8 Vectored Utilities

8.1 Introduction

This section is something of a mixture because it contains a wide variety of different routines within the ROM which can be used in totally different applications. The vectored utilities are so called because they are accessed by looking up their addresses in a table of vectors (pointers to the start of each routine). This means that the routines will work with future versions of QDOS in which the actual address of the routine will very probably have changed.

Accessing the routines via the vector table

The vector table extends from address $C0 in the ROM up to $12B. The *word* entries point to the actual start addresses of the relevant routines. Since each entry takes the form of a word rather than a long word, all vectored routines must be in the lower 32K of memory. The stack space is not checked by the utility routines, so at least 64 bytes should be available for ordinary utilities and 96 bytes should be available for the arithmetic routines. Access to a routine can be obtained using the following lines of 68000 code:

```
MOVE.W   aa.aaaa,An     aa.aaaa is the entry vector
JSR      (An)           jump to subroutine at aa.aaaa
```

If the utility sets a status code in D0, the status register in the 68008 will be set accordingly upon returning return from the routine.

The routines can be split up into five main functional groups, defined by the requirements of the calling code.

Supervisor Mode routines

In certain situations the supervisor mode routines are called by code which is not contained inside QDOS. This may occur for example in parts of the device drivers used by trap #2 or trap #3. There are several rules to which the calling code must conform if it is to use these routines:

1. The calling code must be in supervisor mode before making the call.

2. The calling code must not have been initiated by an interrupt.

3. A6 must point to the base of the system variables.

The following Supervisor Mode routines are available:

Vector	Name	Type	Description
C0	MM.ALCHP	SM	Allocates common heap area
C2	MM.RECHP	SM	Releases common heap space
E8	IO.SERQ	SM	Direct queue handling
EA	IO.SERIO	SM	General IO handling

Simplified Trap routines

There are quite a lot of traps which are used over and over again. In order to reduce the code required for these traps, where the parameters conform to the most common format, simplified vectored routines can be used instead. These routines are listed below. Note that they must all be called in USER mode.

Vector	Name	Type	Description
C4	UT.WINDW	ST	Set up window using supplied name
C6	UT.CON	ST	Set up a console window
C8	UT.SCR	ST	Set up screen window
CA	UT.ERR0	ST	Write error message to channel 0
CC	UT.ERR	ST	Write error message to any channel
CE	UT.MINT	ST	Convert an integer to ASCII
D0	UT.MTEXT	ST	Send message to a channel

General Utility routines

These routines can be called from any code. No special demands are made on the calling code.

Vector	Name	Type	Description
D2	UT.LINK	GU	Link an item into a list
D4	UT.UNLNK	GU	Unlink an item from a list
D8	MM.ALLOC	GU	Allocates an area in a heap
DA	MM.LNKFR	GU	Links free space (back) into heap
DC	IO.QSET	GU	Set up a queue
DE	IO.QTEST	GU	Test queue status
E0	IO.QIN	GU	Put a byte into a queue
E2	IO.QOUT	GU	Extract a byte from a queue
E4	IO.QEOF	GU	Put EOF marker into queue
122	IO.NAME	GU	Decode a device name

Basic Utility routines

The Basic utility routines are supplemented by Basic procedure support routines and the arithmetic package. Any code may use these routines, but should ensure that all addresses passed to the routines are relative to A6.

Vector	Name	Type	Description
E6	UT.CSTR	BU	Compare two strings
EC	CN.DATE	BU	Get date and time
EE	CN.DAY	BU	Get day of week
F0	CN.FTOD	BU	Convert floating point to ASCII
F2	CN.ITOD	BU	Convert an integer to ASCII
F4	CN.ITOBB	BU	Convert a binary byte to ASCII
F6	CN.ITOBW	BU	Convert a binary word to ASCII
F8	CN.ITOBL	BU	Convert binary long word to ASCII
FA	CN.ITOHB	BU	Convert a hex byte to ASCII
FC	CN.ITOHW	BU	Convert a hex word to ASCII
FE	CN.ITOHL	BU	Convert a hex long word to ASCII
100	CN.DTOF	BU	Convert ASCII to floating point
102	CN.DTOI	BU	Convert ASCII to an integer
104	CN.BTOIB	BU	Convert ASCII to a binary byte
106	CN.BTOIW	BU	Convert ASCII to a binary word
108	CN.BTOIL	BU	Convert ASCII to binary long word
10A	CN.HTOIB	BU	Convert ASCII to a hex byte

