Subject: Get Mouse Position using low level assembly language
Posted by Oz on Tue, 07 Oct 2014 11:04:27 GMT
View Forum Message <> Reply to Message

Even if the mouse was nearly never use in games at the Apple IIe/c times, with the Apple IIgs, many terrific games were using the Mouse as primary game controller (Zany Golf, Arkanoid, Defender of the Crown, Dungeon Master...). Because non of them were using the Apple IIgs Graphic User Interface, they needed to find a way to read the mouse pointer coordinates. Games like Arkanoid don't need to show up a pointer on the screen but need anyway to calculate the X coordinate of a virtual mouse pointer to draw the game pad.

Reading mouse on the Apple IIgs is not very complex. You can directly get the information using the ADB registers or simply use the $C027 and $C024 softswitch. The $C027 give us a status of the mouse (are the coordinates ready to be read) and the $C024 give us the mouse button status and the X and Y information, if available.

Even if we see a mouse information like the X and Y coordinates of a Mouse pointer on the screen, the Mouse register ($C024) give us a DeltaX and a DeltaY information. So we don't know where is the pointer, but what was the last movement from the previous position. Up to us to get these DeltaX and DeltaY (a number between 0-63 and a direction) and apply them to the previous known X0,Y0 coordinates to get the new X1,Y1 coordinates. Of course, because we don't want the mouse pointer to go outside of the screen, we have to control the X1,Y1 coordinates to make sure they don't go too far.

In the next sample code, we want our mouse pointer to be restricted to a 320x200 screen. Because the pointer itself is 8 pixels long and 6 pixels high, we let the mouse coordinates to fly into a 0,0 to 312,194 area. Depending on you own pointer size, you might have to change these MaxX and MaxY values.

The code starts by checking the status of the Mouse Register by reading the $C027 (KMSTATUS) softswitch. If the data are not available, we exit immediately. No need to wait here. If the data are available, we read twice the $C024 (once for DeltaX data, once for DeltaY data) and we compute the new coordinates. We read the Mouse Button status during the read of DeltaY value. We could have done that outside of the routine or during the DeltaX read. There is no specific reason for reading Mouse Button status during DeltaY value. The code has to be called with A in 16 bits.

We have kept all intermediate values (DeltaX, DeltaY, DirectionX, DirectionY) because sometimes we need such values if we want, for example, to simulate a joystick using the mouse (we don't care about the Delta, we care more about the direction).

Olivier

```
*-------------------------------------------------------------
*  ReadMouse using low level softswitch ($C024 & $C027)
*-------------------------------------------------------------
```

```
ReadMouse      LDAL  $00C026       ; Get Mouse Status using $C027 (KMSTATUS)
               BMI   RM_Status
               RTS                 ; Mouse not ready, exit
*---
RM_Status      AND   #$0200        ; Bit 1 (0=DeltaX, 1=DeltaY)
               BEQ   RM_Init
               LDAL  $00C024       ; If the DeltaY is available, loop until we get DeltaX first
               BRA   ReadMouse
*---
RM_Init        LDA   #$0000
               STA   MouseButton   ; 0=Button Up, 1=Button Down
               STA   DeltaXSign    ; 0=Positive, 1=Negative
               STA   DeltaYSign    ; 0=Positive, 1=Negative
*-----
RM_DeltaX      LDAL  $00C023       ; Read DeltaX using $C024 (MOUSEDATA)
               BIT   #$4000        ; Sign
               BNE   RM_DX_NEG
               AND   #$3F00        ; DeltaX > 0
               STA   MouseDeltaX
               BRA   RM_DeltaY
*-
RM_DX_NEG      AND   #$3F00        ; DeltaX < 0
               STA   RM_DX_NEG_1+1
               INC   DeltaXSign
               LDA   #$4000        ; 64 is the max value for a Delta
               SEC
RM_DX_NEG_1    SBC   #$0000
               STA   MouseDeltaX   ; Keep DeltaX > 0 and record sign in DeltaXSign
*-----
RM_DeltaY      LDAL  $00C023       ; Read DeltaY + Button #1 Status using $C024
(MOUSEDATA)
               BMI   RM_DY_SIGN
               INC   MouseButton   ; Button #1 is Down
*-
RM_DY_SIGN     BIT   #$4000        ; Sign
               BNE   RM_DY_NEG
               AND   #$3F00        ; DeltaY > 0
               STA   MouseDeltaY
               BRA   RM_X
*-
RM_DY_NEG      AND   #$3F00        ; DeltaY < 0
               STA   RM_DY_NEG_1+1
               INC   DeltaYSign
               LDA   #$4000        ; 64 is the max value for a Delta
               SEC
RM_DY_NEG_1    SBC   #$0000
               STA   MouseDeltaY   ; Keep DeltaY > 0 and record sign in DeltaYSign
*-----
```

```
RM_X        LDA   DeltaXSign    ; Compute X Coordinate
            BNE   RM_X_NEG
*-
            LDA   MouseX        ; DeltaX > 0
            CLC
            ADC   MouseDeltaX+1
            CMP   #$0139        ; 313
            BMI   RM_X_POS1
            LDA   #$0138        ; 312 is the X max
RM_X_POS1     STA   MouseX
            BRA   RM_Y
*-
RM_X_NEG      LDA   MouseX        ; DeltaX < 0
            SEC
            SBC   MouseDeltaX+1
            BPL   RM_X_NEG1
            LDA   #$0000
RM_X_NEG1     STA   MouseX
*-----
RM_Y        LDA   DeltaYSign    ; Compute Y Coordinate
            BNE   RM_Y_NEG
*-
            LDA   MouseY        ; DeltaY > 0
            CLC
            ADC   MouseDeltaY+1
            CMP   #$00C3        ; 195
            BMI   RM_Y_POS1
            LDA   #$00C2        ; 194 is the Y max
RM_Y_POS1     STA   MouseY
            RTS
*-
RM_Y_NEG      LDA   MouseY        ; DeltaY < 0
            SEC
            SBC   MouseDeltaY+1
            BPL   RM_Y_NEG1
            LDA   #$0000
RM_Y_NEG1     STA   MouseY
            RTS
*----
MouseButton    HEX   0000       ; Button Status (0=Up, 1=Down)
MouseX         HEX   0000       ; X Coordinate (0-312)
MouseY         HEX   0000       ; Y Coordinate (0-194)
MouseDeltaX    HEX   000000     ; Delta X
MouseDeltaY    HEX   000000     ; Delta Y
DeltaXSign     HEX   0000       ; Direction X (0=go right, 1=go left)
DeltaYSign     HEX   0000       ; Direction Y (0=go down,  1=go up)

*-------------------------------------------------------
```

