::development::IWM information

Controlling the 3.5 Drive Hardware on the Apple IIGS

Neil Parker

nparker@cie.uoregon.edu
parker@corona.uoregon.edu
Version 1.00
February 1994

========
Abstract
========
This document describes how to control the Apple 3.5 drive hardware withou
going through ProDOS. Accessing the disk drive and the IWM interface chip
are described.

==========
WARNING WARNING WARNING WARNING WARNING WARNING WARNING WARNING WARNING WA
==========
This document is based on information found in several publications, as
listed in the Bibliography at the end of this note, my own disassemblies
of the relevant Apple IIGS ROM routines, and on some experimentation. I
can make no guarantees as to the accuracy of this information: it should
be considered as a starting point for your own explorations rather than
as an authoritative source.

Remember that when you use this information you are dealing directly with
Naked Hardware, and the myriad protective features of the firmware and
operating system are not available. Should you be so foolish as to try ou
this information with a non-expendable disk in the drive, I will not be he
responsible for any lost data.

=============
General Notes
=============
All the sample routines in this article assume the the processor is in
emulation mode or 8-bit native mode.

All I/O locations mentioned are in bank $E0 or $E1, and also in bank 0
or 1 if I/O shadowing is enabled.

============
Introduction
============
Controlling the Apple 3.5 Drive hardware directly requires a knowledge of
two separate pieces of hardware: the disk drive itself, and the Integratec
Woz Machine (IWM) interface chip.

========
IWM Chip
========
The IWM chip in the Apple IIGS is configured to reside in internal slot 6.
Its I/O locations are the same as the original Disk][interface in slot 6

```
        CA0     EQU $C0E0       ;stepper phase 0 / control line 0
        CA1     EQU $C0E2       ;stepper phase 1 / control line 1
        CA2     EQU $C0E4       ;stepper phase 2 / control line 2
        LSTRB   EQU $C0E6       ;stepper phase 3 / control strobe
```

```
ENABLE  EQU $C0E8       ;disk drive off/on
SELECT  EQU $C0EA       ;select drive 1/2
Q6      EQU $C0EC
Q7      EQU $C0EE
```

Each of these I/O locations represents a two-way switch; accessing locatic X turns off the switch; accessing location X+1 turns it on.

For a 5.25-inch drive, the switches CA0...LSTRB control the stepper motor which positions the read/write head over the desired track. For a 3.5-inc drive, these switches have become general-purpose control lines. Using these control lines will be described later.

The ENABLE switch turns the drive off and on. This switch turns on the re "in use" light, holds the disk in the drive, and prepares the drive to receive further commands. Unlike the 5.25-inch drive, it does not start the spindle motor spinning. The command to start the spindle motor will be described later.

The SELECT switch still fully retains its original function: if it is off, drive 1 will be accessed; turning it on selects drive 2.

The switches Q6 and Q7 together form a single four-way switch. The functi of this switch is somewhat complex, and will be covered in detail later.

The following additional memory locations are also important when dealing with the 3.5-inch drive:

```
SLTROMSEL EQU $C02D     ;Clear bit 6 to enable internal slot 6 hardware
DISKREG EQU $C031       ;Additional disk drive control register
CYAREG  EQU $C036       ;System speed and motor-on detect bits
```

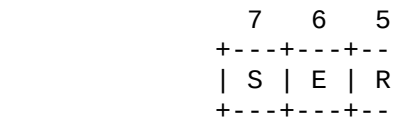
Bit 6 of SLTROMSEL controls whether the internal hardware and firmware for slot 6 is available, or whether an external card in slot 6 is available. Before any access to the disk drive is possible, the internal hardware for slot 6 must be selected by turning off bit 6. Before modifying this register, the original contents should be saved somewhere so that your routine can restore the original system state when it is through with the drive.

The 3.5-inch drive does its I/O twice as fast as the 5.25-inch drive, so it is desirable to set the system speed to "fast" when reading or writing data to avoid getting out of step with the drive. This step is nc absolutely necessary, but it helps a LOT--when the system speed is slow, each data byte from the disk drive must be dealt with in 16 cycles or less which puts uncomfortably tight timing constraints on the code.

