

# Apple II Technical Notes



---

Developer Technical Support

## Apple II Miscellaneous

### #2: Apple II Family Identification Routines 2.2

Revised by: Jim Luther May  
1991

Revised by: Matt Deatherage & Keith Rollin  
November 1988

Revised by: Pete McDonald  
January 1986

This Technical Note presents a new version of the Apple II Family Identification Routine, a sample piece of code which shows how to identify various Apple II computers and their memory configurations.

**Changes since November 1988:** Converted the identification routine from Apple II Assembler/Editor (EDASM) source code to Apple IIGS Programmer's Workshop (APW) Assembler source code. Added the Apple IIe Card for the Macintosh LC to the identification routine's lookup table and memory check routine. Made minor corrections to text.

---

### Why Identification Routines?

Although we present the Apple II family identification bytes in Apple II Miscellaneous Technical Note #7, many people would prefer a routine they can simply plug into their own program and call. In addition, this routine serves as a small piece of sample code, and there is no reason for you to reinvent the wheel.

Most of the interesting part of the routine consists of identifying the memory configuration of the machine. On an Apple IIe, the routine moves code into the zero page to test for the presence of auxiliary memory. (A IIe with a non-extended 80-column card is a configuration still found

in many schools throughout the country.)

The actual identification is done by a table-lookup method.

## What the Routine Returns

[This version (2.2) of the identification routine returns several things:

- A `machine` byte, containing one of seven values:
  - \$00 = Unknown machine
  - \$01 = Apple II
  - \$02 = Apple II+
  - \$03 = Apple III in emulation mode
  - \$04 = Apple IIe
  - \$05 = Apple IIc
  - \$06 = Apple IIe Card for the Macintosh LC

In addition, if the high bit of the byte is set, the machine is a IIGS or equivalent. For all current Apple IIGS computers, the value returned in `machine` is \$84 (high bit set to signify Apple IIGS and \$04 because it matches the ID bytes of an enhanced Apple IIe).

- A `ROMlevel` byte, indicating the revision of the firmware in the machine. For example, there are currently five revisions of the IIc, two of the IIe (unenhanced and enhanced), and three versions of the IIGS ROM (there will always be some owners who have not yet upgraded from ROM 00 to ROM 01). These versions are identified starting at \$01 for the earliest. Therefore, the current IIc will return `ROMlevel` = \$05, the current IIGS will return `ROMlevel` = \$03, etc. The routine will also return correct values for future versions of the IIGS, as a convention has been established for future ROM versions of that machine.
- A `memory` byte, containing the amount of memory in the machine. This byte only has four values—0 (undefined), 48, 64, and 128. Extra memory in an Apple IIGS, or extra memory in an Apple IIe or IIc Memory Expansion card, is not included. Programs must take special considerations to use that memory (if available), beyond those considerations required to use the normal 128K of today's IIe and IIc.
- If running on an Apple IIGS, three word-length fields are also returned. These are the contents of the registers as returned by the ID routine in the IIGS ROM, and they indicate several things about the machine. See Apple II Miscellaneous Technical Note #7 for more details.

In addition to these features, most of the addressing done in the routine is by label. If you wish

things to be stored in different places, simply changing the labels will often do it.

## Limitations and Improvements

As sample code, you might have already guessed that this is not the most compact, efficient way of identifying these machines. Some improvements you might incorporate if using these routines include:

- If you are running under ProDOS, you can remove the section that determines how much memory is in the machine (starting at `exit`, line 127), since the `MACHID` byte (at `$BF98`) in ProDOS already contains this information for you. This change would cut the routine down to less than one page of memory.
- If you know the ROM is switched in when you call the routine, you can remove the sections which save and restore the language card state. Be careful in doing so, however, because the memory-determination routines switch out the ROM to see if a language card exists.
- If you need to know if a IIe is a 64K machine with a non-extended 80-column card, you may put your own identifying routines in after line 284. `NOAux` is only reached if there is an 80-column card but only 64K of memory.

## How It Works

The identification routine does the following things:

- Disables interrupts
- Saves four bytes from the language card areas so they may be restored later
- Identifies all machines by a table look-up procedure
- Calls 16-bit ID routine to distinguish IIGS from other machines of any kind, and returns values in appropriate locations if IIGS ID routine returns any useful information in the registers
- Identifies memory configuration:
  - If Apple /// emulation, there is 48K
  - If Apple ][ or ][+, tests for presence of language card and returns 64K if present, otherwise, returns 48K
  - If Apple IIc or IIGS, returns 128K
  - If Apple IIe, tries to identify auxiliary memory
    - If reading auxiliary memory, it must be there
    - If reading alternate zero page, auxiliary memory is present
    - If none of this is conclusive:
      - Exchanges a section of the zero page with a section of code that switches

memory banks. The code executes in the zero page and does not get switched out when we attempt to switch in the auxiliary RAM.

