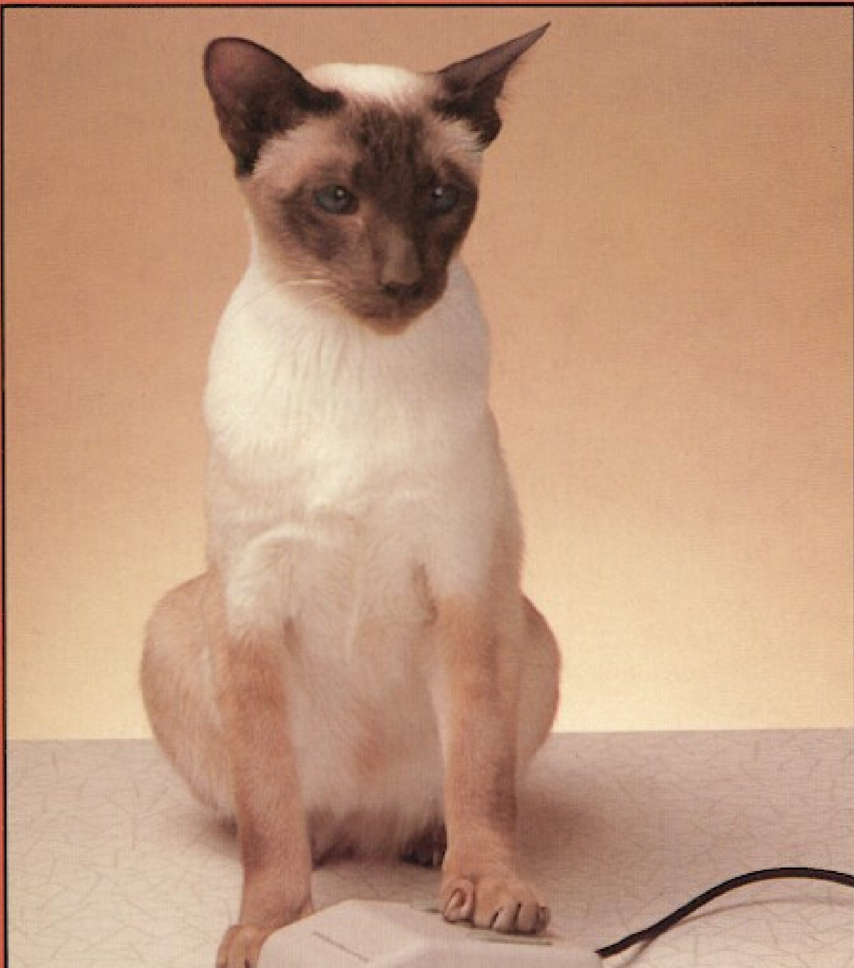
THE COMPLETE MOUSE USER GUIDE FOR THE BBC MICRO



THE COMPLETE MOUSE USER GUIDE FOR THE BBC MICRO

Published in the United Kingdom by
Watford Electronics
Jessa House, 250 Lower High Street
Watford,WD1 2AN. England

Telephone 0923 37774
Telex 8956095
Fax 01 950 8989

This book was computer typeset by
Ideal Software Consultants, 11 Hathaway Close, Luton, Bedfordshire.

CONTENTS

1 INTRODUCTION

The computer mouse is perhaps one of the most useful peripherals currently available. Its ability to facilitate rapid and accurate data entry into the micro makes it an invaluable addition to a host of drawing and CAD packages. It is far faster than calculating coordinates for entry via the keyboard, and much more accurate than all but the most expensive light pens.

A mouse has recently become an integral part of many computer packages available. For older systems, a separate mouse must be purchased, and integrated into the existing hardware and software already present in the micro. When such an addition is required, many pre-written CAD programs are available for use with the mouse, but there is a distinct absence of information about the mouse itself, and how to use it for custom programs and applications.

This manual has been written to reveal the secrets of the mouse. It explains all the principles required by the hardware and associated software, and also example listings for inclusion into custom programs. The manual first details the basic principles of the mouse, and a simple program which uses these principles. This information should be adequate for most applications. However, it is possible to improve the performane of the mouse by expanding on the principles already used in the software. This is again fully explained, and an example program given.

It is possible to gain a full understanding of the mouse from this manual. For those not interested in exactly how the mouse functions, complete example programs are also included. These may be typed directly into the micro, without the need for any understanding of the hardware or software involved, enabling the mouse to be used for custom applications.

This manual is written specifically for the BBC Model 'B' Microcomputer and the AMX/QUEST mouse. It should be noted, however, that most mice are AMX/QUEST compatible, and the information in this manual is therefore still relevant to them. As all the basic principles for using a mouse are fully explained, the information in this manual is still applicable to other computers, and can be used in conjunction with them. It should also be possible for the programs listed to be modified for use with different machines.

2 BASIC PRINCIPLES

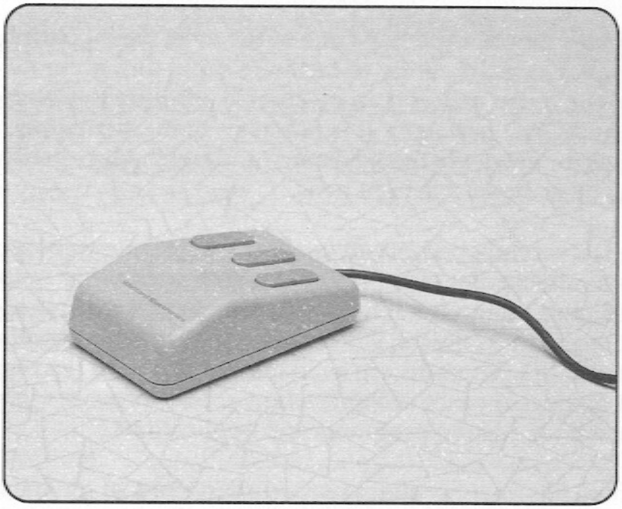
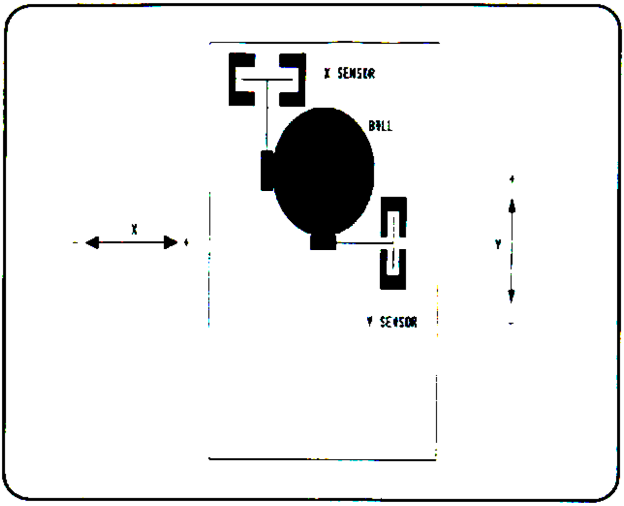


Figure 1
MOUSE

A typical mouse is shown in Fig 1. This consists of a small box, housing the mechanism and electronics, with three push buttons on top, and a cable with a plug on it for connection to the computer. In the case of the BBC Micro, the mouse connects to the User Port on the underside of the machine.

The mouse and associated software produce a pair of coordinates which represent the exact position of the mouse in two dimensions, X and Y. This method of using X and Y coordinates is similar to the method of addressing the graphics screens on the micro. The X coordinate represents the horizontal position of the mouse, and the Y coordinate represents its vertical position.

