Apple IIc
#1: Mouse Differences on IIe and IIc

Revised by: Matt Deatherage November 1988
Revised by: Cameron Birse February 1986

This Technical Note explains differences between the IIe and IIc when working with a mouse and how to write programs which function properly on both machines.

If you use the mouse firmware routines (i.e., `SetMouse`) to control the mouse, then these routines will perform the same function on the IIc as they do on the IIe. However, a program which uses the mouse may not behave the same on both computers, and there are two reasons for the possible differences.

If a program does not properly set the environment prior to calling the mouse firmware routines, it is possible for a program to work on one machine and not the other. In addition, there are differences in machines and although the ROM routines perform the same functions, there may be a noticeable difference in the mouse behavior between the two machines.

This Note explains the fundamental differences between the way the mouse works on the two machines. We point out precautions that you need to take to ensure your assembly language programs work properly on both machines. (With the exception of mouse movement scaling described below, neither BASIC nor Pascal programs need be concerned with setting the proper environment.)

The Apple IIe mouse card has a microprocessor on it which constantly polls the mouse to get status and position information. This data is kept on the card and is available whenever the program requests it through the `ReadMouse` routine. If the mouse is in passive mode, this information will be picked up by the main program whenever it gets around to it.

The `SetMouse` routine can set the mouse card to issue interrupts under certain conditions. When the mouse card determines that such conditions exist, it issues an interrupt. This interrupt stops the main computer and goes to whatever interrupt handling routine has been prepared. This routine then reads the information from where the card processor saved it and puts it in the screen holes. When using a mouse on an Apple with a mouse card, your program is only interrupted if you have requested it, and the data in the screen holes is changed only when the program's interrupt handler or polling routine calls `ReadMouse`. In addition, enabling and disabling interrupts does not affect the card's microprocessor from updating the mouse information.

The Apple IIc mouse does not have a card microprocessor, so mouse information is collected by interrupting the microprocessor of the IIc itself. When the interrupt occurs, the firmware captures it and processes it, which includes updating the screen holes. The interrupt is passed only if `SetMouse` set up the conditions to do so.

Having the mouse interrupt the computer's microprocessor also means that your program is being constantly interrupted, which affects program timing. This interruption also means that the screen holes are constantly updated with X and Y information, even in passive mode, since this information must be stored somewhere and there is no card to keep it in. If you have disabled interrupts, the mouse can never interrupt the microprocessor, so the X and Y values are never updated and calling `ReadMouse` will indicate that there has been no mouse movement.

Since the Apple IIc is constantly interrupted while the mouse is on, the program's performance may be affected. To minimize this effect, the IIc responds one-half as frequently to mouse movements as does the mouse card, which means the mouse must be moved twice as far to create the same on-screen effect. If you want the same behavior on both machines, multiply the IIc X and Y values by two and the clamping value by one half. You do not need to make any changes to these values if your program is running on a IIe.

With this exception for mouse movement, your assembly language program can ignore which machine it is running on by following the precautions listed in Mouse Technical Note #1, Interrupt Environment with the Mouse (you must take these conditions into account if you want your assembly language program to behave similarly on both machines). If you are working in BASIC or Pascal, these conditions are already handled for you.

Further Reference

- *Apple IIc Technical Reference Manual, Second Edition*
- Mouse Technical Note #1, Interrupt Environment with the Mouse

Apple IIc
#2: 40-Column and Double High-Resolution Graphics

Revised by: Matt Deatherage November 1988
Revised by: Cameron Birse February 1986

This Technical Note describes how to properly handle the 40-column screen while using double high-resolution graphics on the Apple IIc.

Many developers using double high-resolution graphics may wish to use 40-column text displays so that the text can be read on a television set. There are a couple of possibilities for accomplishing this task:

1. You can define your own double high-resolution character set with any size characters you desire, then plot them on the double high-resolution screen.
2. You can print text to the Apple IIc text screen and toggle the screen on to display it.

Note: There is no way to display 4 lines of 40-column text at the bottom of the double high-resolution screen in mixed mode since the 80 column hardware must be active while double high-resolution mode is being used.