Setting the speed is not as simple as turning on bit 7 of CYAREG, due to the way the speed setting interacts with disk accesses. If a 5.25-inch drive is connected to slot 6, either through a disk interface card in slot 6 or through the built-in disk drive daisy-chain, the slot 6 motor-on detect feature will be enabled. This causes the system speed to revert to slow whenever that disk is accessed, regardless of the current system speed setting. The system speed must be slowed down to ensure that the 8-bit operating systems, whose timing loops are all written under the assumption that the system speed is 1 MHz, can access the disk drive properly. This automatic slowdown can be disabled by setting bit 2 of CYAREG to 0 before accessing the 3.5-inch drive. As usual, this register should be saved before it is modified, and restored when your coc is through using the drive.

DISKREG contains two bits of interest to the 3.5-inch drive: bit 7 which

is a general purpose control line and bit 6 which enables 3.5-inch drive support. The other bits are reserved and should not be modified. The layout of DISKREG is:



S General purpose control line used in conjunction with CA0...LSTRB switches
E Enables 3.5-inch drive support:
0 = 5.25-inch drive and smartport devices availabl
1 = 3.5-inch drive available
R Reserved

Note:
The Hardware Reference and the Firmware both incorrectly state that bit 7 selects between the upper and lower heads of the drive.

One of the first things a 3.5-inch drive routine should do is turn on bit 6 of DISKREG to ensure that the proper device is accessed. Similarly, the last thing a 3.5-inch drive routine should do is turn this bit back off to prevent other programs from becoming hopelessly confused about which disk drive is available.

Note that SLTROMSEL, CYAREG, and DISKREG each contain bits that your program should not modify. Always use a read-modify-write sequence to change only the bits of interest.

Accessing IWM Registers

The IWM chip has several internal registers available to programs. Access to these registers is controlled by the Q6 and Q7 switches.

Q6	Q7	Register
off	off	Read data register
off	on	Read handshake register
on	off	Read status register
on	on	Write mode register (if drive is off) data register (if drive is on)

The mode register is a write-only register containing several flag bits which control various features if the IWM. To access it, turn off the drive (by accessing ENABLE), turn on Q6 and Q7, and write to any odd-numbered address in the $C0E0...$C0EF range.

Note that the drive may remain active for a second or two after the ENABLE access, and that the write to the mode register will fail unless the drive is fully deactivated. This means that the mode register must be repeatedl written until the status register (see below) indicates that the desired changes have taken effect. The IIGS ROM uses a routine like the following to accomplish this (enter with the desired mode in the Y-register):

```
SELIWM  LDA ENABLE      ;turn drive off
        LDA Q6+1        ;prepare to access mode & status regs
```

```
        BRA SELIWM1
SELIWM2 TYA
        STA Q7+1        ;try writing to mode reg
SELIWM1 TYA
        EOR Q7          ;check status reg
        AND #$1F        ;(only bits 0-4 matter)
        BNE SELIWM2     ;if different, try writing again
RTS
```

The bits of the mode register are laid out as follows:

```
  7   6   5   4   3   2   1   0
+---+---+---+---+---+---+---+---+
| R | R | R | S | C | M | H | L |
+---+---+---+---+---+---+---+---+
```

With the various bit meanings described below:

Bit	Function
R	Reserved
S	Clock speed: 0 = 7 MHz 1 = 8 MHz Should always be 0.
C	Bit cell time: 0 = 4 usec/bit (for 5.25 drives) 1 = 2 usec/bit (for 3.5 drives)
M	Motor-off timer: 0 = leave drive on for 1 sec after program turns it off 1 = no delay Should be 0 for 5.25 and 1 for 3.5.
H	Handshake protocol: 0 = synchronous (software must supply proper timing for writing data) 1 = asynchronous (IWM supplies timing) Should be 0 for 5.25 and 1 for 3.5.
L	Latch mode: 0 = read-data stays valid for about 7 usec 1 = read-data stays valid for full byte time Should be 0 for 5.25 and 1 for 3.5.

Before doing I/O to the 3.5-inch drive, the mode register should be set to $0F. When your routine is done, it should be sure to set the mode register back to $00.

