APPLE//e TECHNOTE #1

Revision of notes on the Apple//e Dec 82*
5-July 84

This technote explains the difference between the Apple//e and Apple][+. It also provides a quick reference for the softswitches and makes some programming suggestions.

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

TECHNICAL OVERVIEW OF THE APPLE IIe

This document is designed for software developers who have some familiarity with the Apple II. Its function is to provide a quick overview of technical information that may affect software design and to a lesser extent hardware design. It is by no means a complete description of the Apple IIe as the manuals provided with the product serve this purpose. An effort has been made to extract from the manuals information that is not obvious. An effort has also been made to point out potential problems resulting from the new design and, where appropriate, to give suggestions on how to avoid the problems.

GENERAL:

1. Full ASCII keyboard with auto-repeat feature, alpha lock and Apple keys.

2. Custom-ICs are used for memory management and I/O control thereby reducing chip count.

3. The language card and slot 0 have been replaced by built in (look alike) RAM called bank-switched memory.

4. An additional slot has been added. It is called the auxiliary slot. This slot has several functions:
   a. It is used for testing and diagnostics.
   b. It serves as the slot for the 80 column card (logically it is mapped as being in slot 3 - $C300).
   c. It serves as the slot for the 80 column / 64K ram card.

5. The back panel is designed for direct mounting of DB-9, DB-19 and DB-25 connectors. This feature allows peripherals to be attached to the back of the Apple IIe rather than to the peripheral cards.

6. It looks like an Apple II.

7. In addition to an introductory booklet,the APPLE IIe OWNER'S manual, there is an APPLE IIe 80-COLUMN TEXT CARD manual, an EXTENDED 80-COLUMN TEXT CARD SUPPLEMENT, and an APPLE IIe REFERENCE manual. Also new revisions of the APPLESOFT TUTORIAL, DOS and APPLESOFT REFERENCE manuals have been written. Additional documentation such as A GUIDE TO THE NEW FEATURES OF THE APPLE IIe COMPUTER and Technical Support handouts have been developed.

8. Applesoft has not been changed at all. Integer BASIC can still be used; it must be loaded into the bank-switched memory (language card).

9. The Autostart ROM has been replaced by a new ROM capable of supporting 80 columns. The autostart entry points have been maintained.

10. The Apple IIe with its auxiliary card functions like an Apple II plus with a language card, upper and lowercase capability and an 80 column card. This means that software and hardware which would operate

improperly in an Apple II plus equipped with these features will probably operate improperly in an Apple IIe. NOTE: the Apple IIe will shift to upper case only if the Alpha Lock key is pressed down.

KEYBOARD:

1. All ASCII code-generating keys will start repeating if held down for more than half a second.
   a. If another key is momentarily pressed while a key is repeating, the new key will begin to repeat.
   b. Since the escape key repeats, care must be taken when using it. In a series of escapes every other escape cancels the effect of the previous escape.

2. ASCII codes
   a. The delete key issues the ASCII code 127 (DEL). In immediate mode a checkerboard is displayed to signify that this key has been pressed.
   b. Up arrow key code is 11, down arrow key code is 10, tab key code is 9.

   c. Pascal 1.1 was designed to funcion with the Apple II keyboard. It therefore has some keyboard related differences when run on the Apple IIe. For example, if the up arrow key is pressed while running Pascal 1.1 without the 80 column firmware active a [will be displayed. Another such difference results in the user needing to press shift-2 in order to generate the @ sign. On the Apple II the @ key was obtained by pressing shift-P.

   NOTE: BASIC still adds 128 to its ASCII codes to signify a keypress.

3. Open & Closed Apple keys
   a. They do not modify or generate ASCII codes.
   b. They cannot be detected by looking at $C000 the normal keypress method.
   c. They are connected directly to the game push buttons. Their key press can be detected by looking for a game push button being pressed. Their presence means there are always game push buttons. This will cause problems for games that determine that there are game paddles by the presence of push buttons. If a joy stick is connected to the Apple IIe and it has a way of locking down one of the buttons or if it is of the Atari type, which has reverse polarity, then when the computer is turned on or control-reset is pressed the computer detects what appears to be an Apple key being pressed and so goes through diagnostics each time reset is pressed.
   d. In combination with control-reset
      1. Open Apple does a cold start after scrambling several bytes of RAM. This combination of key presses replaces the 'give up and start all over again ' key (power switch). It can also substitute for PR#6 with the added advantage that if 80 column firmware is active it will be properly disconnected. PR#6 disconnects the 80 column firmware but leaves the 80 column hardware enabled resulting in improper functioning of the text screen.
      2. Closed Apple sends the computer through an onboard diagnostics test used during production testing. This is only a partial

diagnostic.

3. Pressing both Apple keys results in the diagnostic test being run with output to the speaker.

NOTE: After the diagnostics test has successfully been completed the rather non-informative message "kernel OK" is given. This message means that the diagnostics are completed and that no problems were found. Press reset to reboot the system.

4. The Caps lock key must be down to create the upper case letters needed for BASIC and DOS commands. BASIC permits lower case letters within quotes. When 80 column firmware is active a restrict mode may be selected that will automatically shift anything outside double quotation marks to upper case. Restrict mode is entered by typing PR#3 then escape followed by R.

5. The ARROW keys in combination with ESCAPE function in the same way as I,J, K & M. NOTE: If you want to use an ARROW key to copy something from the screen after 'escaping' to that line press some other key to deactivate the escape mode prior to using the arrow key otherwise you will just continue escaping and will not do any copying.

6. The shift key mod may be duplicated by soldering a 500 ohm resister across the logic board connector at X6..
NOTE: the warranty is voided by doing this.

7. Integer BASIC was not designed to recognize the full set of ASCII characters so some of the keys on the Apple IIe that are not on the Apple II plus may do strange things. Integer BASIC will recognize normal lower case characters but treat them as upper case. Integer and Applesoft BASIC running in bank-switched memory can be made to recognize lower case as explained below.

8. Pressing reset on an Apple IIe does a full 64K reset where the Apple II resets only the lower 48K. This is most noticable by Pascal users who have only one disk drive since they must press reset part of the way through boot-up which then starts the booting process over again. In general a program which runs in the bank-switched or auxiliary memory should set up reset jump vectors which bank in the appropriate memory before jumping back into the program. These jump vectors need to be in the lower 48K of main memory.

9. Pressing reset sets the monitor routines to display video in NORMAL mode. A reset does not inform Applesoft that it should be displaying in NORMAL mode and so it continues to fiddle with the output characters' ASCII code. On the Apple II plus this situation was not detectable. On an Apple IIe the effects of this can be detected. From immediate mode type FLASH then press reset. If you then try to print something to the screen or make a listing some characters are not displayed correctly - (numbers become lowercase letters). Giving any of the following commands corrects the situation: NORMAL, INVERSE, FLASH.

VIDEO OUTPUT:

1. When you boot the Apple IIe via PR#n, DOS 3.3 does not initialize the 80 column firmware nor turn the card on or off. If the 80 column firmware was active when the disk was booted then after booting, the 80 column hardware is still on but the firmware is not. This makes for garbage on the screen. Using control-Open Apple-Reset rather than PR#6 prevents this. Programs that use 80 columns should turn 80 columns off (PRINT CHR$(21)) at the end of the program. To ensure that your program will not be clobbered by this half in and half out situation your program needs to completely turn all of the 80 column card on or off. This is how. First determine the configuration of the computer by using the identification routines. If the computer is an Apple IIe with an 80 column card then do a PR#3 to turn the card on and then if you don't want 80 columns issue the command PRINT CHR$(21) to turn it off.

   NOTE: other 80 column cards on the Apple II or Apple IIe using DOS will have similar problems.

2. To turn the card on from BASIC the command PR#3 must be typed in or the program must issue PRINT CHR$(4);"PR#3". Pascal programs will automatically turn the card on. Runtime Pascal programs may be designed to prevent this from happening. From the monitor type C300G to turn the card on. Assembly language requires a JMP to $C300. Issuing these commands when there is no card in slot 3 or the auxiliary slot will do undetermined things. The Apple IIe will usually reboot the disk drive rather than hang like in the Apple II plus but anything may happen. Reminder issuing a PR#3 to turn on the 80 column card is like issuing a PR#1 to turn on a printer. These commands only set software pointers and the periperal is not actually initialized until the first character is sent to the peripheral. For this reason any screen setting such as VTAB issued after PR#3 but before a PRINT command will be ineffective since it will be changed when the peripheral is initialized.

3. Presence of the 80-column card in the video expansion slot can be determined by writing to a screen location on the card and then checking to see that the value found at that location is the value written. i.e. RAM exists at that location.

4. The Apple IIe can be identified by a six at $FBB3 (64435). Licensed developers can obtain full identification routines from Apple's PCS Technical Support Group.

5. 80 column features are controlled from programs by printing control characters. For example, PRINT CHR$(21) deactivates the 80 column card if one exists. It does nothing if there is no card.

6. 80 column features are controlled from immediate mode by using escape sequences. For example, typing the escape key (don't hold it down) followed by control-Q deactivates the 80 column card.

7. If 80 column firmware is active it can display either 40 or 80 column text.

8. The cursor may be used to identify the status of the computer.

a. BLINKING CURSOR means autostart ROM is active. This will happen if an image of autostart ROM has been placed in the bank-switched memory (language card) and control has been turned over to it.
b. FLASHING CHECKER BOARD CURSOR means the new monitor firmware is active and that the 80 column features are inactive.
c. SOLID INVERSE SPACE CURSOR (40 or 80 column width) means the 80 column features and the new monitor ROM are both active.
d. INVERSE PLUS SIGN CURSOR (40 or 80 column width) means the 80 column features and the new monitor ROM are both active and the escape key has been pressed.

9. The 'other half' of the 80 column screen is located in 1K of RAM on the 80 column or extended 80 column card. The address range of this RAM is from $400 to $7FF (text page one). Data being displayed from this 'other half' is shuffled in with data from text page one on the main board and shrunk by the I/O control chip to produce the 80 column screen. Data from the main board is displayed as the odd columns while data from the 'other half' -the 80 column card- is displayed as the even columns. Display columns are numbered starting from one.

10. A vertical blanking signal is available at $C019 to help graphics animators. Vertical blanking takes approximately 4 milliseconds. Updating of screen characters or switching of pages during vertical blanking prevents the displaying of graphics while the graphics is being updated.

11. The auxiliary slot for the 80 column card has the same memory mapping as slot 3. Therefore, the 80 column card is treated like a peripheral.

12. If the 80 column firmware is inactive and reset is pressed while running in autostart ROM on the language card (either Integer or Applesoft BASIC), the computer returns control to that language (i.e., the language card) without change--DOS does it.

13. Reset deactivates the 80 column card firmware and hardware.

14. Using commas to tab while in Applesoft does not work with 80 columns. You can tab with 80 columns by poking 36,n where n is the column you want to tab to. Poking 36,n will also work for 40 column display as long as n is < 40. Poking 36 with a number >39 probably will cause the program to crash if displaying in 40 columns.

15. If Integer or Applesoft BASIC is running in the bank-switched memory (language card) with the autostart ROM then they will not accept lower case characters. If you want to be able to accept lower case characters then type PR#3 to activate the 80 column firmware. This will replace the autostart ROM by the new ROM. The card may then be deactivated with an escape control-Q and the new monitor ROM will continue to accept lower case.

16. The 80 column display running under the new ROM is markedly slower then 40 columns under autostart ROM.

17. Scrolling windows can be set up anywhere on the 80 column screen. The

width of a scrolling window is limited to an even number. If an attempt is made to set up a odd-numbered window width the width is reduced by one. Therefore, if an attempt is made to set a window width equal 31 by placing the number 31 in the location $33 the actual window width will be 30.

18. There are two built in character sets.
    a. The standard character set displays uppercase characters in NORMAL, INVERSE, and FLASH as is the case with the Apple II plus. It will also display lower case NORMAL characters. NOTE: do not use lowercase normal characters in programs you want to run on an Apple II plus since it cannot display lower case characters.
    b. The alternate set provides for upper and lower case in both normal and inverse.
    c. Atempting to display lower case inverse characters without having the 80 column firmware switched in will not work. Even if the alternate character set is banked in these characters will be displayed as special characters.

    d. When the 80 column firmware is activated the alternate character set is used. This means that software designed to be used with 80 columns must be designed for the alternate character set. Attempting to switch to the standard character set while 80 column firmware is active will result in some of the characters being misinterpreted by Applesoft.

    e. Since both character sets are designed to display the underline character and the descenders of lower case letters, all the characters have been moved up one row of dots. This may cause some visually unpleasant lettering on the top row of a text display which uses inverse video. This can be corrected by placing one row of inverse blanks above the first line of print.

19. Unlike the Apple II plus the Apple IIe's GETLINE routine is affected by the INVERSE FLAG (location $32) setting. On the Apple II plus all BASIC input or Assembly language input using GETLINE is displayed in normal mode. On the Apple IIe input will be displayed in accordance with the value in location $32-(inverse, normal, flash). This is most noticable while in immediate mode. Typing the BASIC command INVERSE results in future keypresses being displayed in inverse. HOME and clear-to-the-end-of-line gives inverse blanks.

20. When displaying in 80 columns, if you look at CH (36) you will find it = 0 even if the cursor is not at the left edge of the screen. This is done to fool BASIC which knows only about 40 columns and to provide windows. Some other 80 column boards set this location to 40. Placing a value into 1403 ($57B) performs an HTAB to that value.

21. Some of the I/O Scratchpad RAM Addresses located in the text screen buffer are used by the 80 column firmware. These are used in accord with the protocol for their use but some programs may have used these areas incorrectly and will have problems. The most common abuse of these protocols is when a lo-res picture is BLOADED into $400-$7FF. When this is done the values in the scratchpad area are changed to match what they

were when the picture was saved. In the past the most common result is that the disk drive 'grinds' since the read head gets lost. A similar 'loss of control' will happen to any peripheral including the 80 column card. Since slot 3 in the Apple IIe is dedicated to the 80 column firmware it uses one scratchpad area dedicated to slot 3 even if there is no 80 column card. Therefore, any BLOADing into the area $400 to $7FF will affect the operation of the output routines, possibly crashing the program. The solution is to BLOAD the picture into a buffer and then move all but the scratchpad area into the screen buffer. A simpler solution, but one that may crash the program if an interrupt occurs during loading, is to save the scratchpad data then restore it after loading the picture. From machine language you could disable interrupts during this operation.

VIDEO SOFT SWITCHES:

1. ALT. CHAR SET - This switch sets up the alternate character set. This switch should be used with care by BASIC programmers as previously pointed out.

2. 80 STORE - If 80 store is active then the PAGE2/NOT PAGE2 switch serves as a bank switching switch rather then video display switch. This is true of the hi res pages also if the extended 80-column card is present. This switch should be used only by experienced programmers.

3. 80 COLUMNS - This switch is designed to assist assembly language programmers who are using their own screen writing routines. This switch turns only the disply hardware on and not the firmware. Programs which use the monitor I/O routines COUT & RDKEY cannot use this switch alone but must use it in combination with PR#3. This is true for both BASIC and the Monitor.

4. TEXT/GRAPHICS, MIXED/NOT MIXED, PAGE2/NOT PAGE2 and HIRES/NOT HIRES serve the same function as in the Apple II plus. The PAGE2/NOT PAGE2 switch serves the additional function of bank switcher as mentioned above. The state of these switches may be found by reading status bytes.

5. VERTICAL BLANKING can be read to determine correct display timing for animated graphics.

6. Two soft switches affect the input/output memory space ($C100 to $C7FF).
   a. The SLOTC3ROM switch is used to select between the space alocated to slot 3 and built in ROM alocated to controling Apple's 80-column cards.
      1. When the computer is reset or turned on it checks for a card in the auxiliary slot. If it finds one the SLOTC3ROM switch is turned off. This banks in the built in C3xx ROM. NOTE: this does not turn on the 80-column card. It simply provides the card with the ROM it will need if a PR#3 or equivalent command is given.
      2. This switch may be turned off -built in ROM banked in- and the 80-column ROM used even if there is no 80-column card. To get into this mode turn off the switch (POKE 49162,0) and give the

PRINT chr$(4);"PR#3" command. Without a card, 80 columns cannot be displayed but features such as upercase restrict are available.

b. The SLOTCXROM switch is used to select between the space alocated to slots 1 through 7 and built in ROM alocated to controling Apple's 80-column card and the built in diagnostics.
   1. If the SLOTCXROM switch is on then the 80 column firmware is mapped in even if the SLOTC#ROM switch is off.
   2. The SLOTCXROM is turned on when one or both of the Apple keys is pressed during reset.

7. The state of the soft switches may be found by reading the appropriate memory locations. With the exception of the SLOTCXROM switch if the value read is >127 then the switch is on.

8. When Pascal 1.1 is initialized it turns on the following soft switches: HIRES, TEXT, NOT-MIXED & NOT-PAGE2. If the 80-column firmware is turned on then 80 STORE is turned on. Since it is not intuitive that the HIRES switch is on even when the program does not use hires and that 80 STORE is on even when not storing things in memory some unexpected things may happen. The unexpected events have most impact on programs directly writing to the text screen and programs using the auxiliary memory. Note that if your program turns off the 80 STORE switch it must turn it back on before it tries to use the 80-column firmware to display 80-column text.

MEMORY MAPPING & ADDRESSING:

1. The memory mapping of the Apple IIe matches the Apple II plus with a language card. Soft switches and the new monitor firmware may be used to bank in additional ROM and RAM.

2. Like the 6502 in the Apple II plus, the 6502A used in the Apple IIe activates the address bus twice during successive clock cycles during an indexed store operation. This may cause a device that toggles each time it is addressed to end up back where it started. In these cases read operations should be used rather than stores.

3. The $D000 to $FFFF memory space functions in a method identical to the language card on the Apple II plus but since it is built in it is referred to as bank-switched memory.

4. Pressing reset switches out bank-switched memory. If operating under DOS 3.3, DOS will switch back to bank-switched memory.

5. While in 80 columns the 1K of auxiliary RAM being used is from $400 to $7FF. The 80 column text card with 1K of RAM uses sparse memory mapping. this means that writing to the location $C00 or $800 on the card is the same as writing to the location $400.

AUXILIARY RAM:

1. 1K of additional RAM exists on the standard 80-column card in address

space $400-$7FF. 64K of additional RAM exists on the extended 80-column card in address space $0000 - $FFFF. This additional RAM is banked in by addressing (writing to) soft switches. To determine if main memory or auxiliary memory is banked in look at the appropriate status bytes.

2. The following softswitch pairs switch between main RAM and auxiliary RAM in the specified ways:
   a. RAMRD - The setting of this switch affects which bank of memory is being read if the read operation is between memory locations $200 and $BFFF.
   b. RAMWRT - The setting of this switch affects which bank of memory is being written to if the write operation is between $200 and $BFFF.
   c. ALTZP - The setting of this switch pair affects which bank of memory is being written to and read from if the read or write operation is to a memory location between $0 and $1FF or between $D000 and $FFFF.
   d. 80STORE - This switch pair in combination with the PAGE2, HIRES and TEXT switch pairs determine in a complex way what display memory is being written to and read from. In general it changes the other switch pairs' functions from screen switching to bank selection. The memory being affected is the same as would be affected by the screen switching.

3. Switching auxiliary memory does not affect the bank-switched memory (language card)$D000 - $FFFF settings. If main board ROM is banked in and then auxiliary memory is switched in, the main board ROM is still active. If bank-switched memory is active and aux mem is switched in then the bank-switched memory in the auxiliary memory will be active.

4. The auxiliary memory provides storage and program expansion capabilities for BASIC, PASCAL and Machine language programs. Machine language programs can very effectivly use the extra memory since the program itself can run in the extra memory if need be. BASIC can use the extra memory to store machine language routines and pictures. Pascal can use the extra memory to store machine language procedures. Both BASIC and Pascal programs are limited to using the standard memory areas since they are unaware of the extra memory. With care BASIC programs could be CHAINED using the extra memory rather than a disk. Also, several programs could be in the computer at once with the posibility of one being in BASIC and the other in Pascal.

