A Floppy Disk Interface

David M Allen
Electronics Consultant
1317 Central Av
Kansas City KS 66102

Once it would have been appropriate to begin this article with a paragraph justifying the use of a floppy disk in a hobbyist system. Today that paragraph is unnecessary for two reasons: First, most of the more ambitious microprocessor users have already convinced themselves of the need for fast programmable memory; second, a growing number of users have begun to discover that the hardware required to interface a floppy disk to a microprocessor doesn't need to be very complex, and certainly not as expensive as some of the purveyors of professional equipment would have us believe. This article presents a straightforward way to control a floppy disk drive with a microprocessor.

While definitely not a project for a beginner, the interface itself is relatively simple and well within the grasp of anyone who has built other computer equipment. It is conceivable that people without an oscilloscope might be able to build the interface blindly and get it working successfully, but I strongly discourage them from doing so. A good oscilloscope, particularly one with dual trace and variable delay features, is highly recommended for testing the interface and drive during construction. Once built and working, the unit should not need any adjustments.

I do not attempt to provide all of the cookbook information necessary to thoroughly educate the novice in the ways of floppy disks. Rather, I provide an overview of one system, cover all the basic features and functions, and draw examples from a working system. I hope to encourage people to build their own systems as an alternative to buying someone else's.

This interface is intended for hardware sectored disks and disk drives, although the basic technique could be extended by the use of more on board buffer memory to permit the use of a software sectored format. This interface is specifically designed for the Memorex 651 drive, but the use of an on board buffer makes the interface independent of the associated processor's speed, and allows it to be used with virtually any processor. At least one person has used a similar interface with a 4004 for three years. The use of an on board buffer did not add to the chip count of the interface; if anything, it reduced it. The chip count is 31, and all chips are small or medium scale integration TTL with the exception of the buffer memory.

Table 1: Power wiring table for the integrated circuits used in figure 1.

Number	Type	+5 V	GND
IC1	7400	14	7
IC2	74175	16	8
IC3	7400	14	7
IC4	2102A6	10	9
IC5	74177	14	7
IC6	74125	14	7
IC7	74125	14	7
IC8	7404	14	7
IC9	7416	14	7
IC10	7416	14	7
IC11	7404	14	7
IC12	7404	14	7
IC13	7408	14	7
IC14	2102A6	10	9
IC15	74177	14	7
IC16	74125	14	7
IC17	7430	14	7
IC18	7411	14	7
IC19	7410	14	7
IC20	74175	16	8
IC21	74177	14	7
IC22	74121	14	7
IC23	74177	14	7
IC24	2102A6	10	9
IC25	74177	14	7
IC26	74177	14	7
IC27	7474	14	7
IC28	7420	14	7
IC29	7430	14	7
IC30	7474	14	7
IC31	7474	14	7
IC32	7404	14	7

