CN.ITOBL Vector F8

Convert a long word to an ASCII string in binary

Call parameters		Return parameters	
D0		D0	undefined
D1		D1	undefined
D2		D2	undefined
D3		D3	undefined
A0	pointer to buffer	A0	pointer to buffer
A1	pointer to stack	A1	pointer to stack
A2		A2	undefined
A3		A3	undefined

Error returns:

undefined

Description:

This routine will convert a long word on the stack (pointed to by A1) into a string of ASCII characters in a buffer (pointed to by A0) forming a 32 character binary number.

CN.ITOHB Vector FA

Convert a byte to an ASCII string in hexadecimal

Call parameters		Return parameters	
D0		D0	undefined
D1		D1	undefined
D2		D2	undefined
D3		D3	undefined
A0	pointer to buffer	A0	pointer to buffer
A1	pointer to stack	A1	pointer to stack
A2		A2	undefined
A3		A3	undefined

Error returns:

undefined

Description:

This routine will convert a byte on the stack (pointed to by A1) into a string of ASCII characters in a buffer (pointed to by A0) which form a two digit hex number.

CN.ITOHW Vector FC

Convert a word to an ASCII string in hexadecimal

Call parameters		Return parameters	
D0		D0	undefined
D1		D1	undefined
D2		D2	undefined
D3		D3	undefined
A0	pointer to buffer	A0	pointer to buffer
A1	pointer to stack	A1	pointer to stack
A2		A2	undefined
A3		A3	undefined

Error returns:

undefined

Description:

This routine will convert a word on the stack (pointed to by A1) into a string of ASCII characters in a buffer (pointed to by A0) which form a four digit hex number.

CN.ITOHL Vector FE

Convert a long word to an ASCII string in hexadecimal

Call parameters		Return parameters	
D0		D0	undefined
D1		D1	undefined
D2		D2	undefined
D3		D3	undefined
A0	pointer to buffer	A0	pointer to buffer
A1	pointer to stack	A1	pointer to stack
A2		A2	undefined
A3		A3	undefined

Error returns:

undefined

Description:

This routine will convert a long word on the stack (pointed to by A1) into a string of ASCII characters in a buffer (pointed to by A0) which form an eight digit hex number.

CN.DTOF Vector 100

Convert a floating point ASCII string to an f.p. number

Call parameters		Return parameters	
D1		D1	undefined
D2		D2	undefined
D3		D3	undefined
D7	0 or buffer end pointer	D7	preserved
A0	pointer to buffer	A0	pointer to buffer
A1	pointer to stack	A1	pointer to stack
A2		A2	undefined
A3		A3	undefined

Error returns:

XP error in conversion (eg. 1..0)

Description:

This routine will convert a string of ASCII characters in a buffer (pointed to by A0) into a floating point number on the stack (pointed to by A1). If there is an error then A0 and A1 are both unchanged. For a correct return, A0 points to the next character in the buffer.

CN.DTOI Vector 102

Convert a decimal ASCII string to an integer

Call parameters		Return parameters	
D1		D1	undefined
D2		D2	undefined
D3		D3	undefined
D7	0 or buffer end pointer	D7	preserved
A0	pointer to buffer	A0	pointer to buffer
A1	pointer to stack	A1	pointer to stack
A2		A2	undefined
A3		A3	undefined

Error returns:

XP error in conversion (eg. 1A)

Description:

This routine will convert a string of ASCII characters in a buffer (pointed to by A0) into an integer number as a long word on the stack (pointed to by A1). If there is an error then A0 and A1 are both unchanged. For a correct return, A0 points to the next character in the buffer.