5. Programs using DOS would need to do all their input and output from either the main memory or the auxiliary memory but not from both unless a copy of DOS were placed in both banks and then both were kept informed of such events as switching the output device. If programs in auxiliary memory need to produce input or output which is not a DOS command they may use the routines COUT1 and KEYIN which do not go through DOS. Great care should be used since 80STORE may need to be used to affect where the output goes.

6. If you write data into the $400 to $7FF space in auxiliary memory and the computer is displaying 80 columns then your data will appear on the screen.

7. The routine AUXMOVE moves a block of data from anywhere in the memory area $200 to $BFFF. Data may be moved from auxiliary memory to main memory or from main memory to auxiliary memory.

8. The routine XFER transfers program control from a machine language program in main memory to one in auxiliary memory or the other way around.

INPUT /OUTPUT:

1. Sending control to a slot which does not have any device connected to it constitutes a NO-NO. In the Apple II plus this NO-NO usually resulted in the computer stopping dead. In the Apple IIe this NO-NO usually results in the disk booting.

2. The SLOTC3ROM soft switch pair selects between internal ROM at $C300 (the 80 column firmware) and slot three.

3. The SLOTCXROM soft switch pair selects between internal ROM from $C100 to $C7FF used by the built in diagnostics and 80 column firmware, and slots one through seven.

4. Very large peripheral cards which stick out the back of the computer will not be able to do so because of the new back panel.

5. Cards which depend on 'piggy backing' to IC sockets to obtain additional signals will probably no longer function properly since the main board is re-designed.

6. The Apple IIe's 80 column card is a peripheral. As is the case with the Apple II plus, two peripherals cannot receive input or send output at the same time. This means that the 80 column firmware must be made inactive before using an output device such as a printer or MODEM and will need to be reactivated when returning to 80 columns. Some software such as the Pascal BIOS does this automatically.

7. There are two locations on the Apple IIe designed for plugging in game paddles. One is a DB-9 connector on the back panel and the other is the same as on the Apple II plus. A game paddle or joy stick may be connected to one or the other location but not to both at the same time.

PASCAL 1.1

As explained earlier the Pascal system and the 80 column firmware when running under Pascal set several soft switches which may create unexpected situations because their settings are not intuitive. Namely, Pascal turns on the HIRES switch during initialization and the 80 column firmware turns on the 80STORE soft switch. As a result of these settings the following unexpected situations may occur.

1. If an Assembly language Pascal procedure is designed to store and retrieve data and it tries to do so from locations $2000 to $4000 in the auxiliary memory it will not do so properly. This is because the HIRES switch is on along with 80STORE and it overrides the RAMRD and

RAMWRT switches. If HIRES is switched off then this area may be used.

2. If a program turns on the PAGE2 soft switch the program may self destruct. If HIRES and 80STORE are still on when PAGE2 is turned on, any data being sent to or retrieved from the $2000 to $4000 memory space will be sending and retrieving from the auxiliary memory space rather than the expected main RAM. If the 80 column card does not have auxiliary memory then the data is coming from and going to thin air. Since the Pascal heap in larger programs will grow into this space the program self destructs. If you must have the PAGE2 switch on then turn off one of the other two switches. NOTE: Pascal does not use the PAGE2 switch to display text or graphics but through trickery it can be turned on.

BACK PANEL:

1. The back panel is designed to support the mounting of DB9, DB19 and DB25 connectors. Cables from peripheral cards run to these connectors. External cables then run from these connectors to the peripheral.

2. The four DB19 mounting holes are reserved for disk drive connections.

3. The Apple IIe's accessory kit contains materials for attaching cables designed for the Apple II plus.

4. Peripherals which use more than 25 lines will need to use two or more of the DB connectors to route their wires through the back panel.

HARDWARE:

1. The Apple IIe uses the 6502A but it still runs at one MHz.

2. There is a 470 ohm resistor on both the open apple and closed apple key. These resistors are on the keyboard.

3. The Apple IIe's data bus is now buffered and may cause timming differences in connection with using the DMA line.

4. The shift-key mod used in the Apple II+ to simulate upper case can be simulated in the Apple//e by soldering the solder blob found on the main board at location X-6.

INTERRUPTS:

When an interrupt occurs the Apple IIe saves the status of the text page (page1 or 2) and SLOTCXROM switches, then sets the page to page 1 and SLOTCXROM to Slot ROM. After the interrupt has been handled these two switch settings are restored.

DOCUMENTATION ERRATA:

1. To connect the game input switches (push buttons) to other hardware use aprox. 500 ohm pull-down resistor connected to ground and a momentary-contact switch to +5V.

2. The MOVE routine in the Apple IIe is the same as in the Apple II plus and therefor the 'Y' register should be set to 0 before calling it.

3. The SLOTCXROM switches are reversed. The slot ROM is selected by writing to 49158 ($C006). The internal ROM is selected by writing to 49159 ($C007).

ADDENDUM TO TECHNICAL OVERVIEW OF THE APPLE//e

1. An unusual condition appears on the text screen using an Apple//e when a text display is switched from inverse to normal or normal to inverse. This only takes place if the change is being made while printing to the bottom line of a scrolling window. If going from normal to inverse the text appears in inverse but the right end of the line is black which is just like on the Apple][+. If going from inverse to normal the text appears in normal but the right end of the line is white. This condition happens because when the screen is scrolled after the printing of the last line, the new bottom line is filled with blanks in the current mode (inverse or normal). This cleans off the old text on that line in preperation for printing text on the line. The screen display is then switched to the new mode and the last line is printed. This condition can be corrected if you must change text modes on a scrolling window. To do this end the last print statement with a semicolon to suppress the scrolling. Follow this by the change of mode and a print statement without any text.

2. If the HOME command is given on an Apple//e while the text mode is in inverse the whole screen becomes white. On the Apple][+ the screen would clear to black..

3. The following is a list of all the special use locations in memory locations $C000 through $C0FF. Note that in some cases different switches are activated depending upon if they are read from (PEEK, LDA) or written to (POKE, STA). If reading a value can indicate the state of a soft switch, the state having the symbol (*) is the state which will return a value greater than 128 ($7F).

LOCATION	EFFECT OF READING	EFFECT OF WRITTING
49152 ($C000)	get keyboard input	pg1&2 sw show diff txt & gr buff
49153 ($C001)		pg1&2 sw bank swich txt & gr buff
49154 ($C002)		read from main memory
49155 ($C003)		read from auxiliary memory
49156 ($C004)		write to main memory
49157 ($C005)		write to auxiliary memory
49158 ($C006)		select card ROM all slots
49159 ($C007)		select internal ROM $C100-$CFFF
49160 ($C008)		read & write main stack,z-pg.,LC
49161 ($C009)		read & write alt. stack,z-pg.,LC
49162 ($C00A)		select internal ROM $C300-$C3FF
49163 ($C00B)		select card ROM slot three
49164 ($C00C)		turn 80-column display off
49165 ($C00D)		turn 80-column display on
49166 ($C00E)		select Apple][char. set
49167 ($C00F)		select new full upper & lower char. set
49168 ($C010)	clear the keyboard strobe	clear the keyboard strobe
49169 ($C011)	indicates if LC first 4K bank one or (*)bank two is in	
49170 ($C012)	indicates if Autostart ROM or (*)LC is banked in	
49171 ($C013)	indicates if main or (*)aux RAM is being read from ($200-$BFFF)	
49172 ($C014)	indicates if main or (*)aux RAM is being written to ($200-$BFFF)	
49173 ($C015)	indicates if card or (*)internal ROM being read ($C100-$CFFF)	
49174 ($C016)	indicates if main stack,z-pg.,LC or (*)aux stack,z-pg.,LC	
49175 ($C017)	indicates if internal or (*)card ROM being read ($C300-$C3FF)	
49176 ($C018)	indicates if storing to main or (*)aux text & graphics buffers	
49177 ($C019)	indicates if vertical blanking or (*)not vertical blanking	
49178 ($C01A)	indicates if displaying graphics or (*)text	
49179 ($C01B)	indicates if displaying full page graphics or (*)mixed txt & gr	
49180 ($C01C)	indicates if displaying page 1 or (*)page 2	
49181 ($C01D)	indicates if displaying in lo-res or (*)hi-res	
49182 ($C01E)	indicates if using Apple][char set or (*)alternate char set	
49183 ($C01F)	indicates if displaying in 40 columns or (*)80-columns	
49184 ($C020)	toggle cassette output switch	
.		
.		
.		
49200 ($C030)	toggle speaker	
.		
.		
.		
49216 ($C040)	utility strobe single pulse	
.		
.		

```
  .
49232 ($C050)   turns graphics mode on          turns graphics mode on
49233 ($C051)   turns text mode on              turns text mode on
49234 ($C052)   turns mixed mode off            turns mixed mode off
49235 ($C053)   turns mixed mode on             turns mixed mode on
49236 ($C054)   display from page 1 buffer      display from page 1 buffer
49237 ($C055)   display from page 2 buffer      display from page 2 buffer
49238 ($C056)   display graphics as lo-res      display graphics as lo-res
49239 ($C057)   display graphics as hi-res      display graphics as hi-res
49240 ($C058)   turn annunciator 0 off          turn annunciator 0 off
49241 ($C059)   turn annunciator 0 on           turn annunciator 0 on
49242 ($C05A)   turn annunciator 1 off          turn annunciator 1 off
49243 ($C05B)   turn annunciator 1 on           turn annunciator 1 on
49244 ($C05C)   turn annunciator 2 off          turn annunciator 2 off
49245 ($C05D)   turn annunciator 2 on           turn annunciator 2 on
49246 ($C05E)   turn annunciator 3 off          turn annunciator 3 off
49247 ($C05F)   turn annunciator 3 on           turn annunciator 3 on
49248 ($C060)   indicates if cassette input toggle has no bit or (*)has a bit
49249 ($C061)   indicates if game push button 0 (open apple) is up or (*)down
49250 ($C062)   indicates if game push button 1 (closed apple) is up or (*)down
49251 ($C063)   indicates if game push button 2 is up or (*)down
49252 ($C064)   indicates if game controller 0 has timed out or (*)not
49253 ($C065)   indicates if game controller 1 has timed out or (*)not
49254 ($C066)   indicates if game controller 2 has timed out or (*)not
49255 ($C067)   indicates if game controller 3 has timed out or (*)not
  .
  .
  .
49264 ($C070)   game controller strobe          game controller strobe
  .
  .
  .
49280 ($C080)   select RAM read bank 2.  Write-protect RAM.
49281 ($C081)   select ROM read. Two or more successive reads write-enables RAM. bank 2
49282 ($C082)   select ROM read. Write protect RAM
49283 ($C083)   select RAM read bank 2. Two or more successive reads write-enables RAM ba
49284 ($C084)   select RAM read bank 2.  Write-protect RAM.
49285 ($C085)   select ROM read. Two or more successive reads write-enables RAM. bank 2
49286 ($C086)   select ROM read. Write protect RAM
49287 ($C087)   select RAM read bank 2. Two or more successive reads write-enables R  ba
49288 ($C088)   select RAM read bank 1.  Write-protect RAM.
49289 ($C089)   select ROM read. Two or more successive reads write-enables RAM. bank 1
49290 ($C08A)   select ROM read. Write protect RAM
49291 ($C08B)   select RAM read bank 1. Two or more successive reads write-enables RAM ba
49292 ($C08C)   select RAM read bank 1.  Write-protect RAM.
49293 ($C08D)   select ROM read. Two or more successive reads write-enables RAM. bank 1
49294 ($C08E)   select ROM read. Write protect RAM
49295 ($C08F)   select RAM read bank 1. Two or more successive reads write-enables RAM ba
49296 ($C090) - 49311 ($C09F) slot 1 device select
49312 ($C0A0) - 49327 ($C0AF) slot 2 device select
49328 ($C0B0) - 49343 ($C0BF) slot 3 device select
49344 ($C0C0) - 49359 ($C0CF) slot 4 device select
49360 ($C0D0) - 49376 ($C0DF) slot 5 device select
49376 ($C0E0) - 49391 ($C0EF) slot 6 device select
```

49392 ($C0F0) - 49407 ($C0FF) slot 7 device select

APPLE//e HARDWARE AND SOFTWARE GUIDE LINES

The following are some suggestions for writing programs for the Apple IIe.

ERAL:

1. Apple has developed interface routines which are designed to help profesional and amateur programmers write 'friendly' interfaces for their programs. These routines also help the programmer avoid some pitfalls associated with using 80-columns. These routines are part of the Applesoft Extension Package. It can be found on the disk supplied with the Apple//e Applesoft Tutorial and Reference Manual. Appendix E of the new Applesoft Tutorial explains how to use this and other supplied routines. A 6502 Machine Language version of these routines will be available soon.

2. Apple has made every effort to maintain the subroutine entry points in the Autostart ROM when the Apple//e ROM was written and will continue to do so in future revisions. This implies that if you use only the entry points supported in the Apple II or Apple//e Reference Manuals your programs should not need to be modified for future revisions. It also implies that if you enter at other locations or if you do such activities as check-summing the ROM, your product may need to be reved when the ROM is reved.

   ogrammers be forwarned
   pple gives no assurance that any locations within the 80-column firmware ($C100-$CFFF) will be maintained. Therefore, programmers should not attempt to 'patch into' any of these routines. The 80-column firmware also uses several 'scratch pad' locations. At this time the only such location which will be maintained between revisions is location 1403 ($57B) which gives the current horizontal cursor location for 80 columns.

3. Use the procedures outlined in the IDENTIFICATION ROUTINES document to recognise the hardware that is available. These routines are available to licensed software developers from PCS Marketing Technical Support.

TWARE SPECIFIC:

1. Before using a peripheral for output be sure the 80 column firmware is inactive.

2. Do not require the use of the reset key during program operation unless you are not concerned that the bank-switched RAM will be switched out.

3. If your software turned on the 80 column firmware be sure it turns it off before ending.

4. Do not check for the absence of game control paddles by having your program 'look' to see if both game buttons have been pressed. An alternate method is to timeout the paddles for, say, twice as long as the normal count of 256; if the 558 timer chip still doesn't timeout, there must not be any paddles.

5. Make sure that an 80 column card exists prior to trying to turn it on since not doing this will lead to unpredictable results.

6. If your program requires DOS or BASIC commands to be typed be sure to instruct the end user to use upper case letters or better still use your program to shift input to upper case.

7. Applesoft BASIC was designed to produce flashing characters. Because of this, incorrect characters appear when lowercase inverse or flashing characters are displayed by an Applesoft program using the standard 40 column display. If an Applesoft program first determines that the 80 column card is there it may correctly display the lowercase inverse characters by turning on the card. A full set of lowercase flashing characters is not available.

8. If your program expects certain string input design it to accept both upper and lower case.

9. Never have your program issue the PR#0 or IN#0 commands while the 80 column card is active.

10. A program running under DOS should turn the 80 column card on by the command PRINT CHR$(4);"PR#3".

11. If your program is generating animated graphics you might want to use the vertical blanking signal to prevent 'blinking'.

12. The 80 COL soft switch $C00D (49165) should not be used if monitor input / output routines are used.

13. If your BASIC / Assembly Language software boots to run, include in your documentation the need to boot by pressing control - open apple - reset rather than by entering a PR#. This is to ensure that the hardware and firmware are in sync. An alternative if you are willing to put up with a momentary flash accross the screen is to have your greeting program's first actions be the following. First, it should determine if an 80 column card is in the system. If one is, then turn the firmware on using the PR#3 command. Finally, if you do not want the card on you may turn it off with a Print CHR$(21) command.

14. If your program is a BASIC program and it uses 80 columns then do not use commas to do tabbing. Instead use POKES to 1403. For example POKE 1403,10 TABS to the 10th. column.

15. If your program has 80 column firmware active (either 80 or 40 columns displayed) and you want to send output to a printer or other output device you must turn off the 80 column firmware before you turn on

the other device. The following is an example: Use Home to clear the screen. Turn off the 80 column firmware by issuing a control character to the screen(PRINT CHR$(21) -control-U). Turn on the printer. When printing is completed or you want an intermediate message on the screen urn the printer off with a PRINT CHR$(4)"PR#0" & PRINT CHR$(4)"IN#0". Then turn the 80 column firmware back on with a PRINT CHR$(4);"PR#3". If you must have a message on the screen durning printing then place the message on the screen (40 columns) after the 80 column firmware is turned off but before turning on the printer. NOTE: the PR#0 and IN#0 is not required by Apple's card but may be by other cards.

16. If the 80 column firmware is active the BASIC GET command and the monitor KEYIN routine will immediatly execute the escape keypress and so escape codes are not available. Therefore, do not use these 'GET' commands when escape sequences are required and the 80 column firmware is active.An Assembly Language or BASIC routine which properly get input by looking at 49152 ($C000) can be used to detect an escape key being pressed.

17. If your program uses a reset trap or in some way is designed to recover from a reset and it uses the bank-switched memory (language card) it must turn the bank-switched memory back on. This would be done by having your reset jump vector point to a reset routine placed somewhere in the the lower 48K of memory. This routine would need to turn the bank-switched memory back on before jumping back into the program.

RDWARE:

1. Don't use any of the four DB19 slots in the back panel since these are reserved for disk drives.

2. Cables should connect to the card at the keyboard end of the card since this gives the user more freedom in selecting the slot into which the card is to be installed. It also prevents cable cramping.

3. Cables should use DB9 or DB25 connectors.

4. Cards which require 'piggy backing' into IC sockets may become obsoleted by this and future revisions of the main board.

5. Do not require cards to be placed in slot three if they are intended to be used in systems having the 80 column card.

6. Cables using DB-25 connectors for parallel I/O devices should block pin seven. This convention should be followed to prevent damage should the connector be accidently pluged into a serial device. Serial devices use this pin for ground.

7. Cards should be identifiable according to the protocol outlined in Pascal's ATTACH document which is excerpt here.

   Pascal 1.1 uses four firmware bytes to identify the peripheral card. Both the identifying bytes and the branch table are near the beginning of the $Cs00 ROM space. The identifiers are listed in

Table 1.

Address	Value
$Cs05	$38
$Cs07	$18
$Cs0B	$01 (the Generic Signature of new FW cards)
$Cs0C	$ci (the Device Signature; see below)

Table 1. Bytes Used for Device Identification

The first digit, c, of the Device Signature byte identifies the device class as listed in Table 2.

Digit	Class
$0	reserved
$1	printer
$2	joystick or other X-Y input device
$3	serial or parallel I/O card
$4	modem
$5	sound or speech device
$6	clock
$7	mass storage device
$8	80-column card
$9	network or bus interface
$A	special purpose (none of the above)
$B-F	reserved for future expansion

Table 2. Device Class Digit

The second digit, i, of the Device Signature byte is a unique identifier for the card, assigned by Apple Technical Support.

NOTE: Our 80 column card identifier is $88

APPLE //e TECHNOTE #3

Original Version
Published by Softalk Magazine
Sept. 1983

This article describes the double hi-resolution display mode which is available in the Apple //c and the Apple //e (with the Extended 80-column card). Double Hi-res graphics provides twice the horizontal resolution and more colors than the standard high-resolution mode. On a monochrome monitor double hi-res displays 560 horizontal by 192 vertical pixels, while on a color monitor, 16 colors are available.

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

PB

DOUBLE HI-RES ON THE APPLE //e

What is It?

The double high-resolution display mode that is available for the Apple //e provides twice the horizontal resolution of the standard high-resolution mode. On a standard black-and-white video monitor, standard hi-res displays 280 columns and 192 rows of picture elements (pixels); the double hi-res mode displays 560x192 pixels. On a color monitor, the standard hi-res mode displays up to 140 columns of colors, each color being selected from the group of six colors available, with certain limitations. Double hi-res displays 140 columns of color, for which all 16 of the low-resolution colors are available.

Table 1. Comparison of Standard and Double Hi-Res Graphics

	Black/White	Color
Standard Hi-Res	280 x 192 pixels	140 columns 6 colors
Double Hi-Res	560 x 192 pixels	140 columns 16 colors

How Do I Install It?