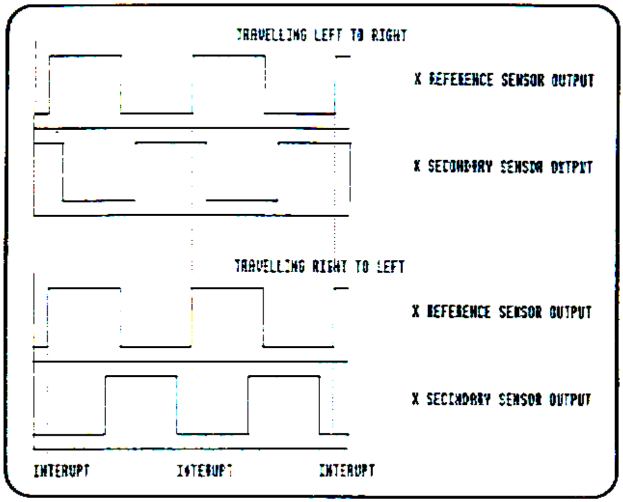

X SENSOR
BALL
X
Y
Y SENSOR

broken by a solid part of the disc, the detector output decreases to zero. In this way, the slots in the disc can be detected.

The distance between the slots, and the position of the IR emitters in respect to this distance, is arranged so that when one beam is fully broken, the other is only partially blocked. The effect of this, when the disc is rotating, is that one beam is broken slightly before or after the other, depending upon the direction of rotation of the disc. By using one beam as a reference, the direction of movement of the mouse can be determined by the state of the output from the other detector in the pair at the start of the pulse generated by the reference. The amount of travel, as opposed to the direction, is measured simply by counting the number of pulses received from one of the detectors in the sensor.

As the discs in the sensors contain many slots, and the mouse is moved fairly rapidly when in use, the frequency of pulses generated by the rotation sensors can be quite high. In order for the software to respond quickly enough, machine code software utilizing interupts is required.

The reference beams from the sensors, are used to generate the interupts. The output from the corresponding secondary beam then indicates the direction of travel of the mouse. Assuming that the interupt is generated on the positive edge of the reference sensor output:- (i) if the reference beam is broken before the secondary beam, the output from the secondary beam will be off at the instant the interupt is generated, (ii) if the mouse is travelling in the opposite direction and the secondary beam is broken before the reference, the reference detector will be on at the instant when the interupt is generated.


TRAVELLING LEFT TO RIGHT
X REFERENCE SENSOR OUTPUT
X SECONDARY SENSOR OUTPUT
TRAVELLING RIGHT TO LEFT
X REFERENCE SENSOR OUTPUT
X SECONDARY SENSOR OUTPUT
INTERUPT
INTERUPT
INTERUPT

3 HARDWARE

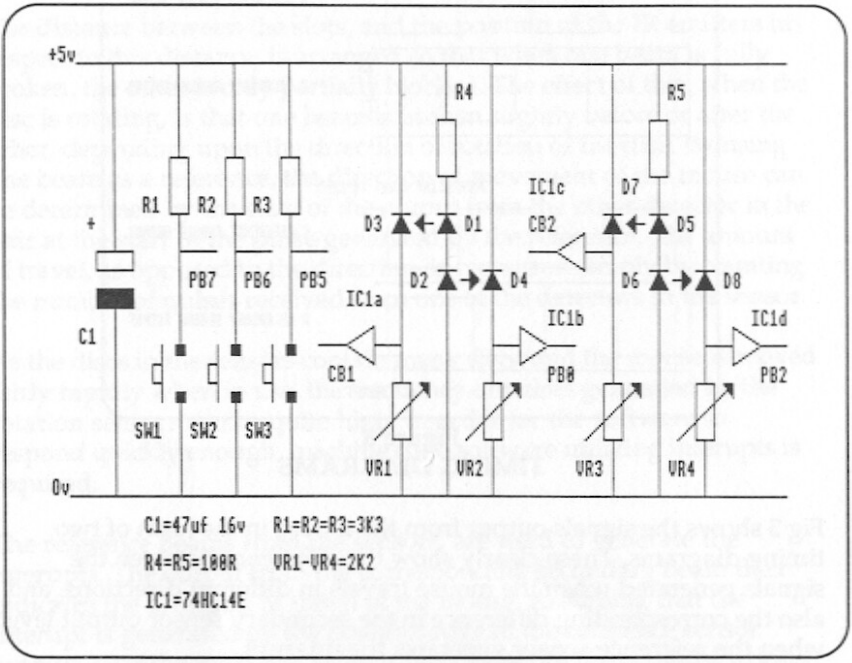


Figure 4
CIRCUIT DIAGRAM

The complete circuit diagram of the mouse is shown in Fig 4.

Push button switches SW1 - SW3 are connected to User Port lines PB7 - PB5 respectively. These lines are normally held high by pull-up resistors R1 - R3. When a switch is pressed, the corresponding input to the User Port is grounded, thus changing the input to the port from a logic '1' to a logic '0'.

The rotation sensors are in the form of two separate slotted opto switches, each having two IR emitter/detector pairs. The two IR emitter diodes of each pair, D1,D2,D5 and D6, are connected across the supply rails in series with the 100 ohm resistors R4 and R5. This limits the voltage across each IR emitter diode to approximately 1.7 volts to prevent damage occuring to the diodes.

Each of the four IR detector diodes, D3,D4,D5 and D6, is connected in series with one of 2.2K ohm preset resistors VR1 - VR4. These presets allow adjustment of the threshold of the output voltage levels from the detectors.

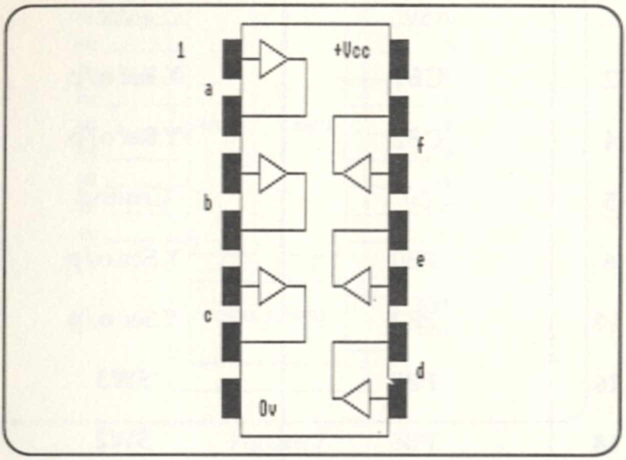


Figure 5
74HC14 PIN CONNECTIONS

The anodes of the IR detector diodes feed inputs to IC1. This a 74HC14 hex inverter, the pin connections for which are shown in Fig 5. A 'HC' device is used in anticipation of the high speeds required. The two unused inputs of this device are connected directly to 0 volts to prevent damage to the chip, and spikes on the supply. The four outputs which are used are connected to the User Port. The reference detector outputs connect to the interupt control lines CB1 and CB2. The secondary detector outputs connect to PB0 and PB2.

When the beam of IR light across a particular emitter/detector pair is blocked, by the corresponding disc, the resistance of the detector increases. The associated input to the inverter is pulled towards 0 volts by the comparatively low resistance of the preset. The inverter thus produces a logic '1' output to the User Port.

When light passes through a slot in the disc allowing it to fall on the IR detector, the detector conducts. This pulls the input to the inverter towards +Vcc due to the now lower resistance of the detector compared to the preset. The output from the inverter now assumes a logic '0' state.

