such as >80xx and >82xx instead of >83xx. These programs will no longer work, since these are now valid addresses in BASIC.

All POKEs and PEEKs as used in previous BASICs will fail. In addition, the ability to PEEK and/or LOAD values into program memory is not predictable. This means LOAD cannot be used to modify a program in memory.

The Value Stack, which contains information like: GOSUB entries, FOR-NEXT entries, and temporary values; is now limited to 3840 bytes (3.75K bytes) rather than 12K bytes less BASIC usage. This means that an infinite GOSUB loop will no longer be able to determine the amount of program space available.

The list of reserved words has been expanded. The following words are additions to the Console BASIC reserved word list. Words preceeded by an asterisk (*) are additions to the Extended BASIC reserved word list. Reserved words may not be used as variable names, and thus may cause an error in existing programs.

```
 ACCEPT
 ALL
*ALPHA
 AND
 AT
 BEEP
 DIGIT
 ERASE
 ERROR
*FREESPACE
 IMAGE
*INTEGER
*LALPHA
 LINPUT
 MAX
 MERGE
 MIN
 NOT
 NUMERIC
 OR
 PI
*REAL
 RPT$
 SIZE
 SUBEND
 SUBEXIT
*TERMCHAR
 UALPHA
 USING
*VALHEX
 VALIDATE
 WARNING
 XOR
```

RESULTS OF SUBPROGRAMS AND STATEMENTS IN DIFFERENT GRAPHICS MODES

	MODE 1	MODE 2	MODE 3/4	MODE 5	MODE 6
DCOLOR	No Effect	No Effect	Yes (Graph) No (Text)	Yes	No Effect
DRAW	Error	Error	Yes (Graph) No (Text)	Yes	Error
DRAWTO	Error	Error	Yes (Graph) No (Text)	Yes	Error
FILL	Error	Error	Yes (Graph) No (Text)	Yes	Error
COLOR	Yes	Error	Yes	Error	Yes
CHAR	Yes	Yes	Yes	Yes	Yes
GCHAR	Yes	Yes	Yes (Graph) No (Text)	Yes	Yes
HCHAR	Yes	Yes	Yes (Graph) No (Text)	Yes	No Effect
VCHAR	Yes	Yes	Yes (Graph) No (Text)	Yes	No Effect
SPRITES	Yes	No Effect	Yes	Yes	Yes
ACCEPT	Yes	Yes	Yes (Text) No (Graph)	Yes, but Invisible	Yes, but Invisible
DISPLAY	Yes	Yes	Yes (Text) No (Graph)	Yes, but Invisible	Yes, but Invisible
INPUT (w/o file)	Yes	Yes	Yes (Text) No (Graph)	Yes, but Invisible	Yes, but Invisible
LINPUT (w/o file)	Yes	Yes	Yes (Text) No (Graph)	Yes, but Invisible	Yes, but Invisible
PRINT (w/o file)	Yes	Yes	Yes (Text) No (Graph)	Yes, but Invisible	Yes, but Invisible

A P P E N D I X

Assembly Language Support Routines

There are several utilities provided to allow access through TI assembly language to many of the resources of the ARMADILLO. With these utilities, the user can change the values in the VDP chip, access the Device Service Routine (DSR) for peripheral devices, scan the keyboard, link a program to GPL routines that perform a variety of useful tasks, and link to the Editor/Assembler Loader. Remember that these can only be used in TI assembly language programs.

The following list gives each of the utilities predefined in the REF/DEF table and describes briefly what each does.

Name	Use
VSBW	Writes a single byte to VDP RAM.
VMBW	Writes multiple bytes to VDP RAM.
VSBR	Reads a single byte from VDP RAM.
VMBR	Reads multiple bytes from VDP RAM.
VWTR	Writes a single byte to a VDP Register.
KSCAN	Scans the keyboard.
GPLLNK	Links a program to Graphics Programming Language routines.
XMLLNK	Links a program to the assembly language routines in the console ROM or in RAM.
DSRLNK	Links a program to Device Service Routines (DSRs).
LOADER	Links a program to the Loader to load TMS9900 tagged object code.

Several general use addresses are predefined with symbols. The following list gives the associated address and describes briefly each symbol.

