Read Timing for the Altair 8 inch Floppy

For those who have looked at Altair disk code, it is well known that the sector loop transfers two bytes per iteration instead of just one. This technique allows the 2mhz 8080 to keep up with the 32us per byte transfer rate of the disk drive. However, the first solution that comes to mind – a single byte transfer per loop iteration – also works. For example, given a pointer to a sector buffer in HL and a byte count in B, the loop could be written as shown below:

Single Byte Transfer per Iteration

```
wtData   in     DSTATUS       ;(10) get drive status byte
         ora    a             ;(4) wait for NRDA flag true (zero)
         jm     wtData        ;(10)

         in     DDATA         ;(10) read the byte
         mov    m,h           ;(7) move the byte to memory
         inx    h             ;(5) increment memory pointer
         dcr    b             ;(5) decrement byte count
         jnz    wtData        ;(10) get next byte
```

This loop is 61 cycles. It must be less than 64 cycles (32us) for the 2mhz Altair 8080 to keep up with the disk transfer rate. With three cycles to spare, this loop can handle about a 4% speed variation between two drives.

I have verified this loop works in a modified CP/M and disk transfer utility. I've had reliable disk interchange between three different drives – a direct-drive FD400, a DC motor belt-drive FD410, and an AC motor belt-drive FD510.

So why was the more complex two byte per loop transfer used? Most likely because the FD400 specs allow for ±1.5% long term speed variance and ±1.5% instantaneous speed variance (ISV). For a short burst, an in-spec drive could be as much as 3% fast or 3% slow. This represents a potential 6% speed difference between two drives. In this case, the 61 cycle loop above could fail to keep up. So it seems that Bill and/or Paul decided to err on the side of caution and created the two byte read loop.

The typical two byte per iteration loop found in Altair code (given a pointer to the sector buffer in HL and a byte count in B) is shown on the following page. This loop is 107 cycles. It must be less than 128 cycles for the 2mhz Altair 8080 to keep up with the disk transfer rate. While the 107/128 cycles appears to give plenty of headroom for speed variance, the limiting factor in this loop is actually the time at which the second byte is read. The second byte is available from the controller during cycles 64 to 128 after NRDA is asserted for the first byte. This code performs the read at 70 to 94 cycles after first byte NRDA is asserted. This provides worst case headroom of six cycles which, in turn, can accommodate a speed variance between drives of about 9%.

Two Byte Transfer per Iteration

```
wtData   in     DSTATUS        ;(10) get drive status byte
         ora    a              ;(4) wait for NRDA flag true (zero)
         jm     wtData         ;(10)

         in     DDATA          ;(10) read the byte
         mov    m,h            ;(7) move the byte to memory
         inx    h              ;(5) increment memory pointer
         dcr    b              ;(5) decrement byte count
         jz     rdDone         ;(10) read is done when B=0

         dcr    b              ;(5) decrement count for 2nd byte
         nop                   ;(4) delay for 2nd byte

         in     DDATA          ;(10) at 70-94 cycles after 1st NRDA
         mov    m,h            ;(7) move the byte to memory
         inx    h              ;(5) increment memory pointer
         dcr    b              ;(5) decrement byte count
         jnz    wtData         ;(10) get next byte

rdDone   ...
```

A loop very similar to the one shown above appears in all versions of Altair Disk BASIC and Altair DOS. Even third-party products like Lifeboat and Burcon CP/M use the same loop. Apparently, however, this loop was not properly implemented in the Altair Disk Boot Loader PROM (DBL).

The DBL PROM does not include the NOP instruction prior to reading the second byte in the loop. The DBL reads the second byte at just 66-90 cycles after NRDA is asserted for the first byte. This provides worst case headroom of just two cycles. A speed variance between two drives of just over 3% could cause boot failure, even though once booted, a 9% difference could be tolerated. There is spare room in the DBL PROM to insert the NOP instruction, or a different bootloader PROM (like CDBL from Martin Eberhard) could be used to provide 9% or better speed tolerance.

Write Timing for the Altair 8 inch Floppy

Similar to the read sector loop, the write sector loop in original Altair software also transfers two bytes per iteration instead of just one. Again, however, the first solution that comes to mind – a single byte transfer per loop iteration – also works. For example, given a pointer to a sector buffer in HL and a byte count in B, the loop could be written as shown below:

Single Byte Transfer per Iteration

```
wtData   in     DSTATUS        ;(10) get drive status byte
         rrc                    ;(4) wait for ENWD flag true (zero)
         jc     wtData          ;(10)

         mov    a,m             ;(7) A=next byte to write
         out    DDATA           ;(10) write the byte
         inx    h               ;(5) increment memory pointer
         dcr    b               ;(5) decrement byte count
         jnz    wtData          ;(10) get next byte
```

This loop is 61 cycles. It must be less than 64 cycles (32us) for the 2mhz Altair 8080 to keep up with the disk transfer rate. However, unlike a read loop which has to accommodate disks written at different rotation rates, write timing is derived from the CPU clock. This means there is no speed variance to worry about – the three cycles of headroom in this loop are plenty. This avoids the messy double-byte transfer used in the typical Altair write loop shown here:

Two Byte Transfer per Iteration

```
         mvi    d,ENWDMSK       ;write data flag mask
         mov    e,m             ;E=first byte to write
         inx    h

wtData   in     drvStat         ;read drive status register
         ana    d               ;write flag (ENWD) asserted (zero)?
         jnz    wtData          ;no, keep waiting

         add    e               ;put byte to write into accumulator
         out    drvData         ;write the byte

         mov    a,m             ;A=next byte to write
         inx    h
         mov    e,m             ;E=byte to write next loop
         inx    h               ;increment source buffer pointer
         dcr    c               ;count byte just written
         jz     wrDone          ;jump if sector done

         dcr    c               ;count byte about to write
         out    drvData         ;write 2nd byte
         jnz    wtData          ;loop if count <> 0

wrDone   ...
```