- Jumps to relocated code on page zero:
  - Switches in auxiliary memory for reading and writing
  - Stores a value at \$800 and sees if the same value appears at \$C00. If so, no auxiliary memory is present (the non-extended 80-column card has sparse memory mapping which causes \$800 and \$C00 to be the same location).
  - Changes value at \$C00 and sees if the value at \$800 changes as well. If so, no auxiliary memory. If not, then there is 128K available
  - Switches main memory back in for reading and writing
- Puts the zero page back like we found it
- Returns memory configuration found (either 64K or 128K)
- Restores language card and ROM state from four saved bytes
- Restores interrupt status
- Returns to caller

```

        keep ID2.2

        list on

        org $2000

        longa off
        longi off

*****
*
*   Apple II Family Identification Program   *
*
*           Version 2.2                     *
*
*           March, 1990                     *
*
*   Includes support for the Apple IIe Card *
*   for the Macintosh LC.                  *
*
*****

; First, some global equates for the routine:

PROGRAM    start

IIplain    equ $01           ;Apple II
IIplus     equ $02           ;Apple II+
IIIem      equ $03           ;Apple /// in emulation mode
IIe        equ $04           ;Apple IIe
IIc        equ $05           ;Apple IIc
IIeCard    equ $06           ;Apple IIe Card for the Macintosh LC

safe       equ $0001         ;start of code relocated to zp
location   equ $06           ;zero page location to use

test1      equ $AA           ;test byte #1
test2      equ $55           ;lsr of test1
test3      equ $88           ;test byte #3
test4      equ $EE           ;test byte #4

```

```

begpage1 equ $400      ;beginning of text page 1
begpage2 equ $800      ;beginning of text page 2
begsprse equ $C00      ;byte after text page 2

clr80col equ $C000     ;disable 80-column store
set80col equ $C001     ;enable 80-column store
rdmainram equ $C002    ;read main ram
rdcardram equ $C003    ;read aux ram
wrmainram equ $C004    ;write main ram
wrcardram equ $C005    ;write aux ram
rdramrd equ $C013     ;are we reading aux ram?
rdaltzp equ $C016     ;are we reading aux zero page?
rd80col equ $C018     ;are we using 80-columns?
rdtext equ $C01A      ;read if text is displayed
rdpage2 equ $C01C     ;read if page 2 is displayed
txtclr equ $C050      ;switch in graphics
txtset equ $C051      ;switch in text
txtpage1 equ $C054    ;switch in page 1
txtpage2 equ $C055    ;switch in page 2
ramin equ $C080       ;read LC bank 2, write protected
romin equ $C081       ;read ROM, 2 reads write enable LC
lcbank1 equ $C08B     ;LC bank 1 enable

lc1 equ $E000         ;bytes to save for LC
lc2 equ $D000         ;save/restore routine
lc3 equ $D400
lc4 equ $D800

idroutine equ $FE1F   ;IIgs id routine

; Start by saving the state of the language card banks and
; by switching in main ROM.

strt      php          ;save the processor state
          sei          ;before disabling interrupts
          lda lc1      ;save four bytes from
          sta save     ;ROM/RAM area for later
          lda lc2      ;restoring of RAM/ROM
          sta save+1   ;to original condition
          lda lc3
          sta save+2
          lda lc4
          sta save+3
          lda $C081    ;read ROM
          lda $C081
          lda #0       ;start by assuming unknown machine
          sta machine
          sta romlevel

IdStart   lda location ;save zero page locations
          sta save+4   ;for later restoration
          lda location+1
          sta save+5
          lda #$FB     ;all ID bytes are in page $FB
          sta location+1 ;save in zero page as high byte
          ldx #0       ;init pointer to start of ID table
loop      lda IDTable,x ;get the machine we are testing for
          sta machine  ;and save it
          lda IDTable+1,x ;get the ROM level we are testing for
          sta romlevel ;and save it
          ora machine  ;are both zero?
          beq matched  ;yes - at end of list - leave

loop2     inx          ;bump index to loc/byte pair to test
          inx
          lda IDTable,x ;get the byte that should be in ROM
          beq matched  ;if zero, we're at end of list

