Adventures in Flatland

Author: Benoît Gilon

Introduction

The points of this article are (in order):

- At first to just reply to some simple query but with some variants about the approach chosen. The simple query is: “is the host Apple 2 monitor (ROM based) supporting the Mini assembler feature?”. The point here is not the result that, given the model and ROM release version of the Apple, we could easily derive the correct answer to that query. Here, we will use some unusual tricks for doing so and follow a progression[1].
- Then, given the I/O architecture of the host operating system (either DOS 3.3 or ProDOS BASIC.SYSTEM interpreter), will extend some monitor features just by applying a simple patch at the OS level. The example provided here is to enable 65C02 disassembly for both the “*unenhanced ROM*” and “*enhanced ROM*” Apple //e. Currently this has been possible only by installing an already patched monitor ROM image in language card. The point of the article would be to offer an alternate way aiming the same goal and usable from within DOS 3.3 and ProDOS 8 (which kernel collides with patched monitor in Language card memory).
- Finally and as a conclusion of this article, the third part will assume the previous section understood by the reader and describes how the mini assembler could be provided or at least enhanced if already there in some related ways (that is bringing in a full 65C02 instruction set “mini” assembler). This can form the starting point for a future article named “BBC inline” and providing an Applesoft based environment to easily integrate ML and Applesoft BASIC in an unusual way.

All code presented within this article are available at the URL below so that you won't even have to type in from the explanatory listing within the current document.

The URL is: http://bgilon.free.fr/apple2/AdventuresInFlatland.zip

---

1 *not unlike what happens to the square shape in the Flatland book when encountering effects originating from the existence of a 3D world (i.e. a third dimension in addition to the basic 2 dimensions of the planar world he lives in since his birth) hence the article title..*

The archive will contain:

- a PDF export of the article text;
- a disk image (DOS 3.3 sector order) containing a working environment for Merlin 2.48 (DOS 3.3 version) along with every Merlin source and object files from this article: they are:
    1. *FLATLANDP1S1.S* (source) and *FLATLANDP1S1* (binary executable) from the 1st approach section;
    2. *FLATLANDP1S2.S* (source) and *FLATLANDP1S2* (binary executable) from the 2nd approach section;
    3. *FLATLANDP1S3.S* (source) and *FLATLANDP1S3* (binary executable) from the 3rd approach section;
    4. *FLATLANDP1S4.S* (source) and *FLATLANDP1S4* (binary executable) from the 4th approach section: the binary file can be converted and copied to a ProDOS volume for testing under a ProDOS BI environment;
    5. *FLATLANDP3.S* (source), *FLATLANDP3xyD33* (binary executable DOS 3.3 version) and *FLATLANDP3xyP8* (binary executable ProDOS BI version). xy being equal to PL (Apple 2 and 2+), EO (Apple //e original ROM), EE (Apple //e enhanced ROM) or CO (Apple //c original ROM). The latter P8 executable could be transferred to a ProDOS volume for testing under ProDOS BI.
    6. An example of STARTUP.FL Applesoft file for binary running the adequate variant of the binary executable file according to both the resident host ROM model and the host operating system.
- The two PDF assembly listing files for the *FLATLANDP3EExyz.S* (one DOS 3.3 (xyz set to D33, one ProDOS BI (xy set to P8)).

Is the Mini assembler feature supported by my Apple 2 ROM?

First approach: checking against the ROM signature

Just to get rid of the simplest and dumbest approach here.. This is the first routine that will check against the ROM version.

```
ROMREAD   EQU  $C081     To enable ROM reading..
COUT1     EQU  $FDF0     Output subroutine

          ORG  $0300
START     BIT  ROMREAD
          LDA  $FBB3
          EOR  $FBC0
          EOR  $FBBF
          LDY  #8-1
]LOOP     CMP  MACMAT,Y
          BEQ  :1
          DEY
          BPL  ]LOOP
          INY       ;Apple 2+ default
:1        LDX  #MES0-MESBASE
          LDA  MCODE,Y
          BPL  *+4
          LDX  #MES1-MESBASE
]LOOP     LDA  MESBASE,X
          BEQ  :2
          JSR  COUT1
          INX
          BNE ]LOOP      Always branch
:2        RTS

MACMAT    HEX  EA2DE6E7F9060502
MCODE     HEX  00        Apple 2+
* Enhanced //e and 2GS have same signature but both
* support the Mini assembler feature..
          HEX  008080    Apple //e and 2GS
          HEX  00808080  Apple //c
MESBASE
MES0      HEX 8D
          ASC  "MINI ASSEMBLER NOT SUPPORTED",8D00
MES1      HEX  8D
          ASC  "MINI ASSEMBLER SUPPORTED",8D00
          ERR  *-MESBASE/256
```

Second approach: checking against the monitor mini assembler character table

This approach is based upon the existence of a well known table which is the encoding of every monitor command/pattern.