Installation of the double hi-res mode on your Apple //e depends on the following three conditions, discussed in detail below:

1. Presence of a Revision B motherboard
2. Installation of an extended 80-column text card with jumper
3. A video monitor with a bandwidth of at least 14 MHz

First, your Apple //e must have a Revision B ("Rev-B") motherboard. To find out whether your //e's motherboard is a Rev-B board, check the part number on the edge of the board nearest the backpanel, above the slots. If the board is a Rev-B board, the part number will be 820-0064-B. (Double hi-res does not work on systems containing a Rev-A motherboard.) If your //e's motherboard is not a Rev-B board, and if you want to obtain one, contact your local Apple dealer.

The second condition for installing double hi-res on your //e is that your //e must have an extended 80-column text card installed. This card must be installed with a jumper connecting the two Molex-type pins on the board.

WARNING: If your //e is a Rev-A machine, do NOT insert into it an extended 80-column card with the jumper connection mentioned above. The system will not work at all if you do.

The last requirement for operation in double hi-res mode is that your video monitor must have a bandwidth of at least 14 MHz. This bandwidth is necessary because a television set that requires a modulator will not reproduce some characters or graphic elements clearly, due to the high speed at which the computer sends out dots in this mode. Because most of the video monitors having a bandwidth of up to 14 MHz are black-and-white, the working examples in this article do not apply to color monitors. If you have a video monitor, please use it -- instead of a television set -- to display the following examples.

Your Turn to be Creative -- or, Volunteers, Anyone?

At this writing, no programs exist that support double hi-res graphics. Moreover, none of the standard hi-res commands (such as HPLOT) work properly in double hi-res mode. Until such routines are available, users must write their own. If you've gotten this far, and want to continue, you'll probably already have used the system monitor, and you'll probably need very few explanations. If not, please refer to the ?Apple_//e_Reference_Manual? and then return to double hi-res operations.

Before going into the subtleties of double hi-res, you should be acquainted with standard hi-res functions. If you aren't, obtain the Apple //e Reference Manual (Part Number A2L2005) or the Apple][Reference Manual (which, however, is out of print), and please read the sections on high-resolution graphics before proceeding with the hands-on practice explained below.

You can find another good explanation of these features in the Apple][Graphics column by Ken Williams, in Softline magazine. We suggest that you start with Volume 1, Number 1 (September 1981), available from Softalk Publishing, Inc. The early columns are especially useful.

The tutorial that occupies the rest of this article assumes you are working at your Apple //e as you read. The second part of the lesson demonstrates the double hi-res mode; therefore, before embarking on the second part, you should

install a jumpered extended 80-column card in your Rev-B Apple //e.

Hands-On Practice with Standard Hi-Res

The Apple //e hi-res graphics display is bit-mapped. In other words, each dot on the screen corresponds to a bit in the Apple //e's memory. For a real-life example of bit-mapping, perform the following procedure, according to the instructions given below. (The symbol "<cr>" indicates a carriage return.)

1. Boot the system, using the DOS system master diskette.

2. When the prompt ("]") appears, press the RESET key.

3. Engage the CAPS LOCK key, and type HGR<cr>. (This instruction should clear the top of the screen.)

4. Type CALL -151 <cr>. (The system is now in the monitor mode, and the prompt should appear as an asterisk (*).)

5. Type 2100:1 <cr>. One single dot should appear in the upper left-hand corner of the screen.

Congratulations! You have just plotted your first hi-res pixel. (Not an astonishing feat, but you have to start somewhere...)

With a black-and-white monitor, the bits in memory have a simple correspondence with the dots (pixels) on the screen. A dot of light appears if the corresponding bit is set (has a value of 1), but remains invisible if the bit is off (has a value of zero). (The dot appears white on a black-and-white monitor, and green on a green-screen monitor, such as Apple's Monitor ///. For simplicity, we shall refer to an invisible dot as a "black" dot or pixel.) Two visible dots located next to each other appear as a single wide dot, and many adjacent dots appear as a line. To obtain a display of another dot and a line, follow the steps listed below:

1. Type 2080:40 <cr>. A dot should appear above and to the right of the dot you produced in the last exercise.

2. Type 2180:7F <cr>. A small horizontal line should appear below the first dot you produced.

From Bits and Bytes to Pixels

The seven low-order bits in each display byte control seven adjacent dots in a row. A group of 40 consecutive bytes in

memory controls a row of 280 dots (7 dots per byte, multiplied by 40 bytes). In the screen display, the least-significant bit of each byte appears as the leftmost pixel in a group of 7 pixels. The second-least-significant bit corresponds to the pixel directly to the right of the pixel previously displayed, and so on. To watch this procedure in action, follow the steps listed below. The dots will appear in the middle of your screen.

1. Type 2028:1 <cr>.

2. Type 2828:2 <cr>.

3. Type 3028:4 <cr>.

The three bits you specified in this exercise correspond to three pixels that are displayed one after another, from left to right.

The most-significant bit in each byte does not correspond to a pixel. Instead, this bit is used to shift the positions of the other seven bits in the byte. For a demonstration of this feature, follow the steps listed below:

1. Type 2050:8 <cr>.

2. Type 2850:8 <cr>.

3. Type 3050:8 <cr>.

You'll notice that the dots align themselves vertically. Now:

4. Type 2450:88 <cr>.

The new dot (that is, the one that corresponds to the bit you just specified) does not line up with the dots you displayed earlier. Instead, it appears to be shifted one "half-dot" to the right.

5. To demonstrate that this dot really is a "new" dot, and not just the "old" dot shifted by one dot position, type 2050:18 <cr>, 2850:18 <cr>.

You'll notice that the dot mentioned under Step 4 above (the dot that was not aligned with the other seven dots) is straddled by the dots above and below it. (The use of magnifying lenses is permitted.)

Shifting the pixel one "half-dot", by setting the high, most-significant bit is most often used for color displays. When the high bit of a byte is set, to generate this shifted

dot (which is also called the "half-dot shift"), then all the dots for that byte will be shifted one half dot. The half-dot shift does not exist in the double hi-res mode for the Apple //e.

The following figure shows the memory map for the standard hi-res graphics mode:

BASE \ HORIZONTAL OFFSET	$00	$01	$02	$03	...	$24	$25	$26	$27
$2000					...				
$2080					...				
$2100					...				
$2180					...				
$2200					...				
$2280					...				
$2300					...				
$2380					...				
$2028					...				
$20A8					...				
$2128					...				
$21A8					...				
$2228					...				
$22A8					...				
$2328					...				
$23A8					...				
$2050					...				
$2000					...				
$2150					...				
$21D0					...				
$2250					...				
$22D0					...				
$2350					...				
$23D0					...				

Standard Hi-res Memory Map

The following figure shows the box subdivisions for the memory map shown in the figure above:

OFFSET FROM BASE	BIT 6	5	4	3	2	1	0 LSB
+$0000							
+$0400							
+$0800							
+$0C00							
+$1000							
+$1400							
+$1800							
+$1C00							

For example, the first memory address of each screen line for the first few lines is as shown below:

$2000
$2400
$2800
$2C00
$3000
$3400
$3800
$3C00
$2080
$2480, etc.

Each of the 24 'boxes' contains 8 screen lines for a total of 192 vertical lines per screen. Each of the 40 'box' per line contains 7 pixels for a total of 280 pixels horizontally across each line.

The Intricacies of Double Hi-Res

Because the double hi-resolution graphics mode provides twice the horizontal dot density as standard hi-res graphics does, double hi-res requires twice as much memory as standard hi-res does. If you spent many hours memorizing the standard hi-res memory map, don't despair. Double hi-res still uses the hi-res graphics page (but only to represent half the picture, so to speak). In the double hi-res mode, the hi-res graphics page is compressed to fit into half of the display. The other half of the display is stored in memory (called the "auxiliary" or "aux" memory) on the Extended 80-Column card. (This article refers to the standard hi-res graphics page, which resides in main memory, as the "motherboard" or "MB" memory.)

The auxiliary memory uses the same addresses used by the standard hi-res graphics page (Page 1, $2000 through $3FFF). The hi-res graphics page stored in auxiliary memory is known as "hi-res page 1X." The graphics pages in auxiliary memory

are bank-switched memory, which you can switch in by activating some of the soft switches. (Adventurous readers may want to skip ahead to "Using the Auxiliary Memory," which appears later in this article.)

The memory mapping for the hi-res graphics display is analogous to the technique used for the 80-column display. The double hi-res display interleaves bytes from the two different memory pages (auxiliary and motherboard). Seven bits from a byte in the auxiliary memory bank are displayed first, followed by seven bits from the corresponding byte on the motherboard. The bits are shifted out the same way as in standard hi-res (least-significant bit first). In double hi-res, the most significant bit of each byte is ignored; thus, no half-dot shift can occur. (This feature is important, as you'll see when we examine double hi-res in color.)

The memory map for double hi-res appears below:

	$0		$1		$2		$3		.....		$25		$26		$27	
	AUX	MB	AUX	MB	AUX	MB	AUX	MB	AUX	MB	AUX	MB	AUX	MB	AUX	MB
$2000									.....							
$2080									.....							
$2100									.....							
$2180									.....							
$2200									.....							
$2280									.....							
$2300									.....							
$2380									.....							
$2028									.....							
$20A8									.....							
$2128									.....							
$21A8									.....							
$2228									.....							
$22A8									.....							
$2328									.....							
$23A8									.....							
$2050									.....							
$20D0									.....							
$2150									.....							
$21D0									.....							
$2250									.....							
$22D0									.....							
$2350									.....							
$23D0									.....							

Double Hi-res Memory Map

Where each box is subdivided exactly the same way it is in the standard hi-res mode.

Obtaining a Double-Hi-Res Display

To display the double hi-res mode, set the following soft switches:

	In Monitor Read	In Applesoft PEEK
HI-RES	$C057	49239
GR	$C050	49232
AN3	$C05E	49246
MIXED	$C053	49235

	In Monitor Write	In Applesoft POKE
80COL	$C00D	49165,0

Annunciator 3 (AN3) must be turned off to get into double hi-res mode. You turn it off by reading location 49246 ($C05E hex). Note that whenever you press CTRL-RESET, AN3 is turned on; therefore, each time you press CTRL-RESET, you must turn AN3 off again.

If you are using MIXED mode, then the bottom four lines on the screen will display text. If you have not turned on the 80-column card, then every second character in the bottom four lines of text will be a random character. (The reason is that although the hardware displays 80 columns of characters, the firmware only updates the 40-column screen, which consists of the characters in the odd-numbered columns. The characters in even-numbered columns then consist of random characters taken from text page 1X in the auxiliary memory.)

To remove the "even" characters from the bottom four lines on the screen, type PR#3<CR> from basic (type 3^P from the monitor). This procedure clears the memory locations on page 1X.

Using the Auxiliary Memory

The auxiliary memory consists of several different sections, which you can select by using the soft switches listed below. A pair of memory locations is dedicated to each switch. (One location turns the switch on; the other turns it off.) You activate a switch by writing to the appropriate memory location. The WRITE instruction itself is what activates the switch; therefore, it doesn't matter what data you write to the memory location. The soft switches are:

		From Monitor Write	From Applesoft POKE
80STORE	off:	$C000	49152,0
	on:	$C001	49153,0
RAMRD	off:	$C002	49154,0
	on:	$C003	49155,0
RAMWRT	off:	$C004	49156,0
	on:	$C005	49157,0
PAGE2	off:	$C054	49236,0
	on:	$C055	49237,0
HIRES	off:	$C056	49238,0
	on:	$C057	49239,0

A routine called AUXMOVE, located in the monitor ROM of the Apple //e, is also very handy, as we'll see below. AUXMOVE is located at address C311.

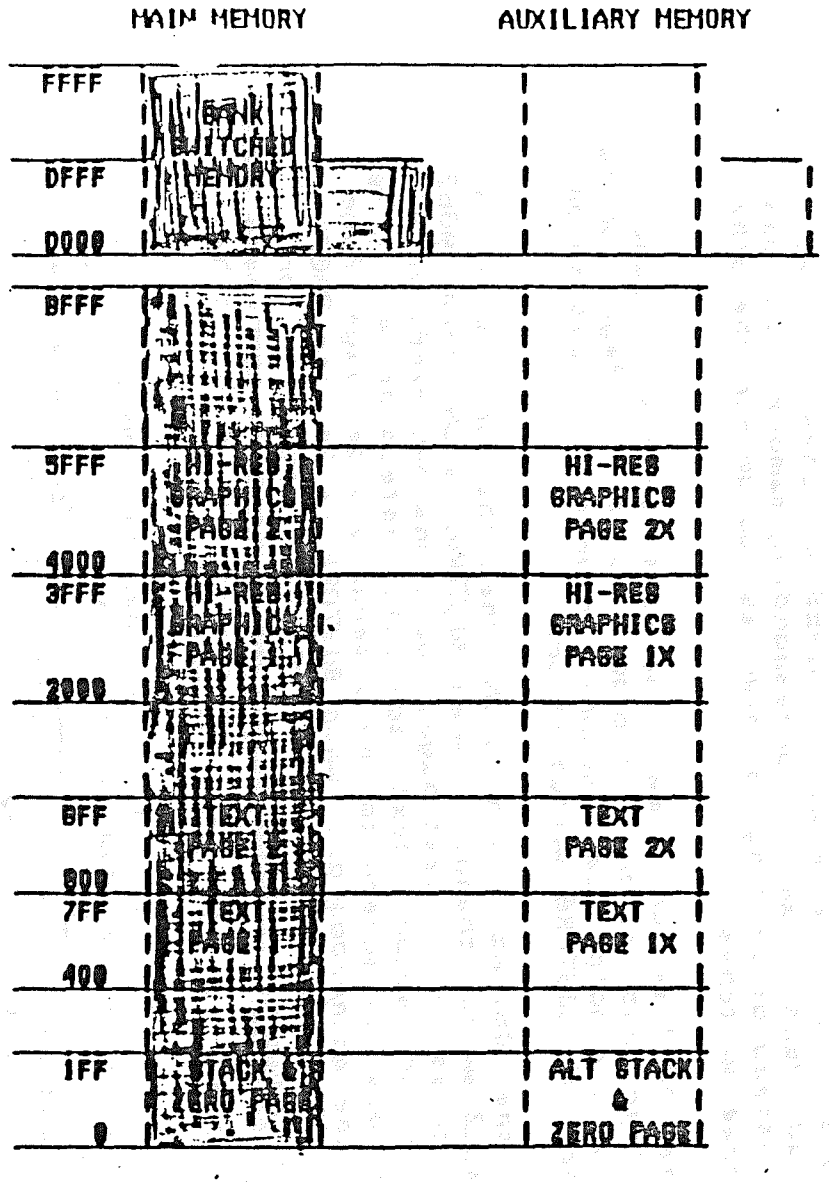
Accessing memory on the auxiliary card with the soft switches has the following characteristics. Memory maps, which help clarify the descriptions, are on the next page.

1). To activate the PAGE2 and HIRES switches, you need only read (PEEK) from the corresponding memory locations (instead of writing to them, as you do for the other three switches).

2). The PAGE2 switch normally selects the display page, in either graphics or text mode, from either Page 1 or Page 2 of the motherboard memory. However, it does so only when the 80STORE switch is OFF.

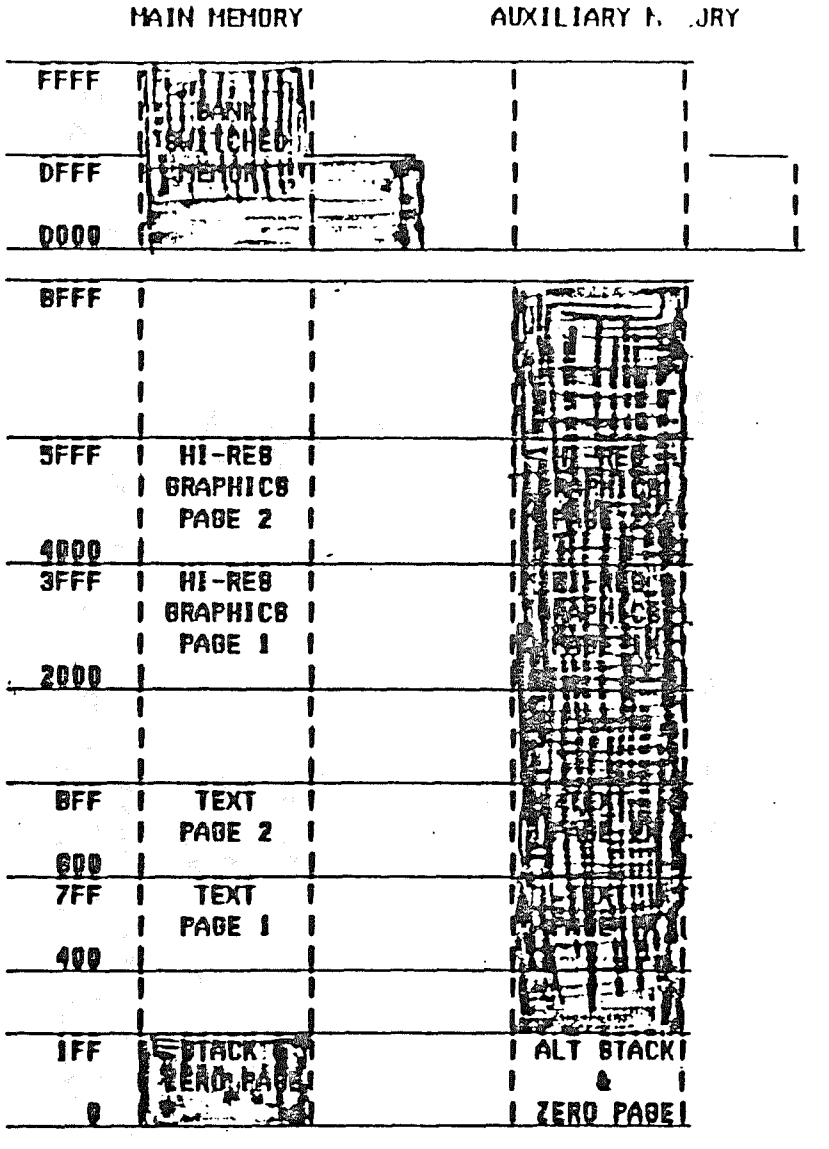
3). If the 80STORE switch is ON, then the function of the PAGE2 switch changes. When 80STORE is ON, then PAGE2 switches in the text page, locations $400-7FF, from auxiliary memory (text page 1X), instead of switching the display screen to the alternate video page (Page 2 on the motherboard). When 80STORE is ON, the PAGE2 switch determines which memory bank (auxiliary or motherboard) is used during any access to addresses $400 through 7FF. When the 80STORE switch is ON, it has priority over all other switches.