The status register is a read-only register which contains information about the current status of the drive and the IWM. To access it, turn Q7 off and Q6 on, and read from any even-numbered address in the $C0E0...$C0EF range.

The bits of the status register are laid out as follows:

```
  7   6   5   4   3   2   1   0
+---+---+---+---+---+---+---+---+
| I | R | E | S | C | M | H | L |
+---+---+---+---+---+---+---+---+
```

Bit	Function
I	Sense input. write-protect indicator (5.25-inch drive) general status line (3.5-inch drive)
R	Reserved.

```
 E      Drive enabled
                0 = no disk drive is on
                1 = a disk drive is on.
 S      Same as S bit in the mode register.
 C      Same as C bit in the mode register.
 M      Same as M bit in the mode register.
 H      Same as H bit in the mode register.
 L      Same as L bit in the mode register.
```

The handshake register is a read-only register used when writing to the disk in asynchronous mode (when bit 1 of the mode register is on). It indicates whether the IWM is ready to receive the next data byte. To read the handshake register, turn switches Q6 off and Q7 on, and read from any even-numbered address in the $C0E0...$C0EF range.

The bits of the mode register are laid out as follows:

```
  7   6   5   4   3   2   1   0
+---+---+---+---+---+---+---+---+
| B | U | R | R | R | R | R | R |
+---+---+---+---+---+---+---+---+

Bit     Function
---     --------
 B      Register Ready
                0 = IWM is busy
                1 = IWM is ready for data
 U      Under-run
                0 = write under-run has occurred (the program took
                    too long to write the next byte)
                1 = no under-run
 R      Reserved.
```

The data register is the register that you read to get the actual data from the disk and write to store data on the disk. To read it, turn Q6 and Q7 off and read from any even-numbered address in the $C0E0...$C0EF range. To write it, turn Q6 and Q7 on and write to any odd-numbered address in the $C0E0...$C0EF range. When reading, the high bit of the data register becomes 1 when the data is valid. The reason the high bit indicates valid data is due to the structure of data on the disk; all valid disk bytes have the high bit set.

Once the disk is properly configured, reading data is quite simple; the following code illustrates the technique:

```
                LDA Q7          ;insure read mode
        R1      LDA Q6          ;ready yet?
                BPL R1          ;if not, try again
                STA DATA1       ;got a valid byte, so save it
        R2      LDA Q6          ;repeat ad nauseam...
                BPL R2
                STA DATA2
        R3      LDA Q6
                BPL R3
                STA DATA3
                etc...
```

Writing data is somewhat more difficult, but mercifully it is not necessar for the user's program to count out precise 32-cycle intervals as it was with the 5.25-inch drive. Instead, the asynchronous mode of the IWM takes care of the counting for you. The following code illustrates the techniqu

```
                BIT Q6+1        ;prepare for writing
                LDA DATA1       ;get first data
                STA Q7+1        ;set write mode and write data at same tim
```

```
                LDA DATA2       ;get second data
        W1      BIT Q6          ;ready yet?
                BPL W1          ;if not, try again
                STA Q6+1        ;write second data
                LDA DATA3       ;do it again...
        W2      BIT Q6
                BPL W2
                STA Q6+1
                LDA DATA4       ;and again...
        W3      BIT Q6
                BPL W3
                STA Q6+1
                etc...
        WLAST   BIT Q6          ;wait until last data underruns
                BVS WLAST
                BIT Q7          ;be VERY SURE to turn off write mode!
                RTS
```

Note that in the write routine, the first byte is written differently than the rest: the STA Q7+1 activates write mode and writes the byte all in one step.

In actual practice, you would probably want to use a loop to read and store (or load and write) the data.

Accessing Disk Drive Status and Control Bits

In addition to programming the IWM, it is also necessary to program the drive itself, which is somewhat "smarter" than the 5.25-inch drive (even though it is a "dumb" device).