Using the second method outlined above requires some special considerations.

The Apple IIc scroll routine continues to use the window parameters when scrolling, but uses the `80COL` softswitch to determine if it should scroll the 80-column screen or 40-column screen. Since the firmware has initialized a 40-column window, the scroll routines will move only the first 40 columns, but the `80COL` flag has been turned on for double high-resolution. Because of the `80COL` flag, the scroll routine takes every even column from auxiliary memory and every odd column from main memory. As a result, only the first 40 columns get scrolled, 20 columns from auxiliary memory and 20 columns from main memory.

One solution to the problem is writing your own scroll routines, while another is writing to the screen so scrolling is not necessary. There is, however, another solution. Turn on the full 80-column mode with `PR#3` or equivalent. Now print your text to `COUT` in the normal manner, and do not exceed 40 characters per line—the 80-column firmware should scroll everything properly. When you are ready to display text, send a Control-Q sequence through `COUT` to toggle to 40-columns and send a Control-R sequence to return to double high-resolution mode. These control characters toggle the display modes, but leave the 80-column firmware active.

When switching between modes, you may experience a momentary glitch. If you send the Control-Q sequence to `COUT` while still in graphics mode, the screen will first switch to the normal high-resolution mode before finally switching to text mode. If you switch to text mode first, the text will be in 80-column mode (with 40 columns displayed on the left of the screen) before ultimately switching to 40-column mode). This same potential glitch may occur when switching back to double high-resolution mode, and it may be only momentary and not present any problems for your application. If, however, it does present a problem, you may wish to make your switch coincide with the video's vertical blanking interval (see the *Apple IIc Technical Reference Manual, Second Edition*).

Further Reference

- *Apple IIc Technical Reference Manual, Second Edition*

Apple IIc
#3: Foreign Language Keyboard Layouts

Revised by: Matt Deatherage November 1988
Revised by: Cindy Roberts January 1985

This Technical Note formerly described the keyboard layouts and ASCII codes for international versions of the Apple IIc keyboard.

The information about international keyboard layouts and key codes which this Note formerly covered is now documented in all current versions of the *Apple IIc Technical Reference Manual, Second Edition.*

Further Reference

- *Apple IIc Technical Reference Manual, Second Edition*

Developer Technical Support

Apple IIc
#4: Dvorak Keyboard Layout

Revised by: Matt Deatherage November 1988
Revised by: Cameron Birse February 1986

This Technical Note discusses the Dvorak keyboard layout on the Apple IIc.

The old, red version of the *Apple IIc Reference Manual* incorrectly illustrated the Dvorak keyboard layout, however, the current *Apple IIc Technical Reference Manual, Second Edition* contains a corrected diagram on page 370.

The diagram in the current manual shows the Dvorak Simplified Keyboard (DSK) as it appears and functions on the Apple IIc today. This layout is the ANSI standard for the Dvorak keyboard layout, which was not available when the original IIc keyboard ROM was created. Previous IIc computers had a DSK layout as follows:

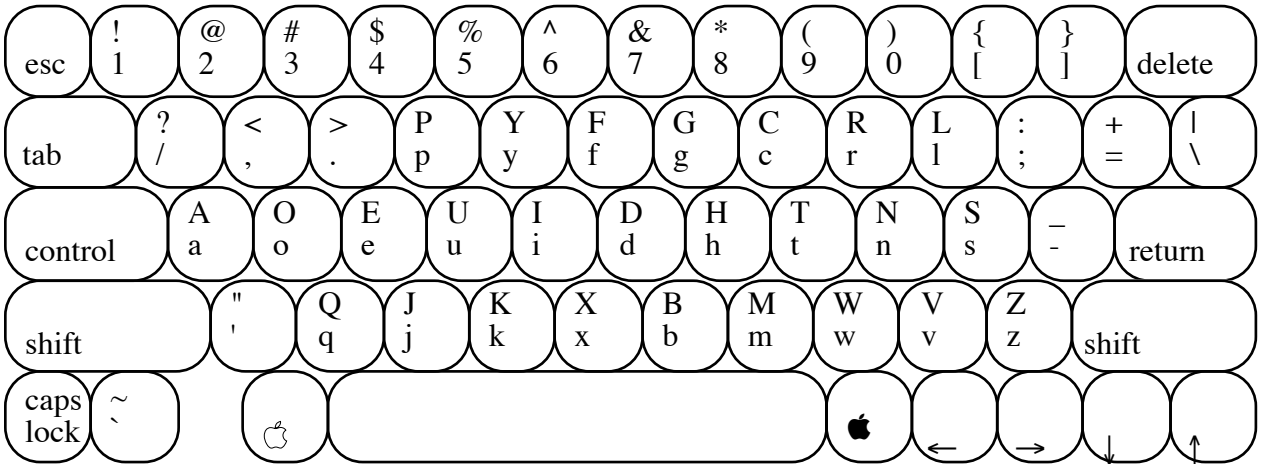

Figure 1–Dvorak DSK Layout on Early IIc Computers

Due to service part changes and other manufacturing considerations, it is not possible to identify which IIc units have which keyboard ROM by looking at identification bytes. If a program **requires** knowledge of this information (i.e., a typing program which draws the Dvorak keyboard), it must ask the user for input.

One possible way to accomplish this would be for a program to draw a blank keyboard layout (except for shift, tab, control, and other keys which do not move between Dvorak and Sholes layouts) and ask the user to press the key to the right of the left shift key, while the drawing on

screen highlights the correct key to press. If the key is a Z, the layout is a standard Sholes layout. If the key is an apostrophe or quotation mark, the layout is the DSK layout shown above. If the key is a semicolon or colon, the layout is the ANSI DSK layout on new IIc models. Since such a program must already ask the user if the keyboard switch is depressed (indicating a Dvorak layout), making this type of inquiry instead will do the trick.

The IIc manual has another DSK diagram in the front, on page 7. This diagram correctly shades those symbols which are in different places in the two DSK layouts.

Further Reference

- *Apple IIc Technical Reference Manual, Second Edition*

Apple IIc
#5: Memory Expansion on the Apple IIc

Revised by: Matt Deatherage November 1988
Written by: Cameron Birse October 1986

This Technical Note describes some important differences in the "memory-expandable" Apple IIc which you should take into account to ensure compatibility.

Beginning with the third Apple IIc, which was announced in September 1986, all new IIc models differ significantly from their predecessors. The most notable of these differences is the addition of a memory expansion capability. The memory expansion card for the IIc is functionally identical to the card for the IIe, but the IIc card "lives" in slot 4 and the firmware is included in the ROM on the IIc motherboard. This architecture means that you cannot depend upon the firmware ID bytes to tell if a card is installed, since unlike other "peripheral cards" in the IIc, the memory expansion card is not necessarily present. For this particular case, you need to interrogate the card and see how many blocks of memory are available. If there are no available blocks, there is no card.

SmartPort

Do a `STATUS` call with a `statcode` = $03 to get the Device Information Block (DIB). This call returns a value of $000000 in the device size fields if there is no RAM card.

In version 3 of the IIc ROM, the value resulting from a status call to device 0 implies that there is always a real card connected; the ROM version 4 returns device connected only when there is RAM card present.

ProDOS

When you do an `ON_LINE` call to the ProDOS MLI and there is no RAM on the Memory Expansion Card, you get an error $2D. This error is not a ProDOS error, rather it is a SmartPort error. The error is `BADBLOCK`, and basically tells you that the block requested was not available. If you try to catalog the RAM disk from BASIC, you will get a `PATH NOT FOUND` error.

Pascal

Formatting the RAM disk (unit #9) with no memory on the card returns no error. Doing a `UnitStatus` call will return zero blocks available, and trying to read the volume directory will result in an `IORESULT` of 8, which means no room is available on the volume. Doing the `V(ols` command from the `F(iler` will result in a `<no dir>` and # of blocks = 0.

DOS 3.3

If there is no memory on the card and you initialize it with an `IN#4` (which returns a slash, appearing to have successfully initialized the RAM disk), you will get an I/O error (`ONERR` code = 8) if you try to read from or write to the RAM disk.