4). If the 80STORE switch is ON, then the PAGE2 switch only switches in the graphics page 1X from the auxiliary memory <u>if the HIRES switch is also ON</u>. (Note that this circumstance is slightly different from that described in Item 3 above.) When 80STORE is ON, and if the HIRES switch is also ON, then the PAGE2 switch selects the

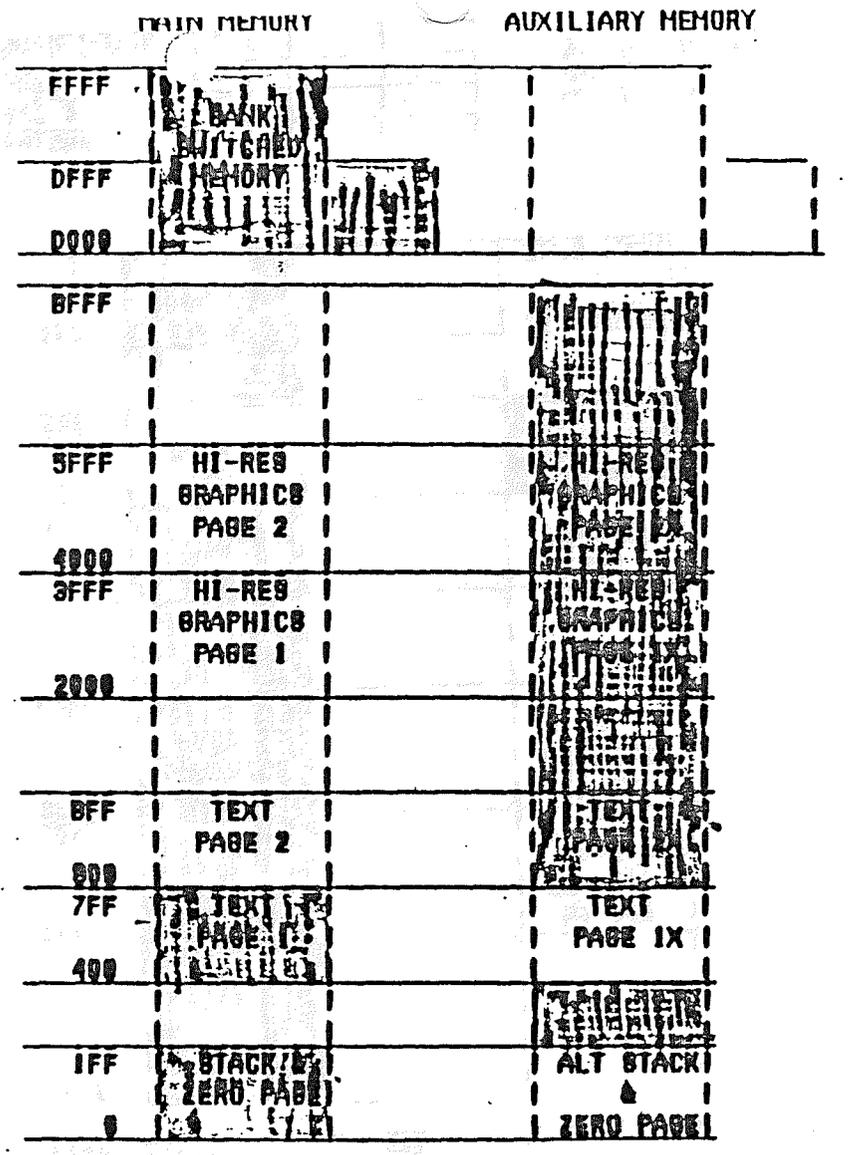


80STORE	OFF	ON	
PAGE 2	X	OFF	
HIRES	X	X	
RAMRD/RAMWRT	OFF	OFF	

ACTIVE MEMORY

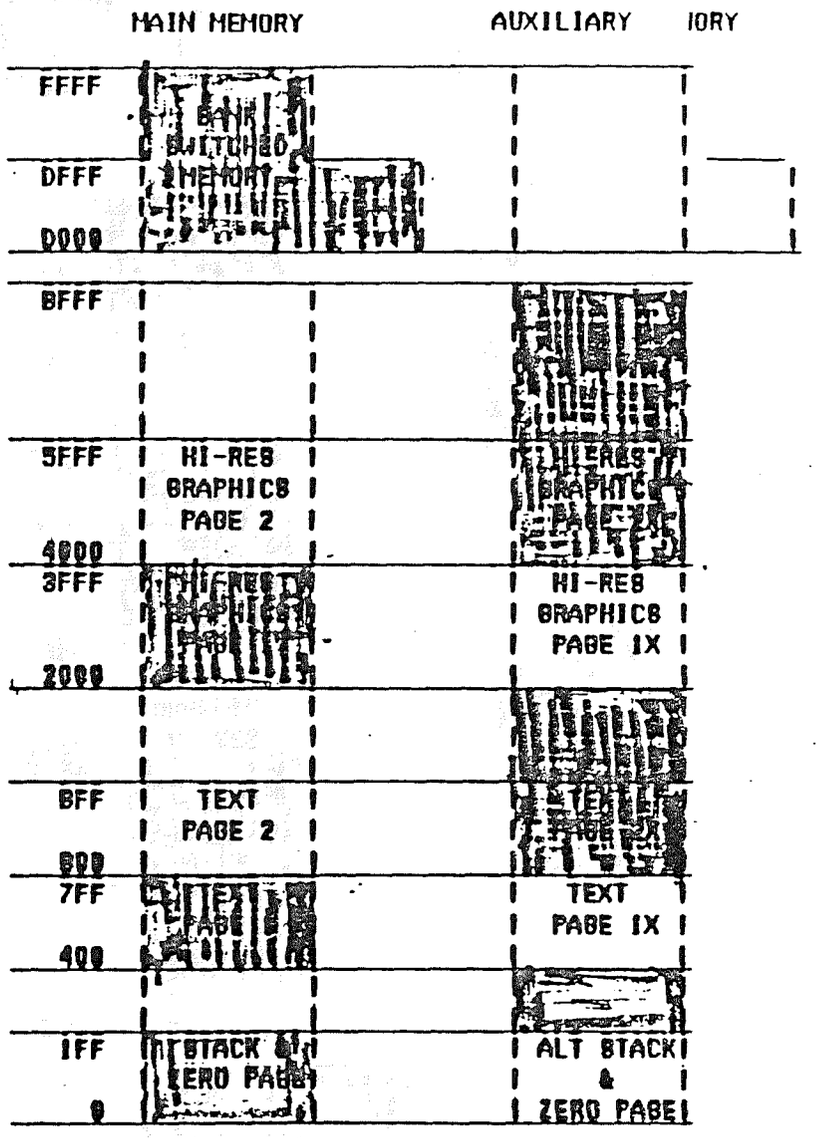


80STORE	OFF	ON	
PAGE 2	X	ON	
HIRES	X	X	
RAMRD/RAMWRT	ON	ON	

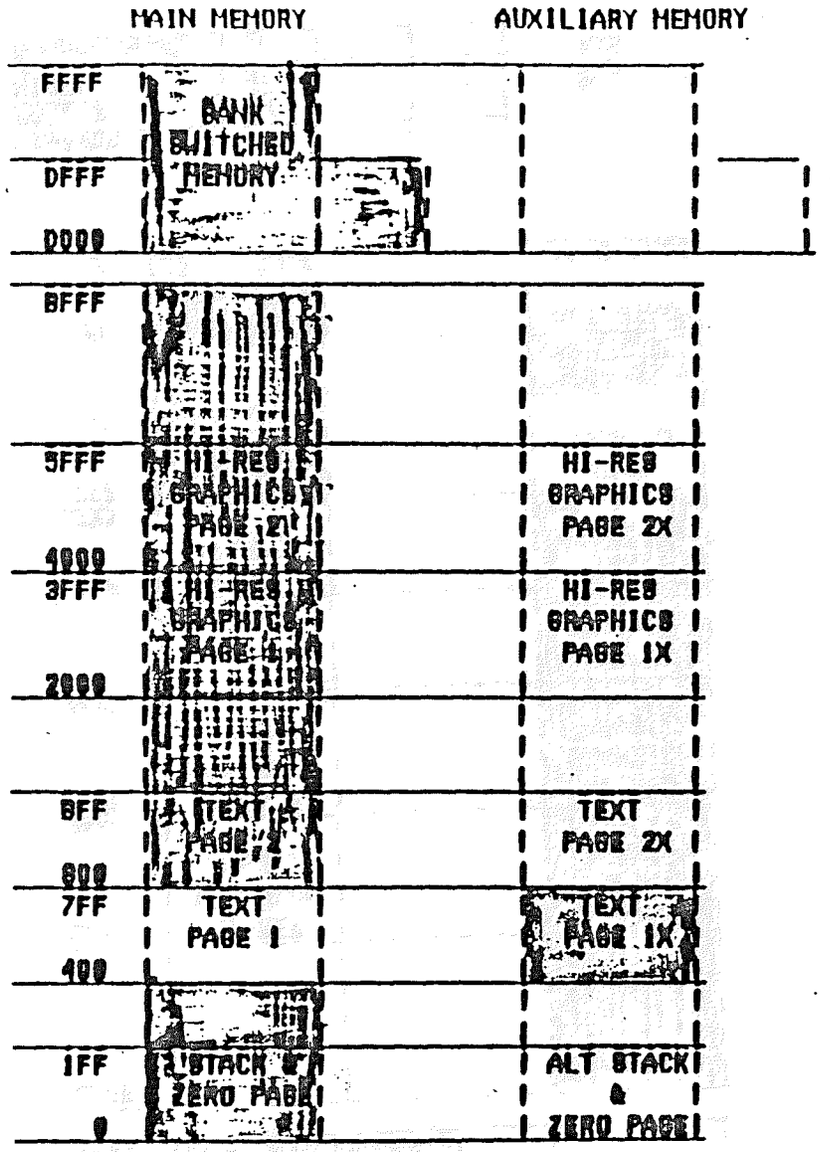


80STORE	ON			
PAGE 2	OFF			
HIRES	OFF			
RAMRD/RAMWRT	ON			

ACTIVE MEMORY

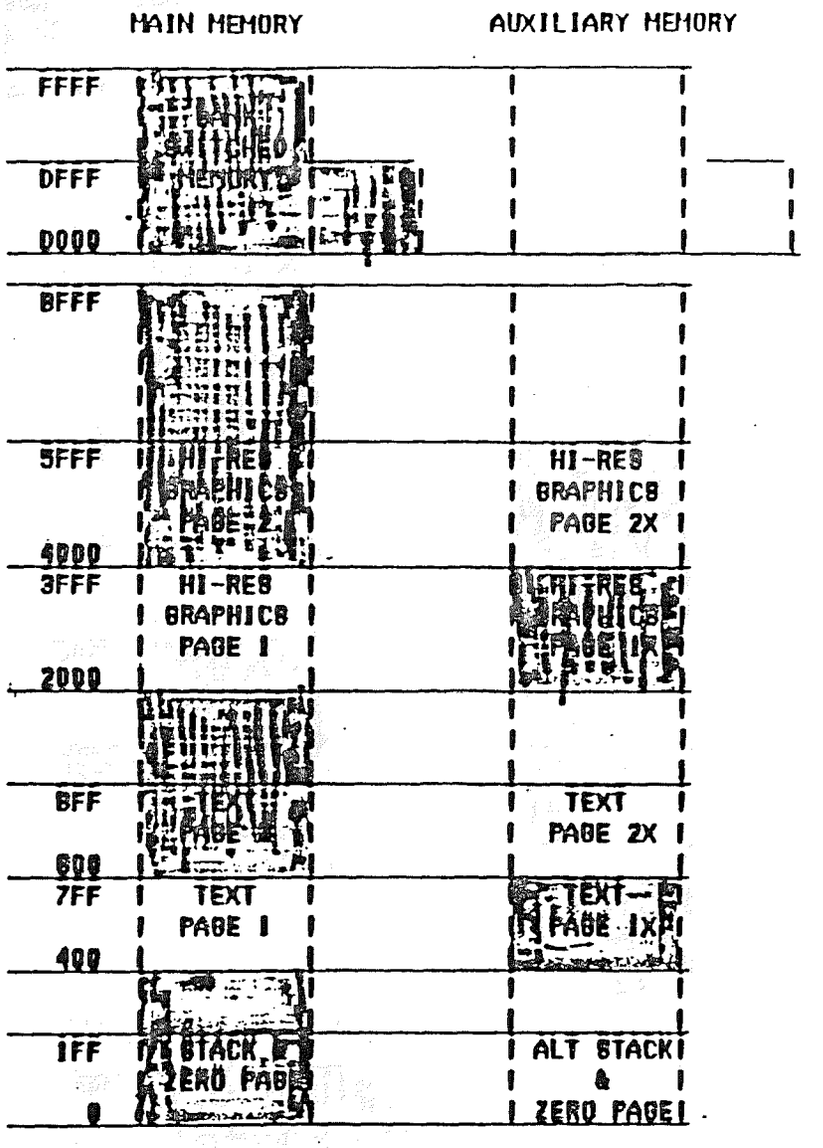


80STORE	ON			
PAGE 2	OFF			
HIRES	ON			
RAMRD/RAMWRT	ON			



80STORE	ON		
PAGE 2	ON		
HIRES	OFF		
RAMRD/RAMWRT	OFF		

ACTIVE MEMORY



80STORE	ON		
PAGE 2	ON		
HIRES	ON		
RAMRD/RAMWRT	OFF		

memory bank (auxiliary or motherboard) for accesses to a memory location within the range $2000 through 3FFF. If the HIRES switch is OFF, then any access to an memory location within the range $2000 through 3FFF uses the motherboard memory, regardless of the state of the PAGE2 switch.

5. If the 80STORE switch is OFF, and if the RAMRD and RAMWRT switches are ON, then any reading or writing to address space $200-$8FFF gains access to the auxiliary memory. If only one of the switches, for example RAMRD, is set then only the appropriate operation, in this case a read, will be performed on the auxiliary memory, while a write operation will access only the motherboard memory. If only RAMWRT is set then all write operations access the auxiliary memory. When The 80STORE switch is ON it has higher priority than the RAMRD and RAMWRT switches.

Shortcuts: Writing to Auxiliary Memory from the Keyboard
First, press CTRL-RESET. Next, type <CALL -151> (to get into the monitor). Then type the following hexadecimal addresses to turn on the double hi-res mode:

C057 (for Hi-res)

C050 (for Graphics)

C053 (for Mixed mode)

C05E Turns off AN3 for double hi-res

C00D:0 Turns on the 80COL switch

This procedure usually causes the display of a random-dot pattern at the top of the screen, while the bottom four lines on the screen contain text. To clear the screen, follow the steps listed below:

1). Type 3D0G to return to BASIC.

2). Type HGR to clear half of the screen. (The characters you type will probably appear in alternating columns. This is not a cause for alarm; as noted above, the firmware simply thinks you are working with a 40-column display.) Remember that hi-res graphics commands don't know about the half of the screen stored on page 1X in the auxiliary memory. Therefore, only page 1 (that is, the first half) of the graphics page on the motherboard is cleared. As a result, in the the screen display, only alternate 7-bit columns appear cleared.

On the other hand, if all of the screen columns were cleared after the HGR command, then chances are good that you're not in double hi-res mode. If your screen was cleared then to determine which mode you're in, type the following instructions:

CALL -151 back into monitor

2000:FF

2001<2000.2027M

If a solid line appears across the top of the screen, you're not in double hi-res mode. (The line that appears should be a dashed or intermittent line: - - - - - - - - across the screen.) If you're not in double hi-res mode, then make sure that you do have a Rev. B motherboard, and that the two Molex-type pins on the Extended 80-Column card are shorted together with the jumper block. Then re-type the instructions listed above.

If you're staring at a half-cleared screen, you can clear the non-blank columns by writing zeros to addresses $2000 through 3FFF on graphics page 1X of auxiliary memory. To do so, simply turn on the 80STORE switch, turn on the PAGE2 switch, and then write to locations $2000, $2001, $2002, and so on up through 3FFF. However: this procedure will not work if you try it from the monitor! The reason is that each time you invoke a monitor routine, the routine sets the PAGE2 switch back to page 1 so that it can display the most recent command that you entered. When you try to write to $2000, etc. on the auxiliary card, instead it will write to the motherboard memory.

Another way to obtain the desired result is to use the monitor's USER command, which forces a jump to memory location $3F8. You can place a JMP instruction starting at this memory location, so that the program will jump to a routine that writes into hi-res page 1X. Fortunately, the monitor already contains such a routine: AUXMOVE.

Using AUXMOVE

You use the AUXMOVE routine to move data blocks between main and auxiliary memory. But the task still remains of setting up the routine so that it knows which data to write, and where to write it. To use this routine, some byte pairs in the zero page must be set up with the data block addresses, and the carry bit must be fixed to indicate the direction of the move. You may not be surprised to learn that the byte pairs in the zero page used by AUXMOVE are also the scratch-pad registers used by the monitor during instruction execution. The result is that while you type the addresses for the monitor's move command, those addresses are being stored in the byte pairs used by AUXMOVE. Thereafter, you

can call the AUXMOVE command directly, using the USER (CTRL-Y) command.

In practice, then, enter the following instructions:

C00A:0	(turns on the 80-Column ROM, which contains the AUXMOVE routine)
C000:0	(reason explained below)
3F8: 4C 11 C3	(the jump to AUXMOVE)
2000<2000.3FFF ^Y	(where "^Y" indicates that you should type CTRL-Y.)

The syntax for this USER (CTRL-Y) command is:

(AUXdest)<(MBstart).(MBend)^Y	Copies the values in the range MBstart to MBend in the motherboard memory into the auxiliary memory beginning at AUXdest. This command is analogous to the MOVE command.

You can use this procedure to transfer any block of data from the motherboard memory to hi-res page 1X. Working directly from the keyboard, you can use a data block transferred this way to fill in any part of a double hi-res screen image. The image to be stored in hi-res page 1X (that is, the image that will be displayed in the even-numbered columns of the double hi-res picture) must first be stored in the motherboard memory. You can then use the CTRL-Y command to transfer the image to hi-res page 1X.

The AUXMOVE routine uses the RAMRD and RAMWRT switches to transfer the data blocks. Because the 80STORE switch overrides the RAMRD and RAMWRT switches, the 80STORE switch must be turned off -- otherwise it would keep the transfer from occurring properly (hence the write to $C000 above).

If the 80STORE and HIRES switches are ON and PAGE2 is off, when you execute AUXMOVE, then any access to an address located within the range from $2000 to $3FFF inclusive would use the motherboard memory, regardless of how RAMRD and RAMWRT are set. Entering the command C000:0 turns off 80STORE, thus letting the RAMRD and RAMWRT switches control the memory banking.

The CTRL-Y trick described above only works for transferring data blocks from the main (motherboard) memory to auxiliary memory (because the monitor always enters the AUXMOVE routine with the carry bit set). To move data blocks from

the auxiliary memory to the main memory, you must enter AUXMOVE with the carry bit clear. You can use the routine listed below to transfer data blocks in either direction:

301:AD 0 3	(loads the contents of address $300 into the accumulator)
304:2A	(rotates the most-significant bit into the carry flag)
305:4C 11 C3	(jump to $C311 <AUXMOVE>)
3F8:4C 1 3	(sets the CNTRL-Y command to jump to address $301)

Before using this routine, you must modify memory location $300, depending on the direction in which you want to transfer the data blocks. If the transfer is from the auxiliary memory to the motherboard, you must clear location $300 to zero. If the transfer is from the motherboard to the auxiliary memory, you must set location $300 to $FF.

Two Double Hi-Res Pages

So far, we've only discussed using graphics pages 1 and 1X to display double hi-res pictures. But -- analogous to the standard hi-res pages 1 and 2 -- two double hi-res pages exist: pages 1 and 1X, at locations $2000 through 3FFF, and pages 2 and 2X, at locations $4000 through 5FFF.The only trick in displaying the second double hi-res page is that you must turn off the 80STORE switch. If the 80STORE switch is ON, then only the first page (1 and 1X) is displayed. Go ahead and try it:

C000:0	to turn off the 80STORE switch
C055	to turn on the PAGE2 switch

The screen will fill up with another display of random bits. Clear the screen using the instructions listed above (in the section entitled "Using AUXMOVE"). However, this time, use addresses $4000 through 5FFF instead. (Don't be alarmed by the fact that the figures you're typing aren't displayed on the screen. They're being "displayed" on text Page 1.)

4000:0

4001<4000.5FFFM

4000<4000.5FFF ^Y

You'll be delighted to learn that you can also use this trick to display two 80-column text screens. The only problem here is that the 80-column firmware continually turns on the 80STORE switch, which prevents the display of the second 80-column screen. However, if you write your own 80-column display driver, then you can use both of the 80-column screens.

Color Madness

It should come as no surprise that color-display techniques in double hi-res are different from color-display techniques in standard hi-res. This is because the "half-dot shift" doesn't exist in double hi-res mode.

Instead of going into a disquisition on how a TV set decodes and displays a color signal, I'll simply explain how to generate color in double hi-res mode. In the following examples, the term "color monitor" refers to either an NTSC monitor or a color television set. Both work; however, the displays will be much harder to see on the color TV.The generation of color in double hi-res demands sacrifices. A 560x192-dot display is not possible in color. Instead, the horizontal resolution decreases by a factor of four (to 140 dots across the screen). Just as with a black-and-white monitor, a simple correspondence exists between memory and the pixels on the screen. The difference is that four bits are required to determine each color pixel. These four bits represent 16 different combinations: one for each of the colors available in double hi-res. (These are the same colors that are available in the low-resolution mode.)