Pin No.	User Port Function	Mouse Function
1	5v	+Vcc
2	CB1	X Ref o/p
4	CB2	Y Ref o/p
5	0v	Ground
6	PB0	X Sec o/p
10	PB2	Y Sec o/p
16	PB5	SW3
18	PB6	SW2
20	PB7	SW1

Figure 6
MOUSE/USER PORT PIN ASSIGNMENTS

The cable from the mouse is terminated in a 20 way IDC header socket. Fig 6 shows a table of the User Port and Mouse pin assignments.

The User Port

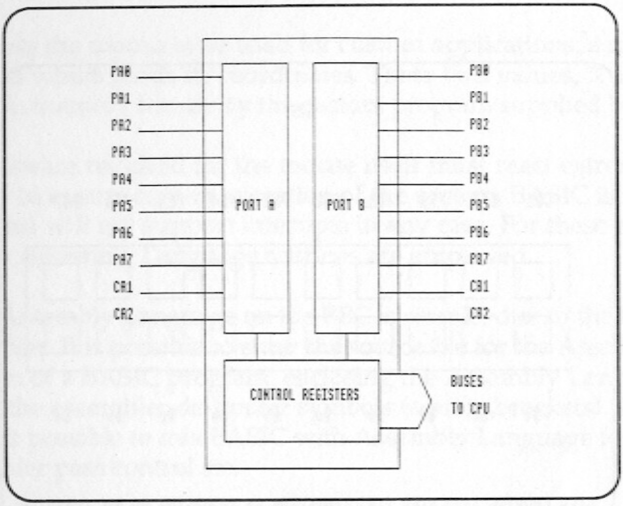


Figure 7
6522 VIA

The User Port of the BBC Micro consists of a 6522 VIA (Versatile Interface Adaptor) as shown in Fig 7. The 6522 has two almost identical ports, A and B, each with eight data lines and two interupt control lines. In the BBC Micro, port A is used for the parallel printer port, an port B is used for the User Port.

The eight data lines from ports A and B are labelled PA0 - PA7 and PB0 - PB7 respectively. Similarly, the interupt control lines are labelled CA1/CA2 and CB1/CB2.

The two ports are virtually identical, but completely independent. The chip contains sixteen internal registers, all programmable by the CPU to determine the exact function of the two sepatate ports. The method of programming these registers is described in detail in the chapter on software. Each of the data lines is independently programmable as either an input or an output. The mode of interupt control and acknowledgement is also determined and recorded using these registers.

The addresses of the user VIA are between Sheila &60 to &6F in the memory map of the micro. Again, further details are given in the next chapter.

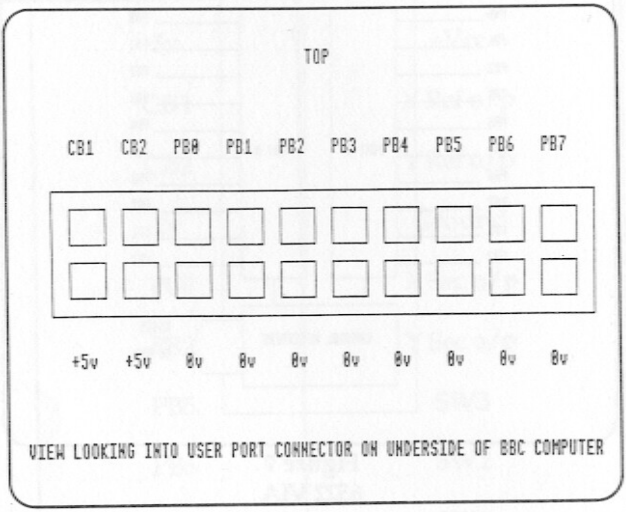


Figure 8
USER PORT PIN CONNECTIONS

The pin connections of the User Port plug are shown in Fig 8.

4 SOFTWARE

To enable the mouse to be used for custom applications, a routine is required which reads its coordinates. These two values, X and Y, are stored in memory for use by the custom program supplied by the user.

The software required for the mouse itself must react extremely quickly to ensure correct operation of the system. BASIC is far too slow, and will not support interupts in any case. For these two reasons Assembly Language routines are employed.

Using Assembly Language on the BBC is simple, due to the built-in Assembler. It is possible to enter the source file for the Assembler in the form of a BASIC program, enclosing the Assembly Language within the assembler de-limiter symbols (square brackets). This also makes it possible to mix BASIC with Assembly Language for Assembler pass control etc.

The programs to control the mouse are shown in Chapters 5 and 6 in Assembler. These programs show the assembly language code, accompanied by reference line numbers. Note that these programs are for demonstration purposes only, and are not complete. The actual program entered into the machine is shown in Chapter 7. This listing is in BASIC. When the program is 'RUN' the Assembler will place the machine code into memory, ready for execution when the mouse is moved and generates interupts.

The actual software required by the mouse is in two parts:-

1) The first part of the software, called the 'Initialization Routine', initialises the User Port and existing Operating System, ready for use with the mouse. Interaction with the OS is required because the mouse is not the only possible source of interupts. Many other devices also rely upon the interupt system. When an interupt is received, the OS polls each device which may have caused the interupt, to determine which device actually originated the call. If polling a device proves positive, the device is serviced by its associated Interupt Service Routine, and program control then reverts to the polling routine. If polling a device is negative, the next device is checked. When all the possible sources of interupts have been checked and serviced as required, normal foreground processing tasks resume.

The mouse software simply adds one more possible source of interupts to the existing ones. It achieves this by modifying a vector called 'IRQ2'. When an interupt is generated, the OS first checks all its own devices (eg RS423 port, keyboard, video processpr etc). It then allows the user to check for any interupts which have been generated by User add-on devices connected to the interupt system. This is done by means of the IRQ2 vector. When the OS has completed its own checks, program control is 'indirected' by the IRQ2 vector. This simply transfers program control to the address pointed to by the contents of the vector. The user program, starting at this address, may check any devices required (in our case the User VIA) and take any action necessary. When the user routine has been completed, program control must be passed back to the code at the location pointed to by the original contents of the IRQ2 vector. By doing this in a structured manner, provision is made for adding further devices to the interupt system for future expansion.

2) The second part of the mouse software called the 'Interupt Service Routine', which constitutes the main volume of the program, actually checks the User Port VIA to determine if the mouse is the source of the current interupt. If the mouse has caused the interupt, the routine takes the necessary action. If it wasn't the mouse that caused the interupt, program control is returned to the OS, or to the next interupt routine if more devices have been added to the system.

Reg. No.	Address	Name	Description
0	&FE60	DRB	Port B Data Reg.
2	&FE62	DDRB	Port B Data Direction Reg.
12	&FE6C	PCR	Peripheral Control Reg.
13	&FE6D	IFR	Interupt Flag Reg.
14	&FE6E	IER	Interupt Enable Reg.

Figure 9
6522 VIA REGISTERS

The Interupt Service Routine makes extensive use of the 6522 VIA. This chip contains 16 internal registers, accessible by the CPU, for its control. Only 5 of these registers are actually used by the mouse software. Fig 9 shows the relevant registers, addresses and functions. Full details of how to use these registers are given in the BBC Advanced User Guide, but the information needed to use the mouse is shown at the appropriate points in the explanation of the routines in the next two chapters.

5 INITIALIZATION PROGRAM

The initialization program integrates the mouse Interupt Service Routine into the existing Operating System interupt handling routine, and configures the User Port for use with the mouse. This program must only be executed once.