Important: Another significant ramification of the memory expansion capability is that the mouse firmware has been moved to slot 7. This change means that programs should scan the slots just as they would on a IIe to find what peripherals are installed. Since most programs have a scan routine in them for the IIe, it should be a relatively minor change to call this routine for whatever machine you are on. In fact, we strongly recommend that programs always scan the slots for peripheral devices regardless of the machine on which they are running.

The firmware ID bytes for this version of the machine are:

Original Expandable IIc
$FBB3—$06 $FBC0—$00 $FBBF—$03

Revised Expandable IIc
$FBB3—$06 $FBC0—$00 $FBBF—$04

Apple IIc Plus
$FBB3—$06 $FBC0—$00 $FBBF—$05

Further Reference

- Apple IIc Technical Note #6, Buffering Blues
- Apple IIc Technical Note #7, Existing Versions
- Apple II Miscellaneous Technical Note #2, Apple II Family Identification Routines 2.1
- Apple II Miscellaneous Technical Note #7, Apple II Family Identification
- Apple II Miscellaneous Technical Note #8, Pascal 1.1 Firmware Protocol ID Bytes

Apple IIc
#6: Buffering Blues

Revised by: Mike Askins November 1988
Written by: Guillermo Ortiz January 1987

This Technical Note describes changes on the memory-expandable IIc which affect the procedures for enabling keyboard and serial input buffering.

When the IIc firmware was reorganized to accommodate the memory expansion card in slot 4, the mouse moved to slot 7, thus causing some screen holes to be reassigned. This change may software which uses keyboard or serial input buffering to crash.

The following list shows the changes in the locations which are used for enabling keyboard and serial input buffering:

Name	Original & 3.5 ROM	Expandable IIc	Comment
`typhed`	$5FA	$5FA	;buffer the keyboard? NO CHANGE
`twkey`	$5FF	$5FC	;storage pointer for type-ahead buffer
`trkey`	$6FF	$6FC†	;retrieve pointer for type-ahead buffer
`aciabuf`	$4FF	$4FC	;Owner of serial buffer, if any
`twser`	$57F	$57C	;storage pointer for serial buffer
`trser`	$67F	$67C	;retrieve pointer for serial buffer

† In the version 3 ROM (original "memory-expandable" IIc) this pointer is still $6FF which causes, among other things, the Terminal Mode to be inoperative. Revision 4 of the IIc firmware fixes this bug.

We can not emphasize enough the need for carefully checking the version of the machine on which a program is running. It is also important to pay attention to the now obvious fact that even in the Apple IIc things can (and most probably will) move around, making any hard-coded slot assignment a sure source of incompatibility. To ensure compatibility, **scan the slots**.

The *Apple IIc Technical Reference Manual, Second Edition* describes how to enable buffering. Using serial buffering as an example, the pertinent instructions in the manual should now be understood as meaning:

Using Serial Buffering Transparently

```
If (machineID = "Memory Expandable IIc") then
  begin
    aciabuf = $04FC;    {Newest IIc with Expanded Memory Capabilities,}
    twser = $057C;      {ROM versions 3 and 4.}
    trser = $067C
  end
else
   begin
     aciabuf = $04FF;    {Original IIc and 3.5 ROM IIc}
     twser = $057F;
     trser = $067F
   end;

Set_Location aciabuf to $C1 or $C2.
Set_Locations twser and trser to $0.
```

Using Serial Interrupts Through Firmware

```
Set_Location aciabuf to a value other than $C1 or $C2
```

Further Reference

- *Apple IIc Technical Reference Manual, Second Edition*
- Apple IIc Technical Note #7, Existing Versions
- Apple II Miscellaneous Technical Note #7, Apple II Family Identification

Apple IIc
#7: Existing Versions

Revised by: Matt Deatherage November 1988
Written by: Guillermo Ortiz November 1987

This Technical Note describes the main differences between the five different IIc ROM versions which encompass the original IIc and four revisions.