10C	CN.HTOIW	BU	Convert ASCII to a hex word
10E	CN.HTOIL	BU	Convert ASCII to a hex long word
110	BP.INIT	BP	Basic procedure initialisation
112	CA.GTINT	BP	Get integers (word)
114	CA.GTFP	BP	Get floating points (6 bytes)
116	CA.GTSTR	BP	Get strings
118	CA.GTLIN	BP	Get long integers (long word)
11A	BV.CHRIX	BP	Reserve space on arithmetic stack
11C	RI.EXEC	AR	Executes an operation
11E	RI.EXECB	AR	Executes a list of operations
120	BP.LET	BP	Return parameter values

Microdrive support routines

The microdrive support routines are vectored to simplify the writing of utilities which need to access particular sectors (eg. file recovery programs).

Vector	Name	Type	Description
124	MD.READ	MD	Read a sector
126	MD.WRITE	MD	Write a sector
128	MD.VERIN	MD	Verify a sector
12A	MD.SECTR	MD	Read a sector header

8.2 Supervisor mode routines

Supervisor mode routines are normally accessed by trap#1 calls. However, under certain circumstances it is useful to be able to call these supervisor mode routines from code which is not resident within QDOS. For example, device drivers which are accessed through trap#2 may need to use supervisor mode routines. Those described here deal with common heap management and simple I/O handling.

Calling code must conform to the following specification:

1. The calling code must be in supervisor mode before making the call.

2. The calling code must not have been initiated by an interrupt.

3. A6 must point to the base of the system variables.

MM.ALCHP Vector C0

Allocate common heap area

Call parameters		Return parameters	
D1.L	space required	D1.L	space allocated
D2		D2	undefined
D3		D3	undefined
A0		A0	base of area allocated
A1		A1	undefined
A2		A2	undefined
A3		A3	undefined

Error returns:

OM out of memory

Description:

This routine will allow common heap area to be allocated. The allocated space must be sufficient to allow for the heap entry header to be added (see Appendix R). This extra space is 16 bytes for simple heap entries or 24 bytes for IOSS channels. Note that A0 returns the address of the base of the heap area and not the base of the area which can be used (cf. trap #1 with D0=18 or 19 is different). The area allocated in the heap is filled with zeros. For a description of *heaps* see section 5.2.4 and 5.2.5.

MM.RECHP Vector C2

Releases common heap area

Call parameters		Return parameters	
D1		D1	undefined
D2		D2	undefined
D3		D3	undefined
A0	base of area to release	A0	undefined
A1		A1	undefined
A2		A2	undefined
A3		A3	undefined

Error returns:

none

Description:

This routine will release an area in the common heap which was previously allocated.

IO.SERQ Vector E8

Serial IO direct queue handler

Call parameters		Return parameters	
D0	routine number	D0	error return
D1	standard IOSS value	D1	standard IOSS value
D2	standard IOSS value	D2	standard IOSS value
D3.W	timeout	D3	preserved
A0	channel ID	A0	preserved
A1	standard IOSS value	A1	standard IOSS value
A2		A2	undefined
A3		A3	undefined

Error returns:

BP undefined action

Description:

This is a direct queue handling routine. In order to use this call, it is necessary to ensure that long words 7 and 8 in the channel definition block (see below) are set to point to the queue for input and output respectively. Input can be inhibited by setting long word 7 to zero and output can be inhibited by setting long word 8 to zero.

The particular type of operation performed is defined by the contents of D0, as for trap #3. The valid values of D0 which can be catered for are $00, $01, $02, $03, $05, $07, $46, $47, $48 and $49. The standard IOSS (input/output sub-system) values refer to the values which would be passed to the associated trap #3 routines.

