Mouse Technical Note #1
15-Mar-84

This technical note explains what you need to be concerned about regarding the computer's interrupt environment with the mouse, regardless of whether you are using interrupts or not.

For further information contact:
PCS Developer Technical Support
M/S 22-W. Phone (408) 996-1010

Disclaimer of All Warranties and Liabilities

/bg

Software developers who are writting mouse based programs in machine language need to be concerned about the computer's interrupt environment even if they are using the mouse in passive mode. Listed below are several conditions which a machine language programmer should take into account if their programs are to run on the Apple // family of computers.

* Do not disable interrupts unless you must. Then be sure to re-enable them.

* Disable interrupts when calling any mouse routine (SEI).

* Do not re-enable interrupts (CLI) or (PLP if previously had done a PHP) after READMOUSE until X & Y data have been removed from the screen holes.

* Be sure to disable interrupts (SEI) before placing position information in the screen holes (POSMOUSE or CLAMPMOUSE)./

* Enter all mouse routines (not required for SERVEMOUSE) with the X register set to $Cn and Y register set to $n0 where n = slot number.

* Some programs may need to turn off interrupts for purposes other then reading the mouse. This is sometimes done on the Apple //e to keep from having to handle interrupts are turned off and then back on, the first call to READMOUSE may give incorrect values. Subsequent calls to READMOUSE will return correct values until interrupts are turned off and on again. Turning off interrupts for mouse calls does not create this problem. If you are watching numbers coming from the mouse while moving it in a direction that would increase values you might see the following: 6, 7, 8, 9, 8, 9, 10. In practice, this momentary 'glitch' in the stream of mouse data has little importance and would probably only be noticed by programmer testing his/her program - no one's hand is that steady. If you must keep this 'glitch' from happening then do not keep interrupts off for more then 40 microseconds or be sure that at least one mouse interrupt has taken place since interrupts were turned back on.

Mouse Technical Note #2
15-Mar-84

This technical note explains how to vary the "VBL" interrupts between 50 Hz (European rate) or 60 Hz (North American rate).

For further information contact:
PCS Developer Technical Support
M/S 22-W. Phone (408) 996-1010

Disclaimer of All Warranties and Liabilities

/bm

This technical note documents a previously undocumented call to the AppleMouse II firmware which allows the user to set the interrupt rate to 50 Hz, or 60 Hz. (60 Hz is the default, and keeps the mousecard-generated "VBL" interrupts synchronized with the actual VBL rate on a standard North American Apple. 50 Hz is necessary for European machines. "60 Hz" and "50 Hz", as used here, are actually shorthand for the Apple video cycle rates used in North America and Europe, respectively).

Call: TIMEDATA

Offset Location: $Cn1C

Input: Accumulator bit 0 = 0 for 60 Hz
= 1 for 50 Hz

Note: All other accumulator bits are reserved, and MUST be set to 0.

Output: carry bit clear.
screenholes unchanged.

This call must be made before INITMOUSE, and then followed by an INITMOUSE call in order to be effective. If you want to change the interrupt rate in the middle of an application, you must call TIMEDATA, with the appropriate value in the accumulator, and then INITMOUSE (until the INITMOUSE is called, no interrupts will be generated). INITMOUSE will, of course, reset the mouse position, mode, clamps, etc., back to their default values.

If TIMEDATA is never called, then the interrupt rate will default to 60 Hz when INITMOUSE is called.

NOTE: This call exists only on the Mousecard for the //e or][+ and should only be used when you know you are working with a //e or][+.

Mouse Technical Note #3
15-Mar-84

This technical note explains what happens when you turn the mouse on and off through t. mode byte of the SETMOUSE routine.

For further information contact:
PCS Developer Technical Support
M/S 22-W. Phone (408) 996-1010

Disclaimer of All Warranties and Liabilities

A) What turning the mouse "off" does:

In the description of SETMOUSE and the mouse mode (see AppleMouse II Users Manual pg. 44), the low-order bit of the mouse mode is said to control "mouse off/mouse on". This is somewhat misleading terminology. When this bit is set to 0, the mouse is off only in the following respects:

(1) the mouse position is not tracked. Any mouse motion is ignored.