CN.BTOIB Vector 104

Convert an 8 bit binary ASCII string to a byte

Call parameters		Return parameters	
D1		D1	undefined
D2		D2	undefined
D3		D3	undefined
D7	0 or buffer end pointer	D7	preserved
A0	pointer to buffer	A0	pointer to buffer
A1	pointer to stack	A1	pointer to stack
A2		A2	undefined
A3		A3	undefined

Error returns:

XP error in conversion (eg. 10110201)

Description:

This routine will convert a string of ASCII characters in a buffer (pointed to by A0) into a byte on the stack (pointed to by A1). If there is an error then A0 and A1 are both unchanged. For a correct return, A0 points to the next character in the buffer.

Note that this routine will not work on QDOS V1.03 and earlier versions.

CN.BTOIW Vector 106

Convert a 16 bit binary ASCII string to an word

Call parameters		Return parameters	
D1		D1	undefined
D2		D2	undefined
D3		D3	undefined
D7	0 or buffer end pointer	D7	preserved
A0	pointer to buffer	A0	pointer to buffer
A1	pointer to stack	A1	pointer to stack
A2		A2	undefined
A3		A3	undefined

Error returns:

XP error in conversion

Description:

This routine will convert a string of ASCII characters in a buffer (pointed to by A0) into a word on the stack (pointed to by A1). If there is an error then A0 and A1 are both unchanged. For a correct return, A0 points to the next character in the buffer.

Note that this routine will not work on QDOS V1.03 and earlier versions.

CN.BTOIL Vector 108

Convert a 32 bit binary ASCII string to an long word

Call parameters		Return parameters	
D1		D1	undefined
D2		D2	undefined
D3		D3	undefined
D7	0 or buffer end pointer	D7	preserved
A0	pointer to buffer	A0	pointer to buffer
A1	pointer to stack	A1	pointer to stack
A2		A2	undefined
A3		A3	undefined

Error returns:

XP error in conversion

Description:

This routine will convert a string of ASCII characters in a buffer (pointed to by A0) into a long word on the stack (pointed to by A1). If there is an error then A0 and A1 are both unchanged. For a correct return, A0 points to the next character in the buffer.

Note that this routine will not work on QDOS V1.03 and earlier versions.

CN.HTOIB Vector 10A

Convert a 2 character hex ASCII string to a byte

Call parameters		Return parameters	
D1		D1	undefined
D2		D2	undefined
D3		D3	undefined
D7	0 or buffer end pointer	D7	preserved
A0	pointer to buffer	A0	pointer to buffer
A1	pointer to stack	A1	pointer to stack
A2		A2	undefined
A3		A3	undefined

Error returns:

XP error in conversion (eg. 1E4R)

Description:

This routine will convert a string of ASCII characters in a buffer (pointed to by A0) into a byte on the stack (pointed to by A1). If there is an error then A0 and A1 are both unchanged. For a correct return, A0 points to the next character in the buffer.

Note that this routine will not work on QDOS V1.03 and earlier versions.

CN.HTOIW Vector 10C

Convert a 4 character hex ASCII string to a word

Call parameters		Return parameters	
D1		D1	undefined
D2		D2	undefined
D3		D3	undefined
D7	0 or buffer end pointer		D7 preserved
A0	pointer to buffer	A0	pointer to buffer
A1	pointer to stack	A1	pointer to stack
A2		A2	undefined
A3		A3	undefined

Error returns:

XP error in conversion

Description:

This routine will convert a string of ASCII characters in a buffer (pointed to by A0) into a word on the stack (pointed to by A1). If there is an error then A0 and A1 are both unchanged. For a correct return, A0 points to the next character in the buffer.

Note that this routine will not work on QDOS V1.03 and earlier versions.

CN.HTOIL Vector 10E

Convert an 8 character hex ASCII string to a long word

Call parameters		Return parameters	
D1		D1	undefined
D2		D2	undefined
D3		D3	undefined
D7	0 or buffer end pointer		D7 preserved
A0	pointer to buffer	A0	pointer to buffer
A1	pointer to stack	A1	pointer to stack
A2		A2	undefined
A3		A3	undefined

Error returns:

XP error in conversion

Description:

This routine will convert a string of ASCII characters in a buffer (pointed to by A0) into an integer number as a long word on the stack (pointed to by A1). If there is an error then A0 and A1 are both unchanged. For a correct return, A0 points to the next character in the buffer.