For undefined actions IO.SERQ will return ERR.BP (if an invalid trap #3 is requested).

Channel definition block

A channel definition block is set up every time that a new channel is opened. All relevant information about the channel is held within this block, such as the device driver which is to be used, the owner Job, the channel number, and the channel status.

CH.LEN	$00 long	length of definition block
CH.DRIVR	$04 long	address of driver
CH.OWNER	$08 long	owner Job
CH.RFLAG	$0C long	address to be sent when space released
CH.TAG	$10 word	channel ID
CH.STAT	$12 byte	status: 0 = OK −1 = A1 absolute $80 = A1 relative to A6 negative = waiting
CH.ACTN	$13 byte	stored action for waiting Job
CH.JOBWT	$14 long	ID of Job waiting on IO
*		
* extended channel definition for serial queue handlers		
*		
CH.QIN	$18 long	pointer to input queue (or zero)
CH.QOUT	$1C long	pointer to output queue (or zero)

IO.SERIO Vector EA

General serial IO handling

Call parameters		Return parameters	
D0	routine number	D0	error return
D1	standard IOSS value	D1	standard IOSS value
D2	standard IOSS value	D2	standard IOSS value
D3.W	timeout	D3	undefined
A0	channel ID	A0	preserved
A1	standard IOSS value	A1	standard IOSS value
A2		A2	undefined
A3		A3	undefined

Error returns:

BP undefined action

Description:

This routine is designed to be used where the simple IO routine IO.SERQ is not sufficient.

As with IO.SERQ, the particular type of operation performed is defined by the contents of D0, as for trap #3. The valid values of D0 which can be catered for are $00, $01, $02, $03, $05, $07, $46, $47, $48 and $49. Other values will cause a return with D0=ERR.BP. The standard IOSS (input/output sub-system) values refer to the values which would be passed to the associated trap #3 routines.

IO.SERIO requires three addresses of routines to test for pending input, fetch bytes and send bytes to be supplied to it. The entry addresses to these routines should be placed in the three long words which follow on from the CALL instruction in the following order:

word1	test for pending input	(next byte returned in D1)
word2	fetch byte	(byte in D1)
word3	send byte	(byte in D1)

Using absolute addresses for these may be difficult to implement, so it is best to put the addresses of the routines in the physical definition block for the driver (which is pointed to by A3) when the device driver is initialised. There are several ways in which this could be implemented. A couple of suggestions are:

Example 1

code put at $28(A3) or similar position

```
Hex             Assembler mnemonics

387800E8        MOVE.W  $E8,A4
4E94            JSR     (A4)
                DC.L    TEST    Address of the test routine
                DC.L    FETCH   Address of the fetch routine
                DC.L    SEND    Address of the send routine
4E75            RTS
```

This code would be called by:

```
                JSR     $28(A3)
```

Example 2

code put at $28(A3) or a similar position

```
Hex             Assembler mnemonics

                DC.L    TEST
                DC.L    FETCH
                DC.L    SEND
4E75            RTS
```

This code would be called by:

```
                PEA     $28(A3)
                MOVE.W  $E8,A4
                JMP     (A4)
```

The three service routines should return D0 as the error code. The other registers D1 to D3 and A1 to A3 can be changed.

8.3 Simplified trap routines

These routines are used for very common operations. They must all be called in user mode.

UT.WINDW Vector C4

Sets up a window using a supplied name

Call parameters		**Return parameters**	
D1		D1	undefined
D2		D2	undefined
D3		D3	undefined
A0	pointer to name	A0	channel ID
A1	ptr to parameter block	A1	undefined
A2		A2	undefined
A3		A3	undefined

Error returns:

BN	bad device name
OM	out of memory
NO	out of channels
OR	window is off the screen

Description:

This routine opens a window using the supplied name. The call statement is followed by a block of 4 byte parameters. These define the colour of the border, the strip, the paper, and the ink. The border width is also defined.