Circuit Description

Several basic functions must be provided

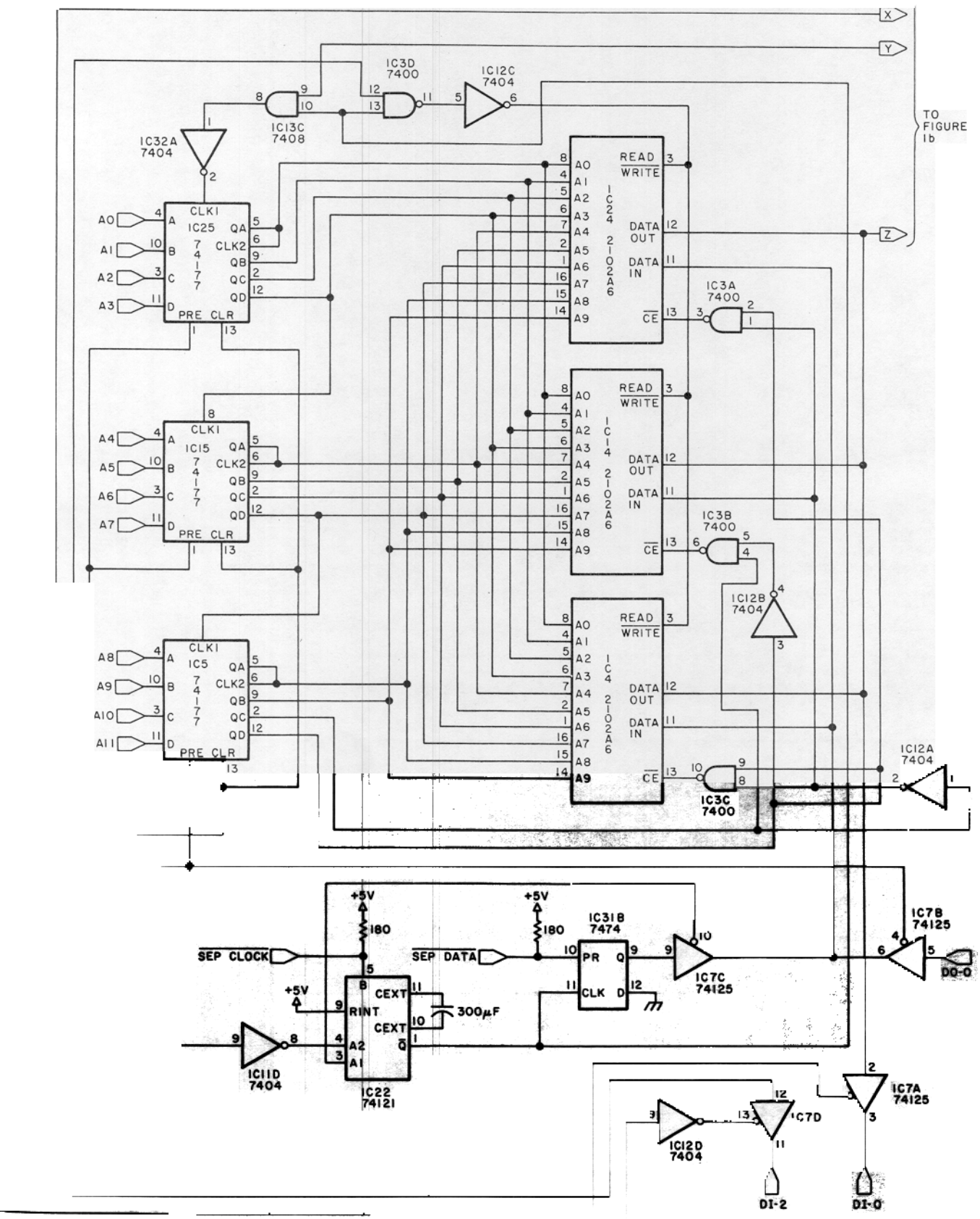
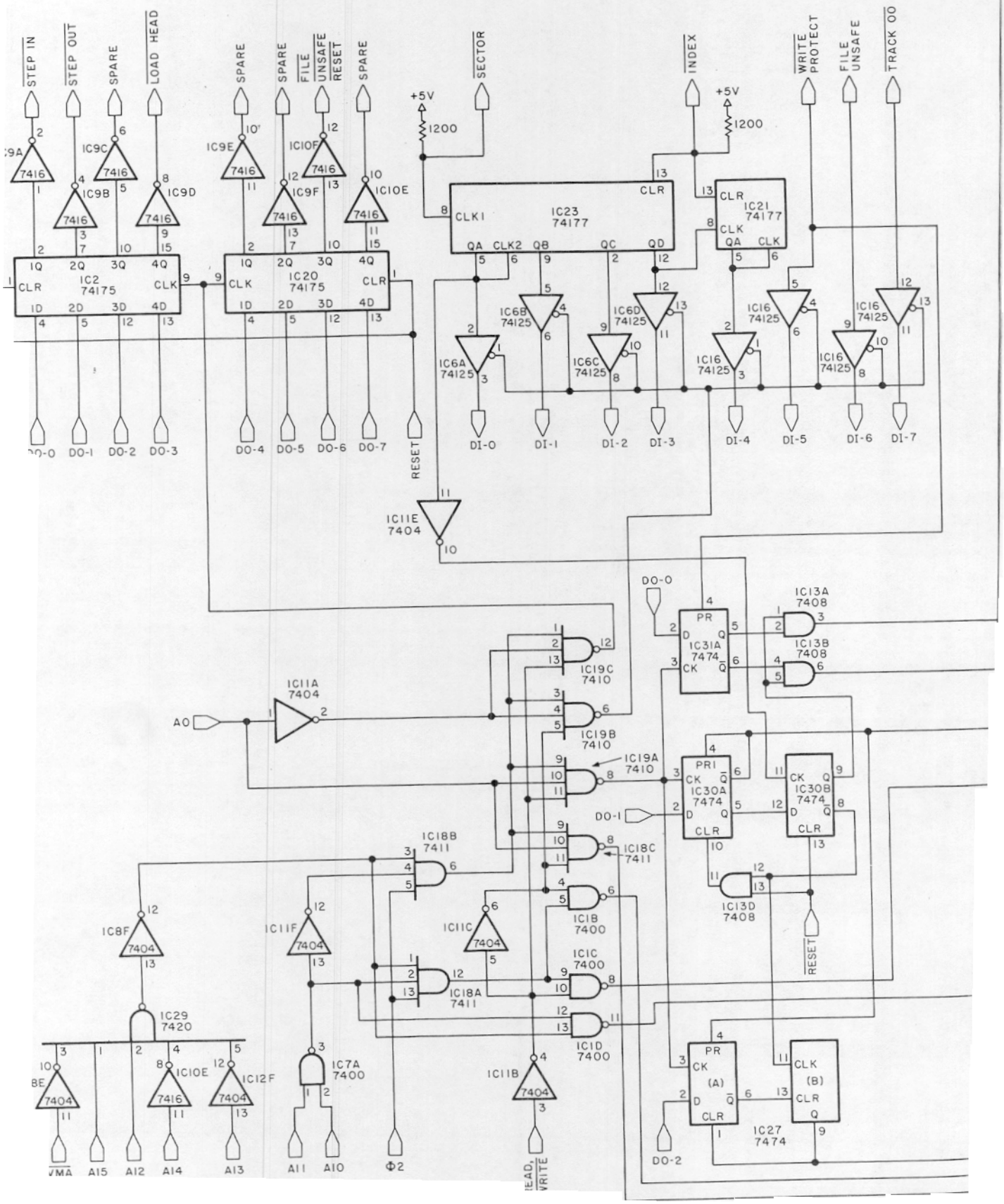