Note that this routine will not work on QDOS V1.03 and earlier versions.

8.5.3 Basic procedure support routines

These routines are listed here purely to ensure that the vectored utility section is complete. Only a cursory glance at the operation of these routines is provided. For a more detailed understanding of the terms used and the types of applications, you should refer to chapter 10 on SuperBASIC Interfacing.

BP.INIT Vector 110

Initialise procedure or function into Bascic name table

Call parameters		Return parameters	
D1		D1	preserved
D2		D2	preserved
D3		D3	preserved
A0		A0	preserved
A1	definition list pointer	A1	undefined
A2		A2	preserved
A3		A3	preserved

Error returns:

none

Description:

This support routine will allow Basic procedures or functions to be linked into the Basic name table list. A1 is set to point to the start of the procedure definition list before calling the routine. See section 10.7.2 for more details.

CA.GTINT Vector 112

Get integer parameters for Basic

Call parameters		Return parameters	
D1		D1	undefined
D2		D2	undefined
D3		D3.W	number of fetched arguments
D4		D4	undefined
D6		D6	undefined
A0		A0	undefined
A1	pointer to stack	A1	pointer to stack
A2		A2	undefined
A3	first parameter pointer	A3	preserved
A5	last parameter pointer	A5	preserved

Error returns:

BP	Bad parameters
XP	Error in expression

Description:

This routine will fetch an indeterminate number of integer (word) procedure arguments, and put them on the arithmetic stack. A3 points to the start of the parameters and A5 points to the end of the parameters. Note that all references should be made relative to A6. For more details see section 10.7.4.

CA.GTFP Vector 114

Get floating point parameters for Basic

Call parameters		Return parameters	
D1		D1	undefined
D2		D2	undefined
D3		D3.W	number of fetched arguments
D4		D4	undefined
D6		D6	undefined
A0		A0	undefined
A1	pointer to stack	A1	pointer to stack
A2		A2	undefined
A3	first parameter pointer	A3	preserved
A5	last parameter pointer	A5	preserved

Error returns:

BP	Bad parameters
XP	Error in expression

Description:

This routine will fetch an indeterminate number of floating point (6 byte) procedure arguments, and put them on the arithmetic stack. A3 points to the start of the parameters and A5 points to the end of the parameters. Note that all references should be made relative to A6. For more details see section 10.7.4.

CA.GTSTR Vector 116

Get string parameters for Basic

Call parameters		Return parameters	
D1		D1	undefined
D2		D2	undefined
D3		D3.W	number of fetched arguments
D4		D4	undefined
D6		D6	undefined
A0		A0	undefined
A1	pointer to stack	A1	pointer to stack
A2		A2	undefined
A3	first parameter pointer	A3	preserved
A5	last parameter pointer	A5	preserved

Error returns:

XP	Error in expression
BP	Bad parameters

Description:

This routine will fetch an indeterminate number of strings (2+n (even) or 3+n (odd) bytes) procedure arguments, and put them on the arithmetic stack. A3 points to the start of the parameters and A5 points to the end of the parameters. Note that all references should be made relative to A6. For more details see section 10.7.4.

CA.GTLIN Vector 118

Get long integer parameters for Basic

Call parameters		Return parameters	
D1		D1	undefined
D2		D2	undefined
D3		D3.W	number of fetched arguments
D4		D4	undefined
D6		D6	undefined
A0		A0	undefined
A1	pointer to stack	A1	pointer to stack
A2		A2	undefined
A3	first parameter pointer	A3	preserved
A5	last parameter pointer	A5	preserved

Error returns:

BP	Bad parameters
XP	Error in conversion

Description:

This routine will fetch an indeterminate number of long integer (long word) procedure arguments, and put them on the arithmetic stack. A3 points to the start of the parameters and A5 points to the end of the parameters. Note that all references should be made relative to A6. For more details see section 10.7.4.

