7 UTILITY ROUTINES

There are a number of useful routines within the QL ROM that would be of great benefit to application programmers. For example, how about a routine to compare two strings, or even better, a complete floating point arithmetic package? All such routines are available to you!

SIMPLIFIED TRAP ROUTINES

It is possible to class a number of the utility routines (seven to be precise) as being simplified TRAP routines. They offer a reduced overhead in accessing common I/O procedures, but they must be called in user mode.

SUPERBASIC UTILITY ROUTINES

The second group of utility routines described in this chapter are the SuperBASIC utility routines. They may be called from any code (i.e., code running in user or supervisor mode). An enforcement on routine accesses is that all addresses passed to a routine must be relative to register A6. If this rule is not obeyed, for every address passed over, there is no telling what the routine may actually do!

As a general rule, to make life easy for yourself, you may set register A6 to zero at the start of your program (e.g., by using SUB.L A6,A6), if your program resides in the transient program area. In this way you can forget about A6 and just concentrate on making the code program counter relative (i.e., position independent). Alternatively you can assemble your code starting at address zero. In this way your position independent code will always end up with addresses relative to a zero base address. TPA jobs are executed with A6 pointing to the base of the code and, therefore, the addresses in your code will automatically be relative to A6.

It is advisable to allocate at least 64 bytes of stack space before using any of these utilities. In the case of the arithmetic package, 96 bytes would be more appropriate. Note that these are minimal requirements.

7.1 Use of 68000 registers

Each utility routine is accessed through its own vector (see Chapter 3) and therefore there is no need to pass over a code indicating which operation is to be performed.

If an error status is returned by a routine it will be as a long-word in register D0. In such cases, if the code returned is not zero then an error has occurred. Small negative error codes are used to indicate standard errors. These error codes are listed in Appendix C. If the trap call invoked some form of additional device driver, the error code returned can be a pointer to a specific error message. In order that the two types of error return code might never be confused, the pointer type error code is in fact a pointer to an address $8000 below that of the true error message. The full descriptions of the routines which return this kind of error status list which additional errors can be returned. It would of course be wise to check for any errors after a routine has been called.

It is important to realize that not all the routines return an error status. Also, some routines will use register D0 to return a result to the calling program (not an error!). It is the programmers responsibility to know whether or not an error status will be returned, and, if not, whether register D0 will be used for some other purpose.

In addition to the use of registers D0 and A6, data registers D1 to D3 and address registers A0 to A3 are variably used to pass values to and from the routines. A few routines will also use D7 or A4. When the appropriate registers have been set for any one call the appropriate routine is accessed via its well-defined vector (see Chapter 3). In cases where the data size qualifier (i.e., '.B', '.W', or '.L') is not specified within the description, the default is long-word (i.e., '.L').

UT.WINDW Vector $C4 (196)

Set window using name

Entry parameters:	A0	Pointer to name
	A1	Pointer to parameter block
Return parameters:	A0	Channel ID
Affected registers:	D1 - D3, A0 - A3	
Additional errors:	OM (-3)	out of memory
	OR (-4)	range error
	NO (-6)	channel not available
	BN (-12)	bad device name

Description

This routine will set up a window defined by the specified name, and with the attributes given in the parameter block. The name should conform to the standard format for a screen device name (see Sec.5.2). The parameter block consists of four bytes as follows:

Byte	Use
0	Border colour
1	Border width
2	Paper colour
3	Ink colour

The strip colour for the window will be set to the same colour as the paper.

Note that the supplied name should be in the form of a standard string. The string definition will consist of a data word, defining the length of the string, followed by the string itself.

UT.CON Vector $C6 (198)

Set up console window

Entry parameters:	A1	Pointer to parameter block
Return parameters:	A0	Channel ID
Affected registers:	D1 - D3, A0 - A3	
Additional errors:	OM (-3)	out of memory
	OR (-4)	range error
	NO (-6)	channel not available

Description

This routine will set up a console window defined by the attributes given in the parameter block. The block consists of four bytes, and four words, as follows:

Offset	Use	
00	Border colour	(byte)
01	Border width	
02	Paper colour	
03	Ink colour	
04	Width of window	(word)
06	Height of window	
08	X origin	
0A	Y origin	

The strip colour for the window will be set to the same colour as the paper. The default keyboard buffer length of 128 bytes will be used.