This table starts within ROM @ address which value (lo/hi format) can be found @ address `$FF7E/7F`. The number of byte wide entries be stored at address `$FF79`.

This table (labeled `CHRTBL` within the "*firmware listing*" part of technical reference manuals) contains encoded values of the original command/pattern monitor character sequence. For the character "`!`" (high ascii meaning b7 set or $A1 in hexadecimal) used to enter into the mini assembler from the monitor prompt ("`*`"), the encoded value is the low byte of the value `($A1 eor $B0) + $89`, that is `$9A`. So what is needed for our subroutine, is just to find a byte of value `$9A` within the (`$FF79`) bytes starting from address (`$FF7E/7F`).

The listing below shows such subroutine

```
ROMREAD   EQU  $C081     To enable ROM reading..
A1L       EQU  $3C
COUT1     EQU  $FDF0     Output subroutine
ENCVALUE  =    "!"!$B0+$89 Encoded value for the "!" char in
CHRTBL

          ORG  $0300
START     BIT  ROMREAD
          LDA  $FF7E
          STA  A1L
          LDA  $FF7F
          STA  A1L+1
          LDY  #0
          LDX  $FF79
]LOOP     LDA  (A1L),Y
          CMP  #ENCVALUE
          BEQ  :1
          INY
          DEX
          BNE  ]LOOP     X = 0 upon loop exit
          HEX  2C        Skip next two bytes
:1        LDX  #MES1-MESBASE
]LOOP     LDA  MESBASE,X
          BEQ  :2
          JSR  COUT1
```

```
          INX
          BNE ]LOOP      Always branch
:2        RTS
MESBASE
MES0      ASC  “MINI ASSEMBLER NOT SUPPORTED”,8D00
MES1      ASC  “MINI ASSEMBLER SUPPORTED”,8D00
          ERR  *-MESBASE/256
```

Third approach: using a User Session Simulation (TLA is USS)

From what you've learnt from the Apple firmware study (usually after a read or two of the famous Apple Technical Reference Manuals), is that the system I/O redirection feature is based on the existence of two special vectors within page zero.

One vector is used to support every output made by program targeting an output device (usually a monitor screen but this can be a printer if a suitable address be put into this vector): The name of such vector is labelled CSW (and its page zero locations are $36 and $37), which contains the address of the routine handling the output of the character which code is in A register. During initialization, the routine COUT1 entry point address ($FDF0) is put into those page zero locations.

Another vector is used to support every input flow that an application requires to carry out its task (usually getting input from the local keyboard, but this can be a serial input connected to a modem or a mouse data sequence from a mouse interface card if a suitable address be put into this vector). The name of such vector is labeled KSW (and its page zero locations are $38 and $39). During initialization, the routine KEYIN entry point ($FD1E) is put into those page zero locations and the routine should return with the A register as its result (value to be handled by the application).

From within the ROM code, most of the code therein, in order to output a character, instead of calling the COUT1 routine directly will call the COUT entry point ($FDED) which itself is nothing more than

```
COUT      JMP  (CSW)
```

The same is true for the ROM code willing to get a character from the user and calling the RDKEY entry point ($FD0C) instead of directly calling the KEYIN subroutine with the RDKEY being coded as:

```
RDKEY     LDY  CH         Do some trick with cursor..
          LDA  (BASL),Y
          PHA             ;The code here can vary according
          AND  #$3F       to the ROM release..
          ORA  #$40       But in all cases, this is
          STA  (BASL),Y   concluded by a JMP (KSW) instruction
          PLA             ; like below.
          JMP  (KSW)
```

The ROM content of every model has evolved but the legacy from the

original Apple 2 computer is still there in the latest ROM in your ROM 03 2GS. So a quick study of the monitor evaluation loop could bring some benefit to the reader knowledge of his computer.
One of the addresses that has never changed with consecutive ROM revisions within the Apple // line is address $FF69 (the one run by a classic CALL-151 from within the Applesoft prompt). Bold instructions are the ones which will interest us in further paragraphs.

```
     IN        EQU  $0200     Monitor input buffer
     COUT      EQU  $FDED
     TOSUB     EQU  $FFBE
     ZMODE     EQU  $FFC7
     PROMPT    EQU  $33
     YSAV      EQU  $34
FF65 MON       CLD
               JSR  BELL
FF69 MONZ      LDA  #"*"
               STA  PROMPT
               JSR  GETLNZ
               JSR  ZMODE
FF73 NXTITM    JSR  GETNUM    Get next non hexa. item
               STY  YSAV
               LDY  #$17
FF7A ]LOOP     DEY
               BMI  MON
               CMP  CHRTBL,Y
               BNE  ]LOOP
               JSR  TOSUB     Call corresponding subroutine
               LDY  YSAV
               JMP  NXTITM

FF3A BELL      LDA  #$87      Code for Dring
               JMP  COUT
     CANCEL    LDA  #$DC      "\" after cancelled line
               JSR  COUT
FD67 GETLNZ    JSR  CROUT
FD6A GETLN     LDA  PROMPT
               JSR  COUT
               LDX  #1
     BCKSPC    TXA
               BEQ  GETLNZ
               DEX
     NXTCHAR   JSR  RDCHAR
               CMP  #$95 Right arrow?
```

```
              BNE  CAPTST
              LDA  (BASL),Y
     CAPTST   CMP  #$E0
              BCC  ADDINP
              AND  #$FF
FD84 ADDINP   STA  IN,X
              CMP  #$8D
              BNE  NOTCR
              JSR  CLREOL    Clear to end of line
FD8E CROUT    LDA  #$8D
              BNE  COUT
FD3D NOTCR    LDA  INVFLG
              PHA
              LDA  #$FF
              NOP
              NOP
              LDA  IN,X
              JSR  COUT
              PLA
              STA  INVFLG
              LDA  IN,X
              CMP  #$88      Left arrow?
              BEQ  BCKSPC
              CMP  #$98      Control-X?
              BEQ  CANCEL
```