BV.CHRIX Vector 11A

Allocate space on the arithmetic stack

Call parameters		Return parameters	
D1.L	number of bytes required	D1	undefined
D2		D2	undefined
D3		D3	undefined
A0		A0	preserved
A1		A1	preserved
A2		A2	preserved
A3		A3	preserved

Error returns:

undefined

Description:

This utility allows user routines to allocate space on the arithmetic stack. The fetch argument routines already reserve enough space on the stack for themselves. The number of bytes to be reserved should be passed in D1. See section 10.7.3 for more details.

BP.LET Vector 120

Returns a parameter value to BASIC

Call parameters		Return parameters	
D1		D1	undefined
D2		D2	undefined
D3		D3	undefined
A0		A0	undefined
A1		A1	undefined
A2		A2	undefined
A3	pointer to table entry	A3	pointer to table entry

Error returns:

BP Bad parameters

Description:

This utility allows values to be returned to a procedure or function through the parameter list. Before the call, the parameter should be on the arithmetic stack. It must be in the correct form required by the calling parameter. A3 is set to point to the appropriate parameter entry in the name table. D0 is set to the error on return. See section 10.7.6 for more details.

8.5.4 The arithmetic routines

The arithmetic package on the QL is available for use in one of two forms. The first form is RI.EXEC which allows a single operation such as adding two numbers together to be carried out. The alternative form is RI.EXECB which allows a whole list of operations to be carried out.

The 68008 itself is able to carry out arithmetic operations on bytes, words and long words, so this does not need to be supported by the arithmetic package here. All operations are therefore carried out in floating point format in this package. Before the particular facilities are described, it is necessary to understand a few basic concepts.

Unlike the 68008, there are no *registers* in the arithmetic package for the storage of data. All operations are carried out from and to the *arithmetic stack*. For example, if there are two Basic variables VAL1 and VAL2, these will each be stored somewhere in memory. In order that they can be added together, each one is copied from memory onto the arithmetic stack. An ADD operation is executed by RI.EXEC, then the result can be popped from the stack back into memory.

The arithmetic stack operates downwards from (A6,A1.L). The two values on the stack which can be dealt with are the value at the top of the stack (called TOS) and the value below that on the stack (called next on stack or NOS). Some operations only operate on TOS and leave the result in place of the old TOS. Other operations operate on both the TOS and the NOS.

The TOS is pointed to by (A6,A1.L) and the NOS is pointed to by 6(A6,A1.L). The stack layout is illustrated in figure 8.1.

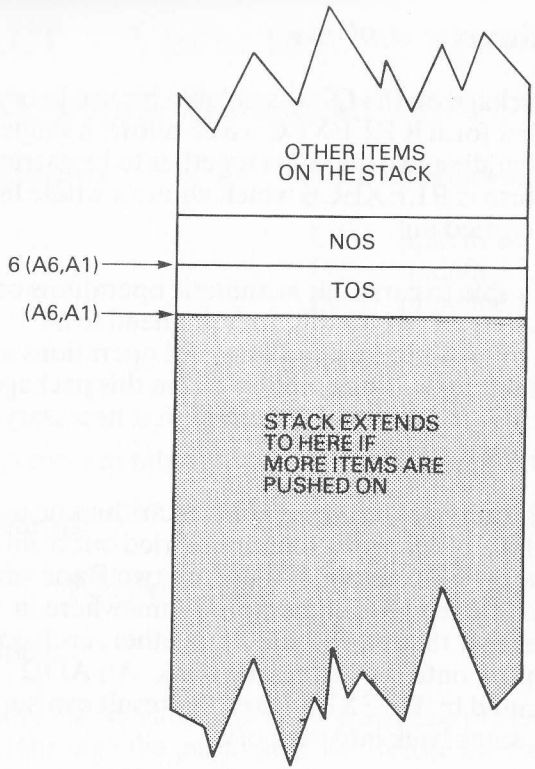

Figure 8.1 – the floating point stack

Floating point numbers on the QL are stored in the format of a long word mantissa and a two byte offset exponent (the most significant four bits of which are zero). This is illustrated below.

