MT.DMODE TRAP#1 D0=10

Sets or reads the display mode

Call parameters		**Return parameters**	
D1.B	−1 read mode 0 mode is 4 colour 8 mode is 8 colour	D1.B	display mode
D2.B	−1 read display 0 monitor 1 625 line TV	D2.B	display type
D3		D3	preserved
A0		A0	preserved
A1		A1	preserved
A2		A2	preserved
A3		A3	undefined

Error returns:

none

Description:

This trap sets or reads the current display mode. It is usually selected at boot up to tell the QL whether a monitor or television is being used as the display medium. All of the displayed windows are affected by this trap. It is unwise to use this trap to change the displayed mode unless the machine is completely idle, as some programs may not have been written to cope with the display mode being changed unilaterally. Using this trap to read the mode is always safe.

5.4.4 Communicating with the IPC

The IPC (intelligent peripheral controller) is a single chip 8049 microprocessor. All data from the keyboard and sound to the sound channel is dealt with by this *second processor*. The 68008 communicates with the 8049 in nibbles (4 bits).

Commands are sent to the IPC by a command nibble. This is then followed by the command parameters, which are a stream of nibbles and/or bytes. Any information returned from the IPC (such as a read row of the keyboard) is in byte format.

The command to be sent to the IPC (using MT.IPCOM) must be stored in memory in the following format. It consists of a header which describes the command, followed by the required number of parameters for that command, terminated by a byte to indicate whether a reply is expected.

1 byte	the IPC command nibble is in the 4 LSBs
1 byte	contains the number of parameter bytes to follow
1 long	describes the number of bits to send from each parameter byte bits 1, 0 = amount of first byte to be sent bits 3, 2 = amount of second byte to be sent etc.
n bytes	the parameter bytes
1 byte	the length of reply encoded in bits 1, 0

Either 0, 4 or 8 bits can be transferred from each parameter byte to the IPC. The number of bits to be transferred are encoded in the long word (see command format above). The encoded 2 bits should be as follows:

00	send least significant 4 bits of parameter byte
01	send nothing
10	send all 8 bits of parameter byte
11	send nothing

Note that IPC communication is **completely unprotected**, so any commands sent to it must **not** contain errors. If errors are present, the entire QL will crash. IPC communication is a very slow

process. It is therefore essential that excessive use is not made of the IPC. For example, it is more efficient just to read the cursor keys from the keyboard (they are all in one row) than it is to read the entire keyboard. If the entire keyboard was read regularly, it would be very expensive on processor overheads.

The majority of IPC commands are designed for use by the operating system, and no attempt should be made to use these from user programs. Such an attempt would probably result in data loss or worse. However, there are three commands which have been designed for use by applications programs. These are:

command 9 — read a keyboard row (one parameter required)
4 bits for the row number
8 bits for the reply
(see fig. 5.3 below for keyboard layout)

command A — initiate sound generation (8 parameters)
8 bits for pitch 1
8 bits for pitch 2
16 bit interval between steps
16 bit sound duration
4 bits step in pitch
4 bits wrap
4 bits randomness of step
4 bits fuzziness
no reply
(see QL User Guide for description of sound parameters)

command B — kill sound – no parameters – no reply

ROW \ COLUMN BIT	7	6	5	4	3	2	1	0
0	7	4	F5	F3	F2	5	F1	F4
1	↓	SPACE	\	→	ESC	↑	←	ENTER
2	¼ £	M	£	B	C	·	Z	]
3	;	G	=	F	S	K	CAPS LOCK	[
4	J	D	P	A	1	M	3	L
5	0	Y	#	R	TAB	I	W	9
6	U	T	Ø	E	Q	6	2	8
7	,	N	/	V	X	ALT	CTRL	SHIFT

Figure 5.3 – the keyboard layout

MT.IPCOM TRAP#1 D0=11

Sends a command to the IPC

Call parameters		Return parameters	
D1		D1.B	return parameter
D2		D2	preserved
D3		D3	preserved
D5		D5	undefined
D7		D7	undefined
A0		A0	preserved
A1		A1	preserved
A2		A2	preserved
A3	pointer to command	A3	preserved

Error returns:

none

Description:

This trap sends a command to the IPC. The command format must conform exactly to that described on the previous page, because no parameter checking is carried out. Failure to comply with this could be disastrous, leading to a total system hang up.

MT.BAUD TRAP#1 D0=12

Sets the baud rate

Call parameters		Return parameters	
D1.W	baud rate	D1	undefined
D2		D2	preserved
D3		D3	preserved
A0		A0	preserved
A1		A1	preserved
A2		A2	preserved
A3		A3	preserved

Error returns:

none

Description:

The two serial ports on the QL operate at the same baud rate in both transmit and receive modes. MT.BAUD is used to select the required baud rate. The following baud rates are valid:

75	transmit and receive
300	transmit and receive
600	transmit and receive
1200	transmit and receive
2400	transmit and receive
4800	transmit and receive
9600	transmit; receive (but only with more than 1 stop bit)
19200	transmit only

Note that parity and number of stop bits are set independently to the baud rate. For more details, see chapter 6 (IO.OPEN and IO.CLOSE).