Name	Address	Description
SCAN	>000E	Address of branch to the keyboard scan utility (KSCAN).
UTLTAB	>8600	Start of the utility variable table.
PAD	>8300	Start of CPU scratch pad RAM.
GPLWS	>83E0	GPL interpreter workspace pointer.
SOUND	>8400	Sound chip register.
SPCHRD	>9000	Speech Read Data Register.
SPCHWT	>9400	Speech Write Data Register.

Some addresses that are useful for accessing memory-mapped devices are predefined with symbols. The following list gives the address and brief description of each symbol.

Name	Address	Description
VDPWA	>8C02	VDP RAM Write Address Register.
VDPRD	>8800	VDP RAM Read Data Register.
VDPWD	>8C00	VDP RAM Write Data Register.
VDPSTA	>8802	VDP RAM Read Status Register.
GRMWA	>9C02	GROM/GRAM Write Address Register.
GRMRA	>9802	GROM/GRAM Read Address Register.
GRMRD	>9800	GROM/GRAM Read Data Register.

GRMWD >9C00 GROM/GRAM Write Data Register.

There are several TI BASIC support utilities which allow the user to access variables and values passed in the parameter list of the TI BASIC subprogram LINK. In addition, ERR allows the user to return an error to the calling TI BASIC program. Remember that these can only be used in TI assembly language programs.

The following list gives the available utilities and describes briefly what each does.

Name	Use
NUMASG	Makes a numeric assignment.
STRASG	Makes a string assignment.
NUMREF	Gets a numeric parameter.
STRREF	Gets a string parameter.
ERR	Reports errors.

Also, seven subprograms are added to TI BASIC for used in interfacing with assembly language programs. They are: INIT, LOAD, LINK, POKEV, PEEK, and PEEKV. These subprograms are described in the User Reference portion of this manual.

The TI Computer 99/8 has expanded the number of utilities available through the Editor/Assembler. As a result, the XMLLNK tables have changed; and old assembly language programs may need updated to reflect these changes. The XMLLNK utility uses the following format:

```
    BLWP  @XMLLNK
    DATA  >xxxx
```

where >xxxx defines the routine to be executed.

An example of this using the Convert Floating Point to Integer routine is:

```
    BLWP  @XMLLNK
    DATA  >1200
```

The following describes the current XMLLNK table.

NOTE: FAC (the Floating Point Accumulator) starts at address >834A.
ARG (which contains arguments) starts at address >835C.
The STATUS byte is at address >837C.
"f.p." means "floating point radix-100 format". (See Appendix F under Internal Numeric Representation for a description of radix-100 format.)

>xxxx	ROUTINE DESCRIPTION
>0100	Round Floating Point Value using Guard Digits. INPUT: FAC contains the f.p. value. OUTPUT: FAC contains the f.p. result after rounding by the contents of the most-significant byte of FAC+8. (FAC+8 contains guard digits which are maintained to guarantee the accuracy of the 14 most-significant digits of the results of f.p. operations.)
>0200	Round Floating Point Value to the position specified by FAC+10. INPUT: FAC contains the f.p. value. OUTPUT: FAC contains the f.p. result after rounding by the contents of FAC+10.
>0300	Floating Point Value Status. INPUT: FAC contains the f.p. value. OUTPUT: Sets the STATUS byte according to the f.p. value in FAC.
>0400	Test Floating Point Value for Overflow or Underflow. INPUT: FAC contains the f.p. value. OUTPUT: FAC contains zero if the f.p. value caused an underflow. FAC contains the largest possible f.p. number (9.999999999999E+128) if the f.p. value caused an overflow. Otherwise FAC contains the original f.p. value.
>0500	Test Floating Point Value for Overflow.

INPUT: FAC contains the f.p. value.

OUTPUT: FAC contains the largest possible f.p. number if the f.p. value caused an overflow.
Otherwise FAC contains the original f.p. value.

>0600 Floating Point Addition--Adds two f.p. values.

INPUT: FAC contains the first f.p. value and ARG contains the second f.p. value.

OUTPUT: FAC contains the f.p. result.

>0700 Floating Point Subtraction--Subtracts two f.p. values.

INPUT: FAC contains the f.p. value to be subtracted.
ARG contains the f.p. value from which FAC is subtracted.

OUTPUT: FAC contains the f.p. result.

>0800 Floating Point Multiplication--Multiplies two f.p. values.

INPUT: FAC contains the f.p. multiplier.
ARG contains the f.p. multiplicand.

OUTPUT: FAC contains the f.p. result.