UT.SCR Vector $C8 (200)

Set up screen window

Entry parameters:	A1	Pointer to parameter block
Return parameters:	A0	Channel ID
Affected registers:	D1 - D3, A0 - A3	
Additional errors:	OM (-3)	out of memory
	OR (-4)	range error
	NO (-6)	channel not available

Description

This routine will set up a screen window defined by the attributes given in the parameter block. The block consists of four bytes, and four words, as follows:

Offset	Use	
00	Border colour	(byte)
01	Border width	
02	Paper colour	
03	Ink colour	
04	Width of window	(word)
06	Height of window	
08	X origin	
0A	Y origin	

The strip colour for the window will be set to the same colour as the paper.

UT.ERRO Vector $CA (202)

Write error message to channel 0

Entry parameters:	D0.L Error code
Return parameters:	none
Affected registers:	none
Additional errors:	none

Description

This routine will write the error message, defined by D0 on entry, to channel zero. If the error code is in fact an address to a specific error message (see Sec.7.1), the message must be in the form of a standard string and it must be terminated by an ASCII <LF> ($0A). The string definition will consist of a data word, defining the length of the string, followed by the string itself. The defined length of the string will include the <LF>.

UT.ERR   Vector $CC (204)

Write error message to channel n

Entry parameters:	DO.L	Error code
	AO	Channel ID
Return parameters:	none	
Affected registers:	none	
Additional errors:	NC (-1) not complete	
	NO (-6) channel not open	

Description

This routine will write the error message, defined by DO on entry, to the specified channel. If the error code is in fact an address to a specific error message (see Sec.7.1), the message must be in the form of a standard string and it must be terminated by an ASCII <LF> ($OA). The string definition will consist of a data word, defining the length of the string, followed by the string itself. The defined length of the string will include the <LF>.

UT.MINT Vector $CE (206)

Convert integer to ASCII and write it to channel n.

Entry parameters:	D1.W	Integer value
	A0	Channel ID
Return parameters:	none	
Affected registers:	D1 - D3, A1 - A3	
Additional errors:	NC (-1) not complete	
	NO (-6) channel not open	

Description

This routine will convert the integer, specified by register D1, into an ASCII string. It will then add an ASCII space to the end of the string and send the whole string to the specified channel.

UT.MTEXT Vector $D0 (208)

Send message to channel n

Entry parameters:	A0	Channel ID
	A1	Base of message
Return parameters:	none	
Affected registers:	D1 - D3, A1 - A3	
Additional errors:	NC (-1) not complete	
	NO (-6) channel not open	

Description

This routine will send the specified message to the specified channel. The message must be in the standard format for a string. The string definition will consist of a data word, defining the length of the string, followed by the string itself. The defined length of the string will include any ASCII <LF> terminator, which may be present if desired.

UT.CSTR Vector $E6 (230)

Compare two strings

Entry parameters:	D0.B	Comparison type
	A0	Base of string 0
	A1	Base of string 1
Return parameters:	D0.L	Result
Affected registers:	none	
Additional errors:	none	

Description

This is a SuperBASIC utility. All addresses must be relative to A6.

The comparison type, specified by register D0 on entry, defines the mode of operation of the comparison operation. Valid types are:

Type	Comparison
0	Simple, character by character
1	As type 0, case of characters ignored
2	As type 0, value of embedded numbers is significant
3	As type 1, value of embedded numbers is significant

The result returned in register D0 will be one of three values:

Result	Reason
-1	String 0 is less than string 1
0	String 0 and string 1 are the same
+1	String 1 is less than string 0

The string definitions will consist of a data word, defining the length of the string, followed by the string itself.

CN.DATE Vector $EC (236)

Get date and time

Entry parameters:	D1.L	Date (internal value)
	A1	Pointer to stack area
Return parameters:	A1	Updated stack pointer
Affected registers:	A1	
Additional errors:	none	

Description

This is a SuperBASIC utility. All addresses must be relative to A6.

This routine will return the data and time as a 20-byte string in standard string format. The date string will have the form:

yyyy mmm dd hh:mm:ss

On entry to the routine, register A1 must point to the top of a stack area. At least 22 bytes must be available at the top of this stack for the routine to store the string, and its length. On returning to the calling program, the pointer will be pointing to the new top of the stack area (which will be holding the length of the string).