The initialization program modifies the contents of the IRQ2 vector to allow the interupts generated by the mouse to be recognised. If the mouse software needs to be reset at any time, the old IRQ2 vector must be restored first. If this is not done, the original value of the IRQ2 vector supplied by the OS will be lost. This will crash the machine, as it will not be able to return from certain interupts correctly. The easiest method of achieving a reset is by pressing the BREAK key. When this occurs, the OS restores the original value of the IRQ2 vector. The initialization program may then be executed again. If a reset is required under software control, a routine must be written which retrieves the original contents of IRQ2, and replaces it in the vector.

Another point to note about the initialization program is that it must not be called before the Interupt Service Routine has been installed. If this happens, the next interupt generated will crash the machine. The interupt will cause the OS to indirect to an address in memory where it expects to find the service routine. As the routine is not present, the code at this location will probably be 'rubbish' and lock the machine. This means program execution will not be transfered back to the OS re-entry point as usual, and the whole system will crash.

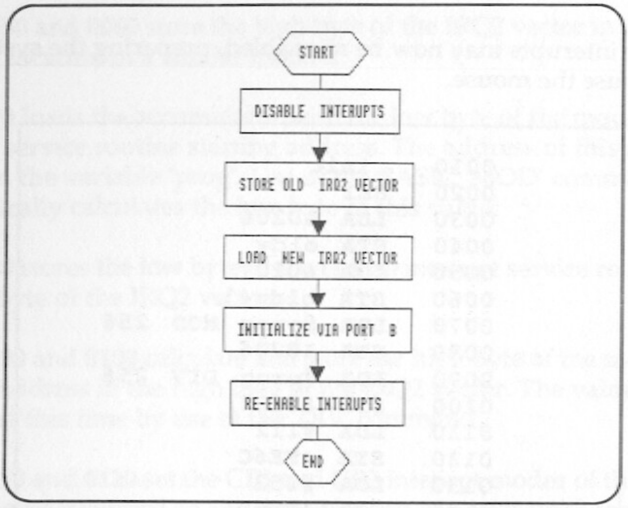


Figure 10
INITIALIZATION PROGRAM FLOWCHART

The flowchart for the initialization program is show in Fig 10.

The first operation this routine performs is to disable all maskable interupts. This is done because if an interupt was called at the instant when the initialization program was in the middle of altering the IRQ2 vector, the vector used would not be correct. The first byte of the vector would have been altered, but the second byte would still be from the initial vector contents. Disabling interupts will delay the interupt call until they are re-enabled. When this happens, the vector will have been modified correctly. Interupts may only be disabled for a maximum of 2mS if the Operating System is not to be disrupted. The initialization program only takes a fraction of this time and is therefore safe to use, as long as interupts are re-enabled afterwards.

Having temporarily disabled interupts, the IRQ2 vector can be modified. The original contents of this vector must be stored in memory for use later when program control is returned to the OS. The new value for the vector can now be loaded into the IRQ2 vector.

Now that the vector has been modified, the VIA must be initialised. This involves setting the required interupt modes, enabling the VIA to generate interupts and resetting any interupts previously flagged in the VIA.

Machine interupts may now be re-enabled, preparing the system ready to use the mouse.

```
0010   .init
0020   SEI
0030   LDA &0206
0040   STA oldv
0050   LDA &0207
0060   STA oldv+1
0070   LDA #prog MOD 256
0080   STA &0206
0090   LDA #prog DIV 256
0100   STA &0207
0110   LDA #112
0120   STA &FE6C
0130   LDA #152
0140   STA &FE6E
0150   LDA #127
0160   STA &FE6D
0170   LDA #0
0180   STA &FE62
0190   CLI
0200   RTS
```

Figure 11
INITIALIZATION PROGRAM LISTING

Fig 11 shows the assembly language listing for the initialization routine.

Line 0010 assigns a name to the routine.

Line 0020 disables maskable interupts by setting the interupt disable flag in the 6502 condition code register. This must be done as previously explained on Page 19.

Line 0030 loads the accumulator with the low byte of the IRQ2 vector.

Line 0040 stores the high byte of the IRQ vector in the location 'oldv'. This location is used to store the vector contents for use later by the Interupt Service Routine as a return address into the OS.

Lines 0050 and **0060** store the high byte of the IRQ2 vector in the next memory location in a similar manner.

Line 0070 loads the accumulator with the low byte of the mouse interupt service routine starting address. The address of this routine is held in the variable 'prog'. Use of the BASIC 'MOD' command automatically calculates the low byte of this value.

Line 0080 stores the low byte of the mouse interupt service routine in the low byte of the IRQ2 vector.

Lines 0090 and **0100** calculate and store the high byte of the mouse ISR starting address in the high byte of the IRQ2 vector. The value is calculated this time by use of the 'DIV' command.

Lines 0110 and **0120** set the CB1 and CB2 interupt modes of the 6522 VIA as being triggered on the positive edge of the interupt signal. CB1 is programmed as a normal interupt, but CB2 is configured as an Independent interupt (This affects the way in which the flags in the IFR are reset as described later). This is done by use of the Peripheral Control Register in the 6522. See Fig 3, page 7, for the actual waveforms of the signals expected on the interupt inputs.

Lines 0130 and **0140** enable the VIA to generate interupt signals to the 6502 CPU. This is done by use of the Interupt Enable Register in the VIA. This only enables the VIA to generate interupts and does not allow the CPU to recognise them. Note that line 0020 disabled the CPU from recognising interupts, not the 6522 from generating them.

Lines 0150 and **0160** reset the Interupt Flag Register in the VIA. When an interupt is generated, the corresponding flag is set in this register. These two lines simply reset the flags in case they are not already clear.

Lines 0170 and **0180** program all the Port B lines of the VIA as inputs to the micro. This is required because the mouse generates all the signals, and the micro reads them.

Line 0190 re-enables machine interupts by resetting the Interupt Disable Flag.

Line 0200 returns control back from the initialization routine.

6 INTERUPT SERVICE ROUTINE

When an interupt has been generated, the Operating System checks all the internal devices to determine if they caused it. The user can then check additional devices, such as the User Port, if this is required.

This system of checking is performed by the Interupt Service Routine (ISR). Having already enabled the User Port to generate interupts, as previously described, the OS indirects program control to the User Interupt Service Routine through the IRQ2 vector. This vector has been altered to point to the start of the User ISR by the previous initialization routine.

The User, or in this case mouse, ISR, must check to see if the current interupt has been caused by the User VIA and hence the mouse. If the mouse has caused the interupt, the ISR must then determine whether the CB1 or CB2 signal caused it and take appropriate action ie increment or decrement the X or Y variables accordingly. These values are stored in the two variables 'xcord' and 'ycord' respectively, by the ISR. These two variables are 16 bit or two byte. The low byte is stored first, followed by the high byte.

When the ISR is complete, program control must be returned to the code at the original address of the IRQ2 vector. The CPU must also enter this routine with the same data in all its registers as was present when the User ISR was called. This necessitates pushing the entire processor status onto the stack at the beginning of the ISR, and retrieving it upon completion of the routine. To return program control to the correct address, the program must 'jump' to the address held in the variable 'oldv'. This variable was loaded with the old contents of the IRQ2 vector at the beginning of the initialization routine.