The 3.5-inch drive contains several internal status bits which the user's program can examine, and several internal control switches which the user's program can use to control various functions of the drive. These status and control bits are accessed by the CA0...LSTRB switches mentioned above and by the SEL line (bit 7 of DISKREG). CA0...CA2 and SEL form a 16-way switch which selects the desired control or status function, and the LSTRB switch signals the drive to perform a control function. The IIGS ROM uses the following routine to select a status or control function (enter with desired function in A-reg):

```
        SEL35   BIT CA0         ;set switches to known state
                BIT CA1+1
                BIT LSTRB
                BIT CA2
                LSR
                BCC SEL35A
                BIT CA2+1       ;if bit 0 on, turn on CA2
        SEL35A  LSR
                PHA
                LDA DISKREG
                AND #$7F        ;if bit 1 off, turn off SEL
                BCC SEL35B
                ORA #$80        ;else turn on SEL
        SEL35B  STA DISKREG
                PLA
                LSR
                BCC SEL35C
                BIT CA0+1       ;if bit 2 on, turn on CA0
        SEL35C  LSR
                BCS SEL35D
                BIT CA1         ;if bit 3 off, turn off CA1
```

```
        SEL35D  RTS
```

To read a status bit, turn Q6 off, Q7 on, and ENABLE on, configure CA0...CA2 and SEL for the desired function, and read the status bit from bit 7 of the IWM status register. The IIGS ROM uses the following code to accomplish this:

```
        STAT35  JSR SEL35       ;select desired status bit
                BIT Q6+1
                BIT Q7          ;test status register
                RTS             ;(returns result in processor N-flag)
```

The status bits are as follows:

CA2	CA1	CA0	SEL	Param for STAT35	Function
off	off	off	off	$00	Step direction. 0 = head set to step inward (toward higher-numbered tracks 1 = head set to step outward (toward lower-numbered tracks)
off	off	off	on	$02	Disk in place. 0 = disk in drive 1 = drive is empty.
off	off	on	off	$04	Disk is stepping. 0 = head is stepping between track 1 = head is not stepping.
off	off	on	on	$06	Disk locked. 0 = disk is write protected 1 = disk is write-enabled.
off	on	off	off	$08	Motor on. 0 = spindle motor is spinning 1 = motor is off
off	on	off	on	$0A	Track 0. 0 = head is at track 0 1 = head is at some other track This bit becomes valid beginning 12 msec after the step that places the head at track 0.
off	on	on	off	$0C	*Disk switched? 0 = user ejected disk by pressing the eject button 1 = disk not ejected.
off	on	on	on	$0E	Tachometer. 60 pulses per disk revoluti
on	off	off	off	$01	Instantaneous data from lower head. Rea this bit configures the drive to do I/O the lower head.
on	off	off	on	$03	Instantaneous data from upper head. Rea this bit configures the drive to do I/O the upper head.
on	on	off	off	$09	Number of sides. 0 = single-sided drive 1 = double-sided drive
on	on	off	on	$0B	*Disk ready for reading? 0 = ready 1 = not ready I am not too sure about this bit. The firmware waits for this bit to go low before trying to read a sector address field.
on	on	on	on	$0F	Drive installed.

0 = drive is connected
1 = no drive is connected

Note:

Functions marked with an asterisk, i.e. "*", are used by the IIGS ROM but not documented in any publication available to me. I am fairly certain of the function of status bit $0C (used by the firmware to test for disk-switched errors), but I am unsure about status bit $0B (if my programs neglect to test for it, the drive displays an annoying tendency to start reading while the head is still stepping).

The settings of most of these bits are "backwards": 0 means "yes" and 1 means "no".

To perform a control function, turn off LSTRB, configure CA0, CA1, and SEL for the desired function, set CA2 to the desired value (all control functions can be turned on or off), and then turn LSTRB on and back off. The IIGS ROM uses the following code to accomplish this:

```
CONT35  JSR SEL35       ;select desired function
        BIT LSTRB+1     ;strobe on
        BIT LSTRB       ;strobe off
        RTS
```

The control functions are as follows:

CA1	CA0	SEL	CA2	Param for CONT35	Function
off	off	off	off	$00	Set step direction inward (toward higher numbered tracks.)
off	off	off	on	$01	Set step direction outward (toward lower numbered tracks.
off	off	on	on	$03	*Reset disk-switched flag? (The firmware uses this to clear disk-switched errors.
off	on	off	off	$04	Step one track in current direction (tak about 12 msec).
on	off	off	off	$08	Turn spindle motor on.
on	off	off	on	$09	Turn spindle motor off.
on	on	off	on	$0D	Eject the disk. This takes about 1/2 se complete. The drive may not recognize f control commands until this operation is complete.