Subject: Re: Get Mouse Position using low level assembly language
Posted by Dagen on Wed, 08 Oct 2014 20:27:18 GMT
View Forum Message <> Reply to Message

I have a few questions.

I wrote some mouse code, when I was younger, that I later discovered did not work on the ROM 3 Apple IIgs, only the ROM 1. I believe the problem was that I was directly using the $C3xx address space, or something like that. Obviously, they changed the way the mouse firmware was implemented on the ROM 3.

Does this code you've posted work on all Apple IIgs machines? I assume the answer is yes.

Does this code rely on interrupts? I.e.- If we disable interrupts, will the mouse data still update?

How often does the data in the registers change? (I'm guessing I can determine this myself by looking at how quick the $C027 high bit changes)

Subject: Re: Get Mouse Position using low level assembly language
Posted by Oz on Thu, 09 Oct 2014 11:42:52 GMT
View Forum Message <> Reply to Message

Dagen,

Quote:Obviously, they changed the way the mouse firmware was implemented on the ROM 3.

They have added the use of Numeric Pad keys to simulate the mouse move. I think it was for simplifying the usage of the Mouse by disable people.
So they had to change few things in the Mouse Firmware but the very low level (ie softswitchs) was not impacted.

Quote:Does this code you've posted work on all Apple IIgs machines? I assume the answer is yes.

This is the kind of code we were using in our games in the 90's, so I would say YES for the well known Apple IIgs models (Rom 1, Rom 3 & Mark Twain).

Quote:Does this code rely on interrupts? I.e.- If we disable interrupts, will the mouse data still update?

We don't use interrupts here. We just read softswitch values. The fact to disable interrupts at the 65c816 level (SEI / CLI) don't change anything about the updates of the softswitchs by the hardware. You can continue to read the Keyboard ($C010), the Mouse ($C024), the Joystick ($C064/$C065 + $C025), Get VBL information ($C02E)...

GS/OS uses the VBL interrupt to update the Mouse pointer on the screen when you are using the Apple Graphic Interface (this explain why, even if your program has crashed, you can still

move the mouse on the screen), but here, because we want to handle the Mouse position in our way, we don't care about interrupts.

Quote:How often does the data in the registers change?

I think nothing change until you decide to move your mouse (or click the button) ! So this is why we exit immediately if we see the registers are not ready (you could wait forever if the GS has no mouse connected).

Once the move has been done, I think it needs few cycles for the ADB controller to update the softswitch with the value. The ADB has a 16 bits value for the Mouse move and has to copy it into a 8 Bit register ($C024). This is why you have to read it twice (one for the DeltaX, one for the DeltaY). I have never really computed the number of cycles required to get the data ready because we were more in a 'not ready = no time to loose here = exit now' strategy.

Olivier