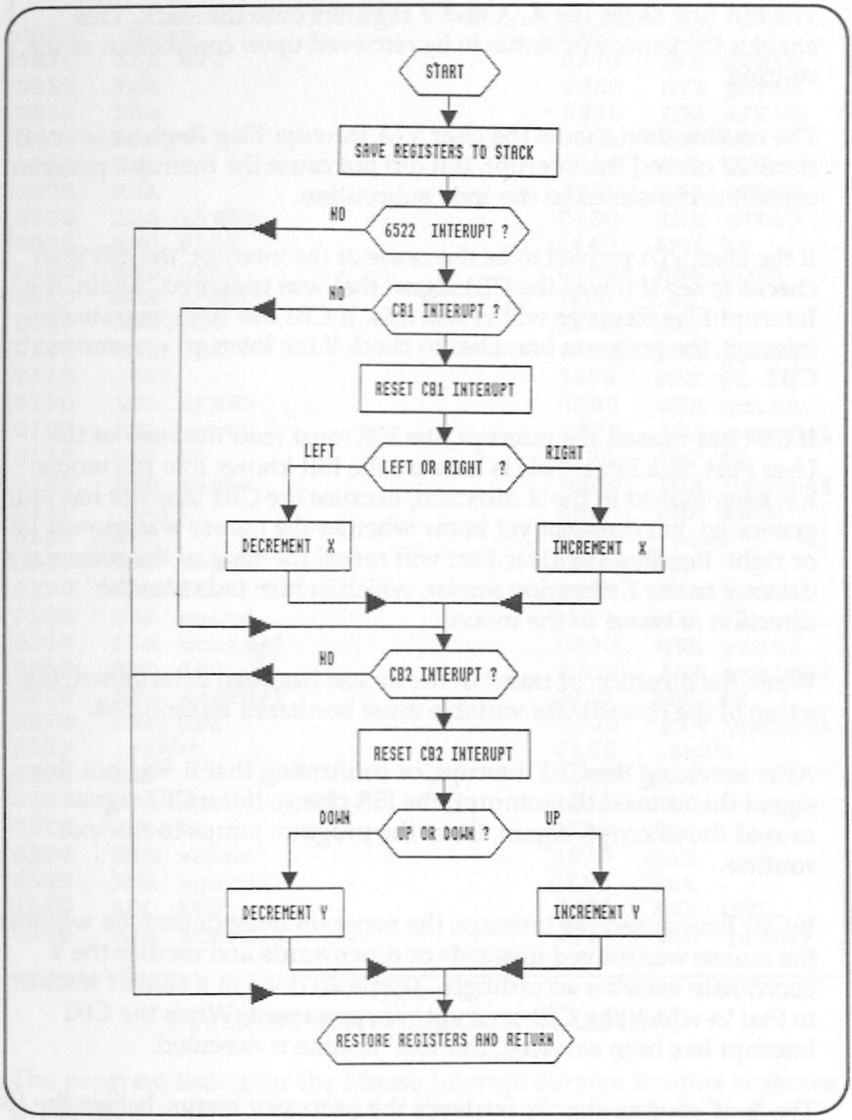


Figure 12
INTERUPT SERVICE ROUTINE FLOWCHART

The flowchart of the Interupt Service Routine is shown in Fig 12.

The ISR first saves the A, X and Y registers onto the stack. This enables the processor status to be retrieved upon completion of the routine.

The routine then checks the User VIA Interupt Flag Register to see if the 6522 caused the interupt. If it did not cause the interupt, program control is transfered to the 'exit' subroutine.

If the User VIA proved to be the cause of the interupt, the ISR then checks to see if it was the CB1 signal that was triggered. Again, the Interupt Flag Register will reveal this. If CB1 did not generate the interupt, the program branches to check if the interupt was caused by CB2.

If CB1 has caused the interupt, the ISR must read the state of the User Port data lines. This is because the ISR knows that the mouse has been moved in the X direction, because the CB1 interupt has been generated, but does not yet know whether the mouse was moved left or right. Reading the User Port will reveal the state of the secondary detector in the X direction sensor, which in turn indicates the direction of travel of the mouse.

When the direction of travel of the mouse has been determined, the value of the co-ordinate variable must be altered as required.

After servicing the CB1 interupt, or confirming that it was not this signal that caused the interupt, the ISR checks if the CB2 signal caused the interupt. Again, if not the program jumps to the 'exit' routine.

If CB2 has caused the interupt, the program must determine whether the mouse was moved upwards or downwards and modify the Y coordinate variable accordingly. This is all done in a similar manner to that in which the CB1 interupt was processed. When the CB2 interupt has been serviced, the 'exit' routine is executed.

The 'exit' routine simply retrieves the processor status, before the ISR was executed, from the stack, and transfers program control to the original code the IRQ2 vector pointed to.

```
0010    .prog
0020    LDA &FC
0030    PHA
0040    TXA
0050    PHA
0060    TYA
0070    PHA
0080    LDA &FE6D
0090    AND #128
0100    BEQ exit
0110    LDA &FE6D
0120    AND #16
0130    BEQ CB2
0140    .CB1
0150    LDA &FE60
0160    STA portb
0170    AND #1
0180    BNE right
0190    .left
0200    SEC
0210    LDA xcord
0220    SBC #1
0230    STA xcord
0240    LDA xcord+1
0250    SBC #00
0260    STA xcord+1
0270    JMP CB2
0280    .right
0290    CLC
0300    LDA xcord
0310    ADC #1
0320    STA xcord
0330    LDA xcord+1
0340    ADC #00
0350    STA xcord+1
0360    .CB2
0370    LDA &FE60
0380    STA portb
0390    LDA &FE6D
0400    AND #8
0410    BEQ exit
0420    STA &FE6D
0430    LDA &FE60
0440    AND #4
0450    BNE down
0460    .up
0470    SEC
0480    LDA ycord
0490    SBC #1
0500    STA ycord
0510    LDA ycord+1
0520    SBC #0
0530    STA ycord+1
0540    JMP exit
0550    .down
0560    CLC
0570    LDA ycord
0580    ADC #1
0590    STA ycord
0600    LDA ycord+1
0610    ADC #0
0620    STA ycord+1
0630    .exit
0640    PLA
0650    TAY
0660    PLA
0670    TAX
0680    PLA
0690    STA &FC
0700    JMP (oldv)
```

Figure 13
INTERUPT SERVICE ROUTINE LISTING

The program listing for the Mouse Interupt Servive Routine is shown in Fig 13.

Line 0010 names the routine as 'prog'. Later, in the BASIC listings, this name is important because it is from this variable that the initialization routine calculates the start address of the new ISR.

Lines 0020 to **0070** save the CPU registers on the stack. Upon entry to the ISR at vector IRQ2, the OS has already saved the contents of the accumulator in address &FC, and saved the Program Counter and Condition Code Register on the stack. Lines 0020 to 0070 load the values of the accumulator (from &00FC), X register and Y register into the accumulator and then push the accumulator on the stack.

Lines 0080 to **0100** check if the 6522 is flagging an interupt by examining bit 7 of the Interupt Flag Register. If this bit is not set, the 6522 has not generated the interupt, and the program will branch to the 'exit' routine.

Lines 0110 to **0130** check if the VIA CB1 input has generated the interupt. The program branches to the routine called 'CB2' if this is not the case.

Lines 0150 and **0160** store the data on the lines of the User Port in a variable called 'portb'. Due to the mode of interupt operation selected, this also has the effect of clearing the CB1 interupt flag in the VIA IFR.

Lines 0170 and **0180** determine if bit 0 of the User port was set when the interupt was generated. If it was, the mouse must have been travelling to the right when the interupt was generated, and the program branches to the routine called 'right'. If bit 0 was not set, the mouse must have been travelling left, and the program continues to execute the routine 'left'.