>0900 Floating Point Division--Divides two f.p. values.

INPUT: FAC contains the f.p. divisor.
ARG contains the f.p. dividend.

OUTPUT: FAC contains the f.p. result.

>0A00 Floating Point Compare--Compares two f.p. values.

INPUT: ARG contains the first f.p. argument and FAC contains the second f.p. argument.

OUTPUT: Sets the STATUS byte. The high bit is set if ARG is logically higher than FAC. The greater than bit is set if ARG is arithmetically greater than FAC. The equal bit is set if ARG and FAC are equal.

Operations >0B00 to >0F00 use VSPTR (located at address >836E) as a pointer into an area of VDP RAM that is being used as a stack. The stack grows toward high memory, and VSPTR points to the top element. Push is pre-increment, pop is post-decrement.

NOTE: This is NOT the stack used by the Basic interpreter, and VSPTR should not be used in this way while in the TI Extended BASIC II environment.

>0B00 Value Stack Addition--Adds using a stack in VDP RAM.

INPUT: VSPTR contains the address in VDP RAM where the left-hand f.p. value is located.
FAC contains the right-hand f.p. value.

OUTPUT: FAC contains the f.p. result.

>0C00 Value Stack Subtraction--Subtracts using a stack in VDP RAM.

INPUT: VSPTR contains the address in VDP RAM where the left-hand f.p. value is located.
FAC contains the f.p. value to be subtracted.

OUTPUT: FAC contains the f.p. result.

>0D00 Value Stack Multiplication--Multiplies using a stack in VDP RAM.

INPUT: VSPTR contains the address in VDP RAM where the f.p. multiplicand is located.
FAC contains the f.p. multiplier.

OUTPUT: FAC contains the f.p. result.

>0E00 Value Stack Division--Divides using a stack in VDP RAM.

INPUT: VSPTR contains the address in VDP RAM where the f.p. dividend is located.
FAC contains the f.p. divisor.

OUTPUT: FAC contains the f.p. result.

>0F00 Value Stack Compare--Compares a f.p. value in the VDP RAM stack to the f.p. value in FAC.

INPUT: VSPTR contains the address in VDP RAM where the f.p. value to be compared is located.
FAC contains the other f.p. value in the comparision.

OUTPUT: Sets the STATUS byte. The high bit is set if the f.p. value pointed to by VSPTR is logically higher than FAC. The greater than bit is set if the f.p. value pointed to by VSPTR is arithmetically greater than FAC. The equal bit is set if the f.p. value pointed to by VSPTR and FAC are equal.

>1000 Convert String to Number (VDP RAM)--Converts an ASCII string in VDP RAM to a f.p. number.

INPUT: FAC+12 is a pointer to the start of the string on input, and to the first unconverted character on output. The normal convention is to terminate the string with an ASCII null (>00) character.

OUTPUT: FAC contains the f.p. result.

>1100 Convert String to Number (CPU RAM)--Like Convert String to Number (VDP RAM), except the string is in CPU RAM.

>1200 Convert Floating Point to Integer--Converts a f.p. value to an integer.

INPUT: FAC contains the f.p. value to be converted.

OUTPUT: FAC contains the one-word integer value. The maximum value is >FFFF. If an error occurs, the byte at FAC+10 is set to a nonzero value.

>1700 VDP RAM stack Push--Push the 8 bytes from FAC onto the VDP RAM stack, using VSPTR as the stack pointer.

>1800 VDP RAM stack Pop--Pop 8 bytes into FAC from the VDP RAM stack, using VSPTR as the stack pointer.

>1001 Greatest Integer Function--Computes the greatest integer contained in the f.p. value.

INPUT: FAC contains the f.p. value.

OUTPUT: FAC contains the result, which is the largest integer not greater than the original f.p. value.

>1101 Involution Routine--Raises a number to a specified power.

INPUT: FAC contains the exponent value.
ARG contains the base value.

OUTPUT: FAC contains the f.p. result.

>1201 Square Root Routine--Computes the square root of a number.

INPUT: FAC contains the input value.

OUTPUT: FAC contains the f.p. result.

>1301 Exponent Routine--Computes the inverse natural logarithm of a number.

INPUT: FAC contains the input value.

OUTPUT: FAC contains the f.p. result.

>1401 Natural Logarithm Routine--Computes the natural logarithm of a number.

INPUT: FAC contains the input value.