CN.DAY Vector $EE (238)

Get day of week

Entry parameters:	D1.L	Date (internal value)
	A1	Pointer to stack area
Return parameters:	A1	Updated stack pointer
Affected registers:	A1	
Additional errors:	none	

Description

This is a SuperBASIC utility. All addresses must be relative to A6.

This routine will return the day of the week as a 3-byte string in standard string format. On entry to the routine, register A1 must point to the top of a stack area. At least six bytes must be available at the top of this stack for the routine to store the string, and its length. On returning to the calling program, the pointer will be pointing to the new top of the stack area (which will be holding the length of the string).

CN.FTOD Vector $F0 (240)

Convert floating point to ASCII

Entry parameters:	A0	Pointer to buffer
	A1	Pointer to stack
Return parameters:	A0	Updated buffer pointer
	A1	Updated stack pointer
Affected registers:	D1 - D3, A0 - A3	
Additional errors:	none	

Description

This is a SuperBASIC utility. All addresses must be relative to A6.

This routine will convert a floating point number into a string of ASCII characters. The string will not be in standard format and, therefore, will not be preceded by a word of data giving the byte count.

On entry, the floating point number must be on the top of a stack area pointed to by A1, and register A0 must point to a buffer area. On return, the string will be stored in the buffer. The buffer must be at least 14 bytes long. Both the stack pointer and the buffer pointer will be updated. The stack pointer will be incremented by 6, and the buffer pointer will point to the byte following the last character stored.

CN.ITOD Vector $F2 (242)

Convert integer to ASCII

Entry parameters:	A0	Pointer to buffer
	A1	Pointer to stack
Return parameters:	A0	Updated buffer pointer
	A1	Updated stack pointer
Affected registers:	D1 - D3, A0 - A3	
Additional errors:	none	

Description

This is a SuperBASIC utility. All addresses must be relative to A6.

This routine will convert an integer number into a string of ASCII characters. The string will not be in standard format and, therefore, will not be preceded by a word of data giving the byte count.

On entry, the integer must be on the top of a stack area pointed to by A1, and register A0 must point to a buffer area. On return, the string will be stored in the buffer. The buffer must be at least six bytes long. Both the stack pointer and the buffer pointer will be updated. The stack pointer will be incremented by 2, and the buffer pointer will point to the byte following the last character stored.

CN.ITOBB Vector $F4 (244)

Convert byte (binary) to ASCII

Entry parameters:	A0	Pointer to buffer
	A1	Pointer to stack
Return parameters:	A0	Updated buffer pointer
	A1	Updated stack pointer
Affected registers:	D1 - D3, A0 - A3	
Additional errors:	none	

Description

This is a SuperBASIC utility. All addresses must be relative to A6.

This routine will convert a binary number, of size byte, into a string of ASCII characters. The string will not be in standard format and, therefore, will not be preceded by a word of data giving the byte count.
On entry, the byte must be on the top of a stack area pointed to by A1, and register A0 must point to a buffer area. On return, the string will be stored in the buffer. The buffer must be at least eight bytes long. Both the stack pointer and the buffer pointer will be updated. The stack pointer will be incremented by 1, and the buffer pointer will point to the byte following the last character stored.

CN.ITOBW Vector $F6 (246)

Convert word (binary) to ASCII

Entry parameters:	A0	Pointer to buffer
	A1	Pointer to stack
Return parameters:	A0	Updated buffer pointer
	A1	Updated stack pointer
Affected registers:	D1 - D3, A0 - A3	
Additional errors:	none	

Description

This is a SuperBASIC utility. All addresses must be relative to A6.

This routine will convert a binary number, of size word, into a string of ASCII characters. The string will not be in standard format and, therefore, will not be preceded by a word of data giving the byte count.
On entry, the word of data must be on the top of a stack area pointed to by A1, and register A0 must point to a buffer area. On return, the string will be stored in the buffer. The buffer must be at least 16 bytes long. Both the stack pointer and the buffer pointer will be updated. The stack pointer will be incremented by 2, and the buffer pointer will point to the byte following the last character stored.