* Again, the asterisk marks a function used by the ROM but not documented in any publication available to me.

Description of Disk I/O

The following pseudo-code is a greatly simplified description of the steps a simple program might take to perform I/O with a 3.5-inch drive.

```
//
// Initialize everything
//
Save SLTROMSEL and CYAREG
Switch in internal slot 6 and set fast speed
Turn off disk I/O switches (to insure a "safe" state)
Select the 3.5-inch drive (turn on bit 6 of DISKREG)
Set IWM mode register to $0F
Select drive 1 or 2 (access SELECT or SELECT+1)
```

```
Turn on drive (access ENABLE+1)
Turn on spindle motor (LDA #$08; JSR CONT35)

//
// if current track number is unknown
//      move to track 0
//
IF we do not know what track we are currently on
        Set step direction = out (LDA #$01; JSR CONT35)
        WHILE Not at track 0 (LDA #$0A; JSR STAT35; BPL ...)
                Step one track (LDA #$04; JSR CONT35)
                WHILE still stepping (LDA #$04; JSR STAT35; BPL ..
                        do nothing
        Set current track = 0

//
// determine how many steps to move to the desired track
//
IF current track < desired track
        Set step direction = in
        Set number of steps = desired track - current track
ELSE IF current track > desired track
        Set step direction = out
        Set number of steps = current track - desired track
ELSE
        Set number of steps = 0

//
// move to the desired track by repeatedly stepping
//
WHILE number of steps > 0
        Step one track
        WHILE still stepping (LDA #$04; JSR STAT35; BPL ...)
                do nothing
        number of steps = number of steps - 1

//
// Set up track and side; wait for disk drive to be ready
//
Set current track = desired track
Select desired side (LDA #$01 or LDA #$03; JSR STAT35)
WHILE not ready to read (LDA #$0B; JSR STAT35; BMI ...)
        do nothing

//
// Perform the desired disk access
//
Read or write your data (this is the FUN part!)

//
// Clean up
//
Turn off spindle motor (LDA #$09; JSR CONT35)
Turn off drive (LDA ENABLE)
Turn off CA0...LSTRB
Set IWM mode register to $00
Deselect 3.5 drive (turn off bit 6 of DISKREG)
Restore slot and speed configuration
Return to caller
```

You will probably notice that I glossed over the most important part: the "read or write your data" part. The basic method is to use routines

like those listed above under the description of the IWM data register. Unfortunately, the data must undergo considerable preparation before writing and after reading.

Those of you who are lucky enough to own a copy of _Beneath Apple DOS_ will understand the kind of work that is necessary. For those not so lucky, I must plead that a proper discussion would require another article every bit as long as this one. Rather than try to tackle that subject here, I will content myself with providing a sample program (with commented source code) which shows one way the above information can be put together to make a working program.

==========
Example of Disk I/O
==========

The program listed below was written to illustrate the steps necessary to control the hardware of the 3.5 Drive from your own programs, without the use of the operating system or the firmware. It is essentially a 3.5-inch version of the DUMP program by Don Worth which was printed in "Beneath Apple DOS." It will read a track from a 3.5-inch disk into your Apple's memory, in its raw, encoded form.

Included below are a commented source code listing and a hex dump suitable for typing directly into the System Monitor (or capturing into a text file and EXECing).

Instructions:

First, boot DOS 3.3 or ProDOS 8. DUMP3.5 should be compatible with either operating system. If you booted ProDOS, get into BASIC.SYSTEM. When you see the] prompt, type "BLOAD DUMP3.5", and then "CALL-151". Store the number of the track you wish to examine in memory location 6, and the disk side you wish to examine (0 for the lower side, anything else for the upper side) in location 7. Put the disk to be examined in Drive 1, and type "900G". The raw track data will then be found in memory locations $1000 through $7FFF (this buffer is much longer than an actual track, so the data will most likely be repeated several times in the buffer).