```

```

        sta location      ;save in zero page

        ldy #0           ;init index for indirect addressing
        lda IDTable+1,x  ;get the byte that should be in ROM
        cmp (Location),y ;is it there?
        beq loop2        ;yes, so keep on looping

loop3    inx              ;we didn't match. Scoot to the end of the
        inx              ;line in the ID table so we can start
        lda IDTable,x    ;checking for another machine
        bne loop3
        inx              ;point to start of next line
        bne loop        ;should always be taken

matched  anop

; Here we check the 16-bit ID routine at idroutine ($FE1F).  If it
; returns with carry clear, we call it again in 16-bit
; mode to provide more information on the machine.

idIIgs   sec            ;set the carry bit
        jsr idroutine   ;Apple IIgs ID Routine
        bcc idIIgs2    ;it's a IIgs or equivalent
        jmp IIgsOut    ;nope, go check memory
idIIgs2  lda machine    ;get the value for machine
        ora #$80       ;and set the high bit
        sta machine    ;put it back
        clc            ;get ready to switch into native mode
        xce
        php            ;save the processor status
        rep #$30       ;sets 16-bit registers
        longa on
        longi on
        jsr idroutine  ;call the ID routine again
        sta IIgsA      ;16-bit store!
        stx IIgsX      ;16-bit store!
        sty IIgsY      ;16-bit store!
        plp            ;restores 8-bit registers
        xce            ;switches back to whatever it was before
        longa off
        longi off

        ldy IIgsY      ;get the ROM vers number (starts at 0)
        cpy #$02       ;is it ROM 01 or 00?
        bcs idIIgs3    ;if not, don't increment
        iny            ;bump it up for romlevel
idIIgs3  sty romlevel  ;and put it there
        cpy #$01       ;is it the first ROM?
        bne IIgsOut    ;no, go on with things
        lda IIgsY+1    ;check the other byte too
        bne IIgsOut    ;nope, it's a IIgs successor
        lda #$7F       ;fix faulty ROM 00 on the IIgs
        sta IIgsA

IIgsOut  anop

*****
* This part of the code checks for the *
* memory configuration of the machine. *
* If it's a IIgs, we've already stored *
* the total memory from above.  If it's *
* a IIC or a IIE Card, we know it's *
* 128K; if it's a ][+, we know it's at *
* least 48K and maybe 64K.  We won't *
* check for less than 48K, since that's *
* a really rare circumstance. *
*****

```

```

exit      lda machine          ;get the machine kind
         bmi exit128         ;it's a 16-bit machine (has 128K)
         cmp #IIc           ;is it a IIc?
         beq exit128         ;yup, it's got 128K
         cmp #IIeCard        ;is it a IIe Card?
         beq exit128         ;yes, it's got 128K
         cmp #IIe           ;is it a IIe?
         bne contextit      ;yes, go muck with aux memory
         jmp muckaux

contextit cmp #IIIem         ;is it a /// in emulation?
         bne exitII         ;nope, it's a ][ or ][+
         lda #48            ;/// emulation has 48K
         jmp exita

exit128   lda #128           ;128K
exita     sta memory

exit1     lda lc1           ;time to restore the LC
         cmp save          ;if all 4 bytes are the same
         bne exit2         ;then LC was never on so
         lda lc2           ;do nothing
         cmp save+1
         bne exit2
         lda lc3
         cmp save+2
         bne exit2
         lda lc4
         cmp save+3
         beq exit6

exit2     lda %C088         ;no match! so turn first LC
         lda lc1           ;bank on and check
         cmp save
         beq exit3
         lda %C080
         jmp exit6

exit3     lda lc2
         cmp save+1        ;if all locations check
         beq exit4         ;then do more more else
         lda %C080         ;turn on bank 2
         jmp exit6

exit4     lda lc3           ;check second byte in bank 1
         cmp save+2
         beq exit5
         lda %C080         ;select bank 2
         jmp exit6

exit5     lda lc4           ;check third byte in bank 1
         cmp save+3
         beq exit6
         lda %C080         ;select bank 2

exit6     plp              ;restore interrupt status
         lda save+4        ;put zero page back
         sta location
         lda save+5        ;like we found it
         sta location+1
         rts              ;and go home.

exitII    lda lcbank1      ;force in language card
         lda lcbank1      ;bank 1
         ldx lc2          ;save the byte there
         lda #test1       ;use this as a test byte
         sta lc2
         eor lc2          ;if the same, should return zero
         bne noLC
         lsr lc2          ;check twice just to be sure
         lda #test2       ;this is the shifted value
         eor lc2          ;here's the second check
         bne noLC
         stx lc2          ;put it back!
         lda #64          ;there's 64K here