The format of the parameter block is:

00	byte	border colour
01	byte	border width
02	byte	paper/strip colour
03	byte	ink colour

UT.CON Vector C6

Set up a console window

Call parameters		**Return parameters**	
D1		D1	undefined
D2		D2	undefined
D3		D3	undefined
A0		A0	channel ID
A1	ptr to parameter block	A1	undefined
A2		A2	undefined
A3		A3	undefined

Error returns:

OM	out of memory
NO	out of channels
OR	window is off the screen

Description:

This routine opens a console window. The call statement is followed by a block of 4 byte parameters and 4 word parameters. These define the colour of the border, the strip, the paper, and the ink. The border width is also defined. Window size is provided in terms of origin (of the upper lefthand corner), the window width and the window height.

The format of the parameter block is:

00	byte	border colour
01	byte	border width
02	byte	paper/strip colour
03	byte	ink colour
04	word	window width
06	word	window height
08	word	X origin
0A	word	Y origin

UT.SCR Vector C8

Sets up a screen window

Call parameters		Return parameters	
D1		D1	undefined
D2		D2	undefined
D3		D3	undefined
A0		A0	channel ID
A1	ptr to parameter block	A1	undefined
A2		A2	undefined
A3		A3	undefined

Error returns:

OM	out of memory
NO	out of channels
OR	window is off the screen

Description:

This routine opens a screen window. The call statement is followed by a block of 4 byte parameters and 4 word parameters. These define the colour of the border, the strip, the paper, and the ink. The border width is also defined. Window size is provided in terms of origin (of the upper lefthand corner), the window width and the window height.

The format of the parameter block is:

00	byte	border colour
01	byte	border width
02	byte	paper/strip colour
03	byte	ink colour
04	word	window width
06	word	window height
08	word	X origin
0A	word	Y origin

UT.ERR0 Vector CA

Write an error message to channel 0

Call parameters		Return parameters	
D0.L	error code	D0.L	preserved
D1		D1	preserved
D2		D2	preserved
D3		D3	preserved
A0		A0	preserved
A1		A1	preserved
A2		A2	preserved
A3		A3	preserved

Error returns:

none

Description:

This routine should be called with the error number for which the message is to be printed out in D0. The error message will be printed on channel 0.

UT.ERR Vector CC

Write an error message to given channel

Call parameters		Return parameters	
D0.L	error code	D0.L	preserved
D1		D1	preserved
D2		D2	preserved
D3		D3	preserved
A0	channel ID	A0	preserved
A1		A1	preserved
A2		A2	preserved
A3		A3	preserved

Error returns:

none

Description:

This routine should be called with the error number for which the message is to be printed out in D0. The error message will be printed on the channel whose number is passed to the routine in A0.

UT.MINT Vector CE

Converts given integer to ASCII and sends it to a channel

Call parameters		Return parameters	
D1.W	integer value	D1	undefined
D2		D2	undefined
D3		D3	−1 (unless A0=0, when D3=0)
A0	channel ID	A0	preserved
A1		A1	undefined
A2		A2	undefined
A3		A3	undefined

Error returns:

NC	not complete (only for channel 0)
NO	channel not open
DF	drive full

Description:

This routine converts the integer which is passed in D1 to ASCII. The converted ASCII string is then sent to the channel whose ID is given in A0. A space is added at the end of the ASCII string.

Interrupt servers, supervisor mode routines and UT.ERR0 can call this routine with A0=0. If the command channel is in use, the routine will try to use channel 1. This operation is not recommended by Sinclair, but it does seem to work.

UT.MTEXT Vector D0

Sends a message to a channel

Call parameters		Return parameters	
D1		D1	undefined
D2		D2	undefined
D3		D3	−1 (unless A0=0, when D3=0)
A0	channel ID	A0	preserved
A1	base of message	A1	undefined
A2		A2	undefined
A3		A3	undefined

Error returns:

NC	not complete (only for channel 0)
NO	channel not open
DF	drive full

Description:

This routine sends a message to a channel. The message is in the form of a text string pointed to by A1. The text string commences with a word containing the number of characters in the string and is followed by the ASCII characters of the message. If a new line is required at the end of the message, the user must include an <LF> in the message.

Interrupt servers and other supervisor mode routines can call this routine with A0=0. If the command channel is in use, the routine will try to use channel 1. This operation is not recommended by Sinclair, but it does seem to work.