X
Y
Z
TO FIGURE 1b
IC3D 7400
IC12C 7404
IC13C 7408
IC32A 7404
IC25 74177
IC24 2102A6
IC15 74177
IC14 2102A6
IC5 74177
IC14 2102A6
CLK1
PRE CLR
READ/WRITE
DATA OUT
DATA IN
CE
IC3A 7400
IC3B 7400
IC12B 7404
IC12A 7404
IC3C 7400
+5V
180
SEP CLOCK
SEP DATA
IC11D 7404
IC22 74121
300μF
IC31B 7474
IC7C 74125
IC7B 74125
IC7A 74125
IC12D 7404
IC7D
DI-2
DI-0
DO-0

Figure 1a: Schematic diagram for the floppy disk interface board. All inputs and outputs are designed and buffered for use with a Motorola 6800 processor and a Memorex 651 floppy disk drive.



in a floppy disk interface, regardless of the actual data transmission method. Outputting step pulses, receiving sector and index pulses, loading the data head, sensing and resetting the file unsafe error bit, and sensing the write protect tab on the diskette are functions which are largely universal to all disk drives. They are handled by the 74175 latches (IC2 and IC20) and 74125 bus drivers (IC6 and IC16) in figure 1. Software takes care of all necessary interlocking and timing relationships between these signals.

One minor exception is in the use of the sector pulses from the drive. I felt that the incorporation of a hardware counter (to convert the individual sector pulses into an address and relieve the processor of the necessity of providing this function in software) was too simple to be left out. The equivalent function in software would yield an excessive and unnecessary overhead in time and software. As such, the two counter packages, which provide the processor with a complete sector address rather than a series of sector pulses, were a welcome expense.

Data passage between the buffer memory and the disk drive is processor independent. The processor simply requests the desired transaction (read or write). Actual data movement is carried out solely by the interface circuits.