Lines 0200 to **0260** subtract one from the current value of the X coordinate.

Line 0270 makes the program jump past the 'right' routine, to the 'CB2' routine.

Lines 0280 to **0350** add one to the current value of the X coordinate.

Line 0360 is the start of the CB2 interupt routine.

Lines 0370 to **0410** check for the CB2 interupt and branch to exit if the appropriate flag is not set in the VIA IFR. These lines also copy the data on the User Port into the 'portb' variable.

Line 0420 resets the CB2 interupt flag in the IFR. The CB2 interupt has been programed to be independent, and therefore requires resetting, unlike the normal CB1 interupt which resets automatically when the data on the User Port is read from the VIA.

Lines 0430 to **0620** determine whether bit 2 of the User Port was set when the CB2 interupt was generated, decide from this the direction in which the mouse is travelling, and then either adds or subtracts one from the Y coordinate accordingly. This process is carried out in a similar manner to the CB1 routine, except this routine uses the CB2 interupt to determine whether the mouse was moved up or down.

Line 0630 is the start of the 'exit' routine. This pulls the registers from the stack in the reverse order to which they were pushed onto it earlier, and restores them.

Line 0700 returns program control back to the routine which the old IRQ2 vector pointed to.

7 IMPROVING THE MOUSE SOFTWARE

Although the description of the mouse and the software needed to drive it given so far are perfectly adequate for nearly all applications, it is possible to improve the software to obtain a higher resolution from the mouse.

Achieving higher resolution means that:- (i) the mouse generates more pulses when moved the same distance and (ii) the mouse generates the same number of pulses when moved a smaller distance. This effectively increases the sensitivity of the mouse ie how far it has to be moved to generate a pulse.

For most applications the simple software already described will be adequate for the mouse. Where a higher resolution is required, for example when using a very high resolution graphics screen, the following information may be used to increase the performance of the mouse.

Refering to the timing diagram on page 7, it can be seen that interupts are only generated by positive edge pulses from the reference sensor. This is the principle the simple mouse software uses.

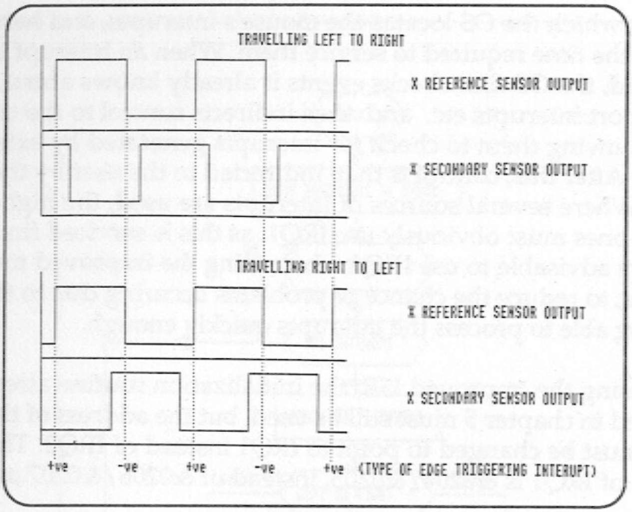


Figure 14
TIMING DIAGRAM

It is, however, possible to uses the negative edges of the pulses as well, the principle being to use positive and negative edges alternately to generate the interupts. The results of doing this are shown in Fig 14. The interupts generated on the positive edges are as before. With the mouse travelling left to right, the secondary sensor output is high when a positive edge triggered interupt is generated, or low when the mouse is travelling right to left.

The interupts generated by the negative edges operate upon exactly the same principle, except that the secondary sensor is at the opposite polarity to before. When using negative edge interupts, the secondary sensor output will be low when the mouse is travelling left to right, or high when travelling right to left.

It is apparent that using this method produces twice as many interupts to the previous routine. This results in the mouse being more sensitive in that it only needs to be moved half the distance to generate the same number of pulses as the previous program.

To use this principle to improve the mouse's performance, the IRQ1 vector must be used instead of IRQ2 as before. This is because the mouse generates interupts at a faster rate than before, which therefore require servicing more quickly. Using IRQ1 increases the speed at which the OS locates the mouse's interupts, and hence reduces the time required to service them. When an interupt is generated, the OS first checks events it already knows about, eg printer port interupts etc, and then indirects control to the user, by IRQ1, allowing them to check for interupts generated by external devices. After this, control is then indirected to the user by the IRQ2 vector. Where several sources of interupts are used, the higher priority ones must obviously use IRQ1, as this is serviced first. It is therefore advisable to use IRQ1 when using the improved mouse software, to reduce the chance of problems occuring due to the CPU not being able to process the interupts quickly enough.

When using the improved ISR, the initialization routine already described in chapter 5 must still be used, but the address of the IRQ vector must be changed to point to IRQ1 instead of IRQ2. The address of IRQ1 is &0204/&0205, instead of &0206/&0207 as for IRQ2.

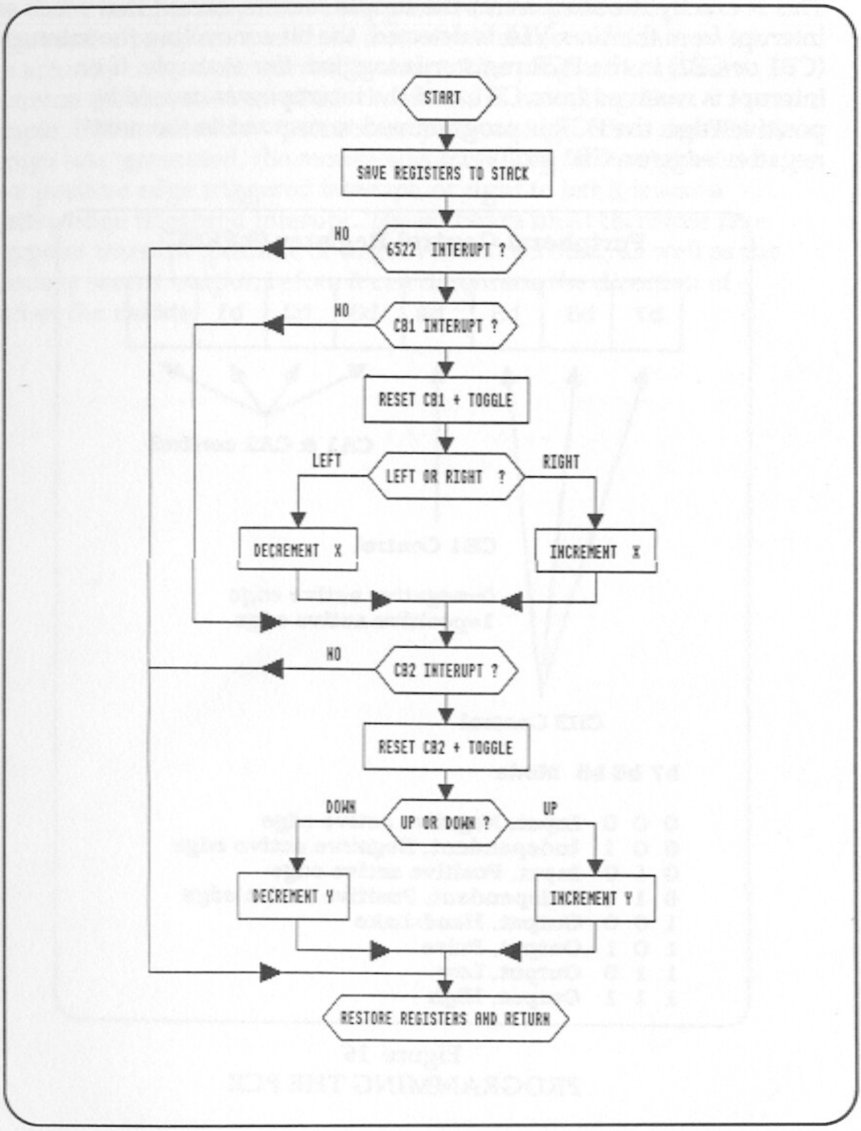
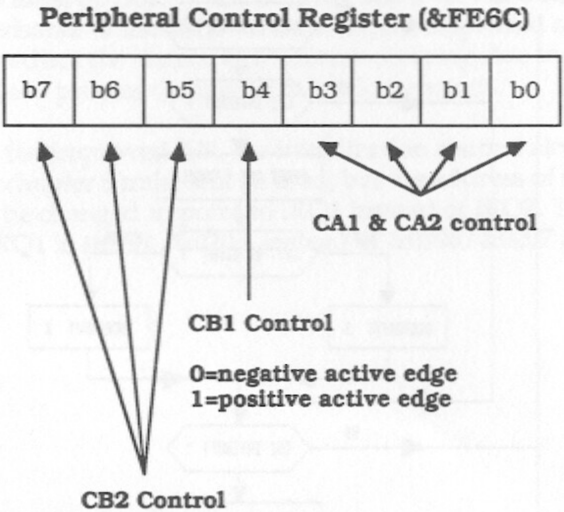


