

## On Downloading COREs

When Reset is pressed, CORE runs an initialization routine starting at location \$F000 in the ROM. This routine checks the value of the two bytes starting at location \$8000 in the RAM. If those two bytes are equal to \$C0 DE, then CORE runs a program beginning at location \$8002. This is what happens in a normal unit.

If the two bytes are equal to anything besides \$C0 DE, control passes to the Download routine in ROM. In order to change the value of \$8000-1, there are 2 methods

1. remove the programming door and move the leftmost switch to the left position. This un-write-protects the RAM. Then hold down the "d" key and press Reset. This writes a zero to \$8000 and jumps to Download. (This procedure will not necessarily work if CORE doesn't have good code in it, in which case use the next method.)
2. remove the programming door and move both switches to the left. This un-write-protects the RAM and deactivates the lithium battery. Then remove a battery, wait a few minutes and replace the battery. CORE will then have completely random code in it and the odds are 65535 to 1 that you will be able to download.

The procedure for downloading using the Apple IIe or IIc is described below. This procedure can be modified for any computer that can send serial data at 19200 baud.

1. There is a BASIC program called DOWNLOAD.ALL which initializes the serial port, loads a binary file called BIGLOAD into memory, asks the user to Reset the CORE and then press a key on the Apple to begin downloading.
2. The binary file BIGLOAD loads at \$3E00-7FFF on the Apple and contains the following:
  - \$3E00-3EFF: a machine language program which outputs data to the CORE, starting with location \$3F00 and ending with \$7FFF. The source code for this program is enclosed here. (Note: there are a couple of versions of this code floating around: this may not be the one on the disk you have.)
  - \$3F00-3FFF: the CORE "bootstrap" program. This program is sent to the CORE, stored in locations \$7F00-7FFF on CORE, then executed. The parameters of this program determine how many more blocks of data are to be received, and what locations they are to be stored in. In the normal case, an additional 64 blocks (of 256 bytes each) are sent and stored at locations \$8000-\$BFFF on CORE.
3. There is a special protocol for sending data to the CORE. The actual bytes from the file (e.g. BIGLOAD) are not sent directly but rather as follows.
  - The first byte of the file is sent directly to the CORE
  - Each subsequent byte is "exclusive or'd" with the previous byte, and the resulting value is sent to the CORE. For example, if the first three bytes of the file are \$55 \$FF \$AA, what will be sent is \$55 \$AA \$55. When the CORE receives the data, each byte is EOR'd with the previous byte (the first byte being EOR'd with zero), so the data would be transformed back to \$55 \$FF \$AA.

```

1 *
2 * Sample Apple IIC download program
3 *
4     org     $3e00
5     lda     #0
6 wakeup   jsr     msdelay
7         dec     a
8         bne     wakeup
9         ldx     #$3f
10 nextpage stx     1
11         lda     #0
12         sta     0
13         tay
14 nextbyte eor     (0),y
15         jsr     sendbyte
16         iny
17         bne     nextbyte
18         inx
19         cpx     #$80
20         bcc     nextpage
21         rts
22 *
23 sendbyte jsr     $fded    output
24 msdelay  pha
25         lda     #0
26 nextwait adc     #1
27         bne     nextwait
28         pla
29         rts

```

LOKE Download code (in ROM)

```
165 *****
166 stopwait jsr  \msdelay
167         clb   inflag      clear false in19200 flag
168 *
169         ldx   #<buf        BUF = $7F00
170         jsr   pageload
171         lda   buf+$ff
172         bne   stopwait     checksum test (8 bit parity)
173         jmp   buf
174 *****
175 pageload stx   ptr+1
176         lda   #0
177         sta   ptr
178         tay
179 byteload jsr   \inwait
180         eor   indata
181         sta   (ptr),y
182         lda   indata
183         iny
184         bne   byteload
185         rts
```

```

1 *
2 * Bootstrap program at $7F00 in CORE
3 pageload equ $f05c
4 *
5     org     $7f00
6     ldx    ##80
7 nextpage jsr    pageload
8     inx
9     cpx    ##c0
10    bcc    nextpage
11    jmp    $b800
12    ds     $7fff-*, $ff
13    dfb    $00

```

*start address (\$8000)*  
*end address (\$c000-1)*  
*run address*  
 FOR CHECKSUM!!!