Movement of data in this system takes place in two phases, regardless of whether the transaction is a write or read. When writing, the processor loads the data destined to be recorded onto a sector of the diskette into the onboard buffer memory. When the entire sector's data has been assembled in this buffer, complete with necessary preambles, data separators, track and sector identifiers, postambles, and whatever else is needed by the chosen format, this block of sector data can be moved to the disk drive in a single operation. The actual operation does not occur until the processor commands the interface board to write.

After receipt of the write command the interface waits until it sees the leading edge of a sector pulse. This sector pulse sets a flip flop, IC30B, which remains set throughout the operation and resets on the leading edge of the next sector pulse. The entire transaction is therefore synchronous with the sector pulses. This same synchronization is present during a read operation as well. For example, reading of disk data into the on board buffer is started and stopped by sector pulses. In this way, format independence is accomplished because the entire sector is read into the buffer in a single operation. The processor is completely uninvolved with this transfer of data. It does not re-enter the picture until the read operation is complete, at which time it begins dissecting the received information. During this dissection process the processor must search through the on board buffer looking for the beginning of data markers (and other landmarks of the particular recording format involved to locate the desired data in the buffer and transform it from serial to 8 bit parallel form.

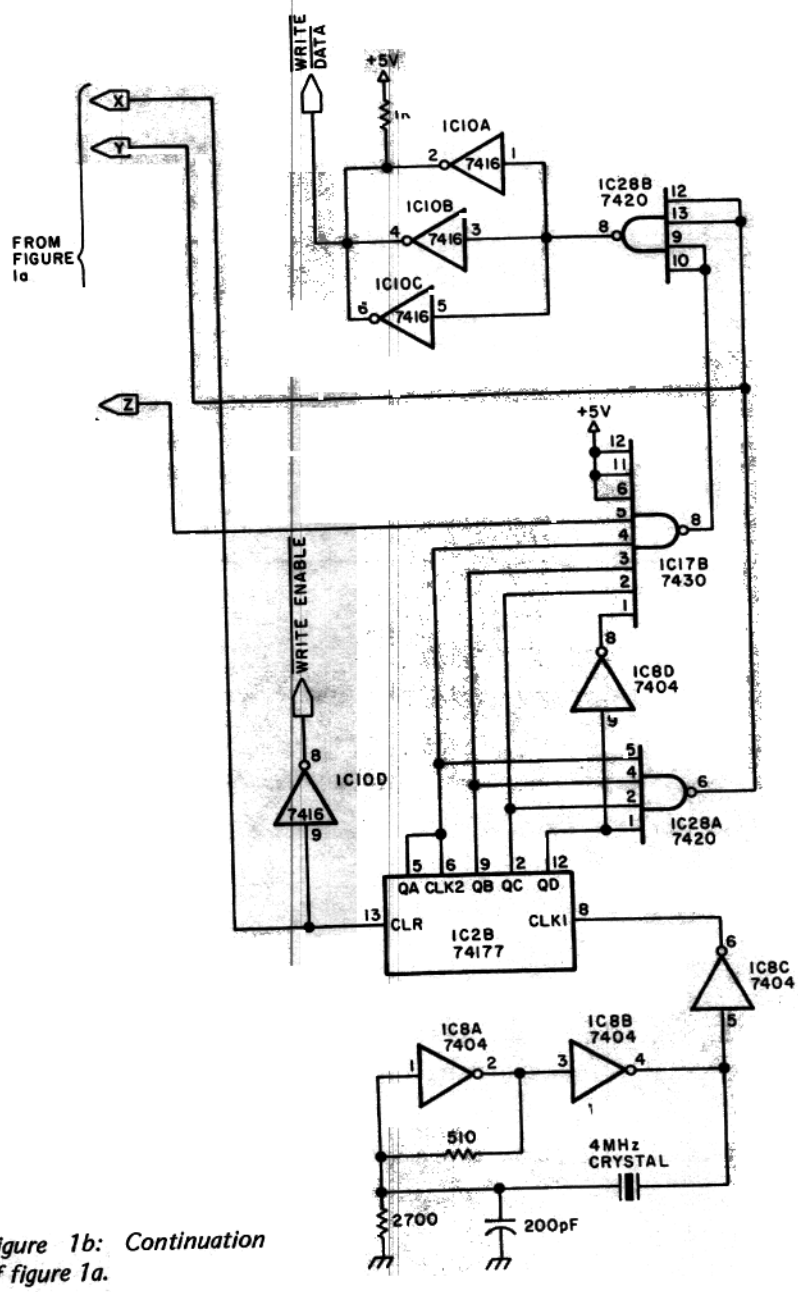


Figure 1b: Continuation of figure 1a.