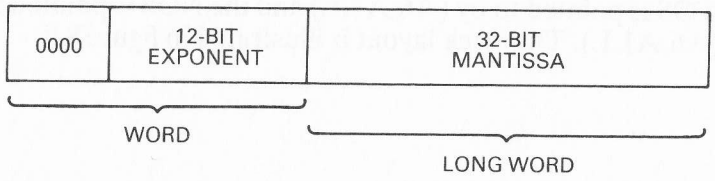

Figure 8.2 – The storage format for floating point numbers

The floating point value is:

mantissa * (2 ↑ (exponent – $800 – $1F))

The floating point package accepts two types of op codes (operation codes). Codes between $02 and $30 are true arithmetic operations. The other byte codes are extended to a negative word between $FFFF and $FF31 inclusive and are instructions to load or store the floating point number to the address given by (A6.L + A4.L + ((opcode OR $FF00) AND $FFEE)). The operation is a load if bit 0 of the opcode is clear. The operation is a store if it is set. Loading causes A1 to be decremented by 6 bytes and saving causes A1 to be incremented by 6. The old NOS becomes the new TOS.

Operation codes for the arithmetic package

Code	Name	Change to A1	Operation
00	RI.TERM		
02	RI.NINT	+4	find nearest integer to TOS
04	RI.INT	+4	truncate TOS to integer
06	RI.NLINT	+2	nearest long integer to TOS
08	RI.FLOAT	−4	convert TOS integer to floating point
0A	RI.ADD	+6	add TOS to NOS
0C	RI.SUB	+6	subtract NOS from TOS
0E	RI.MULT	+6	multiply TOS by NOS
10	RI.DIV	+6	divide TOS into NOS
12	RI.ABS	0	positive value of TOS
14	RI.NEG	0	negate TOS
16	RI.DUP	−6	duplicate TOS
18	RI.COS	0	take cosine of TOS
1A	RI.SIN	0	take sine of TOS
1C	RI.TAN	0	take tangent of TOS
1E	RI.COT	0	take cotangent of TOS
20	RI.ASIN	0	take arcsine of TOS
22	RI.ACOS	0	take arccosine of TOS
24	RI.ATAN	0	take arctangent of TOS
26	RI.ACOT	0	take arccotangent of TOS
28	RI.SQRT	0	take square root of TOS
2A	RI.LN	0	take natural logarithm of TOS
2C	RI.LOG10	0	take logarithm to the base 10 of TOS
2E	RI.EXP	0	take exponential of TOS
30	RI.POWFP	+6	take NOS to the power of TOS
00	RI.LOAD		load operand key if bit 0 clear
01	RI.STORE		store operand key if bit 0 set

RI.EXEC Vector 11C

Executes an arithmetic operation

Call parameters		Return parameters	
D0.W	operation code	D0	error code
D1		D1	preserved
D2		D2	preserved
D3		D3	preserved
D7	should be set to zero	D7	preserved
A0		A0	preserved
A1	arithmetic stack pointer	A1	updated
A2		A2	preserved
A3		A3	preserved
A4	pointer to variable area	A4	preserved

Error returns:

OV arithmetic overflow

Description:

This routine allows an arithmetic operation to be carried out by the arithmetic package, as described in the rest of this section. The operation code should be passed as a word in D0. A1 points to the arithmetic stack. A4 points to the base of the variables area. Note that D7 has to be set to zero for some operations in QDOS version 1.03 or earlier.