For example,

```
]BLOAD DUMP3.5      (Load the program)
]CALL-151           (Enter the Monitor)
*6:20               (Select track $20)
*7:1                (Select upper side)
*900G               (Run DUMP3.5 (do not forget to insert the disk first))
*1000.10FF          (Examine the first 256 bytes of the track)
```

The usual Dire Warnings apply: I make no guarantees whatsoever for this program. I have tested it, and it seems to work on my computer, but I recommend using it ONLY on expendable disks, and ONLY with the write-protect hole open. I assume no responsibility for any damage which may result from the use or misuse of this program.

Be especially careful if you enter either the assemby listing or the hex dump by hand since the slightest typographical error could turn a benign tool into a malevolent disk-eating monster.

I hope this program helps clarify the disk access process. If there is enough interest in an explanation of how to interpret what it accesses, it might be possible to talk me into writing up an explanation of the block encoding process.

I recommend first reading "Beneath Apple DOS", if you can find a copy,
and also the SmartPort chapter of the Firmware Reference.

Appendix A: Loading A Track Into Memory (Assembly Source)

```
;*****************************************************************
; DUMP3.5 -- Dump a track of a 3.5-inch disk to memory.  (IIGS only)
;
; By Neil Parker -- inspired by Don Worth's DUMP program from "Beneath
; Apple DOS"
;
; Inputs: $06 = Track to be dumped
;         $07 = Side to be dumped (0=lower side, non-0=upper side)
; Outputs: $1000-$7FFF = raw track data
;
; Example:
;     *6:20 1      (Select track $20, upper side)
;     *900G        (Run DUMP3.5)
;     *1000.10FF   (Examine part of the track)
;*****************************************************************
          ORG $900
TRACK     EQU 6           ;Track number
SIDE      EQU 7           ;Side number
PTR       EQU 8
BUFFER    EQU $1000       ;Start address for track data
SLTROMSEL EQU $C02D       ;Select internal/external ROMs for slots
DISKREG   EQU $C031       ;Select 3.5/5.25 drive, control SEL line
CYAREG    EQU $C036       ;System speed and motor-on-detect bits
CA0       EQU $C0E0       ;Phase 0, 3.5 drive control
CA1       EQU $C0E2       ;Phase 1, 3.5 drive control
CA2       EQU $C0E4       ;Phase 2, 3.5 drive control
LSTRB     EQU $C0E6       ;Phase 3, control strobe
ENABLE    EQU $C0E8       ;Turn drive off/on
SELECT    EQU $C0EA       ;Select drive 1/2
Q6        EQU $C0EC
Q7        EQU $C0EE
;
          LDA SLTROMSEL   ;Get slot 6 status,
          PHA             ;save it,
          AND #$BF        ;force internal ROM+I/O for Slot 6
          STA SLTROMSEL
          LDA CA0         ;Clear disk I/O latches
          LDA CA1
          LDA CA2
          LDA LSTRB
          LDA ENABLE      ;Insure that drive is off
          LDA SELECT      ;Select drive 1
          LDA Q6          ;Set IWM for reading (a "safe" state)
          LDA Q7
          LDA #$F         ;Configure IWM for 3.5 access
          JSR SELIWM
          LDA DISKREG     ;Save old DISKREG
          PHA
          ORA #$40        ;Select 3.5 drive
          STA DISKREG
          LDA ENABLE+1    ;Turn drive on
          LDA #2          ;Is there a disk in the drive?
          JSR SEL35
          JSR TEST35
          BPL THERE       ;If so, read
```

```
          JMP DONE        ;otherwise quit
THERE     LDA #8          ;Turn motor on
          JSR SEL35
          JSR TRIG35
          LDA #1          ;Set step direction=outward
          JSR SEL35
          JSR TRIG35
TSTTRK0   LDA #$A         ;Are we at track 0 yet?
          JSR SEL35
          JSR TEST35
          BPL ATTRK0      ;If so, go read
          LDA #4          ;otherwise do a step
          JSR SEL35
          JSR TRIG35
SEEKING0  JSR TEST35      ;Step still in progress?
          BPL SEEKING0    ;If so, loop until step done
          BMI TSTTRK0     ;otherwise go see if we are at track 0 yet