8.4 General Utility routines

8.4.1 Linked list management

UT.LINK Vector D2

Link an item into a linked list

Call parameters		Return parameters	
D1		D1	preserved
D2		D2	preserved
D3		D3	preserved
A0	base of unlinked item	A0	preserved
A1	pointer to previous item	A1	updated
A2		A2	preserved
A3		A3	preserved

Error returns:

none

Description:

This routine will link an item into a linked list. Two parameters must be passed to the routine. The first is the address of the item to be linked in (passed in A0). The second is a pointer to an item in the list (passed in A1). If this pointer is to the pointer to the first item in the list, the new item will be linked in at the start of the list. Otherwise, it will be linked in after the item in the list to which it was pointing.

When a new list is being setup, the pointer to the first item in the list should be set to zero.

Note that four bytes must be reserved at the start of each item in the list for the link pointer.

See section 5.4.5 for a description of linked lists.

UT.UNLNK Vector D4

Unlink an item from a linked list

Call parameters		Return parameters	
D1		D1	preserved
D2		D2	preserved
D3		D3	preserved
A0	base of linked item	A0	preserved
A1	pointer to previous item	A1	updated
A2		A2	preserved
A3		A3	preserved

Error returns:

none

Description:

This routine will unlink an item from a linked list. Two parameters must be passed to the routine. The first is the address of the item to be unlinked from the list (passed in A0). The second is a pointer to an item in the list (passed in A1). The only requirement of this item is that it must be before the one which is to be removed from the list. It could of course be a pointer to the pointer to the linked list.

See section 5.4.5 for a description of linked lists.

8.4.2 User heap management

MM.ALLOC Vector D8

Allocates an area in a user heap

Call parameters		Return parameters	
D1.L	required length	D1.L	length allocated
D2		D2	undefined
D3		D3	undefined
A0	ptr to ptr to free space	A0	base of area allocated
A1		A1	undefined
A2		A2	undefined
A3		A3	undefined

Error returns:

OM no free space which is large enough

Description:

User heap allocation is carried out in units of eight bytes. Two long words are allocated per space in the heap. The first contains the length of the space and the second is the relative pointer to the next free space. Relative pointers are used to ensure that user heaps are relocatable. The whole of an allocated area can be used by the code provided that the length of the items in the space is recorded somewhere. When the area is allocated, the first long word in the space contains the length of the area, so it could be retained by the user code if required.

A pointer should be kept by the user code which points to the next area of free space. A0 is set to point to this pointer before calling this routine. If no free space is left in the heap then this pointer must be set to zero.

See section 5.2.5 for a discussion on user heaps. The manager trap MT.ALLOC is an *atomic* version of this utility routine.

MM.LNKFR Vector DA

Links a free space (back) into a user heap

Call parameters		Return parameters	
D1.L	length to be linked in	D1	undefined
D2		D2	undefined
D3		D3	undefined
A0	base of new space	A0	undefined
A1	ptr to ptr to free space	A1	undefined
A2		A2	undefined
A3		A3	undefined

Error returns:

none

Description:

This routine allows user heaps to be created, or free memory to be linked back into a heap. Heaps generated in this way can then have areas within them allocated using MM.ALLOC.

To set up a heap, an area of RAM should be linked into a non-existent heap.

To expand a heap, an area of RAM which is contiguous with the top of the heap should be linked into the heap. Alternatively, some memory in the Job's own workspace or elsewhere can be used. This will ultimately tend to be less efficient than using contiguous memory.

A pointer should be kept by the user code which points to the next area of free space. A1 is set to point to this pointer before calling this routine. If no free space is left in the heap then this pointer must be set to zero.

See section 5.2.5 for a discussion of user heaps. The manager trap MT.LNKFR is an *atomic* version of this routine.

8.4.3 Queue handling routines

IO.QSET Vector DC

Set up a queue

Call parameters		Return parameters	
D1	queue length	D1	undefined
D2		D2	preserved
D3		D3	preserved
A0		A0	preserved
A1		A1	preserved
A2	pointer to queue	A2	preserved
A3		A3	undefined

Error returns:

none

Description:

This routine will set up a queue. The required length of the queue should be passed in D1 and a pointer to the queue should be passed in A2.