OUTPUT: FAC contains the f.p. result.

>1501 Cosine Routine--Computes the cosine of a number expressed in radians.

INPUT: FAC contains the input value.

OUTPUT: FAC contains the f.p. result.

>1601 Sine Routine--Computes the sine of a number expressed in radians.

INPUT: FAC contains the input value.

OUTPUT: FAC contains the f.p. result.

>1701 Tangent Routine--Computes the tangent of a number expressed in radians.

INPUT: FAC contains the input value.

OUTPUT: FAC contains the f.p. result.

>1801 Arctangent Routine--Computes the arctangent of a number expressed in radians.

INPUT: FAC contains the input value.

OUTPUT: FAC contains the f.p. result.

>1901 Convert Number to String--Converts a f.p. number to an ASCII string.

INPUT: FAC contains the input value.
FAC+11 = 0 for free format (this causes all other inputs to be ignored).
FAC+11 > 0 for width, excluding decimal point.
FAC+12 = 0 for underflow to 0, overflow to EEEEEEE.
FAC+12 > 0 for E-format on overflow or underflow.
FAC+13 >= 0 number of digits to right of decimal.
FAC+13 < 0 disables fixed mode.

OUTPUT: FAC is modified.
FAC+12 (byte) contains the length.
FAC+13 (byte) is the least significant byte of a pointer to the answer. The most significant byte is always >83.

>1A01 Convert Integer to Floating Point--Converts an integer to a floating point number.

INPUT: FAC contains the one-word integer value to be converted.

OUTPUT: FAC contains the 8-byte f.p. result.

>4001 Multicolor Mode--Set up VDP Patern Name Table for multicolor mode. (0-31 four times, then 32-63 four times, etc.)

>4101 High-Resolution Mode--Set up VDP Patern Name Table for high-resolution mode. (0-255 three times.)

>4201 Draw Line--Draw a line in high-resolution mode.

INPUT:
FAC : (byte) Graphics Mode (2, 3, or 4). (One less than TI Extended BASIC II's CALL GRAPHICS mode.)
FAC+1 : (byte) Color (foreground/background) each 0-15. (One less than TI Extended BASIC II's CALL COLOR values.)
FAC+2 : (byte) Line Type (-1, 0, or 1).
FAC+4 : (word) Y1 (zero-based).
FAC+6 : (word) X1 (zero-based).
FAC+8 : (word) Y2 (zero-based).
FAC+10 : (word) X2 (zero-based).

>4301 Fill--Fill screen area in high-resolution mode.

INPUT:
FAC : (byte) Graphics Mode (2, 3, or 4). (One less than TI Extended BASIC II's CALL GRAPHICS mode.)
FAC+1 : (byte) Color (foreground/background) each 0-15. (One less than TI Extended BASIC II's CALL COLOR values.)
FAC+4 : (word) Y (zero-based).
FAC+6 : (word) X (zero-based).
FAC+10 : (word) CPU RAM stack pointer.
FAC+12 : (word) CPU RAM stack limit. The stack area is a scratch area used by the FILL routine. A 2K byte area is recommended.

SECTION 1

ARMADILLO BASIC SPECIFICATION

1.1 General Features

Armadillo BASIC will execute user programs from CPU RAM, with program sizes up to 63K bytes. Each array will be to 64K bytes (8K reals, 32K integers, 16K strings). The stack will be limited to 4K bytes. Otherwise the only memory restriction is the total available RAM.

Extended BASIC will be a true extension of the standard interpreter, rather than a separate interpreter that just happens to share a few routines. Designing it this way should allow a smaller (or more powerful) Extended BASIC, and one that is easier to maintain than the current package.

Execution of BASIC programs from GROM will be supported, but access to subprograms in GPL in command modules will not be supported. PRK, Statistics, S-F Administrative, etc. **WILL**NOT**WORK** because of such statements as CALL A(...).

The following subprograms will be added to console BASIC, to make up for the fact that command module subprograms will not be supported.

* CALL SAY,SPGET (as in the Speech Editor)

* CALL INIT,LOAD,LINK,PEEK,POKEV,PEEKV,CHARPAT (as in the Editor/Assembler)

Notice that BASIC programs that use the PRK subprograms or the Carlisle Phillips "cheater" package will not run.

1.2 Added Features

1.2.1 Screen Access.

A new screen handler will be used. It will support faster scrolling, windowing, and access to all VDP modes. Alternate 80-column video may be provided if a definition of it can be supplied in time.