ATTRK0    LDX TRACK       ;What track did the user want?
          BEQ DUMP        ;If track 0, we are already there -- go read
          LDA #0          ;else set step direction=inward
          JSR SEL35
          JSR TRIG35
SEEK      LDA #4          ;Do a step
          JSR SEL35
          JSR TRIG35
SEEKING   JSR TEST35      ;Step still in progress?
          BPL SEEKING     ;If so, loop until step done
          DEX             ;otherwise see if we have stepped enough yet
          BNE SEEK        ;If not, go step again
DUMP      LDA #$B         ;Disk ready for reading yet?
          JSR SEL35
READYT    JSR TEST35
          BMI READYT      ;Loop until disk ready
          LDA SIDE        ;What side did the user want?
          BEQ SIDE1       ;If 0, set lower side
          LDA #3          ;else set upper side
          BNE SETSIDE
SIDE1     LDA #1
SETSIDE   JSR SEL35
          JSR TEST35
          PHP             ;Save interrupt status
          SEI             ;Do not let anything interrupt us
          LDA CYAREG      ;Save old system speed
          PHA
          AND #$FB        ;Set speed=fast
          ORA #$80
          STA CYAREG
          LDA #BUFFER    ;change #> to #< and #< to #>.)
          STA PTR+1
          LDY #0
DUMPLP    LDA Q6          ;Read a byte
          BPL DUMPLP      ;Loop until we have a valid byte
          STA (PTR),Y     ;Store byte in buffer
          INC PTR         ;Advance buffer pointer
          BNE DUMPLP
          INC PTR+1
          LDA PTR+1       ;Buffer full yet?
          CMP #$80
          BCC DUMPLP      ;If not, go read some more
          PLA             ;Done.  Restore system speed
          STA CYAREG
          PLP             ;Restore interrupt status
```

```
          LDA #9          ;Turn motor off
          JSR SEL35
          JSR TRIG35
DONE      LDA ENABLE      ;Turn drive off
          LDA CA0         ;Clear disk I/O latches
          LDA CA1
          LDA CA2
          LDA LSTRB
          PLA             ;Restore old DISKREG value
          STA DISKREG
          LDA #0          ;Configure IWM for 5.25 access
          JSR SELIWM
          PLA             ;Restore original slot configuration
          STA SLTROMSEL
          RTS             ;Amen.
;
;Subroutine to select 3.5 drive status/control registers
;Enter with accumulator=desired status:
;     Bit 0=CA2 status
;     Bit 1=SEL status
;     Bit 2=CA0 status
;     Bit 3=CA1 status
;
SEL35     BIT CA0
          BIT CA1+1
          BIT LSTRB
          BIT CA2
          LSR             ;If bit 0 set, turn on CA2
          BCC S35A
          BIT CA2+1
S35A      LSR             ;If bit 1 set, turn on SEL
          PHA
          LDA DISKREG
          AND #$7F
          BCC S35B
          ORA #$80
S35B      STA DISKREG
          PLA
          LSR             ;If bit 2 set, turn on CA0
          BCC S35C
          BIT CA0+1
S35C      LSR             ;If bit 3 set, turn on CA1
          BCS S35D
          BIT CA1
S35D      RTS
;
;Subroutine to read the status of the 3.5 drive
;First call SEL35 to select register to examine
;Result is in processor N (negative) flag
;
TEST35    BIT Q6+1
          BIT Q7
          RTS
;
;Subroutine to perform a 3.5 drive control function
;First call SEL35 to select function to be performed
;
TRIG35    BIT LSTRB+1
          BIT LSTRB
          RTS
;
;Subroutine to configure the IWM chip
```

```
;Before calling, make sure drive is OFF!
;Call with accumulator=desired Mode Register value
;     A=$00 for 5.25 drive
;     A=$0F for 3.5 drive
;
SELIWM    TAY
          BIT Q6+1        ;Prepare to access Mode & Status Regs.
          JMP SELIWM2     ;First see if it is already set like we want it
SELIWM1   TYA
          STA Q7+1        ;Try writing to Mode Reg.
SELIWM2   TYA
          EOR Q7          ;Compare input to Status Reg.
          AND #$1F
          BNE SELIWM1     ;If not the same, try writing again
          BIT Q6          ;else prepare IWM for data
          RTS
```