(2) READMOUSE calls do not update the status byte or the screen holes (the 6502 firmware makes the READMOUSE command a NOP, and does not even issue the READMOUSE command to the 6805)

(3) button and movement interrupts are not generated, regardless of the other mouse mode bits. "Pure" VBL interrupts can still be generated, however, if bit 3 is set.

B) What turning the mouse "off" doesn't do:

Other mouse functions will continue to work as usual when the mouse is "off". POSMOUSE and CLEARMOUSE will change the mouse position, CLAMPMOUSE will set new clamp values, etc. HOMEMOUSE is an odd case; it will change the mouse position as recorded in the 6805, but this change will not appear in the screen holes until a READMOUSE is done with the mouse "on". In particular:

(1) turning the mouse "off" and "on" with the mode byte does not reset any mouse values, including position, to their defaults. The mouse position retains the last values it had before the mouse was turned off, until it is turned "on" again.

(2) a mode byte of $08 - mouse "off", but VBL interrupt on - will still generate VBL interrupts.

Mouse Technical Note #4
15-Mar-84

This technical note explains a bug in the Mouse Firmware having to do with the way that SERVEMOUSE works.

For further information contact:
PCS Developer Technical Support
M/S 22-W. Phone (408) 996-1010

Disclaimer of All Warranties and Liabilities

Apple Computer, INC. Makes NO warranties, either express or implied, with respect to this technical note or with respect to the software described in this technical note, its quality, performance, merchantability, or fitness for any particular purpose. Apple Computer Software is licensed "as is". The entire risk as to its quality and performance is with the developer. Should the program prove defective following its use, the user (and not Apple Computer, INC., their distributors, or their retailers) assumes the entire cost of all necessary servicing, repair or correction and any incidental or consequential damages. In no event will Apple Computer, INC. be liable for direct, indirect, incidental or consequential damages resulting from any defect in the software, even if they have been advised of the possibility of such damages. Some states do not allow the exculsion or limitation of implied warranties or liability for incidental or consequential damages, so the above limitation or exclusion may not apply to you.

/bm

There is a bug in the AppleMouse II 6805 firmware which may affect the way SERVEMOUSE works in an application program. If the application program takes more than 1 video cycle (normally, about 16 milliseconds) to respond to a mouse-generated interrupt, then there is a chance that SERVEMOUSE will not claim the interrupt: that is, the 6805 will return an interrupt status byte of $00 (i.e. no Mouse interrupt pending), and the 6502 firmware will set the carry bit (although the interrupt will also be cleared by the SERVEMOUSE call). This can be confusing, and under ProDOS or Pascal it can be lethal. We have identified the following solutions, any one of which should work:

(I) If you are not working under an established system (like ProDOS or Pascal):

(A) don't allow unclaimed interrupts to be fatal to your application. Ignore them.

or

(B) Always service mouse interrupts within 1/60 of a second. If you are forced to turn off interrupts for about that length of time or more, first:

use SETMOUSE to set the mouse mode to 0.

call SERVEMOUSE to clear any existing mouse interrupt.

After interrupts are turned back on, restore the mouse mode.

(II) If you are working under an established operating system, like ProDOS or Pascal, for which unclaimed interrupts are fatal:

(A) If the mouse is the only interrupting device: write your interrupt handler so that it claims all interrupts. That is, regardless of whether the mouse admits to generating the interrupt, clear the carry bit before exiting the interrupt handler, to let ProDOS or Pascal know the interrupt was "serviced".

(B) If the mouse is not the only interrupting device:

(1) Write the mouse interrupt handler to claim all unclaimed interrupts, as described in (II)-(A) above, and make sure the mouse interrupt handler is installed last - otherwise the interrupt will never get through to any interrupt handlers whic

follow the mouse's.