MT.RCLCK TRAP#1 D0=13

Reads the clock

Call parameters		Return parameters	
D1		D1.L	time in seconds
D2		D2	undefined
D3		D3	preserved
A0		A0	undefined
A1		A1	preserved
A2		A2	preserved
A3		A3	preserved

Error returns:

none

Description:

The real time clock can be read using this trap. The time is returned as a long word, being the number of seconds since 00:00 1 January 1961. This means that the QL can cope with any time up until 2029! Note that Vectored Utilities CN.DATE and CN.DAY can be used to convert this time in seconds into an ASCII string containing the date and/or day.

MT.SCLCK TRAP#1 D0=14

Sets the clock

Call parameters		Return parameters	
D1.L	time in seconds	D1.L	time in seconds
D2		D2	undefined
D3		D3	undefined
A0		A0	undefined
A1		A1	preserved
A2		A2	preserved
A3		A3	preserved

Error returns:

none

Description:

This trap allows the time in the real time clock to be set.

MT.ACLCK TRAP#1 D0=15

Adjusts the clock

Call parameters		Return parameters	
D1.L	adjustment in seconds	D1.L	time in seconds
D2		D2	undefined
D3		D3	undefined
A0		A0	undefined
A1		A1	preserved
A2		A2	preserved
A3		A3	preserved

Error returns:

none

Description:

The clock can be adjusted using this trap. Note that setting the clock takes a significant time, so if MT.ACLCK is called with D1=0 QDOS will not even attempt to adjust the time.

MT.ALBAS TRAP#1 D0=16

Allocate Basic program area

Call parameters		Return parameters	
D1.L	no. of bytes required	D1.L	number of bytes allocated
D2		D2	undefined
D3		D3	undefined
A0		A0	undefined
A1		A1	undefined
A2		A2	undefined
A3		A3	undefined
A6	base address	A6	new base address
A7	user stack pointer	A7	new stack pointer

Error returns:

OM out of memory

Description:

MT.ALBAS will allocate more memory for use by the Basic command interpreter. The memory may be required either for additional program lines or for data storage. The command interpreter is a very special Job, because it is the only one which is allowed to expand its memory in this way. The command interpreter executes in *user mode* as Job 0.

MT.REBAS TRAP#1 D0=17

Release Basic program area

Call parameters		Return parameters	
D1.L	no. of bytes to release	D1.L	number of bytes released
D2		D2	undefined
D3		D3	undefined
A0		A0	undefined
A1		A1	undefined
A2		A2	undefined
A3		A3	undefined
A6	base address	A6	new base address
A7	user stack pointer	A7	new stack pointer

Error returns:

none

Description:

MT.REBAS will release Basic program area for use by other Jobs. Space is only released when a `NEW` or `CLEAR` command is executed.

MT.ALCHP TRAP#1 D0=18

Allocate common heap area

Call parameters		Return parameters	
D1.L	no. of bytes required	D1.L	number of bytes allocated
D2.L	owner Job ID	D2	undefined
D3		D3	undefined
A0		A0	base address of area
A1		A1	undefined
A2		A2	undefined
A3		A3	undefined

Error returns:

OM out of memory
NJ Job does not exist

Description:

MT.ALCHP will allow a Job to allocate some space for itself in the common heap area. The allocated space does not have to be owned by the creating Job. When the Job which owns the space is removed, all of the space which had been allocated to that Job will be cleared and made available for use by an application.

MT.RECHP TRAP#1 D0=19

Release common heap area

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

Error returns:

none

Description:

MT.RECHP will release a common heap area which was allocated using MT.ALCHP.

5.4.5 Extending the operating system

It is possible to extend QDOS by adding in routines to service interrupts and device drivers. This is performed by linking new routines into the *linked lists* maintained by QDOS. The diagrams and explanation below show how this works.