Let's start by exploring the pattern that must be stored in memory to draw a single colored line across the screen. Start by pressing RESET; then load the program "COLOR TEST" from the DOS 3.3 sample programs disk (with the old Apple][+ DOS system master use the program "COLOR DEMOSOFT"). Use this program to adjust the colors displayed by your monitor. After you've adjusted the colors, exit from the color-demo program.

The instructions that appear below are divided into groups separated by blank lines. Because it's very difficult (and, on a TV set, almost impossible) to read the characters you're typing in as they appear on the screen, face it: you will make typing errors. If the instructions appear not to work, then start again from the beginning of a group of instructions.

CALL -151	(to get into the Monitor routine/program)
C050	(This set of instructions puts the
C057	computer into double hi-res mode.)
C05E	
C00D:0	

```
2000:0                  (This set of instructions clears first
2001<2000.3FFFM         one half of the screen, and then the
3F8: 4C 11 C3           other half of the screen.)
2000<2000.3FFF^Y

2100:11 4               (2 red dots appear on top left of screen)
2102<2100.2126M         (A dashed red line appears across screen)

2150:8 22               (Two green dots appear near bottom left)
2152<2150.2175M         (Dashed green line appears across screen)

2100<2150.2177^Y        (Fills in the red line)
```

In contrast to conditions in standard hi-res, no half-dot shift occurs, and the most-significant bit of each byte is not used.

As noted above, four bits determine a color. You can "paint" a single-color line across the screen simply by repeating a four-bit pattern across the screen. But it is much easier to write a whole byte rather than just change four bits at a time. Since only 7 bits of each byte are displayed (as noted earlier in our discussion of black-and-white double hi-res) and the pattern is four bits wide, it repeats itself every 28 bits or four bytes. Use the instructions listed below to draw a line of any color across the screen by repeating a four byte pattern for the color as shown in Table III below.

```
2200: mb1 mb2           (Colored dots appear at the left edge)
2202<2200.2226M         (A dashed, colored line appears)

2250: aux1 aux2
2250<2250.2276M

2200<2250.2276^Y        (Fills in line, using the selected color)
```

[see Table III on next page]

TABLE III. The Sixteen Colors

COLOR	aux1	mb1	aux2	mb2	REPEATED BINARY PATTERN
BLACK	00	00	00	00	0000
MAGENTA	08	11	22	44	0001
BROWN	44	08	11	22	0010
ORANGE	4C	19	33	66	0011
DARK GREEN	22	44	08	11	0100
GREY1	2A	55	2A	55	0101
GREEN	66	4C	19	33	0110
YELLOW	6E	5D	3B	77	0111
DARK BLUE	11	22	44	08	1000
VIOLET	19	33	66	4C	1001
GREY2	55	2A	55	2A	1010
PINK	5D	3B	77	6E	1011
MEDIUM BLUE	33	66	4C	19	1100
LIGHT BLUE	3B	77	6E	5D	1101
AQUA	77	6E	5D	3B	1110
WHITE	7F	7F	7F	7F	1111

In this table, "aux1" indicates the first, fifth, ninth, thirteenth, etc. byte of each line (i.e., every fourth byte, starting with the first byte). The heading "mb1" indicates the second, sixth, tenth, fourteenth, etc. byte of each line (i.e., every fourth byte, starting with the second byte). The "aux2" and "mb2" headings indicate every fourth byte, starting with the third and fourth bytes of each line, respectively. "Aux1" and "aux2" are always stored in auxiliary memory, while "mb1" and "mb2" are always stored in the motherboard memory.

As you'll infer from Table III, the absolute position of a byte also determines the color displayed. If you write an "8" into the first byte at the far left side of the screen (i.e., in the "aux1" column), then a red dot is displayed. But if you write an "8" into the third byte at the left side of the screen (the "aux2" column), then a dark green dot is displayed. Remember -- the color monitor decides which color to display based on the relative position of the bits on each line (i.e., on how far the bits are from the left edge of the screen).

So far, so good. But suppose you want to display more than one color on a single line. It's easy: just change the four-bit pattern that is stored in memory. For example, if you want the left half of the line to be red, and the right half to be purple, then store the "red" pattern (8, 11, 22, 44) in the first 40 bytes of the line, and then store the

"purple" pattern (19,33,66,4C) in the second 40 bytes of the line. Table III is a useful reference tool for switching from one color to another, provided you make the change on a byte boundary. In other words, you must start a new color at the same point in the pattern at which the old color ended. For example, if the old color stops after you write a byte from the "mbl" column, then you should start the new color by storing the next byte in memory with a byte from the "aux2" column. This procedure is illustrated below:

```
2028:11 44 11 44 11 44 11 77 5D 77 5D 77 5D
                (creates a dashed line that is red, then yellow)

2128: 8 22 8 22 8 22 8 22 6E 3B 6E 3B 6E

2028<2128.2134^Y          (fills in the rest of the colors)
```

Switching Colors in Mid-Byte

If you want a line to change color in the middle of a byte, you'll have to re-calculate the column, based on the information in Table III. Suppose you want to divide the screen into three vertical sections, each a different color. The left-hand third of the screen ends in the middle of the 27th character from the left edge -- that is, in an "aux2" column of the color table. (Dividing 27 by 4 gives a remainder of 3, which indicates the third column, or "aux2".) Your pattern should change from the first color to the second color after the 5th bit of the 27th byte. You can change the color in the middle of a byte by selecting the appropriate bytes from the "aux2" column of Table III, and concatenating two bits for the second color with five bits for the first color.

However, because the bits from each byte are shifted out in order from least significant to most significant, the two most significant bits (in this case I mean bits 5 and 6, because bit 7 is unused) for the second color are concatenated with the five least significant bits for the first color. For instance, if you want the color to change from orange (the first color) to green (the second color), then you must append the two most significant bits (5 and 6) of "green" to the five least significant bits (0-4) of "orange." In Table III, the "aux2" column byte for green is 19, and the two most significant bits are both clear. The "aux2" column byte for orange is 33, and the five least significant bits are equal to 10011. The new byte calculated from appending green (00) to orange (10011) yields 13 (0010011). Therefore, the first 26 byte of the line come from the table values for orange; the 27th byte is 13, and the next 26 bytes come from the table values for green.

```
2300: 19 66                (puts orange line on screen)
2302<2300.2310M
```

The double hires screen has 140 columns, numbered 0 through 139, and 192 rows, numbered 0 to 191. Just like the standard hi-res screen, the origin is in the upper left corner, while the point 139,191 is in the bottom right corner.

The color codes are the same as for lo-res graphics:

0:black
1:magenta
2:dark blue
3:violet
4:dark green
5:grey1
6:medium blue
7:light blue
8:brown
9:orange
10:grey2
11:pink
12:green
13:yellow
14:aqua
15:white

Some exercises you may want to try include painting the left half of the screen with grey1 and the right half with grey2 to see if they are different or moving a colored ball on different colored background. For the adventerous type, you may want to rewrite brickout (super brickout).

The following program shows off double hi-res. It starts with the color bar demo, except in this case the color bars can be much narrower than was possible in low resolution graphics. The next screen shows a simple picture of an orange line drawn diagonally on a green background. These two colors are also available in standard hi-res, but as you'll see in the next picture there are certain limitations.

[[RUN DEMO]]

In double hi-res the most significant bit is not used, and any color can appear next to any other color, anywhere on the screen (though "fringing" can occur where the colors join). In standard hi-res the most significant bit of each byte limits that byte to four of the six colors. If the MSB is set than than the only colors displayed by that byte are white, black, blue, and orange. Therefore since green and orange can't be displayed in the same byte, the whole byte becomes orange, and the stair step line appears.

By the way, if Annunciator 3 (AN3) is turned off when a jumpered extended 80-column card is present, then the most significant bit of standard hi-res isn't used either. This

means that any standard hi-res picture will display only black, white, violet or green. If the picture contains blue or orange, then those colors will be converted to violet or green. Go ahead and try it: pull out a game that uses all four colors, turn the AN3 off with PEEK (49246), and then, without pressing RESET (since that sets AN3 on), run the program (RUN HELLO sometimes works).

Now you've got the tools and the rules to the double hi-res mode. As you can see double hi-res has more color with higher resolution than standard hi-res. You can even develop games that do fancy animation or scroll orange objects across green backgrounds. In black and white, word processing programs that use different fonts or proportional character sets can be developed. Have fun playing with the this new mode and I hope I'll see some of your programs soon.

[[I've got two more demo programs if there is room:]]

```
RUN DOUBLENET     (Remember Brians's theme)
BRUN QIX
```

APPLE //e TECHNOTE #4

Revision of RDY TECHNOTE 1-April 83*
1-July 84

This article describes an input signal into the 6502 microprocessor called the RDY line. The RDY line allows a peripheral card to halt the microprocessor with the output address lines reflecting the current address being fetched. If a peripheral device can not get data on the bus fast enough to meet the set up time of the 6502 then the peripheral card can pull the RDY line low and tell the 6502 to wait. This allows the peripheral device enough time to get the proper data on the bus. This article describes the timing for as event such as this.

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

PB

Peter Baum
20525 Mariani Ave
Cupertino, Ca. 95014

Apple Computer
M.S. 22-W

July 1, 1984

Using the RDY Line on the Apple //e and Apple][+

Though the 6502 was one of the first commercially successful microprocessors sold, the designers had forsight to include some very useful functions. Because many early peripherals products were very slow devices a microprocessor could not read from the device directly. To connect these slow devices onto the Apple peripheral bus, so that the 6502 can read data from them, requires either buffering the device or slowing down the processor. Though most people would try to buffer the device, sometimes it is not feasible. For example, the 2 ms. access time of a 1-megabit CMOS ROM makes buffering a nightmare, since both the address and data bus have to be buffered. When buffering isn't possible then a peripheral device can pull the RDY line to slow down the processor long enough to read a byte. This technique can be used by slow devices to communicate with the 6502.

The RDY line allows a peripheral card to halt the microprocessor with the output address lines reflecting the current address being fetched. If a peripheral device can not get data on the bus fast enough to meet the set up time of the 6502 then the peripheral card can pull the RDY line low and tell the 6502 to wait. This can not be done during a 6502 write cycle because the 6502 will not hold up.

In order for the 6502 to read a valid data byte from a peripheral card, the card has about 800 ns. from the time the addresses are valid to put the data on the bus. The data must be set up on the bus within approximately 400 ns. from the time that the I/O STROBE, I/O SELECT, or DEVICE SELECT signal on the peripheral slot goes true. If a device pulls the RDY line low for one cycle then the device will have 1.4 usec., instead of the 400 ns., to put out valid data. The RDY line can be pulled low for more than one cycle; in fact, there is no limit. A device that takes 100 us. to send data can just hold the RDY line low for 100 cycles. Hence, this technique will allow any slower device to get on the bus and send data to the 6502.

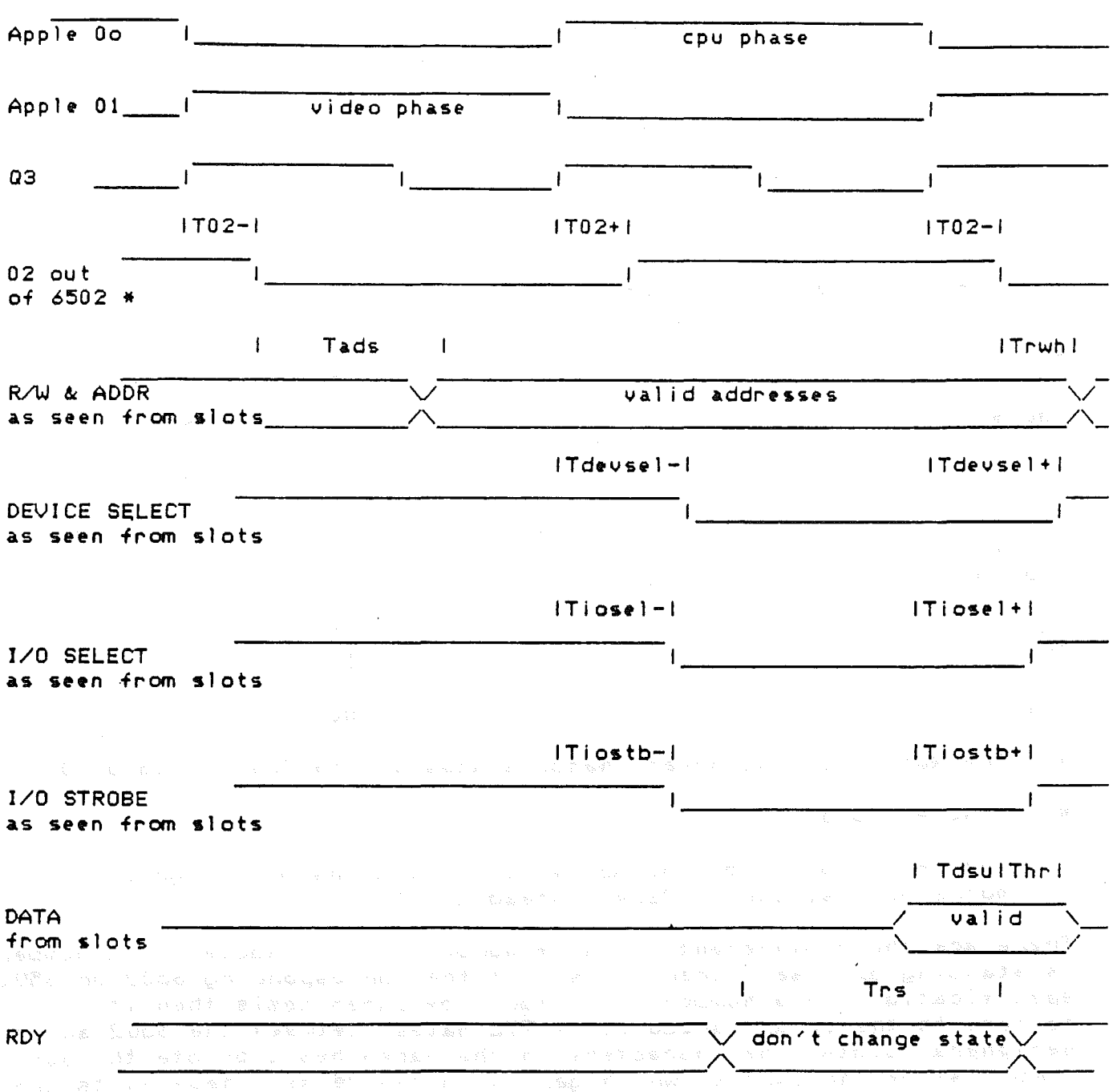
The RDY line is typically pulled low during 01, but the specification sheets for the 6502 show that it can be pulled anytime before the last 200 ns. of 02. The 02 line is not used by the Apple, but is an unused output from the 6502. It is basically the same as the 0o line with a little delay. Before I explain when to use (or not use in some cases) the RDY line, let us first look at some timing diagrams of the Apple system.

The timing diagram on the next page shows the relationship between the 6502 and Apple //e and Apple][+. The timing specifications have been adjusted to reflect the signals as they are seen from the peripheral slots. For example the 6502 (1 MHZ.) specification guarantees that the address bus will be valid within 225 ns. from 02 out. But the peripheral slots do not see these address lines directly; Instead the address lines go thru a buffer and then out to the peripheral slots. This adds a maximum delay of 13 ns. in the Apple][and 18 ns. in the Apple //e. The timing diagrams will show, in the case of an Apple][, that the address bus will be valid to the peripheral slots within 238 ns. (225+13) of 02 falling edge.

The major differences in timing between the Apple][+ and the Apple //e are due to the processor. The Apple][uses a 1 MHZ. 6502, while the Apple //e uses a 6502A, which is a 2 MHZ. part. This does not mean that the system clock in the Apple //e runs any faster, only that the 6502A is capable of running faster. This results in better timing margins. For example, the address and data busses are set up faster in the Apple //e by the 6502A than the 6502 sets them up in the Apple][. (This was done because the custom chips in the Apple //e are slower than the discrete logic in the Apple][and the 6502A was needed to compensate for this).

A peripheral card which uses the RDY line can only be used under certain circumstances. Because pulling the RDY line low halts the processor, any program with a software timing loop will not work properly. These programs assume that each instruction will take a fixed amount of time, which is not true when the processor stops in the middle of an instruction. An Apple][disk is an example of a peripheral which requires timing loops and won't run properly if the RDY line is used.

TIMING SIGNALS AS SEEN FROM PERIPHERAL SLOTS



* - 02 is an output signal from the 6502 which is not used by the Apple. It is a delayed 0o.

FIGURE 1

TIMING SPECIFICATIONS FOR FIGURE 1
(all times in ns.)

	Apple II 1 MHZ. 6502		Apple //e 2 MHZ. 6502A	
Symbol	min.	max.	min.	max.
T02- #	15	50+20 (LS08)	15	50+5 (S02)
T02+ #	30	80+15 (LS08)	30	80+5 (S02)
Tads		225+13 (8T97)		140+18 (LS244)
Trwh	30		30	
Tdevsel-		96 (3 x LS138)		65 (LS154+LS138)
Tiosel-		64 (2 x LS138)		38 (LS138)
Tiostb-		32 (LS138)		15 (LS10)
Tdevsel+		18 (LS138)		30 (LS154)
Tiosel+		36 (2 x LS138)		18 (LS138)
Tiostb+		18 (LS138)		15 (LS10)
Tdsu	100+17 (8T28) ^		50+12 (LS245)	
Thr	10		10	
Trs *	200		200	

* - The RDY line must never change states within Trs to end of 02.

- load = 100 pf.

^ - The RFI versions of the Apple II+, revisions A through D motherboards, use an 8304 instead an 8T28.

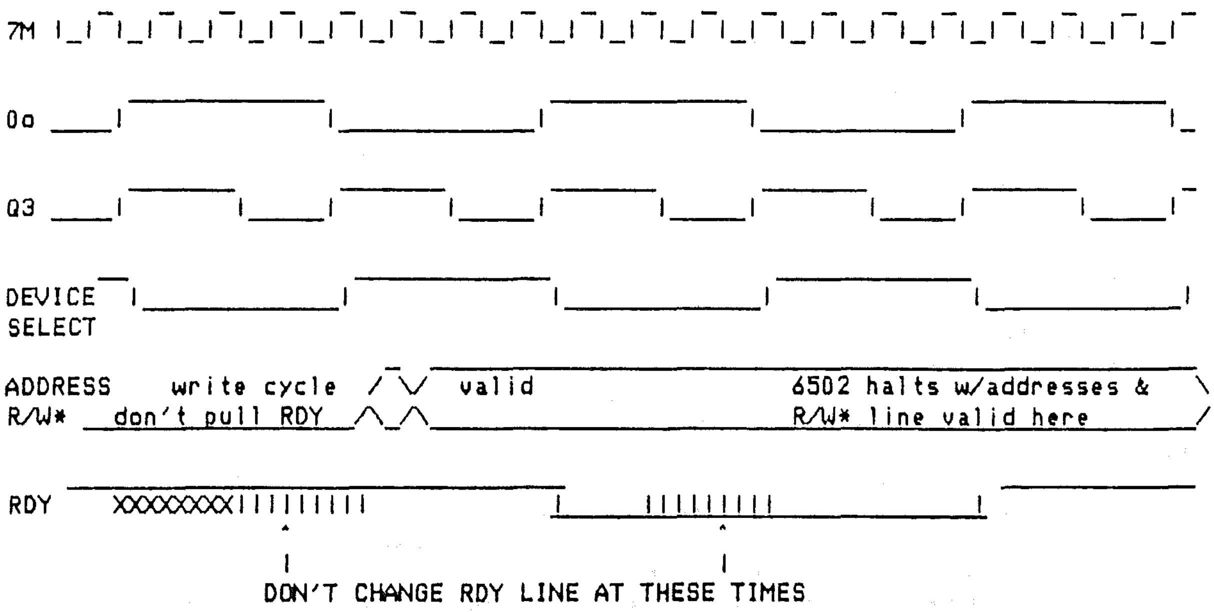
There are three different types of numbers listed above. If a number is standing by itself then it is just the corresponding 6502 or 6502A specification. If a number is followed by parenthesis then it represents the delay, produced by TTL gates, between the 6502 and the peripheral slots. The characters in the parenthesis denote the part number(s) of the part(s) which generated the delay. These parts are typically 74' series TTL except for the 8T28 and 8T97. If there are two numbers in a column with a "+" then the first number signifies the 6502 specification and the second the TTL delay, with the corresponding part number. Most of the TTL delay times are from the Texas Instrument data books. The 6502 specifications are from the Synertek 6502 data sheet and from Synertek application note AN2 - SY6500.