Note: This solution may cause curser flicker by delaying the application's response to VBL interrupts.

or

(2) Write a spurious interrupt handler (also known as a "demon"), not associated with any device, which claims all unclaimed interrupts (that is, clears the carry bit and then exits). For the reason just mentioned, this interrupt handler must be installed last.

Note: Under ProDOS, this cuts down the number of interrupting devices that can be used to 3.

or

(3) Include code in every interrupt handler to check if that interrupt handler is last. If it is, then that interrupt handler should claim any previously unclaimed interrupts, even if its device was not generating it.

Under ProDOS, this would permit 4 interrupting devices; but it may be tricky to implement, and it requires identical code in each interrupt handler which must be executed every time the handler is called.

Note: This bug will be fixed in future versions of the mouse card (but that's not much help, is it?).

Mouse Technical Note #5
4-May-84

This technical note explains how you can go about checking to see if the mouse firmware card you have identified through software is able to support interrupts.

For further information contact:
PCS Developer Technical Support
M/S 22-W. Phone (408) 996-1010

Disclaimer of All Warranties and Liabilities

Apple Computer, Inc. makes no warranties, either express or implied, with respect to this documentation or with respect to the software described in this documentation, its quality, performance, merchantability, or fitness for any particular purpose. Apple Computer, Inc. software is sold or licensed "as is". The entire risk as to its quality and performance is with the vendor. Should the programs prove defective folowing their purchase, the vendor (and not Apple Computer, Inc., its distributor, or its retailer) assumes the entire cost of all necessary servicing, repair, or correction and any incidental or consequential damages. In no event will Apple Computer, Inc. be liable for direct, indirect, incidental, or consequential damages resulting from any defect in the software, even if Apple Computer, Inc. has been advised of the possibility of such damages. Some states do not allow the exclusion or limitation of implied warranties or liability for incidental or consequential damages, so the above limitation may not apply to you.

Notice

Apple Computer, Inc. reserves the right to make improvements in the product described in this document at any time and without notice.

/gs

Checking To See Whether A Mouse Type Device Supports Interrupts.

After identifying a card as a mouse type device it is important to check if the card supports interrupts. There is a convention defined in the mouse firmware protocol that does just that. It defines location $CN11 as a flag to indicate whether or not interrupts can be supported. The value at location $CN11 will be a 0 if interrupts are supported. It is important to check this byte if your program uses interrupts. The reason that you must check this is that the device may not be a mouse at all, rather some other type of device that is emulating a mouse, without interrupt generating capability. (This could be a track ball, graphics tablet, etc.) If the byte at $CN11 is a non zero value then that device does not support interrupts and must be used passively.

If you are a hardware developer and would like your device to emulate a mouse you must follow the mouse protocol as described in the AppleMouse II users guide. (pg 43-49) Your device must also have the same signature bytes as the mouse card. These are $CN0C=$20 and CNFB=$D6. (The N in the above addresses represents the slot number that the card happens to be in at the time. So for slot 4, $CNFB you would have $C4FB.)

Note: The use of location $CN11 is not described in the AppleMouse II users guide. Having your program check this byte is highly recommended since it is quite likely that devices which emulate the mouse will be developed, and some of them may not support interrupts. Through this byte you have a simple way to check if the device supports interrupts.

MOUSE TECHNOTE #6

Revision of general handout on the Apple//e Dec 83*
5-July 84

This technote explains changes that will be made to the Apple//e ROM so that it will support text icons. These icons will be used by the new 'mouse' interface tool kit.

For further information contact:
PCS Developer Technical Support
M/S 22w. Phone (408). 996-1010

Notice

Apple Computer, Inc. reserves the right to make improvements in the product described in this document at any time and without notice.

Dear Developers:

THIS IS TO NOTIFY YOU THAT APPLE COMPUTER WILL BE MAKING A CHANGE TO THE CHARACTER SET IT USES IN ITS APPLE//e. THIS MAY AFFECT YOUR SOFTWARE.

This new character set will be available to the public in 1984.