Figure 15
IMPROVED INTERUPT SERVICE ROUTINE FLOWCHART

The flowchart for the improved ISR is shown in Fig 15.

This is exactly the same as for the simple routine, except that when an interupt from the User VIA is detected, the bit controlling the interupt (CB1 or CB2) in the PCR register is toggled. For example, if an interupt is received from CB1, and the interupt was caused by a positive edge, the PCR is programmed to respond to the next negative edge on CB1 etc.



b7	b6	b5	Mode
0	0	0	Input, Negative active edge
0	0	1	Independent, Negative active edge
0	1	0	Input, Positive active edge
0	1	1	Independent, Positive active edge
1	0	0	Output, Handshake
1	0	1	Output, Pulse
1	1	0	Output, Low
1	1	1	Output, High

Figure 16
PROGRAMMING THE PCR

The method of determining whether the current interupt was positive or negative edge triggered is to interrogate the Peripheral Control Register (PCR) in the VIA, and note what type of edge the interupt was programmed to respond to when it was generated. Fig 16 shows how the PCR is programmed.

Now that interupts may have been positive or negative edge triggered, the program must take into consideration which type of edge actually caused the current interupt. The program may then determine the direction in which the mouse is travelling. For example, if the secondary X sensor output was high when the CB1 interupt was generated, the mouse was travelling left to right if it was a positive edge triggered interupt, or right to left if it was a negative edge triggered interupt. The software must therefore take the type of interupt ,positive or negative, into account, as well as the secondary sensor output, before it can determine the direction of travel of the mouse.

```
0010 .prog
0020 LDA &FC
0030 PHA
0040 TXA
0050 PHA
0060 TYA
0070 PHA
0080 LDA &FE6D
0090 AND #128
0100 BNE skip1
0110 JMP exit
0120 .skip1
0130 LDA &FE6D
0140 AND #16
0150 BEQ CB2
0160 .CB1
0170 LDA &FE6C
0180 EOR #16
0190 STA &FE6C
0200 AND #16
0210 BNE CB1N
0220 .CB1P
0230 LDA &FE60
0240 STA portb
0250 AND #1
0260 BNE right
0270 JMP left
0280 .CB1N
0290 LDA &FE60
0300 STA portb
0310 AND #1
0320 BNE left
0330 JMP right
0340 .left
0350 SEC
0360 LDA xcord
0370 SBC #1
0380 STA xcord
0390 LDA xcord+1
0400 SBC #00
0410 STA xcord+1
0420 JMP CB2
0430 .right
0440 CLC
0450 LDA xcord
0460 ADC #1
0470 STA xcord
0480 LDA xcord+1
0490 ADC #00
0500 STA xcord+1
0510 .CB2
0520 LDA &FE60
0530 STA portb
0540 LDA &FE&D
0550 AND #08
0560 BNE skip2
0570 JMP exit
0580 .skip2
0590 STA &FE6D
0600 LDA &FE6C
0610 EOR #64
0620 STA &FE6C
0630 AND #64
0640 BNE CB2N
0650 .CB2P
0660 LDA &FE60
0670 AND #4
0680 BNE down
0690 JMP up
0700 .CB2N
0710 LDA &FE60
0720 AND #4
0730 BNE up
0740 JMP down
0750 .up
0760 SEC
0770 LDA ycord
0780 SBC #01
0790 STA ycord
0800 LDA ycord+1
0810 SBC #00
0820 STA ycord+1
0830 JMP exit
0840 .down
0850 CLC
0860 LDA ycord
0870 ADC #01
0880 STA ycord
0890 LDA ycord+1
0900 ADC #00
0910 STA ycord+1
0920 .exit
0930 PLA
0940 TAY
0950 PLA
0960 TAX
0970 PLA
0980 STA &FC
0990 JMP (oldv)
```

Figure 17
IMPROVED ISR LISTING

The program listing for the improved ISR is shown in Fig 17.

Lines 0010 to **0070** push the processor's registers onto the stack, as before.

Lines 0080 to **0110** test to see if the User VIA is flagging an interupt. If it is, the program branches to 'skip1', the start of the actual mouse routine. If the User VIA is not flagging an interupt, control jumps to 'exit'. Note that the program cannot use a BEQ exit command, as the code is now too long to do this, ie it would require a branch of more than 127 bytes.

Lines 0120 to **0150** check the CB1 interupt flag, and branch to 'CB2' if it is not set.

Lines 0160 to **0190** program the Peripheral Control Register (PCR) of the VIA ready for the next interupt. This is done by exclusively OR'ing with 16 to toggle bit 4 of the PCR. Looking at Fig 16, it can be seen that b4 determines which edge of the interupt signal actually causes the interupt.

Lines 0200 and **0210** test if the CB1 is now set for a positive or negative edge. Remembering that the type of interupt has just been altered, the program branches to CB1N (negative edge routine) if the mode for the next interupt is positive. If the next interupt will be negative edge triggered, the program continues execution of CB1P.

Lines 0220 to **0270** are the same as the basic routine. They determine whether the mouse is moving left or right by reading b0 of the user port, and then jump to the appropriate routine to increment or decrement the x coordinate as appropriate.

Lines 0280 to **0330** work in the same way as the previous 6 lines, except that the interupt was caused by a negative edge. As the polarity of the secondary sensor output will therefore be inverted, this routine again tests b0 of the user port, but this time branches to the opposite routine to before (ie left instead of right etc).

Lines 0340 to **0500** work exactly as the basic program, decrementing and incrementing 'xcord' etc.

Lines 0500 to **0570** test for CB2 interupts using the same method as for CB1, and jump to 'exit' if a CB2 interupt is not being flagged.

Lines 0570 to 0740 toggle the interupt mode for CB2, determine the mode of the previous interupt, and jump to the appropriate routine to either increment or decrement 'ycord' as required.

Lines 0750 to 0910 decrement and increment 'ycord' as before.

Lines 0920 to 0990 retrieve the processor status from the stack, and return control from the ISR, as before.