CN.ITOBL Vector $F8 (248)

Convert long-word (binary) to ASCII

Entry parameters:	A0	Pointer to buffer
	A1	Pointer to stack
Return parameters:	A0	Updated buffer pointer
	A1	Updated stack pointer
Affected registers:	D1 - D3, A0 - A3	
Additional errors:	none	

Description

This is a SuperBASIC utility. All addresses must be relative to A6.

This routine will convert a binary number, of size long-word, into a string of ASCII characters. The string will not be in standard format and, therefore, will not be preceded by a word of data giving the byte count.

On entry, the long-word must be on the top of a stack area pointed to by A1, and register A0 must point to a buffer area. On return, the string will be stored in the buffer. The buffer must be at least 32 bytes long. Both the stack pointer and the buffer pointer will be updated. The stack pointer will be incremented by 4, and the buffer pointer will point to the byte following the last character stored.

CN.ITOHB Vector $FA (250)

Convert byte to hexadecimal ASCII

Entry parameters:	A0	Pointer to buffer
	A1	Pointer to stack
Return parameters:	A0	Updated buffer pointer
	A1	Updated stack pointer
Affected registers:	D1 - D3, A0 - A3	
Additional errors:	none	

Description

This is a SuperBASIC utility. All addresses must be relative to A6.

This routine will convert a single byte into a string of ASCII characters. The string will not be in standard format and, therefore, will not be preceded by a word of data giving the byte count.
On entry, the byte must be on the top of a stack area pointed to by A1, and register A0 must point to a buffer area. On return, the string will be stored in the buffer. The buffer must be at least two bytes long. Both the stack pointer and the buffer pointer will be updated. The stack pointer will be incremented by 1, and the buffer pointer will point to the byte following the last character stored.

CN.ITOHW Vector $FC (252)

Convert word to hexadecimal ASCII

Entry parameters:	A0	Pointer to buffer
	A1	Pointer to stack
Return parameters:	A0	Updated buffer pointer
	A1	Updated stack pointer
Affected registers:	D1 - D3, A0 - A3	
Additional errors:	none	

Description

This is a SuperBASIC utility. All addresses must be relative to A6.

This routine will convert a word of data into a string of ASCII characters. The string will not be in standard format and, therefore, will not be preceded by a word of data giving the byte count.
 On entry, the data word must be on the top of a stack area pointed to by A1, and register A0 must point to a buffer area. On return, the string will be stored in the buffer. The buffer must be at least four bytes long. Both the stack pointer and the buffer pointer will be updated. The stack pointer will be incremented by 2, and the buffer pointer will point to the byte following the last character stored.

CN.ITOHL Vector $FE (254)

Convert long-word to hexadecimal ASCII

Entry parameters:	A0	Pointer to buffer
	A1	Pointer to stack
Return parameters:	A0	Updated buffer pointer
	A1	Updated stack pointer
Affected registers:	D1 - D3, A0 - A3	
Additional errors:	none	

Description

This is a SuperBASIC utility. All addresses must be relative to A6.

This routine will convert a long-word of data into a string of ASCII characters. The string will not be in standard format and, therefore, will not be preceded by a word of data giving the byte count.

On entry, the long-word must be on the top of a stack area pointed to by A1, and register A0 must point to a buffer area. On return, the string will be stored in the buffer. The buffer must be at least eight bytes long. Both the stack pointer and the buffer pointer will be updated. The stack pointer will be incremented by 4, and the buffer pointer will point to the byte following the last character stored.

CN.DTOF Vector $100 (256)

Convert ASCII to floating point

Entry parameters:	D7	Pointer to end of buffer
	A0	Pointer to buffer
	A1	Pointer to stack
Return parameters:	A0	Updated buffer pointer
	A1	Updated stack pointer
Affected registers:	D1 - D3, A0 - A3	
Additional errors:	XP (-17) expression error	

Description

This is a SuperBASIC utility. All addresses must be relative to A6.

This routine will convert a string of ASCII characters into a floating point number. The string should not be in standard format and, therefore, will not be preceded by a word of data giving the byte count. Conversion will stop either when A0 equals D7, or when a non-permissible character is found in the ASCII string.

On entry, register A1 must point to the top of a stack area, and register A0 must point to the buffer area containing the string. The stack must have at least six bytes free. Register D7 may either point to the end of the buffer, or be zero. On return, the floating point number will be loaded onto the top of the stack. Both the stack pointer and the buffer pointer will be updated if no error occurs. The stack pointer will be decremented by 6, and the buffer pointer will point to the byte following the last valid character scanned.