WHEN THE RDY LINE CAN BE CHANGED AND WHEN IT CAN'T

As can be seen from these diagrams, the RDY line should not be gated with the 0o trailing edge since this happens around the same time as the falling edge of 02. This would violate the Trs specification and probably force the 6502 to perform erratically. Gating the RDY line with the trailing edge of Q3 during 0o might work, but this could leave as little as 25 ns. for the signal to be valid. In other words Q3 must enable the RDY line low within 25 ns. of Q3 changing states. If this output cannot be guaranteed stable, then the RDY line might violate the Trs specification.

The safest time to pull the RDY line is using the 0o rising edge, but this edge occurs before I/O SELECT, I/O STROBE, or DEVICE SELECT is enabled. Therefore this scheme will not work if any of these three enables is used by the peripheral card. For example, many peripheral cards use memory mapped I/O to transfer data, with the cards registers designed to reside in the DEVICE SELECT memory space. Location C0n0 (where n = 8 + slot number of peripheral card) might hold the status of the card, and location C0n1 might be used to read a device such as a disk or an A/D converter. The card uses the DEVICE SELECT signal, pin 41 on the slot, and the 4 low order address lines to determine if the 6502 wants to read the status register or read from the A/D converter. Typically, the status register can put its data on the bus within 200 ns., easily meeting the set-up requirements of the 6502. But the A/D converter might take at least 100 us. before it can respond with data. The RDY line must be pulled low to allow time for the A/D converter to set up the data bus. Notice that the peripheral card doesn't know that it should pull the RDY line low until after the)EVICE SELECT signal has gone low. This signal doesn't go low until after 0o goes high, so the 0o rising edge can't be used to enable the RDY line for this peripheral card.

There are a few ways around this problem. One solution would be to decode the C0n0 address on the peripheral card and not use DEVICE SELECT. This also requires either putting user selectable switches on the card for setting the slot number, or making the card slot dependent. Another solution is to pull the RDY line low using one of the first three edges, trailing or leading, of the 7M clock. These edges occur at 70, 140, and 210 ns. into 0o and are trailing, leading, then trailing edges, respectively. The best solution is to use the DEVICE SELECT signal to enable the RDY line. The following timing diagram should help.



DON'T PULL RDY DURING WRITE CYCLES

Because there is no acknowledge response from the 6502, the peripheral card must do some of its own housekeeping and check if a write cycle is taking place. On write cycles the 6502 will not halt, but continue running until the next read cycle. After a slow peripheral pulls the RDY line and before it tries to get on the bus, it must make sure the 6502 is not in the middle of a write cycle. Otherwise there will be a bus crash, as both the peripheral card and 6502 try to drive the bus. One simple way to prevent this bus crash from occuring is to make sure the peripheral card doesn't pull the RDY line low during a write cycle. This can be guaranteed by checking the R/W* line when 0o goes high or DEVICE SELECT goes low. The R/W* line will be stable by this time.

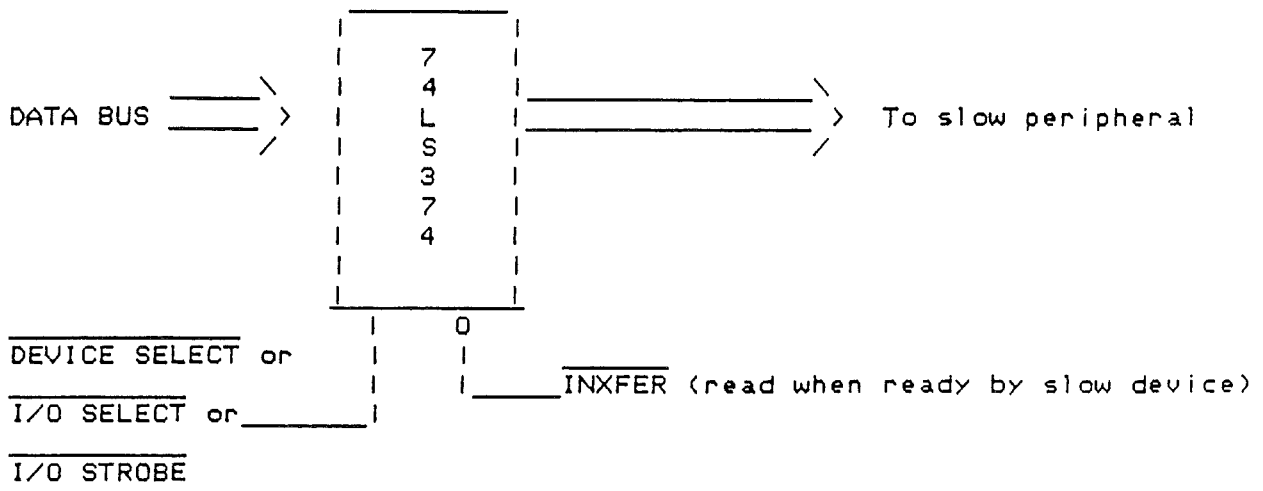
RELEASING THE RDY LINE

When the RDY line is released the 6502 will continue the cycle that was originally halted and allow the 6502 to read the data bus. Data will be read on the next trailing edge of 02 by the 6502, as long as RDY doesn't change within Trs of the end of 02. When the peripheral device has set the data bus up with the correct data it can release the RDY line to complete the read cycle. Releasing the RDY line has exactly the same constraints as pulling the line; Do not change RDY within 200 ns. of the end of 02.

The RDY line can be released before data has been set up, if the data will be valid within specification. This means that RDY can be released in the middle of 01 if the data bus will be valid 117 ns. before 02 trailing edge, for the Apple][(62 ns. for the Apple //e).

SLOW WRITES

Since the 6502 can't be halted during write cycles, if a device requires longer than one cycle to receive data then the data must be buffered. Here is an example of how to accomplish this:



NOTE: It is very easy to overrun the slow peripheral using this scheme, since it only buffers one byte at a time. Don't send data twice to the buffer within the maximum allowed time between slow peripheral reads.

APPLE //e TECHNOTE #5

5-JAN 84

One of the new features of the Apple //e is the ability to add more memory or overide existing memory from a peripheral card. This feature, which uses the INH (inhibit) line on the peripheral slots, has been expanded from its original purpose on the Apple II+ of disabling the onboard ROM and allowing the language card (RAM) to reside in the same address space. The Apple //e allows any part of memory to be replaced by memory on a peripheral card. This article explains how a peripheral card should use the INH line.

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

PB

Peter Baum
Developer Technical Support
M.S. 22-W

Apple Computer
20525 Mariani
Cupertino, Ca. 95014

Jan. 5, 1984

Using the INH line on the Apple //e

One of the new features of the Apple //e is the ability to add more memory or overide existing memory from a peripheral card. This feature, which uses the INH (inhibit) line on the peripheral slots, has been expanded from its original purpose on the Apple][+ of disabling the onboard ROM and allowing the language card (RAM) to reside in the same address space. The Apple //e allows any part of memory to be replaced by memory on a peripheral card.

USES

Presently, only a few peripheral devices use the INH line in the //e for memory expansion. One type of card uses INH for RAM expansion by switching in extra language cards, while another class of cards uses it to extend the built-in 80-column ROM code by replacing it with their own ROM code. Other cards use INH so that they can have more than one stack and zero page. Future peripheral cards can take advantage of the INH line to do even fancier memory expansion, such as keeping multiple programs running in memory at the same time.

More memory, either ROM or RAM, can be added by mapping the memory into the same address space as existing memory. The processor can then select which memory, the onboard or the additional, it wants to use by setting a register (or softswitch). This technique of switching different blocks of memory into the same address space is called bank switching. An example of this technique for extending memory is found in the Apple][+ language card and in the bank switched memory on the //e.

HOW IT WORKS

When the INH line, pin 32 in slots 1-7, is pulled low, all memory on the motherboard and in the auxiliary slot is disabled (including memory on the 80-column and extended 80-column cards). This allows a peripheral card, in slots 1-7, to enable it's memory onto the bus.

When the 6502 reads a byte from memory the data typically comes from one of three places: motherboard ROM, motherboard RAM, or RAM on one of the 80-column cards in the auxiliary slot. When the INH line is pulled low, all of the above mentioned ROM anD RAM is disabled and will not drive the data bus. This allows the peripheral slots to drive the bus by enabling data onto it. The 6502 will then read data from the peripheral card instead of a location on the motherboard or auxiliary slot.

During a 6502 write cycle, if the INH line is pulled low, then motherboard and auxiliary card RAM are both disabled. A peripheral card can then read a byte off the data bus and store it away.

IMPLEMENTATION

Because pulling the INH line low disables all of memory, the peripheral card must be very careful when it does this. If only a small piece of memory is to be banked into a specific address space, then the INH line should only be pulled on memory references to that address space. Otherwise the motherboard memory will be disabled and the processor will read/write to the wrong memory and the program won't work properlyn For example, if a peripheral card wants to replace the zero page with memory on the card, then the INH line should be pulled low only on references to the address space between $0 and $FF. If the INH line is pulled during a processor instruction fetch from the monitor ROM at $F800, the 6502 will read the wrong instruction (or a floating bus) and probably crash the program.

Pulling the INH line at specific addresses is called select decoding. The hardware on the peripheral card does this by checking the address bus of the 6502, and if the address falls in the correct range the card pulls the INH line low. In the earlier example of a new zero page, if the address bus was in the range $0-$FF the card would pull INH low.

DIFFERENCES: //e vs. II+

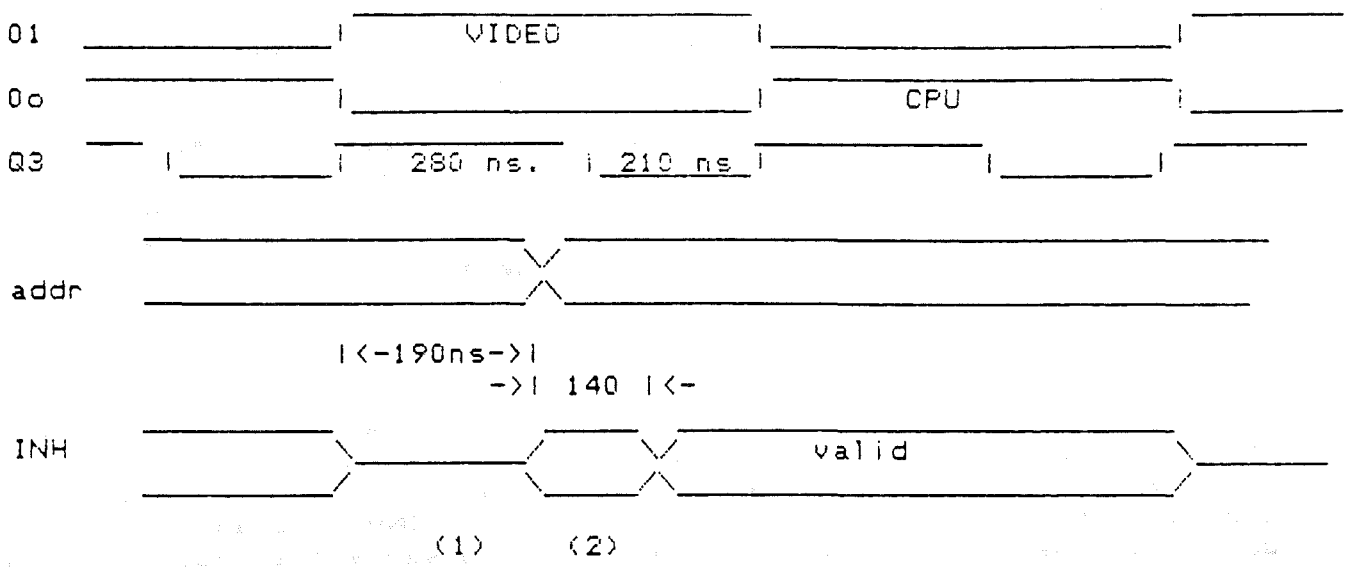
On the Apple II+, select decoding was not necessarily needed because the INH line only affected the ROM and not the RAM. If the Apple II+ peripheral card wanted to bank in extra language cards at 4he addresses $D000-$FFFF then it could pull the INH Line and keep it low during any memory access. This would just disable the onboard ROM and not any other memory accesses such as zero page or stack. This same card would not work in the //e, since the next instruction fetch to RAM after pulling INH low would read a floating bus because all the memory would be disabled.

ANOTHER FEATURE

For those of you who love to muck around in the guts of the Apple //e one more feature has been added to the INH function. The INH line will also overide DMA accesses to memory on the motherboard. This means that if a peripheral card uses DMA to read or write to memory, another peripheral card could pull the INH line and process the DMA access. An example of this would be a co-processor card using the memory on a RAM card in another slot. Rather than have the co-processor write to the memory on the motherboard and then have the 6502 write to the RAM card, the co-processor can write to an address that the RAM card recognizes. The RAM card could then pull the INH line and it would be free to read or write the data bus. This technique could also be used by a co-processor to write directly to a printer card in another slot.

TIMING

The peripheral card must wait for the address bus to settle, which occurs a maximum of 190 ns. after the falling edge of 0o, before pulling the INH line. (The 6502A maximum address setup time is 140 ns. from 02, with a worst case 6502A skew of 50 ns. from 0o to 02.) To guarantee that the RAM is disabled and a write doesn't accidentally take place to the motherboard, the INH line must be pulled low within 330 ns. of 0o.



(1) The INH line can be pulled high at this time.
(2) The INH line can be pulled low (or high) after the addresses are valid at 190ns, but before 330 ns. (from 0o).

CIRCUITS

A simple example of a circuit that can be used to implement the INH function is shown below.

```
                 ________
                |        |
A15 ----------- | select |
A14 ----------- | decode |                  _________
 .        .     | logic  |                |         |
 .        .     |        |_______________|D   7   Q|_______INH*  J[1-7]32
A0  ----------- |________|  PULLINH*     |    4    |
                                          |    L    |
                  |\                      |    S    |
Q3---------------| >o-------------------|>   3    |
                  |/                      |_   7    |
0o-------------------------------------|G   9    |
                                          |_________|
```

AN APPLICATION

The following circuit can be used to replace the code in the monitor ROM, from location $FC00 to $FFFF, with custom code. Any time the address space between $FC00-$FFFF is accessed the INH line is pulled low, the motherboard memory is disabled, and the circuit's 1K RAM is enabled instead. Part of this feature can be disabled and the motherboard memory can be read by keeping the switch connected to +5 volts (READDIS). Whenever the system writes to any location in the address space $FC00-$FFFF, the circuit will disable any RAM on the motherboard and instead write into the 1K RAM.

Here is a series of commands that can be used with the circuit to replace the reset vector at $FFFC and $FFFD. A new reset routine can be written that will print the screen or save the status of all the registers whenever the reset key is pressed.

Start the system with the circuit's switch connected to +5 (READDIS). This will enable the system to read the monitor ROM during power up, before the 1K RAM has been initialized.

Get into the monitor by typing CALL -151. The system prompt should now be a '*'.

Copy the monitor ROM into the 1K RAM with the command FC00<FC00.FFFFM <CR>

Change the reset vector so that it jumps to location $300 with this command, FFFC:0 <CR>. Copy your new reset routine into memory starting at location $300.

Set the switch to ground (READEN) so that all future read accesses to $FC00-$FFFF will read the 1K RAM.

For example if these instructions are stored in memory starting at location $300, then when reset is pressed the system will clear the screen and then continue execution in the monitor (prompt='*').

```
$300:20 58 FC    JSR HOME (clears screen)
$303:4C 65 FF    JMP to MON (resume execution in monitor)
```

One of the problems with this circuit is that it also overrides any accesses to the language card. Therefore any program that uses the language card will not work with this circuit. The circuit doesn't keep track of which memory is enabled, ROM or language card RAM, in the $FC00-$FFFF space.

APPLE //e TECHNOTE #6

6-May 84

This article describes the paddle circuit used in the Apple // family of computers. The article starts with a simple description of the circuit used and then takes the reader through a thorough example of a typical paddle read routine. Finally, a few of the anomalies of the paddle circuit are discussed.

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

PB

Peter Baum
20525 Mariani Ave.
Cupertino, Ca. 95014

Apple Computer
MS 22-W

May 6, 1984

A Treatise on the Apple Paddles/Joysticks

This article describes the paddle circuit used in the Apple // family of computers. The article starts with a simple description of the circuit used and then takes the reader through a thorough example of a typical paddle read routine. Finally, a few of the anomalies of the paddle circuit are discussed.

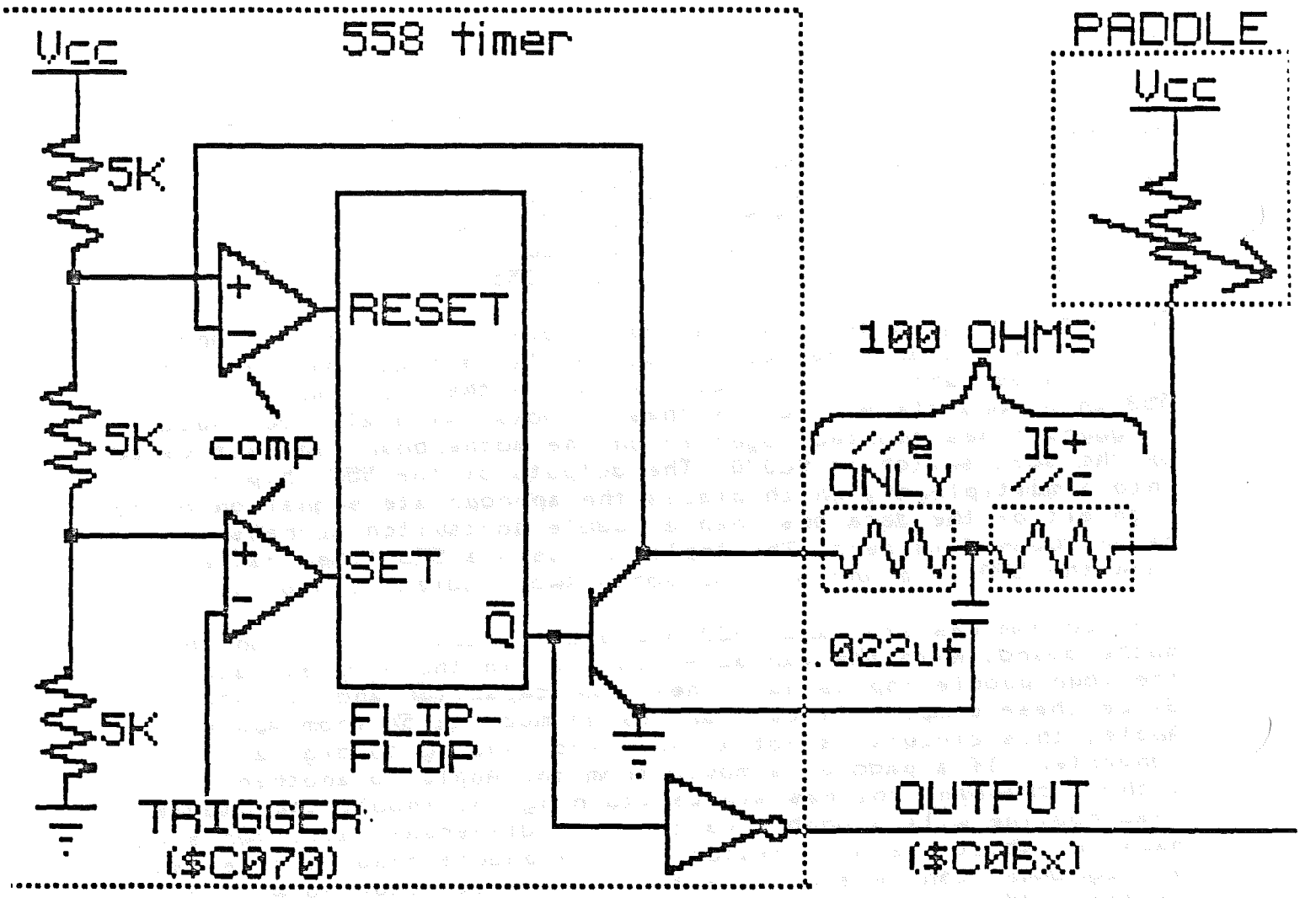
Circuit Description

The value of the Apple paddles (or joystick) is determined by a software timing loop reading a change of state in a timing circuit. The paddles consist of a variable resistor (from 0-150K ohms) which makes up part of the timing circuit. There is a routine in the Monitor ROM, called PREAD, which counts the time until a state change occurs in the paddle circuit. This time is translated into a value between 0 and 255.