8 COMPLETE BASIC PROGRAMS

The complete Basic program listings for the mouse routines are shown at the end of this chapter in Fig 18 and Fig 19. These listings contain the Initialization and Interupt Service Routines, together with the necessary Assembler control functions.

Fig 18 shows the code for the simple mouse software, using only positive edge triggered interupts. Fig 19 shows the code for the improved version, using both positive and negative edge triggered interupts. Both programs are complete on their own, so only the one required need be typed in.

To use either program, simply press 'BREAK' and type the listing required into the micro. Once the program is in the machine, check it and 'SAVE' it a few times for future use.

The BASIC program must now be 'RUN' to assemble the code into memory. If the assembler reports any errors, re-check the BASIC program, correct it and 'SAVE' the new version. Do not forget to press <BREAK>, type 'OLD' <RETURN> and 'RUN' the BASIC program again if any alterations are made !

The code to drive the mouse is now in the micro. To start the machine code program, type 'CALL init' <RETURN>. This integrates the mouse routine into the Operating System.

The mouse program should now be operational. If the machine 'locks up' immediately, or after the mouse is moved, an error must have been made whilst entering the BASIC program. Press <CTRL> and <BREAK> to reset the machine, and load the BASIC program back for correction. When the mistake has been found, proceed as before to assemble the program in the computer etc.

The function of the mouse software is to generate two numbers to represent the values of the X and Y coordinates of the mouse. To test the program, type the following line of BASIC into the computer:-

```
CLS:REPEAT:PRINTTAB(0,0);?xcord+256*?(xcord+1),?
ycord+256*?(ycord+1):UNTIL FALSE <RETURN>
```

Two numbers should be displayed in the top left hand corner of the screen. The first number, the X coordinate of the mouse, should increase and decrease as the mouse is moved right and left respectively. The second number, the Y coordinate, should increase and decrease as the mouse is moved up and down respectively.

Figure 18
COMPLETE BASIC PROGRAM LISTING

```
 10 REM MOUSE DRIVER
    ROUTINE V1.3
 20 REM  (C) I.HEWITT
    1988
 30 DIM MC% 200
 40 FOR opt%=0 TO 3
    STEP 3
 50 P%=MC%
 60 [
 70 OPT opt%
 80 .init
 90 SEI
100 LDA &0206
110 STA oldv
120 LDA &0207
130 STA oldv+1
140 LDA #prog MOD 256
150 STA &0206
160 LDA #prog DIV 256
170 STA &0207
180 LDA #112
190 STA &FE6C
200 LDA #152
210 STA &FE6E
220 LDA #127
230 STA &FE6D
240 LDA #0
250 STA &FE62
260 CLI
270 RTS
280 .prog
290 LDA &FC
300 PHA
310 TXA
320 PHA
330 TYA
340 PHA
350 LDA &FE6D
360 AND #128
370 BEQ exit
380 LDA &FE6D
390 AND #16
400 BEQ CB2
410 .CB1
420 LDA &FE60
430 STA portb
440 AND #1
450 BNE right
460 .left
470 SEC
480 LDA xcord
490 SBC #1
500 STA xcord
510 LDA xcord+1
520 SBC #0FIG 18
530 STA xcord+1
540 JMP CB2
550 .right
560 CLC
570 LDA xcord
580 ADC #1
590 STA xcord
600 LDA xcord+1
610 ADC #0
620 STA xcord+1
```

```
630 .CB2
640 LDA &FE60
650 STA portb
660 LDA &FE6D
670 AND #8
680 BEQ exit
690 STA &FE6D
700 LDA &FE60
710 AND #4
720 BNE down
730 .up
740 SEC
750 LDA ycord
760 SBC #1
770 STA ycord
780 LDA ycord+1
790 SBC #0
800 STA ycord+1
810 JMP exit
820 .down
830 CLC
840 LDA ycord
850 ADC #1
860 STA ycord
870 LDA ycord+1
880 ADC #0
890 STA ycord+1
900 .exit
910 PLA
920 TAY
930 PLA
940 TAX
950 PLA
960 STA &FC
970 JMP (oldv)
980 .oldv EQUW0000
990 .xcord EQUW0000
1000 .ycord EQUW0000
1010 .portb EQUB0
1020 ]
1030 NEXT opt%
```

Figure 19
IMPROVED BASIC
PROGRAM LISTING

```
10 REM MOUSE DRIVER
   ROUTINE V2.3
20 REM (C) I.HEWITT
   1988
30 DIM MC% 400
40 FOR opt%=0 TO 3
   STEP 3
50 P%=MC%
60 [
70 OPT opt%
80 .init
90 SEI
100 LDA &0204
110 STA oldv
120 LDA &0205
130 STA oldv+1
140 LDA #prog MOD 256
150 STA &0204
160 LDA #prog DIV 256
170 STA &0205
180 LDA #112
190 STA &FE6C
200 LDA #152
210 STA &FE6E
220 LDA #127
230 STA &FE6D
240 LDA #0
250 STA &FE62
260 CLI
270 RTS
280 .prog
290 LDA &FC
300 PHA
310 TXA
320 PHA
330 TYA
340 PHA
350 LDA &FE6D
360 AND #128
370 BNE skip1
380 JMP exit
390 .skip1
400 LDA &FE6D
410 AND #16
420 BEQ CB2
```

```
430  .CB1
440  LDA  &FE6C
450  EOR  #16
460  STA  &FE6C
470  AND  #16
480  BNE  CB1N
490  .CB1P
500  LDA  &FE60
510  STA  portb
520  AND  #1
530  BNE  right
540  JMP  left
550  .CB1N
560  LDA  &FE60
570  STA  portb
580  AND  #1
590  BNE  left
600  JMP  right
610  .left
620  SEC
630  LDA  xcord
640  SBC  #1
650  STA  xcord
660  LDA  xcord+1
670  SBC  #00
680  STA  xcord+1
690  JMP  CB2
700  .right
710  CLC
720  LDA  xcord
730  ADC  #1
740  STA  xcord
750  LDA  xcord+1
760  ADC  #00
770  STA  xcord+1
780  .CB2
790  LDA  &FE60
800  STA  portb
810  LDA  &FE6D
820  AND  #08
830  BNE  skip2
840  JMP  exit
850  .skip2
860  STA  &FE6D
870  LDA  &FE6C
880  EOR  #64
890  STA  &FE6C
900  AND  #64
910  BNE  CB2N
920  .CB2P
930  LDA  &FE60
940  AND  #4
950  BNE  down
960  JMP  up
970  .CB2N
980  LDA  &FE60
990  AND  #4
1000 BNE  up
1010 JMP  down
1020 .up
1030 SEC
1040 LDA  ycord
1050 SBC  #1
1060 STA  ycord
1070 LDA  ycord+1
1080 SBC  #0
1090 STA  ycord+1
1100 JMP  exit
1110 .down
1120 CLC
1130 LDA  ycord
1140 ADC  #1
1150 STA  ycord
1160 LDA  ycord+1
1170 ADC  #0
1180 STA  ycord+1
1190 .exit
1200 PLA
1210 TAY
1220 PLA
1230 TAX
1240 PLA
1250 STA  &FC
1260 JMP  (oldv)
1270 .oldv  EQUW0000
1280 .xcord  EQUW0000
1290 .ycord  EQUW0000
1300 .portb  EQUB0
1310 ]
1320 NEXT  opt%
```

Jessa House, 250 High Street, Watford, WD1 2AN, England
Tel: Watford (0923) 37774, Telex: 8956095 WATFRD, FAX: 01 950 8989