So, if you update the CSW vector within page zero and testing on the new user routine the value output from the program. It is easy to check whether or not the mini assembler is ROM resident. Just:

- put the two characters $A1 (“!”) and $8D (carriage return) to IN and IN+1 address, set X to 1;
- jump to the address at $FF70
- Here is the sample new user output routine:

```
MYCOUT    CMP  #$87 This is from the JSR BELL at address
$FF66
          BEQ  :NOK
* The $A1 comes from the JSR COUT at address $FD6C while
* the prompt (pz0 $33) is set to “!”
          CMP  #$A1
          BEQ  :OK
          RTS       ;Discard any other output characters
```

The following code extract is an illustration of the complete mechanism.

```
ROMREAD   EQU  $C081     To enable ROM reading..
PROMPT    EQU  $33
CSWL      EQU  $36
CSWH      EQU  $37
IN        EQU  $0200
COUT1     EQU  $FDF0     Output subroutine

          ORG  $0300
START     BIT  ROMREAD
          LDA  CSWH
          PHA
          LDA  CSWL
          PHA
          LDA  PROMPT
          PHA
          LDA  #MYCOUT
          STA  CSWL
          LDA  #>MYCOUT
          STA  CSWH
          LDA  #"!"
          STA  IN
          LDA  #$8D
          STA  IN+1
          TSX            ;Save the stack pointer
          STX  STKINIT   to be restored from within MYCOUT
          LDX  #1
          JMP  $FF70
MYCOUT    CMP  #"!"
          BEQ  :OK
          CMP  #$87
          BEQ  :NOK
          RTS            ;Discard any othrer character
:NOK      LDY  #0
          HEX  2C   Skip next two bytes
:OK       LDY  #MES1-MESBASE
          LDX  STKINIT   Restore stack pointer
          TXS
          PLA            And updated pz0 cells..
          STA  PROMPT     from stack..
          PLA
          STA  CSWL
          PLA
          STA  CSWH
]LOOP     LDA  MESBASE,Y
```

```
          BEQ  :2
          JSR  COUT1
          INY
          BNE  ]LOOP
:2        RTS
MESBASE
MES0      ASC  "MINI ASSEMBLER NOT SUPPORTED",8D00
MES1      ASC  "MINI ASSEMBLER SUPPORTED",8D00
          ERR  *-MESBASE/256
STKINIT   DS   1
```

If you like, you might want to also simulate user data entry into the monitor by replacing the `KSW` vector.. see the `FLATLANDP1S3.S` Merlin source file for details about how to do that...

Fourth approach: dwelling with host operating systems I/O redirection

Hereby we will take the third approach described in the previous section as the basis, but instead of doing a simple CSW/KSW replacement, we consider the fact that the OS has already put its own vectors into CSW and KSW, saving the user vectors somewhere within the DOS/ProDOS data memory segment.

So, instead of saving the CSW and KSW contents onto stack directly, we will save (i.e. push onto stack) the content of those memory slots containing the true vector owners from a user point of view. Also, at the conclusion of putting our own value in the pz0 locations, a call to some OS routine will both save them in dedicated memory area with OS memory and replace their p0 locations with OS own intercept vectors. This is this mechanism that allow “in fine” the Apple 2 to be “responsive” as a user has entered a DOS/ProDOS command either at the Applesoft prompt (“]”) or within a program text (by prefixing the output string with a CHR$(4) character).

The table below will provide details about where the true vector addresses are stored and the subroutine entry point to call to reinstate DOS interception vector in page zero.