RI.EXECB Vector 11E

Executes a list of arithmetic operations

Call parameters		Return parameters	
D1		D1	preserved
D2		D2	preserved
D3		D3	preserved
D7	should be set to zero	D7	preserved
A0		A0	preserved
A1	arithmetic stack pointer	A1	updated
A2		A2	preserved
A3	operation list pointer	A3	preserved
A4	pointer to variable area	A4	preserved

Error returns:

OV arithmetic overflow

Description:

This routine allows a list of arithmetic operations to be carried out by the arithmetic package, as described in the rest of this section. The operation codes should be passed as a table of *bytes*, pointed to by A3. The end of the table is signified by a zero byte. A1 points to the arithmetic stack. A4 points to the base of the variables area. Note that D7 has to be set to 0 for some functions in QDOS version 1.03 or earlier.

8.5.5 SINH as a resident function

This example program illustrates how many of the utilities described in the rest of section 8.5 can be used. A resident function is set up to evaluate the value of SINH(x) where 'x' is some floating point value.

You should now refer to the program listing for sinh at the end of this sub-section.

The Basic procedure initialisation utility BP.INIT is called. This causes the procedure/function definition block at PROC_TAB to be linked into the Basic procedure/function list. Once initialised in this way, the routine SINH can be accessed just like any other function from Basic.

Once entered, the routine SINH will allocate space for 12 floating point numbers on the arithmetic stack (using BV.CHRIX). The parameters passed to the function can then be obtained using CA.GTFP. If more or less than one parameter was provided, an error will be generated at this stage.

The standard algorithm for evaluating sinh of a function is:

$$\sinh(x) = (e^{x} - e^{-x})/2$$

The arithmetic stack must now be prepared by putting all the required values onto it. If the parameter is designated by 'X', we put 'X', '1.0' and '0.5' onto the stack. The two constants will be used later.

The sequence of operation by the arithmetic package is now:

load X to TOS
take exponential of TOS
load 1.0 to TOS
load exp(X) to TOS
divide TOS into NOS to generate exp(-x)
subtract NOS from TOS
load 0.5 onto stack at TOS
multiply TOS by NOS
put TOS back in X
and reset A1

the result SINH(X) now exists on the top of the stack. So that this value is returned as the value of the function, A1 (the stack pointer) is stored in BV_RIP.

```
*
* SINH
*
*
BP.INIT  EQU    $110
CA.GTFP  EQU    $114
BV.CHRIX EQU    $11A
RI.EXECB EQU    $11E
RI.SUB   EQU    $0C
RI.MULT  EQU    $0E
RI.DIV   EQU    $10
RI.EXP   EQU    $2E
RI.STORE EQU    $1
*
BV_RIP   EQU    $58
*
ERR.BP   EQU    -15
*
         LEA    PROC_TAB(PC),A1 load address of procedure
         MOVE.W BP.INIT,A2      definition table and add to
         JSR    (A2)            BASIC's table
         MOVEQ  #0,D0           no error return
         RTS
*
PROC_TAB
         DC.W   0               NO procedures
         DC.W   0               end of procedures
         DC.W   1               two functions
         DC.W   SINH-*
         DC.B   4,'SINH',0      5 bytes so extra 0 to align
         DC.W   0               end of functions

*
* SINH function
*
SINH
         MOVEQ  #72,D1          reserve space for 12
         MOVE.W BV.CHRIX,A2     floating point numbers
         JSR    (A2)
*
         MOVE.W CA.GTFP,A2      get any number of ¤oating
         JSR    (A2)            point numbers
         BNE.S  SINH_RTS        ... oops
         SUBQ.W #1,D3           but we only wanted one
         BNE.S  ERR_BP
```

```
*
        LEA     6(A1),A4        set top of stack pointer
        CLR.W   -2(A6,A1.L)             put 1.0 on
        MOVE.L  #$08014000,-6(A6,A1.L)
        CLR.W   -8(A6,A1.L)             put 0.5 on
        MOVE.L  #$08004000,-12(A6,A1.L)
        SUB.W   #12,A1
        LEA     OP_TAB(PC),A3   set up operation table
        MOVE.W  RI.EXECB,A2     now use arithmetic package
        JSR     (A2)
*
        ADD.W   #12,A1          reset arithmetic stack
        MOVE.L  A1,BV_RIP(A6)
        MOVEQ   #2,D4           and set type
SINH_RTS
        RTS
ERR_BP
        MOVEQ   #ERR.BP,D0
        RTS
*
* contents of RI stack
*
RIS_VAL EQU     -6
RIS_1.0 EQU     -12
RIS_0.5 EQU     -18
RIS_EXP EQU     -24
*
OP_TAB
        DC.B    RIS_VAL         Stack   X
        DC.B    RI.EXP                  exp(X)
        DC.B    RIS_1.0                 exp(X),1.0
        DC.B    RIS_EXP                 exp(X),1.0,exp(X)
        DC.B    RI.DIV                  exp(X),exp(-X)
        DC.B    RI.SUB                  exp(X)-exp(-X)
        DC.B    RIS_0.5                 exp(X)-exp(-X),0.5
        DC.B    RI.MULT                 (exp(X)-exp(-X))/2
        DC.B    RIS_VAL+RI.STORE
        DC.B    0
```