* CALL SCREEN(background-color) (current usage)

* CALL SCREEN(background, foreground)

* CALL SCREEN(background, foreground, mode)

* CALL SCREEN(background, foreground, mode, left-margin, right-margin, top-margin, bottom-margin)

Modes will include pattern, multicolor, text and bit-map. The default is pattern mode with left and right margins of 2 and top and bottom margins of 0.

CALL LINE(y1,x1,y2,x2,color) will draw a line segment in bitmap or multicolor mode. If the color is omitted, black is assumed.

CALL DOT(y,x,color) will draw a dot of the given color in bitmap or multicolor mode. If the color is omitted, black is assumed.

1.2.2 Sprites.

Sprites will be accessible in all modes except text. They will be accessed as in Extended BASIC, except that 32 sprites will be available.

1.2.3 Integers.

BASIC will implement an integer data type, for speed and space efficiency. The default numeric type is still real.

1.2.4 Multiple-statement Lines.

Standard BASIC will include multiple-statement lines and the corresponding enhancements to the IF-THEN-ELSE statement.

1.2.5 Program Chaining.

Standard BASIC will implement RUN as a statement, for chaining. RUN will be enhanced to accept an arbitrary string expression as an argument, instead of just a constant.

1.2.6 Accept and Display.

Standard BASIC will include a simplified form of accept and display, and a function TERM to return the function code that ended the input (proceed, enter, up, down, etc.). Only the AT and SIZE clauses will be supported.

1.2.7 Loading and Saving Programs.

While BASIC is running, only 24K bytes of RAM can be mapped into the logical address space. Thus memory image saves and loads (i.e. PROGRAM files) will be limited to 24K. (This assumes that all of the DSRs are written so that they can use CPU RAM buffers.) Programs that are larger than 10K will be saved in the same sequential file format that Extended BASIC currently uses for programs too large to fit in VDP RAM. (INTERNAL, VARIABLE 254).
Problem: A cassette cannot currently handle variable files or records longer than 192.
BASIC will also be able to load DISPLAY, VARIABLE 80 text files of the type produced by LIST and by the text editors. This type of loading will crunch each line as it is input. Loading will be slower than the standard binary loads, but this will permit loading foreign programs (from RS232) as well as using text editors on BASIC programs.

1.2.8 Assembly Language Support.

The default condition is that BASIC will use all of available RAM for program and data. Thus the user must allocate some RAM before assembly language can be loaded or accessed. This will be done in the CALL INIT statement.

CALL INIT will allocate 8K bytes of RAM for the user and will load utility routines into the first 2 or 3 K of it. CALL

INIT(X) will allocate X K bytes of RAM for the user. The minimum is 4, and the maximum is 64 (or as much as remains after allowing for the current BASIC program & data and DSR allocations). A value of 3 or less will be treated as CALL INIT(0), which will return all allocated RAM to BASIC.

The utilities will include NUMREF, NUMASG, STRREF, STRASG, LOADER, DSRLNK, GPLLNK, XMLLNK, etc. as in the Editor/Assembler and Extended BASIC. Upon a CALL LINK, the BLWP vector table will be mapped in at >2000 (compatible with Extended BASIC) and all other allocated RAM at >A000. Note that the loader can directly load only 24K of RAM, even though 64K may be allocated. Absolute origin code which loads on the 4A may not load properly on the Armadillo. (e.g. the SAVE utility in the Editor/Assembler or the text-to-speech tables in the disk text-to-speech package.

1.3 Speed Enhancements

The following functions will be faster than 99/4A BASIC, due primarily to translation of GPL to machine code, and extra scratchpad RAM.

* Screen access and scrolling
* Formatting numbers for print (convert-number-to-string)
* Listing and editing programs
* Intrinsic functions (SIN, EXP, etc.)
* String handling
* Random number generation

1.4 Extended BASIC Features

Extended BASIC will still have the following features not found in Standard BASIC:

* User-written subprograms with parameters and local variables
* Program merging
* Functions PI, MAX, MIN, RPT
* Logical operators AND, OR, XOR, NOT
* Block statements DO, LOOP, UNTIL, WHILE, EXIT (From new ANSI standard; new for Armadillo version)
* Full ACCEPT and DISPLAY statements
* Formatted output via PRINT USING and DISPLAY USING
* Tail remarks
* Error, warning, break handling under user control

SECTION 1

BASIC SUPPORT ROUTINES

POPSTK

CALL: BL @POPSTK or XML POPSTK

Assumes that value stack is mapped into memory and VSPTR is a logical memory pointer to last 8-byte entry. The last 8 bytes are transferred to ARG area, and VSPTR is incremented by 8, as is the physical pointer, STKTP4.