Appendix B: Loading A Track Into Memory (Hex Dump)

Here is the hext dump corresponding to the above assembler listing. This can be entered by hand into the Monitor, or you can capture it into a text file, put "CALL-151" at the beginning and "3D0G" and "BSAVE DUMP3.5,A$900,L$145" at the end, and EXEC it to create the program.

```
900:AD 2D C0 48 29 BF 8D 2D C0 AD E0 C0 AD E2 C0 AD
910:E4 C0 AD E6 C0 AD E8 C0 AD EA C0 AD EC C0 AD EE
920:C0 A9 0F 20 2E 0A AD 31 C0 48 09 40 8D 31 C0 AD
930:E9 C0 A9 02 20 F2 09 20 20 0A 10 03 4C D5 09 A9
940:08 20 F2 09 20 27 0A A9 01 20 F2 09 20 27 0A A9
950:0A 20 F2 09 20 20 0A 10 0F A9 04 20 F2 09 20 27
960:0A 20 20 0A 10 FB 30 E7 A6 06 F0 18 A9 00 20 F2
970:09 20 27 0A A9 04 20 F2 09 20 27 0A 20 20 0A 10
980:FB CA D0 F0 A9 0B 20 F2 09 20 20 0A 30 FB A5 07
990:F0 04 A9 03 D0 02 A9 01 20 F2 09 20 20 0A 08 78
9A0:AD 36 C0 48 29 FB 09 80 8D 36 C0 A9 00 85 08 A9
9B0:10 85 09 A0 00 AD EC C0 10 FB 91 08 E6 08 D0 F5
9C0:E6 09 A5 09 C9 80 90 ED 68 8D 36 C0 28 A9 09 20
9D0:F2 09 20 27 0A AD E8 C0 AD E0 C0 AD E2 C0 AD E4
9E0:C0 AD E6 C0 68 8D 31 C0 A9 00 20 2E 0A 68 8D 2D
9F0:C0 60 2C E0 C0 2C E3 C0 2C E6 C0 2C E4 C0 4A 90
A00:03 2C E5 C0 4A 48 AD 31 C0 29 7F 90 02 09 80 8D
A10:31 C0 68 4A 90 03 2C E1 C0 4A B0 03 2C E2 C0 60
A20:2C ED C0 2C EE C0 60 2C E7 C0 2C E6 C0 60 A8 2C
A30:ED C0 4C 39 0A 98 8D EF C0 98 4D EE C0 29 1F D0
A40:F4 2C EC C0 60
```

Annotated Bibliography

Apple Computer, Inc.
Apple IIGS Firmware Reference
Contains a lengthy description of the SmartPort firmware, including some clues as to the functioning of the 3.5 Drive hardware and a diagram of the layout of an individual block of data. You will also need Apple IIGS Technical Note 25, which corrects some errors.

Apple Computer, Inc.
Apple IIGS Hardware Reference
Contains a description of the disk interface register (DISKREG, $C031) and the internal registers of the IWM chip. You will also need Apple IIGS

Technical Note 30, which corrects numerous errors in the IWM descriptions.

Apple Computer, Inc.
Inside Macintosh, Volume III
Contains a description of most of the 3.5 Drive status and control bits.

Apple Computer, Inc.
Macintosh Family Hardware Reference
The 3.5 Drive information from Inside Macintosh is also reprinted in this book, in several different locations.

Don Worth
Pieter Lechner
Beneath Apple DOS
Quality Software
Reseda, CA
1981
THE classic reference for anything and everything having to do with DOS 3. and the 5.25 Drive hardware. Although the 3.5 Drive is a much more comple and powerful device, and uses a slightly different data format, much of th low-level information in this book is still quite relevant.

Don Worth
Pieter Lechner
Beneath Apple ProDOS
Reston Publishing Company
Reston, VA
1984
This does for ProDOS what _Beneath Apple DOS_ did for DOS 3.3. It contair a somewhat abbreviated version of the previous volume's description of low-level formatting, and in addition offers some valuable information on the functioning of the disk interface hardware.