```

```

noLC      jmp exita
          lda #48          ;no restore - no LC!
          jmp exita      ;and get out of here

muckaux   ldx rdtext      ;remember graphics in X
          lda rdpage2     ;remember current video display
          asl A           ;in the carry bit
          lda #test3     ;another test character
          bit rd80col    ;remember video mode in N
          sta set80col   ;enable 80-column store
          php            ;save N and C flags
          sta txtpage2   ;set page two
          sta txtset     ;set text
          ldy begpage1   ;save first character
          sta begpage1   ;and replace it with test character
          lda begpage1   ;get it back
          sty begpage1   ;and put back what was there
          plp
          bcs muck2     ;stay in page 2
          sta txtpage1   ;restore page 1
muck1     bmi muck2     ;stay in 80-columns
          sta %c000     ;turn off 80-columns
muck2     tay           ;save returned character
          txa           ;get graphics/text setting
          bmi muck3
          sta txtclr    ;turn graphics back on
muck3     cpy #test3    ;finally compare it
          bne nocard    ;no 80-column card!
          lda rdramrd   ;is aux memory being read?
          bmi muck128   ;yup, there's 128K!
          lda rdaltzp   ;is aux zero page used?
          bmi muck128   ;yup!
          ldy #done-start
move      ldx start-1,y  ;swap section of zero page
          lda |safe-1,y  ;code needing safe location during
          stx safe-1,y   ;reading of aux mem
          sta start-1,Y
          dey
          bne move
          jmp |safe     ;jump to safe ground
back      php           ;save status
          ldy #done-start ;move zero page back
move2     lda start-1,y
          sta |safe-1,y
          dey
          bne move2
          pla
          bcs noaux
isaux     jmp muck128   ;there is 128K

* You can put your own routine at "noaux" if you wish to
* distinguish between 64K without an 80-column card and
* 64K with an 80-column card.

noaux     anop
nocard    lda #64       ;only 64K
          jmp exita
muck128   jmp exit128   ;there's 128K

* This is the routine run in the safe area not affected
* by bank-switching the main and aux RAM.

start     lda #test4    ;yet another test byte
          sta wrCARDram ;write to aux while on main zero page
          sta rdCARDram ;read aux ram as well
          sta begpage2  ;check for sparse memory mapping
          lda begsprse  ;if sparse, these will be the same

```

```

        cmp #test4      ;value since they're 1K apart
        bne auxmem      ;yup, there's 128K!
        asl begsprse    ;may have been lucky so we'll
        lda begpage2    ;change the value and see what happens
        cmp begsprse
        bne auxmem
        sec              ;oops, no auxiliary memory
        bcs goback
auxmem   clc
goback   sta wrmainram  ;write main RAM
        sta rdmainram  ;read main RAM
        jmp back       ;continue with program in main mem
done     nop            ;end of relocated program marker

```

\* The storage locations for the returned machine ID:

```

machine  ds  1          ;the type of Apple II
romlevel ds  1          ;which revision of the machine
memory   ds  1          ;how much memory (up to 128K)
IIgsA    ds  2          ;16-bit field
IIgsX    ds  2          ;16-bit field
IIgsY    ds  2          ;16-bit field
save     ds  6          ;six bytes for saved data

```

```

IDTable  dc  I1'1,1'    ;Apple ][
        dc  H'B3 38 00'

        dc  I1'2,1'    ;Apple ][+
        dc  H'B3 EA 1E AD 00'

        dc  I1'3,1'    ;Apple /// (emulation)
        dc  H'B3 EA 1E 8A 00'

        dc  I1'4,1'    ;Apple IIe (original)
        dc  H'B3 06 C0 EA 00'

```

```

; Note: You must check for the Apple IIe Card BEFORE you
; check for the enhanced Apple IIe since the first
; two identification bytes are the same.

```

```

        dc  I1'6,1'    ;Apple IIe Card for the Macintosh LC (1st release)
        dc  H'B3 06 C0 E0 DD 02 BE 00 00'

        dc  I1'4,2'    ;Apple IIe (enhanced)
        dc  H'B3 06 C0 E0 00'

        dc  I1'5,1'    ;Apple IIc (original)
        dc  H'B3 06 C0 00 BF FF 00'

        dc  I1'5,2'    ;Apple IIc (3.5 ROM)
        dc  H'B3 06 C0 00 BF 00 00'

        dc  I1'5,3'    ;Apple IIc (Mem. Exp)
        dc  H'B3 06 C0 00 BF 03 00'

        dc  I1'5,4'    ;Apple IIc (Rev. Mem. Exp.)
        dc  H'B3 06 C0 00 BF 04 00'

        dc  I1'5,5'    ;Apple IIc Plus
        dc  H'B3 06 C0 00 BF 05 00'

        dc  I1'0,0'    ;end of table

```

end

**Further Reference**

---

- Apple II Miscellaneous Technical Note #7, Apple II Family Identification