WE ARE NOTIFYING YOU OF THIS CHANGE AT THIS TIME BECAUSE SOME CURRENT SOFTWARE MAY NO LONGER FUNCTION CORRECTLY WITH THE NEW CHARACTER SET. UNDER THE RIGHT CONDITIONS AN INVERSE UPPERCASE LETTER WILL NOW BE A GRAPHIC ICON.

The following will help you identify if the changes we are making will affect you or not.

1. If your program is written entirely in BASIC or Pascal or your Assembly Language program calls the COUT routine to put characters on the screen you will not be affected. The only exception would be if you are using BASIC pokes to Poke inverse upper case characters directly to the text screen.

2. If your program is using the standard character set (checkerboard cursor) you will not be affected.

3. If your program is using the alternate character set (solid cursor) and is directly POKING (storing) values to the text display area you will have problems if your character values are from 64 ($40) to 95 ($5F). These values now display inverse uppercase characters plus some special characters. In the future these values will display graphic icons. To recreate the origional displays (which include inverse uppercase plus some special characters) use values in the range from 0 ($0) to 31 ($1F) rather then the values from 64 ($40) to 95 ($5F). Note that using these lower values will work properly on the current character set.

We at Apple are excited about this new extension to the Apple//e's alternate character set. The new icons are similar to those used in LISA and will enhance the use of pointing devices such as a mouse on the Apple//e. If used effectively, the icons, in connection with pointing devices, can significantly simplify the human interface of your programs.

What we foresee as the ways to access these ICONS from various programming languages are described below. We have also included a sample of the current ICON set. More detailed and accurate information will be provided prior to making the ICONS available.

The following method will probably be used for showing ICONS from BASIC:

1) Set up the alternate character set by POKING 49162 ($C00A) with any value then doing a PR#3. If an 80 column card is present you may remain in 80 columns. If there is no card or you want to be in 40 columns PRINT CHR$(11).
2) PRINT CHR$(27) to enable the mouse characters.
3) Use the INVERSE command to set inverse mode.
4) PRINT the appropriate capital letter for the desired ICON. See attached

ICON chart.
Disable the ICONS by PRINTing CHR$(24).

Machine Language programs are expected to follow the same procedure as BASIC. Use calls to COUT to perform the print operations. The following is a sample Machine Language program which will 'print' two ICONS followed by the two inverse uppercase letters that have the same ASCII values.

```
START    STA   $C00A          ;FLIP IN 80 COLUMN FIRMWARE
         LDA   #$A0          ;USE A BLANK TO
         JSR   $C300         ;TURN ON VIDEO FIRMWARE
         LDY   #0            ;INIT COUNTER
LOOP     LDA   STR,Y         ;GET VALUE
         JSR   $FDED         ;SEND IT THROUGH COUT ROUTINE
         INY
         CPY   STRLEN
         BNE   LOOP          ;=>NOT DONE YET
         RTS
STR      DFB   $1B,$46,$47,$18,$46,$47 ;ICONS ON, SHOW, ICONS OFF, SHOW
STRLEN   EQU   *-STR         ;LENGTH OF STR
```

NOTE: 'printing' ICONS on the text screen by directly poking or storing ICON values into the text buffer is not supported.

The probabable method for using the ICONS from Pascal 1.1 will be as follows:

1) Output a chr(27), an escape character, to enable ICONS.
2) Output a chr (15) to turn on inverse video.
3) Output the appropriate capital letter for the desired ICON. See attached ICON chart.
Disable the ICONS by outputting chr(24).

Pascal sample program:

```
program Output_mouse_icon;
var cmd : packed array [ 0..1] of 0..255;
begin
  cmd [ 0 ] := 27;  cmd [ 1 ] := 15;
  Unitwrite (1, cmd, 2 );  {turn on ICON mode }
  {code to display icons...
            .
            .
            .
  }
  cmd [ 0 ] := 24;
  Unitwrite ( 1, cmd, 1 ); {turn off ICON mode }
end.
```