CN.DTOI Vector $102 (258)

Convert ASCII to integer

Entry parameters:	D7	Pointer to end of buffer
	A0	Pointer to buffer
	A1	Pointer to stack
Return parameters:	A0	Updated buffer pointer
	A1	Updated stack pointer
Affected registers:	D1 - D3, A0 - A3	
Additional errors:	XP (-17) expression error	

Description

This is a SuperBASIC utility. All addresses must be relative to A6.

This routine will convert a string of ASCII characters into an integer. The string should not be in standard format and, therefore, will not be preceded by a word of data giving the byte count. Conversion will stop either when A0 equals D7, or when a non-permissible character is found in the ASCII string.

On entry, register A1 must point to the top of a stack area, and register A0 must point to the buffer area containing the string. The stack must have at least four bytes free. Register D7 may either point to the end of the buffer, or be zero. On return, the integer will be loaded onto the top of the stack. Both the stack pointer and the buffer pointer will be updated if no error occurs. The stack pointer will be decremented by 2, and the buffer pointer will point to the byte following the last valid character scanned.

CN.BTOIB Vector $104 (260)

Convert ASCII to byte

Entry parameters:	D7	Pointer to end of buffer
	A0	Pointer to buffer
	A1	Pointer to stack
Return parameters:	A0	Updated buffer pointer
	A1	Updated stack pointer
Affected registers:	D1 - D3, A0 - A3	
Additional errors:	XP (-17) expression error	

Description

This is a SuperBASIC utility. All addresses must be relative to A6.

This routine will convert a string of ASCII ones and zeros into a byte of data. The string should not be in standard format and, therefore, will not be preceded by a word of data giving the byte count. Conversion will stop either when A0 equals D7, or when a non-permissible character is found in the ASCII string.

On entry, register A1 must point to the top of a stack area (at an even address boundary), and register A0 must point to the buffer area containing the string. The stack must have at least four bytes free. Register D7 may either point to the end of the buffer, or be zero. On return, the byte (as the least significant byte of a word) will be loaded onto the top of the stack. Both the stack pointer and the buffer pointer will be updated if no error occurs. The stack pointer will be decremented by 1, and the buffer pointer will point to the byte following the last valid character scanned.

This routine will not work on QDOS versions 1.03 and earlier.

CN.BTOIW Vector $106 (262)

Convert ASCII to word

Entry parameters:	D7	Pointer to end of buffer
	A0	Pointer to buffer
	A1	Pointer to stack
Return parameters:	A0	Updated buffer pointer
	A1	Updated stack pointer
Affected registers:	D1 - D3, A0 - A3	
Additional errors:	XP (-17) expression error	

Description

This is a SuperBASIC utility. All addresses must be relative to A6.

This routine will convert a string of ASCII ones and zeros into a word of data. The string should not be in standard format and, therefore, will not be preceded by a word of data giving the byte count. Conversion will stop either when A0 equals D7, or when a non-permissible character is found in the ASCII string.

On entry, register A1 must point to the top of a stack area, and register A0 must point to the buffer area containing the string. The stack must have at least four bytes free. Register D7 may either point to the end of the buffer, or be zero. On return, the word of data will be loaded onto the top of the stack. Both the stack pointer and the buffer pointer will be updated if no error occurs. The stack pointer will be decremented by 2, and the buffer pointer will point to the byte following the last valid character scanned.

This routine will not work on QDOS versions 1.03 and earlier.

CN.BTOIL   Vector $108 (264)

Convert ASCII to long-word

Entry parameters:	D7	Pointer to end of buffer
	A0	Pointer to buffer
	A1	Pointer to stack
Return parameters:	A0	Updated buffer pointer
	A1	Updated stack pointer
Affected registers:	D1 - D3, A0 - A3	
Additional errors:	XP (-17) expression error	

Description

This is a SuperBASIC utility. All addresses must be relative to A6.

This routine will convert a string of ASCII ones and zeros into a long-word of data. The string should not be in standard format and, therefore, will not be preceded by a word of data giving the byte count. Conversion will stop either when A0 equals D7, or when a non-permissible character is found in the ASCII string.