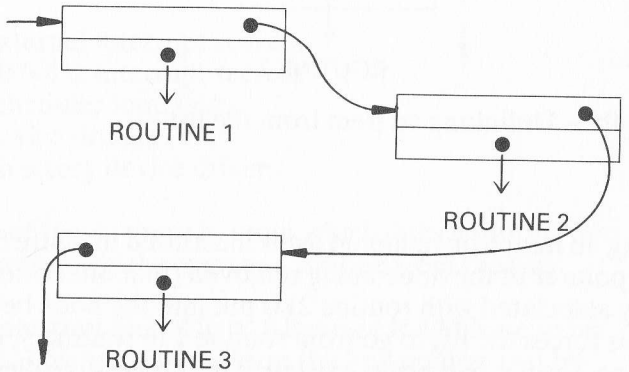

Figure 5.4a – Original linked list

An item in a linked list consists of a *link pointer* which contains the address of the next link in the list. Associated with each link pointer is an address (or series of addresses), which point to the service routines. A list is scanned by starting off with a link pointer. This has associated with it the address of the first service routine to be called (routine 1 in the diagram). When that routine returns, the OS uses the the old link pointer to find the new link pointer. The old pointer is scrapped. The new pointer has a routine address associated with it (routine 2). This routine is then called. Upon returning, the whole process repeats until the end of the list is found.

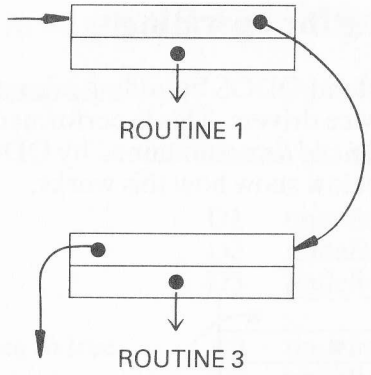

Figure 5.4b – Unlinking an item from the list

Removing an item from a linked list is illustrated in figure 5.4b. The link pointer of the node being removed (that one which was originally associated with routine 2) is put into the node before it. Doing this forces the list to go from routine 1 to routine 3, missing out routine 2 as if it had never existed. This process is called *unlinking* an item from the list.

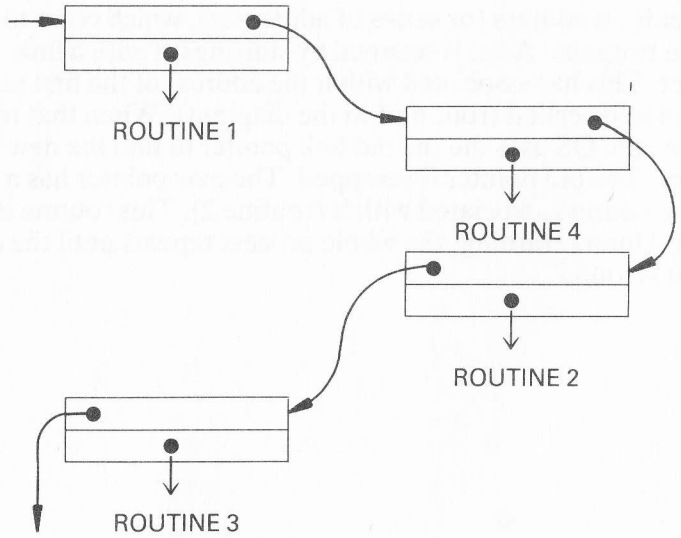

Figure 5.4c – Linking an item into a list

When a new item is to be linked into a list, the link pointer of the previous node must be put into the new node. The original node is then set to point at the new node. In this example, the old pointer from node 1 has been put into the new node (node 4). A new pointer is placed has been node 1 has been point at node 4. This ensures that the list remains complete and can be extended up to any size (within the memory constraints of the machine).

There are five linked lists in the QL. These are:

1. external interrupt servers
2. 50/60Hz interrupt servers
3. scheduler loop tasks
4. device drivers
5. directory device drivers

For each of these, there is a trap which links a routine into the list, and a trap which removes a routine from the list.

For the interrupt linked lists, 8 bytes of RAM have to be allocated. The first 4 bytes form the link pointer (set by MT.LXINT, MT.LPOLL and MT.LSCHD). The second 4 bytes form a long word which contains the address of the linked routine.

For the device drivers, 16 bytes of RAM have to be allocated. The first long word is the link pointer. The second, third and fourth long words point to the input/output, open and close routines respectively. For directory drivers, at least 40 bytes of RAM are required.

Before any links are made, it is essential that the RAM into which the links are to go should have been allocated by a Job. The allocated RAM should be in the resident procedure area if appropriate (device driver code in RAM), or alternatively in the common heap (device driver code in ROM). If the common heap is used, the space should be owned by Job 0. If it is owned by any other Job and that Job is force removed before it has unlinked the relevant routines from the list, the operating system is guaranteed to crash.