A general description of I/O queues can be found in section 9.7.

IO.QTEST Vector DE

Test the status of a queue

Call parameters		Return parameters	
D1		D1	next byte
D2		D2	free space
D3		D3	preserved
A0		A0	preserved
A1		A1	preserved
A2	pointer to queue	A2	preserved
A3		A3	undefined

Error returns:

NC	queue is empty
EF	end of file has been reached

Description:

This routine will test the status of a queue. The free space in the queue is returned in D2 and the next byte to be taken out is in D1 (the byte is not actually removed by this routine). The routine returns ERR.NC if the queue is empty.

A general description of I/O queues can be found in section 9.7.

IO.QIN Vector E0

Put a byte into a queue

Call parameters		Return parameters	
D1	byte to be put in	D1	preserved
D2		D2	preserved
D3		D3	preserved
A0		A0	preserved
A1		A1	preserved
A2	pointer to queue	A2	preserved
A3		A3	undefined

Error returns:

NC	queue is full

Description:

This routine will put a byte into a queue. The byte to be put in should be passed in D1.

A general description of I/O queues can be found in section 9.7.

IO.QOUT Vector E2

Extract a byte from a queue

Call parameters		Return parameters	
D1		D1	next byte
D2		D2	preserved
D3		D3	preserved
A0		A0	preserved
A1		A1	preserved
A2	pointer to queue	A2	preserved
A3		A3	undefined

Error returns:

NC	the queue is empty
EF	end of file has been reached

Description:

This routine will remove a byte from a queue. The returned byte is in D1.

A general description of I/O queues can be found in section 9.7.

IO.QEOF Vector E4

Put an end of file marker into a queue

Call parameters		Return parameters	
D1		D1	preserved
D2		D2	preserved
D3		D3	preserved
A0		A0	preserved
A1		A1	preserved
A2	pointer to queue	A2	preserved
A3		A3	preserved

Error returns:

NC	queue is full

Description:

This routine will put an end of file marker into a queue.

A general description of I/O queues can be found in section 9.7.

8.4.4 IO Device name utility

IO.NAME Vector 122

Decodes a device name

Call parameters		Return parameters	
D1		D1	undefined
D2		D2	undefined
D3		D3	undefined
A0	pointer to name	A0	preserved
A1		A1	undefined
A2		A2	undefined
A3	pointer to parameters	A3	preserved

Error returns:

NF	not recognised
BN	name recognised, but bad parameters were given

Description:

This utility is a useful aid to help with the decoding of device names. It checks the device name and then evaluates any optional parameters.

ERR.NF returns directly after the call instruction. ERR.BN returns 2 bytes after the call and if there were no errors, the routine returns 4 bytes after the call. The device name description (in ASCII) commences 6 bytes after the call instruction.

8.5 Basic utility routines

Basic utilites can be called from any code, but all addresses passed to these utilities must be relative to A6.

8.5.1 Comparison of strings

UT.CSTR Vector E6

Compares two strings

Call parameters		Return parameters	
D0.B	comparison type	D0.L	−1, 0 or +1
D1		D1	preserved
D2		D2	preserved
D3		D3	preserved
A0	base of string 0 wrt A6	A0	preserved
A1	base of string 1 wrt A6	A1	preserved
A2		A2	preserved
A3		A3	preserved
A6	base address register	A6	preserved

Error returns:

none

Description:

This routine will compare two strings. The first string is pointed to by (A6,A0) and the second string is pointed to by (A6,A1). After the call, D0 contains one of three possible values. It is −1 if the first string is less than the second string, 0 if both strings are the same and +1 if the first is greater than the second.

In order for this explanation to make sense, it is necessary to have a very well defined idea of how strings are compared. This is really quite complex on the QL, because several different comparison options are available.

Comparisons

Two strings may be compared and sorted into one of the following categories:

Less than the strings are compared until a mismatch of characters (or string segment in the case of numbers) is found. The first string is less than the second string according to the defined order of the mismatched characters (see below).