The block diagram below shows the paddle circuit for the Apple][+, Apple //c and the Apple //e. The large block on the left illustrates part of the circuitry inside the 558 timer chip. The 558 chip consists of four of these blocks, with all four paddle triggers lines shorted together on the motherboard and activated by the soft switch at $C070. The outputs of the 558 chip run into a multiplexor, which places the appropriate signal onto the high bit of the data bus when a paddle softswitch address in the range $C064-7 is read. The Apple //c uses a 556 timer rather than the 558 chip and only supports two paddles, 0 and 1.

The 100 ohm resistor and .022 microfarad capacitor are on the motherboard, with the variable resistor in the paddle. Each of the four paddle inputs have their own capacitor and resistor. Since these components can vary by as much as 5% from Apple to Apple, this circuit is not a very exact analog to digital converter. If a paddle is moved from one Apple to another without changing the resistance (turning the knob), the paddle read routine will probably calculate a different value for each machine. About the only feature of the paddle read routine that a programmer can depend on is that the value returned will rise if the paddle resistance increases (or fall if the resistance decreases).

The paddle timing circuit on the Apple][+ and Apple //c is slightly different than the one on the Apple //e. On the Apple //e the 100 ohm fixed resistor is between the transistor and the capacitor, while the variable resistor in the paddle is connected directly to the capacitor. On the Apple][+ and //c the capacitor is connected directly to the transistor and the fixed resistor is in series with paddle resistor.



Paddle Circuit for Apple][+, //c and //e Showing 558 Timer

An Example of Typical Paddle Read Routine

The timing circuit works by discharging a capacitor through a transistor, then shutting the transistor off and letting the paddle charge the capacitor by supplying current through the variable resistor. The rate at which the capacitor charges is a function of the variable resistance; the lower the paddle resistance the greater the current and the faster the capacitor charges. When the capacitor reaches a predetermined value it changes the state of a flip-flop. The paddle read routine counts the time it takes for the capacitor to rise and change the flip-flop.

Let's step through an example of a typical paddle read operation. For now we'll assume the capacitor has already been discharged and in a few pages I'll explain when this assumption can be made and when it can't.

The software starts by reading the softswitch at location $C070, which strobes the trigger lines on the 558 timer. This causes two events to occur, the output signal (which is read at $C064-$C067 for paddle 0-3, respectively) goes high and the transistor turns off.

The software, after initially strobing the trigger line, executes a timing loop which reads the state of the output signal. When the output signal changes from high to low the the software jumps out of the timing loop and returns a value indicating the time. The monitor PREAD routine consists of a 11 usec. loop and will return a value between 0 and 255. (NOTE: The firmware listing is wrong and says the loop is 12 Usec.). The timing loop returns 255 if the circuit takes longer than 2.82 msec. for the state change to occur.

```
* PADDLE READ ROUTINE
* ENTER WITH PADDLE NUMBER (0-3) IN X-REG

FB1E:AD 70 C0   PREAD    4  LDA  PTRIG      ;TRIGGER PADDLES
FB21:A0 00               2  LDY  #0         ;INIT COUNTER
FB23:EA                  2  NOP             ;COMPENSATE FOR 1ST COUNT
FB24:EA                  2  NOP
FB25:BD 64 C0   PREAD2   4  LDA  PADDL0,X   ;COUNT EVERY 11 USEC.
FB28:10 04               2  BPL  RTS2D      ;BRANCH WHEN TIMED OUT
FB2A:C8                  2  INY             ;INCREMENT COUNTER
FB2B:D0 F8               3  BNE  PREAD2     ;CONTINUE COUNTING
FB2D:88                     DEY             ;COUNTER OVERFLOWED
FB2E:60                     RTS             ;RETURN W/VALUE 0-255
```

Inside the 558 timer chip, when the trigger is strobed low, the comparator that feeds the set input of the flip-flop is triggered, which in turn sets the output of the 558 timer. At the same time the transistor, which has held the capacitor near ground by sinking current from it, is shut off. The capacitor

can now charge up using the current supplied by the paddle. The smaller the paddle's resistance the more current the paddle will supply and the faster the capacitor charges. After some time, the capacitor will charge to the threshold value of 3.3 volts, which is set by the voltage divider network in the 558 timer, and the comparator that feeds the reset input on the flip-flop will trigger. This sets the output signal ($C06x) of the 558 timer low, which indicates to the software that the circuit has timed out.

```
                                                                  <=====THRESHOLD
TRIGGER $C070   __|__|_____________________________
                                  _.......-------|
FEEDBACK TO                     _.-'             |
RESET COMP     _____.-'                          \_________

                                _____________________
OUTPUT         _______|                          |___________

                        |<--TIMING VALUE-->|

                        0-2.82 MILLISECONDS
```

Resetting the flip-flop turns the transistor on, which discharges the capacitor very quickly (normally less than 250 ns). That paddle can then be read again.

A Closer Look at the Hardware

The First Anomaly

Notice that the last sentence states that the paddle can be read again and not the paddles. If another paddle is read immediately after the first, it may yield the wrong value. To show this I'll step through an example of reading a second paddle immediately after finishing the first.

In this example I'll assume that the first paddle has been set with a very low resistance, while the second paddle has a high resistance. The first paddle will time out very quickly and return with a small value, while the second paddle will take longer and yield a larger value.

We start reading the paddles by testing the paddle outputs to see if they're low, which indicates that the capacitor has been discharged. Assuming that the outputs are low, the next step is to trigger the 558 timer ($C070), which turns off the transistor and allows the capacitors to charge. Since all of the trigger input lines are shorted together all four of the capacitors will charge up, but at different rates since the paddle resistances have been set to different values. The voltage on the capacitor

for the first paddle will reach the threshold voltage very quickly since the paddle resistance has been set low, and therefore the timing loop will time out quickly.

At this point the capacitor for the second paddle is still charging and has not reached the threshold value yet, since the paddle resistance was set to a high value. The transistor for the second paddle is still turned off due to the initial trigger used for reading paddle one. This means that the capacitor for the second paddle has not been discharged.

Any attempts at reading the second paddle now will only yield false results. The capacitor is partly charged and therefore will reach the threshold value much faster than if the capacitor had been completely discharged. If the timing loop is used it will return with a smaller value than it would if the capacitor had been completely discharged. Notice that retriggering (reading location $C070) the 558 timer will not help, since that only keeps the transistor turned off and doesn't help discharge the capacitor. The only way for the capacitor to discharge is to let the circuit timeout completely by letting the capacitor charge until it resets the flip-flop.

To read the second paddle the capacitor must first be discharged, which is only done when the threshold value is reached and the 558 timer flip-flop is reset. The only way to guarantee that the capacitor is discharged is if the transistor is on. This condition is met when the paddle output is low. Therefore start every paddle read either by waiting for at least 3 ms. before strobing the trigger input or testing to make sure that the paddle output is low.

If after 4 ms. the paddle output is not low then there is a good chance that there is no paddle connected. It may also indicate that a peripheral with a larger maximum value resistor than the 150K ohms used by the Apple paddles is attached. Some peripheral devices use this technique of a larger variable resistor so that more than 256 points of resolution can be determined. Of course this requires a custom software driver and the Monitor PREAD routine can't be used.

The Apple //e Anomaly

The problem with Apple //e paddle input is that the capacitor may not be discharged by the transistor. Typically, the transistor will discharge the capacitor in less than 250 ns. on the Apple II+. But on the Apple //e if the paddle resistance is very low then the paddle may supply enough current to always keep the capacitor charged.

Because the fixed resistor (100 ohms) on the Apple //e motherboard is between the capacitor and the transistor, there will be a voltage drop across the resistor if the capacitor stays charged. When the transistor is shut off by the trigger

strobe, this voltage drop will disappear and the capacitor, which may be near the threshold voltage, will trigger the reset comparator earlier than it would if the capacitor had been discharged completely. The net affect of this is that the paddles will read zero on the Apple //e when they would read a small value on the Apple][+ or //c.

Other circuits which expect the capacitor to discharge completely may not work properly. A circuit which attempts to simulate a paddle through active components such as a digital to analog converter may be able to source enough current that the capacitor never discharges and the paddle always reads zero.

Hopefully, this article has given the reader a good feel for the paddle circuitry and the routines which determine the paddle values. To reinforce the material covered you should try writing your own paddle read routine. For example, you could write a read routine that would read two paddles at once. The software loop will not have the 11 usec. resolution of the PREAD routine, but you'll find it stills works just fine. Happy programming!

APPLE //e TECHNOTE #7

3-April 84

This article describes three different types of interfaces, serial, parallel, and IEEE-488, that are currently used to connect a printing device to an Apple //. The interface cards available from Apple and the protocol to connect to an Apple printer are briefly described. Pin out configuration and switch settings for these interfaces cards and printers is also included.

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

PB

Peter Baum
Apple Computer
20525 Mariani Ave.
Cupertino, Ca. 95014

Apr. 3, 1984

Connecting a Printer to an Apple //

This article describes three different types of interfaces, serial, parallel, and IEEE-488, that are currently used to connect a printing device to an Apple //. The interface cards available from Apple and the protocol to connect to an Apple printer are briefly described. Pin out configuration and switch settings for these interfaces cards and printers is also included.

Serial

Currently, Apple sells a card, called the Super Serial Card (SSC), that can be used to connect an Apple printer to an Apple // (Apple sells a dot-matrix printer, called the Imagewriter, a daisy-wheel printer, and a plotter, which all use a serial interface). The SSC replaces both the Communications card and the Hi-speed Serial card. The SSC supports the firmware (Pascal 1.1) protocol except for the optional control and interrupt handling routines. For more information on the firmware protocols see Appendix A of the Super Serial Card manual (Part # A2L0044).

If the SSC is to be connected to an Imagewriter printer then the switches should be set as follows:

```
                SSC                                        Imagewriter
_____________________________________                    _______________
|     1234567 1234567  modem      |                    |   87654321  |
|ON     ' '' '  ''       v        |DB-25      DB-25|ON    ''       |
|OFF ,,, ,     ,,  ,, terminal    |================|OFF ,,  ,,,,   |
|      SW1      SW2               |   (590-0037)   |      SW1    |
|                                 |                |             |
|_________________                |                |     4321    |
                   |____________|                | ON     ''   |
                                                  | OFF  ,,     |
' = switch is in up (closed/on) position          |      SW2    |
, = switch is in down (open/off) position         |_____________|
v = The jumper block should point toward terminal
```

Note: Switch 1-5 on the printer must be in the on (up) position for the printer to work with the Apple //e.

These switch settings set the serial interface to use 9600 baud printer mode with 8 data bits, one stop bit, no delay after carriage return (CR), 80 column line width, no echo to screen, and automatic line feed sent after CR.

The Apple part number for a cable that connects the SSC to an Imagewriter is 590-0037. This cable consists of two male DB-25 connectors with pins 1-8, 12, 13, 19, 20, 23 wired pin to pin and shielded.

The SSC has a 10-pin header on it, but comes with a cable which connects the header to a female DB-25 connector. The DB-25 can be configured as either a modem (DCE) or as a terminal (DTE) using a jumper block (in the latter case the jumper block acts as a modem eliminator). Though the pin out configuration of the DB-25 connector is well defined, there is no standard use of the handshake signals. Different printers will use the handshake lines for different functions. The following table shows the pinout for the DB-25 on the SSC. Consult the printer manual for more specific information on which signals are used.

10-pin Header	Signal Name		Female DB-25 pinout Terminal	Modem	notes
1	Frame Ground	(FRMGND)	1	1	
2	Transmit Data	(TxD)	3	2	
3	Receive Data	(RxD)	2	3	
4	Request To Send	(RTS)	8	4	
5	Clear To Send	(CTS)	8	5	
6	Data Set Ready	(DSR)	20	6	
8	Signal Ground	(SGLGND)	7	7	
10	Data Carrier Detect	(DCD)	4,5	8	*1
7	Secondary Clear to Send	(SCTS)	19	19	*2
9	Data Terminal Ready	(DTR)	6	20	

notes:

*1 - only if SW1-7 is closed (on) with SSC
*2 - only if SW2-7 is closed (on) with SSC

```
 1 |\
   | ) 14
   | |
   | |     DB-25
   | |
   | ) 25
13 |/
```

To illustrate an example of a serial interface, I'll use the Imagewriter printer. Here is the pinout and interface specification:

Pin no.	Symbol	Descripition	Direction
1	FG	Frame Ground	
2	TxD	Send Data	Output
3	RxD	Receive Data	Input
4	RTS	Request to Send	Output
7	SG	Signal Ground	
14	FAULT	Fault	Output
20	DTR	Data Terminal Ready	Output

Functional Description:

FG = Grounding line for circuit protection
TxD = Serial transmission line from printer to computer
RxD = Serial transmission line from computer to printer
RTS = True when printer is turned on
Fault = False when printer deselected; true when selected
DTR = True if printer on and ready to receive

The printer uses a hardware handshaking scheme, called the Data Transfer Ready protocol, to receive data. Whenever the capacity of the input buffer is less than 30 characters, the printer sends a busy signal by setting the DTR line false. The computer must stop transmission within the next 27 characters or the printer will ignore the excess data. The DTR line is also set false when the printer is deselected, and when it receives a DC3 character. The DTR line is true whenever there is room for at least 100 characters in the input buffer, when the printer is turned on, selected, and has received a DC1 character.

Parallel

Apple currently ships a parallel card, called appropriately the Parallel Interface Card (PIC), which can be used to connect a parallel printer to an Apple // (Apple used to sell a dot-matrix printer called the DMP, but has discontinued shipping any printers with a parallel interface). The PIC replaces the Parallel Printer Interface Card and the Centronics Interface Card. The PIC doesn't support the firmware protocol, so Pascal identifies the card as a printer card (described in Pascal protocols).

Most commonly used printers will operate properly if the switches on the PIC are set as follows:

```
 _____________________
|        1234567      |
|  ON      ''         |
| OFF   ,,,   ,,      |
|_____________________|
```

This sets the parallel interface to transfer data using a 1 microsecond strobe pulse of negative polarity when sending data, while receiving a negative acknowledge signal, with interrupts disabled.

The PIC has a 26-pin header, but comes with a cable which connects the header to a female DB-25. The Parallel Printer Card and the Centronics Card used a 20-pin header. Most parallel printers (90%) use a 'microribbon 36' as the connector. The pinout varies from printer to printer, but the following table covers most printers (Apple DMP, Epson). For other printers refer to page 7 of the Parallel Interface Card manual (Part # A2L0045).

PIC Function	Printer Function	26-pin header	DB-25 conn.	36-pin microribbon	20-pin header
Ground	Ground	3	2	19	1
Ground	Ground	22	24	16	20
Ground	Ground	7	4		
Ground	Ground	14	20		
ACK	Acknowledge	6	16	10	2
Strobe	Strobe	4	15	1	8
DO 0	Data 1	9	5	2	10
DO 1	Data 2	11	6	3	11
DO 2	Data 3	15	8	4	12
DO 3	Data 4	18	22	5	13
DO 4	Data 5	20	23	6	14
DO 5	Data 6	21	11	7	15
DO 6	Data 7	23	12	8	16
DO 7	Data 8 (#2)	25	13	9	17
DI 3	Fault	24	25	32	6
DI 4	Busy	2	14	11	7
DI 5	Paper out	12	19	12	9
DI 6	Select	16	21	13	18
DI 7	Enable (#1)	10	18 7	35	19
Apple internal part # for cable		590-0049B		590-0042B	

(#1) - Pin 7 is blocked on the female DB-25 connector and omitted on the male DB-25 connector to prevent the insertion of serial connectors into parallel ports.

(#2) - This may be assigned a 'hard' value for some printers to distinguish between graphics and normal character sets.

<u>Functional Description of Signal for Typical Printer</u>

Strobe = Printer clocks data in on falling edge
ACK = Set low by printer to indicate it has processed last character and is ready for another
Fault = Set low if printer detects fault condition
Busy = Set high by printer to indicate not ready
Paper out= Used by printer to indicate out of paper
Select = Output from printer, set high if printer selected
Enable = Set high by printer to indicate printer active

Since the PIC can also be used to input parallel data and doesn't act as only a printer card, it is no longer referred to as a printer card, but instead as a general purpose parallel card.

IEEE-488

Though most printing instruments on the market today use either a serial or parallel interface, another standard interface, IEEE-488, is also available. These devices can be connected to the Apple // through the Apple IEEE-488 Interface Card. Currently Apple doesn't sell any printer devices that use the IEEE-488 interface, but other companies supply them. One of the advantages of the IEEE-488 bus over either the parallel or serial (RS-232) busses is that more than one type of printer can be attached to the bus at the same time. This means that both a fast dot-matrix and a daisy wheel printer can be hooked to the Apple with only one peripheral card.

The IEEE-488 bus standard is a well defined 8-bit parallel, byte serial, asynchronous data transfer interface. The standard has been thoroughly documented with the most complete description available from the Institute of Electrical and Electronic Engineers (IEEE) in New York. Standard cables are manufactured by many companies, and usually advertised as either IEEE-488, General Purpose Interface Bus (GPIB), or Hewlett-Packard Interface Bus (HPIB) cables.

The IEEE-488 card doesn't support the firmware protocols, so an assembly language driver must be used to access the card from Pascal (See Appendix F of the IEEE-488 Interface User's Guide, part number A2L0037).

Appendix A

Product	Order Part #
Super Serial Card	A2B0044
SSC to Imagewriter Accessory Kit *	A2C0352
SSC to Imagewriter external Cable	590-0037
Imagewriter	A9M0303
Apple Daisy Wheel Printer (DWP)	A3M0025
SSC to Apple DWP Accessory Kit *	A2C0351
Apple Color Plotter	A9M0302
SSC to Color Plotter Accessory Kit *	A2C0302
Parallel Card	A2B0021
IEEE-488 Interface Card	A2B0015
SSC manual	A2L0044
Parallel Interface Card manual	A2L0045
ProDOS Technical Reference Manual	A2W0010
Apple //e Reference Manual	A2L2005
Apple //e Design Guidelines	A2F2116

* The accessory kit includes a cable and manuals

Apple][Monitor Entry Points

2 August 1984

This document lists all supported entry points in the Apple][family $F800 Monitor ROM. This is NOT a programming guide. Since each member of the Apple][family has variations in the implementation of the Monitor, it is the individual programmer's responsibility to identify the machine type and take appropriate action when calling these routines. The only purpose of this document is to reassure software developers that the entry points for these routines will remain intact and that there is no committment to keep any other Monitor code in the same locations.

Disclaimer of all Warranties and Liabilities

Notice

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

CJS

$F800 PLOT Plot on the low-resolution screen

PLOT puts a single block of the color value set by SETCOL on the low-resolution display screen. The block's vertical position is passed in the accumulator, its horizontal position in the Y register. PLOT returns with the accumulator scrambled, but X and Y intact.

$F819 HLINE Draw a horizontal line of blocks

HLINE draws a horizontal line of blocks of the color set by SETCOL on the low-resolution graphics display. Call HLINE with the vertical coordinate of the line in the accumulator, the leftmost horizontal coordinate in the Y register, and the rightmost horizontal coordinate in location $2C. HLINE returns with A and Y scrambled, X intact.

$F828 VLINE Draw a vertical line of blocks

VLINE draws a verticle line of blocks of the color set by SETCOL on the low-resolution display. You should call VLINE with the horizontal coordinate of the line in the Y register, the top vertical coordinate in the accumulator, and the bottom vertical coordinate in location $2D. VLINE will return with the accumulator scrambled.