MT.LXINT TRAP#1 D0=1A

Links an external interrupt service routine into QDOS

Call parameters		Return parameters	
D1		D1	preserved
D2		D2	preserved
D3		D3	preserved
A0	address of link	A0	preserved
A1		A1	undefined
A2		A2	preserved
A3		A3	preserved

Error returns:

none

Description:

see section 5.4.5 for a description of linked lists

MT.RXINT TRAP#1 D0=1B

Removes an external interrupt service routine from QDOS

Call parameters		Return parameters	
D1		D1	preserved
D2		D2	preserved
D3		D3	preserved
A0	address of link	A0	preserved
A1		A1	undefined
A2		A2	preserved
A3		A3	preserved

Error returns:

none

Description:

see section 5.4.5 for a description of linked lists

MT.LPOLL TRAP#1 D0=1C

Links a polling 50/60Hz service routine into QDOS

Call parameters		Return parameters	
D1		D1	preserved
D2		D2	preserved
D3		D3	preserved
A0	address of link	A0	preserved
A1		A1	undefined
A2		A2	preserved
A3		A3	preserved

Error returns:

none

Description:

see section 5.4.5 for a description of linked lists

MT.RPOLL TRAP#1 D0=1D

Removes a polling 50/60Hz service routine from QDOS

Call parameters		Return parameters	
D1		D1	preserved
D2		D2	preserved
D3		D3	preserved
A0	address of link	A0	preserved
A1		A1	undefined
A2		A2	preserved
A3		A3	preserved

Error returns:

none

Description:

see section 5.4.5 for a description of linked lists

MT.LSCHD TRAP#1 D0=1E

Links a scheduler loop task into QDOS

Call parameters		Return parameters	
D1		D1	preserved
D2		D2	preserved
D3		D3	preserved
A0	address of link	A0	preserved
A1		A1	undefined
A2		A2	preserved
A3		A3	preserved

Error returns:

none

Description:

see section 5.4.5 for a description of linked lists

MT.RSCHD TRAP#1 D0=1F

Removes a scheduler loop task from QDOS

Call parameters		Return parameters	
D1		D1	preserved
D2		D2	preserved
D3		D3	preserved
A0	address of link	A0	preserved
A1		A1	undefined
A2		A2	preserved
A3		A3	preserved

Error returns:

none

Description:

see section 5.4.5 for a description of linked lists

MT.LIOD TRAP#1 D0=20

Links an IO device driver into QDOS

Call parameters		Return parameters	
D1		D1	preserved
D2		D2	preserved
D3		D3	preserved
A0	address of link	A0	preserved
A1		A1	undefined
A2		A2	preserved
A3		A3	preserved

Error returns:

none

Description:

see section 5.4.5 for a description of linked lists

MT.RIOD TRAP#1 D0=21

Removes an IO device driver from QDOS

Call parameters		Return parameters	
D1		D1	preserved
D2		D2	preserved
D3		D3	preserved
A0	address of link	A0	preserved
A1		A1	undefined
A2		A2	preserved
A3		A3	preserved

Error returns:

none

Description:

see section 5.4.5 for a description of linked lists

MT.LDD TRAP#1 D0=22

Links a directory device driver into QDOS

Call parameters		Return parameters	
D1		D1	preserved
D2		D2	preserved
D3		D3	preserved
A0	address of link	A0	preserved
A1		A1	undefined
A2		A2	preserved
A3		A3	preserved

Error returns:

none

Description:

see section 5.4.5 for a description of linked lists

MT.RDD TRAP#1 D0=23

Removes a directory device driver from QDOS

Call parameters		Return parameters	
D1		D1	preserved
D2		D2	preserved
D3		D3	preserved
A0	address of link	A0	preserved
A1		A1	undefined
A2		A2	preserved
A3		A3	preserved

Error returns:

none

Description:

see section 5.4.5 for a description of linked lists

6 Input/Output Allocation

6.1 Introduction to I/O

This chapter deals with the allocation of input and output resources on the QL. The input/output resources cover every aspect of communicating with the computer; keyboard, serial ports, screen, microdrives etc. However, it is not necessary to learn a different set of rules for dealing with all of these different devices. The QL has been designed so that communication with all of these devices is carried out in a well structured manner.

Input and output on the QL can be thought of as writing data to and reading data from a *logical file*. This *logical file* is in a standard device independent form. Data which is being sent to a serial port for printing, or to a microdrive for storage is all treated in a similar way. The specific details of interfacing to any particular device are left up to the *device drivers*. Device drivers have been specially written for each of the devices on the QL. The system will automatically find any device drivers which are in the system. These are all in a special device driver format (see section 9.2). Details of input and output are explained in greater detail in chapter 7.

I/O allocation can be divided into one of two general categories. That of opening a file so that data can be input from it and output to it, and that of closing a file. This section explains how to open and close files. It is necessary to know how to specify a channel and all of the information about that channel before it can be opened. Section 6.2 therefore explains the way in which device names should be specified.

The following IO Allocation traps are available:

D0	Name	Description
01	IO.OPEN	Opens a channel
02	IO.CLOSE	Closes a channel
03	IO.FORMT	Formats a sectored medium
04	IO.DELET	Deletes a file