Equal to all of the characters and/or numbers in the two strings are identical.

Greater than the strings are compared until a mismatch between them is found. The first string's value is greater than the second string's value according to the defined order of the mismatched characters (see below).

Order of comparison

From the point of view of comparing strings, it is necessary to be able to sort them in a directory type of order. This order is well defined and can be modified by the selection of one of four modes.

Space is defined as the first character

Punctuation is in standard ASCII order except for '.' which is last.

All punctuation is defined to come before letters or digits (eg. **A?** comes before **AA?**)

As an optional selection, embedded numbers can be compared in numerical order (eg. **Size6B** comes before **Size12B** and also **Size6.12** comes before **Size6.4**)

All digits or numbers are defined to come before all letters (eg. **car1** comes before **cart1**)

An upper case letter comes before the corresponding lower case letter, but after the previous lower case letter (eg. **Car** is before **car** but after **aid**)

As an optional selection, upper case letters can be treated as equivalent to lower case letters.

Summarised order

```
SPACE
!"#$%&'()*+,-/:;<=>?@[\]^_£{|}~©.
Digits or numbers
AaBbCcDdEeFfGgHhIiJjKkLlMmNnOoPpQqRrSsTtUuVvWwXxYyZz
```

Note that this order will have to be modified for foreign character sets.

Comparison types

Several comparison types can be selected, depending upon the t pe of application to which the string comparison is being applied.

Type 0 comparisons are made directly on a character by character basis. Letter case is used to distinguish strings.

Type 1 the same as type 0, but the letter case is ignored.

Type 2 the embedded number value is used to make comparisons rather than operating on a character by character basis. Upper and lower case letters are distinguished.

Type 3 as type 2, but upper and lower case letters are not distinguished.

File and variable name comparisons use type 1.

Basic <, <=, =, >=, > and <> use type 2.

Basic == (equivalence) operator uses type 3.

8.5.2 Conversion routines

CN.DATE Vector EC

Get ASCII string for date and time

Call parameters		Return parameters	
D1.L	date (internal value)	D1	preserved
D2		D2	preserved
D3		D3	preserved
A0		A0	preserved
A1	pointer to stack	A1	pointer to stack
A2		A2	preserved
A3		A3	preserved

Error returns:

none

Description:

This routine converts the time in long word format into an ASCII string for the date. This string takes the form:

yyyy mmm dd hh:mm:ss

where yyyy is the year, mmm is the month, dd is the day, hh is the hour, mm is the minute and ss is the second for the time. The string result is put on the A1 stack. Some 22 bytes are required on the stack for this.

See also CN.DAY which produces the day in a similar manner.

CN.DAY Vector EE

Get ASCII string for day of the week

Call parameters		Return parameters	
D1.L	date (internal value)	D1	preserved
D2		D2	preserved
D3		D3	preserved
A0		A0	preserved
A1	pointer to stack	A1	pointer to stack
A2		A2	preserved
A3		A3	preserved

Error returns:

none

Description:

This routine converts the time in long word format into an ASCII string for the day of the week. This string takes the form of a three letter day of the week. The string result is put on the A1 stack. Some 6 bytes are required on the stack for this.

See also CN.DATE which produces the date in a similar manner.

CN.FTOD Vector F0

Convert floating point number to ASCII string

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 floating point number on the stack (pointed to by A1) into a string of ASCII characters in a buffer (pointed to by A0).

CN.ITOD Vector F2

Convert an integer to an ASCII string in decimal

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 an integer number on the stack (pointed to by A1) into a string of ASCII characters in a buffer (pointed to by A0) forming a decimal number.

CN.ITOBB Vector F4

Convert a byte to an ASCII string of 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 byte on the stack (pointed to by A1) into a string of ASCII characters in a buffer (pointed to by A0) forming a binary number of 8 characters. For example, if the byte $C5 is converted, it will produced the characters '11000101'.

CN.ITOBW Vector F6

Convert a 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 word on the stack (pointed to by A1) into a string of ASCII characters in a buffer (pointed to by A0) forming a 16 character binary number.