On entry, register A1 must point to the top of a stack area, and register A0 must point to the buffer area containing the string. The stack must have at least four bytes free. Register D7 may either point to the end of the buffer, or be zero. On return, the long-word will be loaded onto the top of the stack. Both the stack pointer and the buffer pointer will be updated if no error occurs. The stack pointer will be decremented by 4, and the buffer pointer will point to the byte following the last valid character scanned.

This routine will not work on QDOS versions 1.03 and earlier.

CN.HTOIB Vector $10A (266)

Convert hexadecimal ASCII to byte

Entry parameters:	D7	Pointer to end of buffer
	A0	Pointer to buffer
	A1	Pointer to stack
Return parameters:	A0	Updated buffer pointer
	A1	Updated stack pointer
Affected registers:	D1 - D3, A0 - A3	
Additional errors:	XP (-17) expression error	

Description

This is a SuperBASIC utility. All addresses must be relative to A6.

This routine will convert a string of ASCII characters into a byte of data. The string should not be in standard format and, therefore, will not be preceded by a word of data giving the byte count. Conversion will stop either when A0 equals D7, or when a non-permissible character is found in the hexadecimal ASCII string.

On entry, register A1 must point to the top of a stack area (at an even address boundary), and register A0 must point to the buffer area containing the string. The stack must have at least four bytes free. Register D7 may either point to the end of the buffer, or be zero. On return, the byte (as the least significant byte of a word) will be loaded onto the top of the stack. Both the stack pointer and the buffer pointer will be updated if no error occurs. The stack pointer will be decremented by 1, and the buffer pointer will point to the byte following the last valid character scanned.

This routine will not work on QDOS versions 1.03 and earlier.

CN.HTOIW Vector $10C (268)

Convert hexadecimal ASCII to word

Entry parameters:	D7	Pointer to end of buffer
	A0	Pointer to buffer
	A1	Pointer to stack
Return parameters:	A0	Updated buffer pointer
	A1	Updated stack pointer
Affected registers:	D1 - D3, A0 - A3	
Additional errors:	XP (-17) expression error	

Description

This is a SuperBASIC utility. All addresses must be relative to A6.

This routine will convert a string of ASCII characters into a word of data. The string should not be in standard format and, therefore, will not be preceded by a word of data giving the byte count. Conversion will stop either when A0 equals D7, or when a non-permissible character is found in the hexadecimal ASCII string.

On entry, register A1 must point to the top of a stack area, and register A0 must point to the buffer area containing the string. The stack must have at least four bytes free. Register D7 may either point to the end of the buffer, or be zero. On return, the word of data will be loaded onto the top of the stack. Both the stack pointer and the buffer pointer will be updated if no error occurs. The stack pointer will be decremented by 2, and the buffer pointer will point to the byte following the last valid character scanned.

This routine will not work on QDOS versions 1.03 and earlier.

CN.HTOIL Vector $10E (270)

Convert hexadecimal ASCII to long-word

Entry parameters:	D7	Pointer to end of buffer
	A0	Pointer to buffer
	A1	Pointer to stack
Return parameters:	A0	Updated buffer pointer
	A1	Updated stack pointer
Affected registers:	D1 - D3, A0 - A3	
Additional errors:	XP (-17) expression error	

Description

This is a SuperBASIC utility. All addresses must be relative tɔ A6.

This routine will convert a string of ASCII characters into a long-word of data. The string should not be in standard format and, therefore, will not be preceded by a word of data giving the byte count. Conversion will stop either when A0 equals D7, or when a non-permissible character is found in the hexadecimal ASCII string.

On entry, register A1 must point to the top of a stack area, and register A0 must point to the buffer area containing the string. The stack must have at least four bytes free. Register D7 may either point to the end of the buffer, or be zero. On return, the long-word of data will be loaded onto the top of the stack. Both the stack pointer and the buffer pointer will be updated if no error occurs. The stack pointer will be decremented by 4, and the buffer pointer will point to the byte following the last valid character scanned.

This routine will not work on QDOS versions 1.03 and earlier.

RI.EXEC Vector $11C (284)

Execute single arithmetic operation

Entry parameters:	D0.B	Operation code
	D7	0
	A1	Pointer to arithmetic stack
	A4	Pointer to base of load/store area
Return parameters:	A1	Updated stack pointer
Affected registers:	A1	
Additional errors:	OV (-18) arithmetic overflow	

Description

This is a SuperBASIC utility. All addresses must be relative to A6. Also, it is advised that data register D7 be set to zero.