Original IIc ($FBBF = $FF)

- Can use the IIc external drive only
- No AppleTalk firmware
- `PR#7` boots the second drive
- Mouse firmware maps to slot 4
- Serial firmware does not mask incoming linefeed characters
- Serial firmware does not support XON/XOFF protocol

3.5 ROM IIc ($FBBF = $00)

- Can use the IIc external drive and the UniDisk 3.5 drive
- AppleTalk firmware maps to slot 7
- `PR#7` returns the message “AppleTalk Off Line”
- Mouse firmware maps to slot 4
- Serial firmware defaults to mask all incoming linefeed characters
- Serial firmware supports XON/XOFF protocol

Original “Memory-Expandable” IIc ($FBBF = $03)

- Can use the IIc external drive, the UniDisk 3.5 drive, and the IIc Memory Expansion Card
- Mouse firmware maps to slot 7
- No AppleTalk firmware
- `PR#7` kills the system
- Serial firmware defaults to mask all incoming linefeed characters
- Serial firmware supports XON/XOFF protocol

Revised "Memory-Expandable" IIc ($FBBF = $04)

Same as Original Memory-Expandable, plus:

- Keyboard buffering firmware bug fixed
- Firmware returns correct information when the Memory Expansion Card is not present

Apple IIc Plus ($FBBF = $05)

- Can use the external IIc drive, the UniDisk 3.5 drive, the Apple 3.5 drives, but not the original IIc Memory Expansion Card.
- Contains a Memory Expansion Card connector
- 3.5" internal drive replaces 5.25" internal drive
- Mouse maps to slot 7
- `PR#7` kills the system
- 4 MHz 65C02 microprocessor
- Accelerator chip and static RAM cache permit operation up to 4 MHz
- Keyboard replaced with Apple Standard Keyboard (minus numeric keypad)
- Internal power supply
- Internal modem connector
- Serial ports refitted with mini-DIN 8 connectors
- Headphone jack has been removed
- Volume control relocated above the keyboard
- 40/80 column switch replaced by keyboard (Sholes/Dvorak) switch

Further Reference

- *Apple IIc Technical Reference Manual, Second Edition*
- Apple IIc Technical Note #5, Memory Expansion on the Apple IIc
- Apple IIc Technical Note #6, Buffering Blues
- Apple II Miscellaneous Technical Note #2, Apple II Family Identification Routines 2.1
- Apple II Miscellaneous Technical Note #7, Apple II Family Identification
- Apple II Miscellaneous Technical Note #8, Pascal 1.1 Firmware Protocol ID Bytes

Apple IIc
#8: Single-Sided 3.5” Media and the Apple IIc Plus

Written by: Llew Roberts May 1989

This Technical Note describes a media limitation on the internal drive of the Apple IIc Plus.

With the exception of the internal drive on the Apple IIc Plus, single-sided 3.5” disks are supported on all Apple 3.5” drives, including external disk drives connected to an Apple IIc Plus. The IIc Plus internal disk drive assumes that all disks have an 800K capacity, so it returns valid reads on blocks which occur on the formatted side and I/O errors on blocks which occur on the unformatted side. A disk may appear to work when the disk-reading algorithm has read blocks only from the formatted side.

For these reasons, we suggest that you do **not** ship programs on single-sided media.

Further Reference

- *Apple IIc Technical Reference Manual, Second Edition*

Apple IIc
#9: Detecting VBL

Written by: Dan Strnad November 1990

This Technical Note describes how the `VBLInt` flag at $C019 behaves differently than documented in the *Apple IIc Technical Reference Manual*, Second Edition, when being polled.

The *Apple IIc Technical Reference Manual*, Second Edition, claims that reading $C019 reads and resets the `VBLInt` flag. This is not correct. After reading $C019 once the high bit has been set to flag VBL, the high bit remains set. A program polling VBL at $C019 would have to access either `PTrig` at $C070 or `RdIOUDis` at $C07E to reset the high-bit for $C019. Note that `IOUDis` must have been turned off by writing to $C07F then `ENVBL` accessed at $C05B in order to poll for $C019 on the IIc.

Developers are encouraged to detect VBL using an interrupt handler with one of the VBL-based mouse modes on the IIc, if possible.

Further Reference

- *Apple IIc Technical Reference Manual, Second Edition*