What	DOS 3.3	ProDOS 8 BI (all versions)
Where CSW true user routine vector address is stored	$AA53/54	$BE30/31
Where KSW true user routine vector address is stored	$AA55/56	$BE32/33
Subroutine entry point to call for reinstating DOS intercepts	$A851 (DOS 3.3 48K slave January 1983) or $03EA (vector in page 3: all versions)	SAVIOTRU ($9A8D)

Here is the resulting source code (handling both DOS 3.3 and ProDOS 8 BI environments).

```
* Hardware equates
TXTPAG1   EQU  $C054
TXTPAG2   EQU  $C055
RDINTCXR  EQU  $C015     b7 set if internal ROM active
SLTCXROM  EQU  $C006     C1-C7 pages to slot ROMs
```

```
INTCXROM  EQU  $C007     C1-C7 pages to internal ROMs
* Monitor equates
CSWL      EQU  $36
CSWH      EQU  $37
KSWL      EQU  $38
KSWH      EQU  $39
WNDLFT    EQU  $20
WNDWDTH   EQU  $21
CH        EQU  $24
BASL      EQU  $28
PROMPT    EQU  $33
* My own equates
AUXPTR    EQU  $06
* DOS 3.3 equates
DTRUCSW   EQU  $AA53     Where the true CSW owner address is
stored
DTRUKSW   EQU  $AA55     Same for KSW
INITPTRS  EQU  $03EA     Address to call to reestablish OS control
* ProDOS BI equates (all versions)
PTRUCSW   EQU  $BE30     Where the true CSW owner address is
stored
PTRUKSW   EQU  $BE32     Same for KSW
SAVIOTRU  EQU  $9A8D     Address to call to reestablish BI control
PSAVAREA  EQU  $BE3E     Placeholder for storing registers
          DUMMY PSAVAREA
PREGA     DS   1
PREGX     DS   1
PREGY     DS   1
          DEND
* 80 col. Firmware equates
OURCH     EQU  $057B
* We cannot place the new ML routine in page 3 anymore
* (not enough space), so instead, the program loads at
* address $2000, which leaves pages $08 to $1F for potential
* Applesoft program text which would not be disturbed by
* running this small test.
          ORG  $2000
START     LDX  #0
          CLC
* For ProDOS BI to be detected, value $4C must be
* on the five bytes starting at $BE00 up to $BE0E
* with a step of +3..
]LOOP     LDA  $BE00,X
          EOR  #$4C
```

```
          BNE  :1
          INX
          INX
          INX
          CPX  #$0F
          BCC  ]LOOP
:1        LDX  #1
          ROR
          STA  ISPRODOS  Only b7 is meaningful here
          BMI  *+3       ;Branch iif ProDOS 8 BI
          DEX
          STX  IDOS
* In the Enhanced Apple //e, the path to mini assembler
* points to an address in page $C1 and the INTCXROM status
* is not restored until return to monitor..
* Thus we have to save the original status of the softswitch
* and restore it back to its original setting while quitting
* before the mini assembler returns itself to monitor.
          LDA  RDINTCXR
          STA  MINTCXR
          LDA  TOFB,X    base address is $AA53
          STA  AUXPTR     for DOS 3.3
          LDA  TOFT,X     and $BE30
          STA  AUXPTR+1   for ProDOS BI (all versions)
* Store the true CSW/KSW owner addresses onto stack.
          LDY  #4-1      2 bytes for CSW, 2 bytes for KSW
]LOOP     LDA  (AUXPTR),Y
          PHA
          DEY
          BPL  ]LOOP
          LDA  PROMPT
          PHA
* All the PHA operations below are to properly handle the
* CLREOL operation performed from within the GETLN processing
* as a $8D (carriage return) key is pressed at the keyboard.
          LDA  CH
          PHA
          LDA  OURCH
          PHA
          LDA  WNDLFT
          PHA
          LDA  WNDWDTH
          PHA
          LDX  $C01D     RDHIRES
```

```
          BIT  $C056     HIRESOFF
          STA  $C001     80STORON
          LDY  #0
          BIT  TXTPAGE1
          LDA  (BASL),Y
          PHA
          BIT  TXTPAGE2
          LDA  (BASL),Y
          PHA
          BIT  TXTPAGE1
          TXA
          BPL  *+5
          BIT  $C057     HIRESON
* Backup our stack pointer to be restored @ the MYCOUT
* subroutine level
          TSX
          STX  STKINIT
* Do our own stuff as per the previous code sample..
          LDA  #MYCOUT
          STA  CSWL
          LDA  #>MYCOUT
          STA  CSWH
          LDA  #"*"
          STA  PROMPT
          LDA  #MYRDKEY
          STA  KSWL
          LDA  #>MYRDKEY
          STA KSWH
          LDY  #0
          STY  IDX
* The few lines below just to properly handle the CLREOL