$F832 CLRSCR Clear the low-resolution screen

CLRSCR clears the low-resolution graphics display to black. If CLRSCR is called while the video display is in text mode, it fills the screen with inverse at-sign (@) characters. CLRSCR destroys the contents of A and Y.

$F836 CLRTOP Clear the low-resolution screen

CLRTOP is the same as CLRSCR, except that it clears only the top 40 rows of the low-resolution display. (Mixed-mode)

$F847 GBASCALC Calculate base address for low resolution graphics

GBASCALC calculates the base address of the line on which a particular pixel is to be plotted. The accumulator is scrambled.

$F85F NXTCOL Increment color by 3

NXTCOL adds 3 to the current color (set by SETCOL) used for low-resolution graphics. The accumulator is scrambled.

$F864 SETCOL Set low-resolution graphics color

SETCOL sets the color used for plotting in low-resolution graphics to the value passed in the low nybble of accumulator. The colors and their values are listed in the technical reference manual. The accumulator is scrambled.

$F871 SCRN Read the low-resolution graphics screen

SCRN returns the color value of a single block on the low resolution graphics display. Call it with the vertical position of the block in the accumulator and horizontal position in the Y register. Call it as you would call PLOT (above). The color of the block will be returned in accumulator. No other registers are changed.

$F88E INSDS2 Set-up indexes for opcode in A register

INSDS2 expects to find the opcode in the accumulator. It then sets up formats, modes, and indexes into the mnemonic table. Upon entry, the X register must be zero. Upon exit the accumulator and X register are scrambled.

$F8D0 INSTDSP Display disassembled instruction

INSTDSP disassembles and displays one instruction pointed to by the program counter (PCL-PCH). None of the registers are preserved.

$F940 PRNTYX Print contents of Y and X registers as hex

PRNTYX prints the contents of the Y and X registers as a four-digit hexadecimal value. The Y register contains the first byte output, the X register contains the second. On return, the contents of the accumulator are scrambled.

$F941 PRNTAX Print A and X in hexadecimal

PRNTAX prints the contents of the A and X registers as a four-digit hexadecimal value. The accumulator contains the first byte output, the X register contains the second. On return, the contents of the accumulator are scrambled.

$F944 PRNTX Print contents of X register as hex

PRNTX prints the contents of the X register as a two digit hexadecimal value. On return, the contents of the accumulator are scrambled.

$F948 PRBLNK Print 3 spaces

PRBLNK outputs three blank spaces to the standard output device. On return, the accumulator usually contains $A0, the X register contains 0.

$F94A PRBL2 Print many blank spaces

PRBL2 outputs from 1 to 256 blanks to the standard output device. Upon entry, the X register should contain the number of blanks to be output. If X=$00, then PRBL2 will output 256 blanks. On return, the accumulator usually contains $A0, the X register contains 0.

$F953 PCADJ Adjust program counter

PCADJ increments the program counter by 1, 2, or 3 depending on the LENGTH byte stored at $2F, 0 = 1 byte, 1 = 2 bytes, 2 = 3 bytes. All registers are scrambled.

$FA40 IRQ IRO handler

IRQ first determines if the interrupt request was from a BRK instruction. If not, control is sent to IRQLOC ($3FE). The accumulator is stored (at $45 with the][,][+, and original //e monitors and pushed on the stack with the "ICON" //e, and //c monitors). When the $03FE interrupt handler terminates with an RTI, all registers are restored. (Generally called by operating system, not user.)

$FA4C BREAK BRK handler
BREAK saves the registers and JMPs to BRKV ($3F0).

$FA62 RESET Hardware reset handler
RESET sets normal video out and keyboard in, re-initialize system, set and clear various annunciators (depending on system type), clear keyboard, and falls through to NEWMON.

$FAA6 PWRUP System cold start
PWRUP prints system type at top of screen, sets page 3 vectors equal to cold start of current BASIC. It then falls through to SLOOP.

$FABA SLOOP Disk controller slot search loop
SLOOP is the disk controller search loop. It searches for a disk controller beginning at the peripheral ROM space pointed to by $00-$01. If a disk controller is found, it JMPs to the ROM code. Otherwise, it cold-starts BASIC. (Required to support the ProFile card boot code.)

$FAD7 REGDSP Display contents of registers
REGDSP sets location A3 ($40-$41) equal to $0045, then displays the contents of the registers (from locations $45 thru $49) with labels. (Setting A3 prepares the user for modifying memory beginning at $45.) The accumulator and X register are not preserved.

$FB19 RTBL Register names table
RTBL contains the ASCII codes for "AXYPS" (hi-bit set), the names of the registers.

$FB1E PREAD Read a hand controller
PREAD returns a number that represents the position of a hand control. You pass the number of the hand control in the X register. If this number is not valid (not equal to 0, 1, 2, or 3), stange things may happen. PREAD returns with a number from $00 to $FF in the Y register. The accumulator is scrambled.

$FB2F INIT Initialize system
Clears $48, the 6502 status register save locations, and sets softswitches to LO-RES, PAGE 1, TEXT, then falls through to SETTXT.

$FB39 SETTXT Set text mode
SETTXT sets text mode and LDA #0 to set window top then JMPs to SETWND.

$FB40 SETGR Set graphics mode
SETGR sets mixed graphics mode and clears the graphics portion of the screen then LDA #20 to set window top and falls through to SETWND.

$FB4B SETWND Set text window
SETWND sets a full width text window with the window top set to the value in the accumulator and bottom set to the bottom of the screen. It then VTABs to line 23.

$FB5B TABV Vertical tab
TABV merely stores the value in the accumulator in location CV ($25) and calls VTAB ($FC22).

$FB60 APPLEII Print machine type
APPLEII clears the screen and prints the machine type centered at the top of the screen. A and Y are scrambled.

$FB6F SETPWRC Create power-up byte
SETPWRC calculates the "funny" complement of the high byte of the RESET vector and stores it at PWREDUP ($3F5).

$FB78 VIDWAIT Check for a pause (CONTROL-S)
VIDWAIT checks the keyboard for a CONTROL-S if it is called with an $8D in the accumulator. If a CONTROL-S is found, it falls through to KBDWAIT. If not, control is sent on to VIDOUT where the character is printed and the cursor advanced.

$FB88 KBDWAIT Wait for keypress
KBDWAIT waits for a keypress. The keyboard is cleared unless the keypress is a control-C then control is sent on to VIDOUT where the character is printer and the cursor advanced.

$FBB3 VERSION Monitor ROM identification byte
VERSION is a byte used to aid in identifying which monitor ROM is installed.

$FBC0 ZIDBYTE Monitor ROM sub-identification byte
This byte provides more detailed identification of the monitor ROM.

$FBC1 BASCALC Text base address calculator
BASCALC calculates the base address of the line for the next text character on the forty column screen. The value is stored at BASH and BASL ($28-$29).

$FBDD BELL1 Beep the speaker
BELL1 toggles the speaker on and off at 1000 hz rate for 0.1 sec.

$FBF0 STORADV Place a printable character on the screen
STORADV stores the value in the accumulator at the next position in the text buffer and falls through to ADVANCE.

$FBF4 ADVANCE Increment the cursor position
ADVANCE advances the cursor by one position. If the cursor is at the window limit it branches to CR.

$FBFD VIDOUT Place a character on the screen
VIDOUT sends printable characters to STORADV. Return, linefeed, forward and reverse space, etc., are vectored to appropriate special routines. (NOTE: This routine does not work in 80-columns on][,][+, and original //e.)

$FC10 BS Back-space

BS decrements the cursor one position. If the cursor is at the beginning of the window, the horizontal cursor position is set to the right edge of the window and the routine falls through to UP. (NOTE: 40-columns only.)

$FC1A UP Move up a line

UP decrements the cursor vertical location by one line unless the cursor is currently on the first line. (NOTE: 40-columns only.)

$FC22 VTAB Vertical tab

VTAB loads the value at CV ($25) into the accumulator and falls through to VTABZ. (NOTE: This routine does not update OURCV in 80-columns.)

$FC24 VTABZ Vertical tab (alternate entry)

VTABZ uses the value in the accumulator to update the base address used for storing values in the text screen buffer.

$FC42 CLREOP Clear to end of page

CLREOP clears the text window from the cursor position to the bottom of the window. CLREOP destroys the contents of A and Y.

$FC58 HOME Home cursor and clear

HOME clears the current window and places the cursor in the home position: the upper left corner of the screen.

$FC62 CR Begin a new line

CR sets the cursor horizontal position back to the left edge of the window and increments the cursor vertical position. It then falls through to LF. (NOTE: 40-columns only.)

$FC66 LF Line-feed

If the cursor vertical position is not past the bottom line, the base address is updated, otherwise the routine falls through to SCROLL. (NOTE: 40-columns only.)

$FC70 SCROLL Scroll the screen

SCROLL moves all characters up one position within the current text window.

$FC9C CLREOL Clear to end of line

CLREOL clears a text line from the cursor position to the right edge of the window. CLREOL destroys the contents of A and Y.

$FC9E CLEOLZ Clear to end of line

CLEOLZ clears a text line to the right of the window, starting at the location given by base address BASL indexed by the contents of the Y register. CLEOLZ destroys the contents of A and Y.

$FCA8 WAIT Delay

WAIT delays for a specific amount of time, then returns to the program that called it. The amount of delay is specified by the contents of the accumulator. With A the contents of the accumulator, the delay is 1/2(26+27A+5A^2) microseconds. WAIT returns with the accumulator zeroed and the X and Y registers undisturbed.

$FCB4 NXTA4 Increment pointer A4

NXTA4 increments the 16 bit pointer, A4 ($42-$43) and then falls through to NXTA1.

$FCBA NXTA1 Compare A1 with A2 and increment A1

NXTA1 does a 16 bit compare of A1 ($3C-$3D) with A2 ($3E-$3F) and then increments pointer A1.

$FCC9 HEADR Write a header to cassette tape (][,][+, //e only)

HEADR writes a header to cassette tape.

$FD0C RDKEY Get an input character

RDKEY is the character input subroutine. It places an appropriate cursor on the display at the cursor position and jumps to the subroutine whose address is stored in KSW (locations $38-$39), usually the standard input subroutine KEYIN, which returns with a character in the accumulator.

$FD1B KEYIN Read the keyboard

KEYIN is the keyboard input subroutine. It reads the Apple's keyboard, waits for a keypress, and randomizes the random number seed at $4E-$4F. When a key is pressed, KEYIN removed the cursor from the display and returns with the keycode in the accumulator. (NOTE: On //e with 80-columns, it interprets escape codes and forward arrows.)

$FD35 RDCHAR Get an input character or ESC code

RDCHAR is an alternate input subroutine that gets characters from the standard input subroutine, and also intreprets the escape codes listed in the technical reference manual.

$FD67 GETLNZ Get an input line

GETLNZ is an alternate entry point for GETLN that sends a carriage return to the standard output, then continues into GETLN.

$FD6A GETLN Get an input line with prompt

GETLN is the standard input subroutine for entire lines of characters, as described in the various technical reference manuals. Your program calls GETLN with the prompt character in location $33; GETLN then falls through to GETLNO.

$FD6C GETLNO Get an input line with prompt (alternate entry)

GETLNO outputs the contents of the accumulator as the prompt. The routine will return with the input line in the input buffer ($200-$2FF) and the X register holding the length of the input line. If the user cancels the input line, either with too many backspaces or a CONTROL-X, then the contents of PROMPT ($33) will be issued as the prompt when it gets another line.

$FD6F GETLN1 Get an input line, no prompt

GETLN1 is an alternate entry point for GETLN that does not issue a prompt before it accepts the input line. If, however, the input line is cancelled, with too long a line, with too many backspaces or with a CONTROL-X, then GETLN1 will issue the contents of PROMPT ($33) as a prompt when it gets another line.

$FD8B CROUT1 RETURN with clear to end-of-line

CROUT1 clears the screen from the current cursor position to the edge of the text window, then falls through to CROUT.

$FD8E CROUT Carriage return output

CROUT sends a RETURN character to the current output device.

$FD92 PRA1 Print RETURN and A1 in HEX

PRA1 sends out a RETURN character followed by the contents of the 16 bit pointer, A1 ($3C-$3D) in hex follwed by a hyphen.

$FDDA PRBYTE Print a hexadecimal byte

PRBYTE outputs the contents of the accumulator in hexadecimal on the current output device. The contents of the accumulator are scrambled.

$FDE3 PRHEX Print a hexadecimal digit

PRHEX prints the lower nybble of the accumulator as a single hexadecimal value. On return, the contents of the accumulator are scrambled.

$FDED COUT Output a character

COUT calls the current output subroutine. The character to be output should be in the accumulator. COUT calls the subroutine whose address is stored in CSW (locations $36 and $37), which is usually the standard character output COUT1.

$FDF0 COUT1 Output to screen

COUT1 displays the character in the accumulator on the Apple's screen at the current output cursor position and advances the output cursor. It places the character using the setting of the Normal/Inverse location. It handles the control charcters [RETURN], linefeed, backspace, and bell. It returns with all registers intact.

$FE2C MOVE Move a block of memory

MOVE copies the contents of memory from one range of locations to another. This subroutine is the same as the MOVE commands in the Monitor, except it takes its arguments from pairs of locations in memory, low-byte first. The destination address must be in A4 ($42-$43), the starting source address in A1 ($3C-$3D), and the ending source address in A2 ($3E-$3F) when your program calls MOVE.

$FE5E LIST Disassemble and list 20 instructions
LIST will disassemble and list to the current output device, 20 assembly language instructions beginning at the location pointed to by A1 ($3C-$3D).

$FE80 SETINV Set inverse text mode
SETINV sets INVFLG so that subsequent text output to the screen will appear in inverse mode.

$FE84 SETNORM Set normal text mode
SETNORM sets INVFLG such that subsequent text output to the screen will appear in normal mode.

$FE89 SETKBD Re-set input to keyboard
SETKBD re-sets the the input hooks ($38-$39) to point to the Keyboard.

$FE8B INPORT Re-set input to a slot
INPORT re-sets the input hooks ($38-$39) to point to the ROM space reserved for a perphireal card (or port) in the slot (or port) designated by the value in the accumulator. (NOTE: In new //e and //c monitor, does a quit if the video firmware was on.)

$FE93 SETVID Re-set output to screen
SETVID re-sets the output hooks ($36-$37) to the screen display routines.

$FE95 OUTPORT Re-set output to a slot
OUTPORT re-sets the output hooks ($36-$37) to point to the ROM space reserved for a peripheral card (or port) in the slot (or port) designated by the value in the accumulator.

$FEB6 GO Begin code execution
GO begins execution of the code pointed to by A1 ($3C-$3D).

$FECD WRITE Write a record on a cassette tape (][,][+, and //e only)
WRITE converts the data in a range of memory to a series of tones at the cassette output port. Before calling WRITE, the address of the first data byte must be in A1 ($3C-$3D) and the address of the last byte in A2 ($3E-$3F). The subroutine writes a ten-second continuous tone as a header, then writes the data followed by a one byte checksum.

$FEFD READ Read data from a cassette tape (][,][+, and //e only)
READ reads a series of tones at the cassette input port, converts them to bytes, and stores the data in a specified range of memory locations. Before calling READ, the address of the first byte must be in A1 ($3C-$3D) and the address of the last byte in A2 ($3E-$3F).

$FF2D PRERR Print ERR
PRERR sends the word ERR, and falls through to BELL. On return, the accumulator contains $87.

$FF3A BELL Output a bell character
BFLL writes a bell [CONTROL]-G character to the current output device. It leaves the accumulator holding $87.

$FF3F RESTORE Restore all registers
RESTORE loads the 6502's internal registers with the contents of memory locations $45 through $48, as saved by BREAK.

$FF4A SAVE Save all registers
SAVE stores the contents of the 6502's internal registers in locations $45 through $49 in the order A, X, Y, P, S. The contents of A and X are changed and the decimal mode is cleared.

$FF58 = $60 Known RTS instruction (IORTS)
This byte must always contain $60.

$FF65 MON Standard Monitor entry with beep
MON beeps the speaker and falls through to MONZ.

$FF69 MONZ Standard Monitor entry point (CALL -151)
MONZ displays the "*" prompt and sends control to GETLNZ.

$FF8A DIG Shift hex digit into A2
DIG shifts an ASCII representation of a hex digit in the accumulator into A2 ($3E-$3F).

$FFA7 GETNUM Transfer hex input into A2
GETNUM scans input buffer starting at position Y. Shifts hex digits into A2 ($3E-$3F). Stops when non-hex digit encountered.

$FFAD NXTCHR Translate next character
NXTCHR is the loop used by GETNUM to parse each character in the input buffer and convert it to a value in A2 ($3E-$3F).

$FFFA NMI Non-maskable interrupt vector
NMI is a two byte pointer to the non-maskable interrupt handler.

$FFFC RESET Reset vector
RESET is a two byte pointer to the RESET handler routine.

$FFFE IRQVECT IRQ vector
IRQVECT is a two byte pointer to the interrupt request handler.

```
2350: 4C 33
2352<2350.2360M
2300<2350.2360^Y

230D: 33 4C 33 4C 33 4C 33 4C      (puts green line next to it)
235D: 13 66 19 66 19 66 19 66      (note first byte)
230D<235D.2363^Y
```

There you have it: a basic explanation of how double hi-res works -- except for one or two anomalies. The first anomaly is that NTSC monitors have a limited display range. The second anomaly shows one of the features of double hi-res versus a limitation of standard hi-res.

An NTSC color monitor decides what color to display based on its "view" of four bit "windows" in each line, starting from the left edge of the screen. The monitor looks at the first four bits, determines which color is called for, and then shifts one bit to the right and determines the color for this new four-bit window. But remember the color depends not only on the pattern, but also the positon of the pattern. To compensate for relative position from the left edge of the screen, the monitor keeps track of where on each line each of these window starts. (For those of you of the technical persuasion, this is done through the use of the color burst signal, which is a 3.58 MHz. clock).

Try this example:

```
2000:0                    Clears screen
2001<2000.3FFFM
2000<2000.3FFF^Y

2001:66                   Draws orange box in uppper left
2401:66
2801:66
2C01:66
3001:66


2050:33                   Draws blue box below and
3402<2050.2050^Y              to the right of the orange
3802<2050.2050^Y
3C02<2050.2050^Y
```

Notice that if the blue box was drawn at the top of the screen, next to the orange box, they would overlap. Yet, the boxes were drawn on two different columns, orange on mb2 and blue on aux1. This can be explained by the previous paragraph, and the sliding windows. The monitor will detect the pattern for orange slightly after the mb2 column, while the pattern for blue shows up before column aux1.

orange pattern:

```
00000001011001110000000        look at four-bit windows and you'll see
  aux2 |  mb2  |  aux1         an orange pattern overlaps on both sides
```

If a pattern is repeated on a line, this overlap doesn't cause a problem, since the same color just overlaps itself. But watch what happens when a new pattern is started next to a different pattern:

```
3002<2050.2050^Y          Puts blue pattern next to orange
2C02<2050.2050^Y
2802<2050.2050^Y
```

Where the blue overlaps the orange, you'll see a white dot. This is because one of the four-bit windows the monitor sees is all 1's. If two colors are placed right next to each other, the monitor will sometimes display a third color, or fringe, right at the boundary. "Fringing" is especially noticeable when there are a lot of narrow columns of different colors next to each other. (Next time you run COLOR TEST take a look at the boundaries between the colors).

```
        orange    blue

00000001011001111001100        note the four 1's in a row
  aux2 |  mb2  |  aux1         at the boundary between
                                orange and blue
```

THE DOUBLE HI-RES ROUTINES

The second anomaly presents a good lead in to the last part of this series, the double hi-res routines, which plot lines. These routines work like the standard hi-res Applesoft commands, HGR, HCOLOR , and HPLOT, except they use the Applesoft ampersand function.

[[At this point BRUN COLOR DBL HIRES]]

There are four ampersand functions:

&H	Clears double hi-res screen
&Cn	Sets the double hires color to n
&Px,y	Plots a point at x,y
&Lx,y	Draws a line from the last point to x,y

TEXT	Returns to 40-column text mode
POKE 49164,0	
POKE 49247,0	