8.6 Microdrive support routines

MD.READ Vector 124

Read a sector on a microdrive

Call parameters		Return parameters	
D1		D1	file number
D2		D2	block number
D7		D7	undefined
A0		A0	undefined
A1	pointer to buffer start	A1	pointer to buffer end
A2		A2	undefined
A3	$18020	A3	$18020

Error returns:

normal = failed
return+2 = OK

Description:

This routine reads a sector on a microdrive. All registers except A3 and A6 are volatile. A3 must be set to point to the microdrive control register at entry and the interrupts must be disabled. D0 is not set on return, but there are two possible returns. The normal return implies that there was failure to read. The normal return plus 2 means that the read was successful.

The vector to this routine points to $4000 bytes before the actual code. The following code may therefore be used to invoke it:

```
MOVE.W      aa.aaaa,An
JSR         $4000(An)
RET1.W      return at this point means the read failed
RET2.W      return here means that read was successful
```

MD.WRITE Vector 126

Writes a sector on a microdrive

Call parameters		Return parameters	
D1		D1	undefined
D2		D2	undefined
D7		D7	undefined
A0		A0	undefined
A1	pointer to buffer start	A1	pointer to buffer end
A2		A2	undefined
A3	$18020	A3	$18020

Error returns:

none

Description:

This routine writes a sector on a microdrive. All registers except A3 and A6 are volatile. A3 must be set to point to the microdrive control register at entry and the interrupts must be disabled. Before MD.WRITE is called, the stack pointer must be set to point to two words. The first word is the file number and the second word is the block number of the sector to be written.

The vector to this routine points to $4000 bytes before the actual code. The following code may therefore be used to invoke it:

```
MOVE.W      aa.aaaa,An
JSR         $4000(An)
```


MD.VERIN Vector 128

Verify a sector on a microdrive

Call parameters		Return parameters	
D1		D1	file number
D2		D2	block number
D7		D7	undefined
A0		A0	undefined
A1	pointer to buffer start	A1	pointer to buffer end
A2		A2	undefined
A3	$18020	A3	$18020

Error returns:

normal = failed
return+2 = OK

Description:

This routine verifies a sector on a microdrive. All registers except A3 and A6 are volatile. A3 must be set to point to the microdrive control register at entry and the interrupts must be disabled. D0 is not set on return, but there are two possible returns. The normal return implies that there was failure to verify. The normal return plus 2 means that the verify was successful.

The vector to this routine points to $4000 bytes before the actual code. The following code may therefore be used to invoke it:

```
MOVE.W      aa.aaaa,An
JSR         $4000(An)
RET1.W      return at this point means failed to verify
RET2.W      return here means that sector verified OK
```