* We simulate a CH value of zero, a WNDLFT value of zero,
* a WNDWDTH value of 1 so that to minimize the screen area
* written to by the CLREOL processing within the video
* firmware (either 40 col. Or 80 col.) called by GETLN.
          STY  CH
          STY  WNDLFT
          INY
          STY WNDWDTH
* Save OS entry context into private area to be restored later
          JSR  STORAUX
]LOOP     LDA  (AUXPTR),Y
          STA  MSAVAR,Y
          DEY
```

```
           BPL  ]LOOP
           CLC            ;Full reconnect here..
           JSR  COMREDIR  Reconnect installed DOS..
           JMP  $FF6D     Run monitor! But get back soon…
MYRDKEY    LDX  IDX
           LDA  INPUT,X
           INC  IDX
           RTS
MYCOUT     CMP  #"!"
           BNE  *+6
           CMP  PROMPT
           BEQ  :OK
           CMP  #$87
           BEQ  :NOK
           RTS
:NOK       LDY  #MES0-MESBASE
           HEX  2C   Skip next two bytes
:OK        LDY  #MES1-MESBASE
           LDX  STKINIT
           TXS
           LDY  $C01D     RDHIRES
           BIT  $C056     HIRESOFF
           STA  $C001     80STORON
* All the PLA operations below are to properly handle the
* CLREOL operation performed from within the GETLN processing
* as a $8D ("carriage return/enter") key is pressed at the
* keyboard.
           LDX  #0
           PLA
           BIT  TXTPAGE2
           STA  (BASL,X)
           PLA
           BIT  TXTPAGE1
           STA  (BASL,X)
           TYA
           BPL  *+5
           BIT  $C057     HIRESON
           PLA
           STA  WNDWDTH
           PLA
           STA  WNDLFT
           PLA
           STA  OURCH
           PLA
```

```
          STA  CH
          PLA
          STA  PROMPT
* Pop the four bytes forming the CSW and KSW values
* and stores them in proper cells within page zero
          LDX  #0
]LOOP     PLA
          STA  CSWL,X
          INX
          CPX  #4
          BCC  ]LOOP
          TYA
          PHA
          LDY  #MESMA-MESBASE
          JSR  PRMSG     Print 8D,"MINI ASSEMBLER "
          PLA
          BNE  :0
          TAY
          JSR  PRMSG     Print "NOT "
:0        LDY  #MES1-MESBASE
          JSR  PRMSG     Print "SUPPORTED",8D
          JSR  STOAUX
]LOOP     LDA  MSAVAR,Y
          STA  (AUXPTR),Y
          DEY
          BPL  ]LOOP
* Restore the SLT/INTCXROM status
          STA  SLTCXROM
          BIT  MINTCXR
          BPL  *+5
          STA INTCXROM
          SEC

* Whichever is the installed OS, let it control the I/O
* redirection..
COMREDIR  BIT  ISPRODOS
          BMI  *+5
          JMP  INITPTRS  Reestablish DOS 3.3 control over I/O
          JMP  SAVIOTRU  Reestablish BI control over I/O
          BCC  :0
          LDX  #4-1
]LOOP     LDA  PTRUCSW,X
          STA  CSWL,X
          DEX
          BPL  ]LOOP
```

```
          LDA  #$88      Backspace/left arrow
          LDX  #0        Already at 1st postion within INBUF
          LDY  PREGY
          STA  PREGA
          STX  PREGX
:0        RTS
* Subroutine to print a message thru a non vectored ROM routine
PRMSG     LDA  MESBASE,Y
          BEQ  :2
          JSR  COUT1
          INY
          BCC  PRMSG
          RTS
* Subroutine to store OS save area gein address to AUXPTR
STOAUX    LDX  IDOS
          LDA  TSVB,X
          STA  AUXPTR
          LDA  TSVT,X
          STA  AUXPTR+1
          LDY  TLENM1,X
          RTS
* Subroutine substitute for ROM KEYIN: could'nt be simpler
MYRDKEY   LDX  IDX
          LDA  INPUT,X
          INC  IDX
          RTS
MESBASE
MES0      ASC  "NOT ",00
MES1      ASC  "SUPPORTED",8D00
MESMA     HEX  8D
          ASC  "MINI ASSEMBLER ",00
          ERR  *-MESBASE/256
TOFB      DFB  DTRUCSW, PTRUCSW
TOFT      DFB  >DTRUCSW, >PTRUCSW
* 3 tables below for keeping track of the CPU registers set
* upon entry to DOS/BI
TLENM1    DFB  4-1,3-1   length of actual table minus 1.
TSVB      DFB  DSAVAREA,PSAVAREA
TSVT      DFB  >DSAVAREA,>PSAVAREA
INPUT     HEX  A18D      "!" character followed by EOL.
MSAVAR    DS   4         Private area for register set on DOS ent.
STKINIT   DS   1         To keep track of working stack pointer
IDX       DS   1
ISPRODOS  DS   1
```

```
IDOS      DS   1
MINTCXR   DS   1
```