Uses GPL registers 5,6,7,11.
WINDOWS: >A stack

FBSYMB

CALL: BL @FBS or XML FBSYMB
DATA NOTFOUND BR NOTFOUND

INPUT: name length in @FAC+15
name in @FAC through @FAC+14

OUTPUT: (IF FOUND)
Symbol table pointer in FAC+4 through FAC+7.
(Physical)
Leaves matching symbol table entry in symbol window.

Uses GPL registers 0,1,2,3,11.
WINDOWS: symbol

SYM

CALL: BL @SYM or XML SYMB

INPUT: PGMPTR pointing to first character of identifier. (Not necessarily mapped in.) From 9900, assumes R9 is subroutine stack pointer to free top.

OUTPUT: PGMPTR advanced to next character beyond name. FBS called to look up name. Syntax error if name not found in symbol table. Symbol table pointer in @FAC+4 through @FAC+7. Matching symbol left in symbol window.

Uses GPL registers 2,6,8,9,11(,0,1,3).
WINDOWS: symbol, text

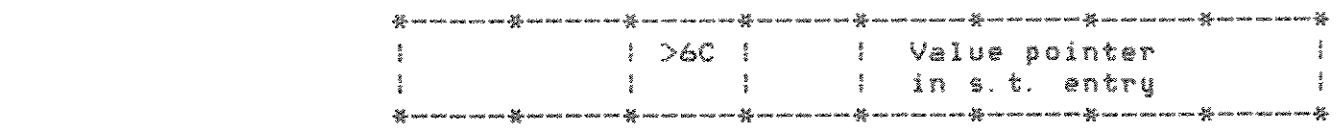
SMB

CALL: BL @SMB or XMB SMBB

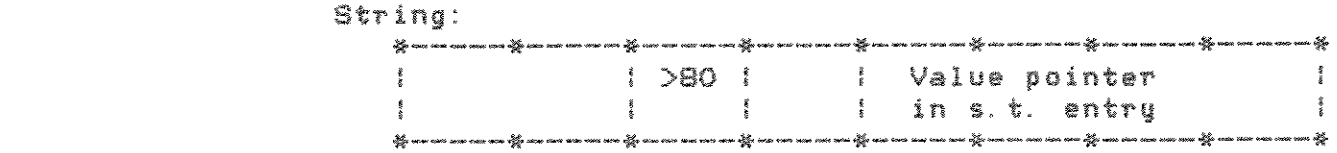
INPUT: Symbol table pointer in FAC+4 to FAC+7, PGMCHR pointing to next byte beyond symbol name.

OUTPUT: PGMPTR advanced beyond complete symbol. (i.e. variable with all subscripts.) Entry in FAC to FAC+7 built describing entry. Error if array subscripts out of bounds or otherwise improper.

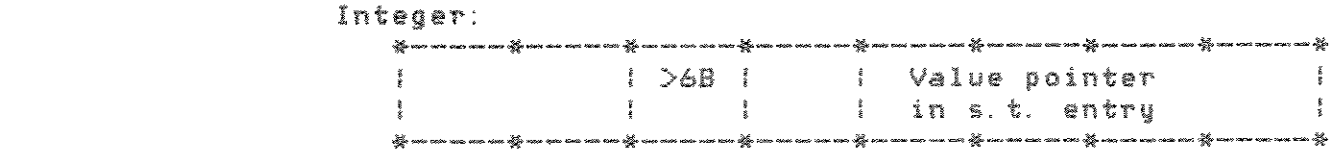
Real:



String:



Integer:



NOTE: May call PARSE recursively to get array indices.

ASSG

INPUT: Entry that was built in SYM and SMB must be on the stack. FAC must contain the 8-byte descriptor of the string, floating point number, or integer.

OUTPUT: The symbol table for the variable has the correct variable space data (address of string if a string entry, 2 byte value if integer entry, 8 byte value if floating point value). If a string entry, the strings backpointer points to the symbol table entry.