The arithmetic package operates on floating point numbers that exist on a specified arithmetic stack. The package will operate on the floating point number at the top of the stack (TOS), which is pointed to by '0(A6,A1.L)', and if required for any one operation the package will also operate on the next floating point number on the stack (NOS), which is pointed to by '6(A6,A1.L)'. The format of floating point numbers is discussed in Chapter 8.

The package will accept one of two types of operation code. First, there are the true arithmetic operations. These have codes in the range $02 to $30 inclusive:

Code	Name	Operation
02	RI.NINT	TOS to nearest integer, A1=A1+4
04	RI.INT	TOS to truncated integer, A1=A1+4
06	RI.NLINT	TOS to nearest long integer, A1=A1+2
08	RI.FLOAT	Integer TOS to floating point, A1=A1-4
0A	RI.ADD	Add TOS to NOS, A1=A1+6
0C	RI.SUB	Subtract TOS from NOS, A1=A1+6
0E	RI.MULT	Multiply TOS by NOS, A1=A1+6
10	RI.DIV	Divide TOS into NOS, A1=A1+6 (NOS/TOS)
12	RI.ABS	Positive value of TOS
14	RI.NEG	Negate TOS
16	RI.DUP	Duplicate TOS, A1=A1-6
18	RI.COS	Cosine of TOS
1A	RI.SIN	Sine of TOS
1C	RI.TAN	Tangent of TOS
1E	RI.COT	Cotangent of TOS
20	RI.ASIN	Arcsine of TOS

22	RI.ACOS	Arccosine of TOS
24	RI.ATAN	Arctangent of TOS
26	RI.ACOT	Arccotangent of TOS
28	RI.SQRT	Square root of TOS
2A	RI.LN	Natural logarithm of TOS
2C	RI.LOG	LOG10 of TOS
2E	RI.EXP	Exponential of TOS
30	RI.POWFP	NOS to power of TOS, A1=A1+6

Some of these operations are seen to affect the stack. Remember that stacks grow downwards. A few examples will clarify the situation. First, let us look at RI.COS (code $18). This does not affect the stack pointer (A1). The floating point number on the top of the stack is taken as the argument for the cosine operation, and the resultant cosine is placed on the top of the stack. A floating point number is removed, and a floating point number is put back, so the stack pointer remains unmoved. Second, let us look at RI.NINT (code $02). This will take the floating point number from the top of the stack and convert it to the nearest integer. This integer result is then placed on the stack. Floating point numbers are six bytes long, but integers are only two bytes long, therefore the stack pointer is incremented by 4 (i.e., four less bytes will now be required on the stack). By looking at the effect on the stack pointer (A1), you should now be able to deduce what is happening to the stack for each of the operations listed.

|34 - FF| = 34 Vic

The second type of operation code is a negative code in the range $FF to $31 (interpreted as the range $FFFF to $FF31 inclusive). These codes indicate a load operation (i.e., from memory to stack) if the least significant bit of the code is zero, or they indicate a store operation (i.e., from stack to memory) if the least significant bit of the code is set. The memory address used for the load or store is given by:

A6.L + A4.L + ((opcode AND $FE) OR $FF00)

Only floating point values may be transferred. A load operation will cause the stack pointer A1 to be decremented by 6 (creating a new TOS). A save operation will cause the stack pointer A1 to be incremented by 6 (NOS will become new TOS).

RI.EXECB Vector $11E (286)

Execute list of arithmetic operations

Entry parameters:	D7	0
	A1	Pointer to arithmetic stack
	A3	Pointer to operation list
	A4	Pointer to base of variable area
Return parameters:	A1	Updated stack pointer
Affected registers:	A1	
Additional errors:	OV (-18) arithmetic overflow	

Description

This is a SuperBASIC utility. All addresses must be relative to A6. Also, it is advised that data register D7 be set to zero.

This routine enables a list of operations to be carried out by the arithmetic package. Any one operation may be as defined under RI.EXEC (vector $11C). The byte list of operation codes are pointed to by register A3 on entry to the routine, and the list must be terminated by a zero byte.