65C02 instructions in monitor disassembly listings

Introduction

Now that we have answered a basic query such as “Does the ROM resident monitor support the mini assembler feature?”, it is time to trick the system by extending ROM based features with extra features that you surely would have like to be there from the start.

Having upgraded your non enhanced //e with an “upgrade kit”, the first thing you want to try out is whether the disassembler is now supporting your new CPU instruction set. Hélas (in French in the text), the output below will surely frighten you a bit:

```
]CALL -151
*300:80 FE N 300L

0300-   80       ???
0301-   FE FF FF INC   $FFFF,X
0304-   FF       ???
```

$80 is the OpCode for the BRA <*relativeAddress*>

The transparency of such patch will be an important aspect of such patch. After having binary run the patch, and trying again:

```
]BRUN FLATLANDP2D33
]CALL -151
*300:80 FE N 300L

0300-   80 FE    BRA   $0300
0302-   FF       ???
```

One might object that there is a current way of obtaining this feature and is used by storing in the Language card (from address $F800 to $FFFF) an already patched copy of the monitor. The SOURCEROR disassembler (by Glen Bredon) is provided with a MON65C02 file that copied itself to LC for doing so and thus allowing SOURCEROR to output the whole 65C02 instruction set at its text stream output (disassembly).

FLATLANDP2 aim is to be used both for DOS 3.3 and for ProDOS BI, therefore, instead of putting its stuff in LC, will occupy a small memory segment within main 48K memory[2].

2 “*Elegance, elegance above all*” Clive Sinclair

ROM monitor study (all Apple 2 models)

When KSW is reached while in monitor entry phase, the stack will look like the one below:

Index within stack	Content	Comment
S + 6	$FF	HiByte($FF6F) that is the location where GETLNZ is called from main monitor loop.
S + 5	$6F	LoByte($FF6F)
S + 4	$FD	HiByte($FD77) that is the location where the RDCHAR/ESCRDKEY is called from GETLN routine
S + 3	$77	LoByte($FD77)
S + 2	vH	Intermediary routine in charge of calling RDKEY: HiByte
S + 1	vLo	Intermediary routine in charge of calling RDKEY: LoByte
S		Next location to be used by subsequent PUSH

The values for vHi/vLo have different values according to the Apple 2 model.

Apple 2 ROM model	vHi/vLo
//e unenhanced	$FD37
][and][plus	$FD37
Enhanced //e	$FD37
Original //c	$CCF4

What will be performed at Keyboard intercept step from with OS is to substitute the two locations (S + 5, S + 4) to #>MyOwnRoutine, #<MyOwnRoutine whenever the stack on entry follows the layout given above.

DOS 3.3 patch

Here is the original keyboard intercept routine from DOS 3.3. Only the line marked as such will be updated per the patch.

```
KBDINTRC  JSR  SAVREG
          LDA  CURSTAT
          BEQ  :2   Branch iif not coldstarting or reading a file
```

```
          PHA
          LDA  ASAVE
          STA  (BASL),Y
          PLA
          BMI  :1         If coldstarting..
          JMP  READBYTE   read a byte from file
:1        JSR  FIRSTIME   $9D0A
          LDY  CH
          LDA  #$60
          STA  (BASL),Y
:2        LDA  EXCFLG          Will be replaced with a JSR PX
          BEQ  :3
          JSR  EXERD
:3        LDA  #3
          STA  CSWSTATE
          …
* Somewhere within the dedicated memory segment
PX        LDA  PROMPT
          CMP  #"*"
          BNE  :1
          LDX  SSAVE
          LDY  #5
]LOOP     INX
          LDA  $0100,X
          EOR  STKL,Y
          BNE  :1
          DEY
          BPL  ]LOOP
          LDA  #>MyOwnRoutine
          STA  $0100,X
          DEX
          LDA  #MyOwnRoutine
          STA  $0100,X
:1        LDA  EXFLG
          RTS
STKL      HEX  FF6FFD77FD37   Those values for //e, ][ and ][+
```

I'll not describe here the necessary patch details required for initing an unpatched DOS image whenever the user issues an INIT command. Please consult the source file for such information to the attention of interested readers.

ProDOS BI patch

A similar approach to the one related to DOS 3.3 could be adopted for ProDOS BI.

```
KBDINCPT  EQU  *
KSTATE0   BIT  EXACTV
          BPL  :1
          JSR  REGSAV    Will be replaced with JSR PX
          JMP  EXECREAD
:1        JSR  SETIOTRU
          JSR  RDKEY+4
          CMP  #cr
          BNE  KBDEXIT
          JSR  REGSAV    Will be replaced with JSR PX
          STA  INBUF,X
          …
PX        JSR  REGSAV
          LDA  PROMPT
          CMP  #"*"
          BNE  :1
          TSX
          INX       ;Pass above the caller return address
          INX
          LDY  #5
]LOOP     INX
          LDA  $0100,X
          EOR  STKL,Y
          BNE  :1
          DEY
          BPL  ]LOOP
          LDA  #>MyOwnRoutine
          STA  $0100,X
          DEX
          LDA  #MyOwnRoutine
          STA  $0100,X
:1        JMP  REGRST
STKL      HEX  FF6FFD77FD37   Those values for //e, ][ and ][+
```

Flatland part 2: a user manual

Flatland DOS 3.3 and ProDOS BI share the same source file FLATLANDP2.S which can be assembled in Merlin 8 (v2.48).

Two possible object files result from such assembly:

- FLATLANDP2D33 which can directly be run from within DOS 3.3;
- FLATLANDP2P8, which is the ProDOS BI version, has to be copied to a ProDOS volume by using either a COPY 2+ aoftware or the Apple CONVERTER system program from the ProDOS disk or using the DOP companion utility (exchanging files between DOS

and ProDOS from the DOS side).

The version assembled is set according to the OPTBI label's value at the beginning of the source file.

Both versions use a safe guard against lockups caused by extra installation of the same patch within either DOS or ProDOS BI.

```
]BRUN FLATLANDP2D33
]BRUN FLATLANDP2D33
It seems that this patch or another
similar is already installed

]
```

Also if you try to install this patch on an Apple 2 model which ROM monitor already supports 65C02 instruction set disassembly, an adequate message is displayed and installation is canceled.

```
]BRUN FLATLANDP2D33
Your Apple already supports disassembly
of 65C02 instructions within its ROM.
There is no need to install this patch!

]
```

Thus the patch can only be installed on an Apple //e (original or enhanced ROM) or an Apple 2 or 2+. All other Apple model support this feature.

The ProDOS BI version installs itself by calling the GETBUFR system wide memory allocation routine from ProDOS BI (2 pages are claimed). If such request could not be satisfied, then a message is returned to the user.

```
]BRUN FLATLANDP2P8
Not enough memory to install
this patch!

]
```

Once the patch is installed, you can load object modules which you know include some 65C02 instructions just to test their proper disassembly format..

```
]BLOAD TESTDIS
]CALL-151

*300L

0300-   DA          PHX
0301-   FA          PLX
0302-   5A          PHY
0303-   7A          PLY
```

```
0304-   1A          INC
0305-   3A          DEC
0306-   80 FE       BRA   $0306
0308-   89 FF       BIT   #$FF
030A-   64 00       STZ   $00
030C-   9C FF FF    STZ   $FFFF
030F-   74 00       STZ   $00,X
0311-   04 00       TSB   $00
0313-   0C FF FF    TSB   $FFFF
0316-   14 00       TRB   $00
0318-   1C FF FF    TRB   $FFFF
031B-   7C 00 10    JMP   ($1000,X)
031E-   72 06       ADC   ($06)
0320-   32 06       AND   ($06)
0322-   D2 06       CMP   ($06)
0324-   52 06       EOR   ($06)

*L

0326-   B2 06       LDA   ($06)
0328-   12 06       ORA   ($06)
032A-   F2 06       SBC   ($06)
032C-   92 06       STA   ($06)
032E-   00          BRK
...
```

Here are the monitor's major missing features matrix per Apple ROM model. Just to remind ourselves that there are many cases which still indicate a *No* string pattern and that there might be features not listed which might be listed according to your priorities (e.g. the memory multi bank display, copy and verify in case you have a A.E. RAMWorks compatible board).

Apple // ROM version	Disassembly	Mini assembler	Step	Trace
Apple 2 Integer	6502 **now 65C02**	Yes but 6502 only	Yes	Yes
Apple 2+	6502 **now 65C02**	No	No	No
Apple //e original ROM	6502 **now 65C02**	No	No	No
Apple //e enhanced ROM	6502 **now 65C02**	6502 only	No (replaced by a SEARCH feature)	No
Apple //e debug ROM	65C02	65C02	Yes	Yes
Apple //c original ROM	65C02	No	No	No
Apple //c Unidisk 3.5 ROM	65C02	65C02	Yes	Yes
Apple //c memory expanded ROM	65C02	65C02	Yes	Yes
Apple //c revised memory expanded ROM	65C02	65C02	Yes	Yes
Apple 2GS ROM 0	65816	65816	Yes	Yes
Apple 2GS ROM 1	65816	65816	Yes	Yes
Apple 2GS ROM 3	65816	65816	Yes	Yes

Flatland part 3: rebirth of the mini (the assembler not the british car ;-))

Introduction

Just to show the reader how easy it is to start from the point left in our previous part (whole 65C02 instruction set disassembled despite unsupported by ROM resident monitor), in order to be able to mini assemble the whole 65C02 IS. Here is the broad description of how to use this feature.

The mini assembler is available by issuing a “!” command character at the monitor prompt. From this on, the user can manually keyboard enter the 65C02 mnemonics (and not only 6502) and the assembly will take over in order to replace the user entry with a disassembled view of this instruction. Details can be covered at the Apple //e and //c TRM as well as in the 2GS firmware manual.

Readers can also get their hand on the WOZPAK (second edition) by published by *callapple.org*. If we get a look at the feature matrix first shown in the article previous part. Below is the new vision of the features supported per ROM model. In addition to the features that part 2 brought in (marked as *P2* in column labelled “Disassembly), the new features are in **bold** style.

Apple // ROM version	Disassembly	Mini assembler	Step	Trace
Apple 2 Integer	65C02 (P2)	Was 6502 only, **now supports 65C02**	Yes	Yes
Apple 2+	65C02 (P2)	**65C02**	No	No
Apple //e original ROM	65C02 (P2)	**65C02**	No	No
Apple //e enhanced ROM	65C02 (P2)	Was 6502 only, **now supports**	No (replaced by a	No

		65C02	SEARCH feature)	
Apple //e debug ROM	65C02	65C02	Yes	Yes
Apple //c original ROM	65C02	**65C02**	No	No
Apple //c Unidisk 3.5 ROM	65C02	65C02	Yes	Yes
Apple //c memory expanded ROM	65C02	65C02	Yes	Yes
Apple //c revised memory expanded ROM	65C02	65C02	Yes	Yes
Apple 2GS ROM 0	65816	65816	Yes	Yes
Apple 2GS ROM 1	65816	65816	Yes	Yes
Apple 2GS ROM 3	65816	65816	Yes	Yes

Flatland part 3 mini user manual

For this part 3 to be very easily tailored from a user point of view, this small Applesoft program (File *STARTUP.FL* in the disk image) could be used to automatically install the patch version per ROM model and per host operating system (either DOS 3.3 or ProDOS 8 BI).

```
 0  GOTO 10
 1  DEF  FN V(VP) = (VP < 58) * (VP - 48) + (VP > 64) * (VP - 55): RESTORE: RETURN
 2  FOR IC = 1 TO  LEN (P$) STEP 2:VX =  FN V( ASC ( MID$ (P$,IC,1))) * 16 +  FN V( ASC ( MID$
(P$,IC + 1,1)))
 3  POKE PT,VX:PT = PT + 1: NEXT IC: RETURN
 4 B = 0: FOR PT = 48640 TO 48652 STEP 3:B =  PEEK (PT) <  > 76: ON  NOT B GOTO 5:PT = 48652
 5  NEXT PT:S$ = "D33": ON B GOTO 6:S$ = "P8"
 6  RETURN
 10  HOME : GOSUB 1:PT = 769: FOR I = 0 TO 1 STEP 0: READ P$: ON  LEN (P$) > 0 GOTO 20:I = 1
 20  IF  NOT I THEN  GOSUB 2
 30  NEXT I
 31  DATA A007ADB3FB4DC0FB4DBFFBD94003F0048810F8C8
 32  DATA C002D023AD5CFCC9EBD01CA00818FB08C230201FFE
 33  DATA 8C800328FBA00CAD8103D006AD80030908A88C000360
 34  DATA EA2DE6E7F9060502
 39  DATA  ""
 40  CALL 769:RY =  PEEK (768): IF RY > 11 THEN RY = 11
 41  FOR I =  - 1 TO RY - 1: READ P$: NEXT : READ P$: IF  LEFT$ (P$,1) = "/" THEN P$ =
CHR$ (8) + P$
 42  PRINT "MODEL DETECTED: APPLE 2"P$: FOR I = 0 TO 1: READ P$:I =  LEN (P$) = 0: NEXT
 43  DATA "","," OR 2+","//e original ROM","//e enhanced ROM","//e debug ROM"
 44  DATA "//c original ROM","//c Unidisk 3.5 ROM","//c Memory expansion ROM","//c Revised memory
expansion ROM”
 45  DATA "GS ROM0","GS ROM1","GS ROM3","Future GS",""
 50  GOSUB 4: FOR I =  - 1 TO RY - 1: READ P$: NEXT : READ NF$
 51  ON  LEN (NF$) > 0 GOTO 52: PRINT "There is no need to run the FLATLAND": PRINT "patch on your
Apple computer": END
```

```
 52 NF$ = "FLATLANDP3" + NF$ + S$
 55  DATA  "","PL","EO","EE","","CO","","","","","",""
 60 CV =  PEEK (37): PRINT  CHR$ (4)"BRUN "NF$: ON  PEEK (37) > CV GOTO 70:NF$ = "Flatland patch
install complete": ON RY > 0 GOTO 61:NF$ = "FLATLAND PATCH INSTALL COMPLETE"
 61  PRINT NF$
 70  END
```

From the study of the *FLATLANDP3.S* Merlin source file, you'll find out that The version built (see the SAV sequence at the end of the assembly) depend on two label values, the *OPTBI* could be set to either 0 (DOS 3.3) or 1 (ProDOS BI) and the *ROMMODEL* could be set to one of those string (*R2P*, *R2E*, *R2EE* or *R2CO*).

Flatland part3 conclusion

Here we've reached a point where all Apple computers have a mini assembler (either ROM resident, or being somewhere within the main 48K RAM) in order to support the BBC inline macro assembler feature which will be covered in a future article. I do not have any clone on hand, and so we'll be happy to hear from brave hobbysts who adapted this mechanism to exotic architectures.