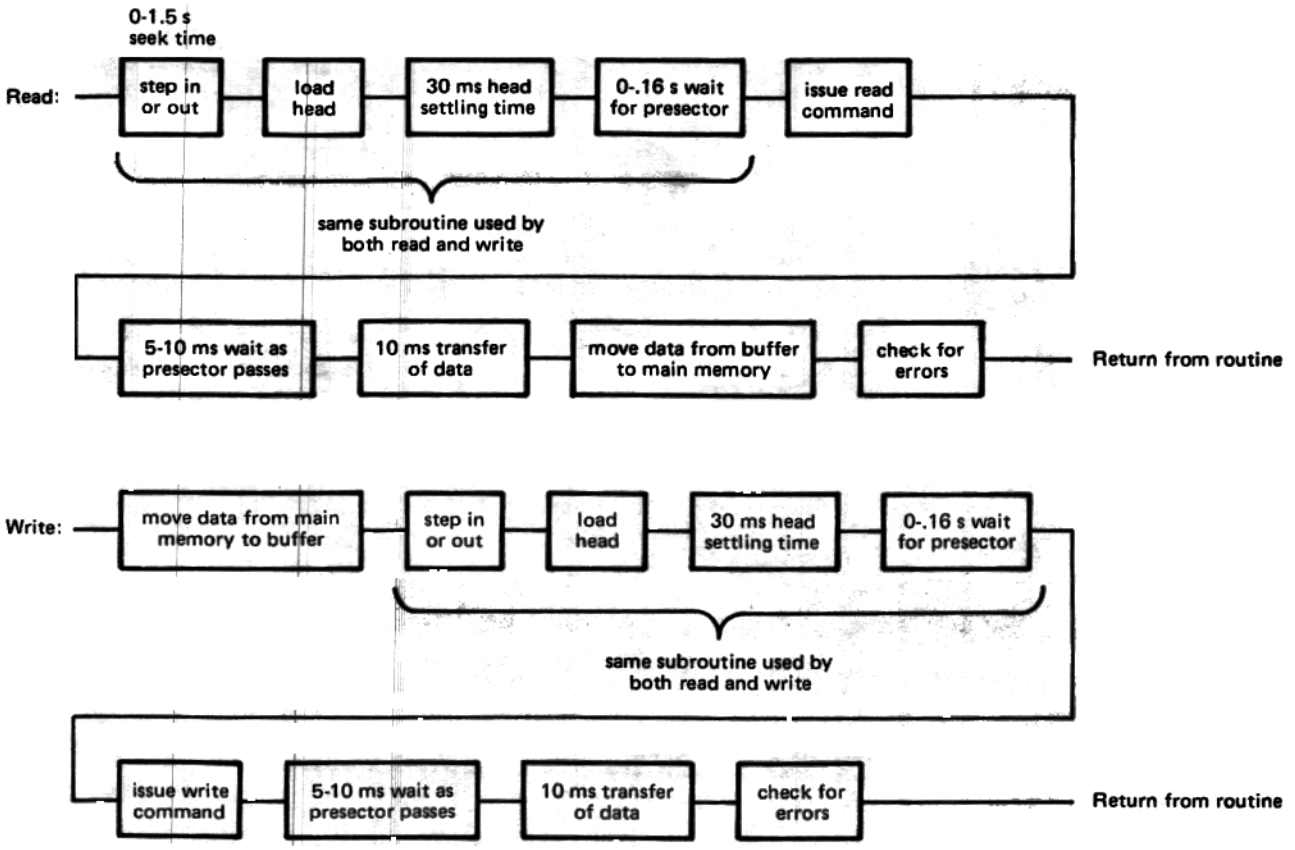
Note that, in this system, the actual SECTOR pulses from the disk drive are not used to start and stop the transactions; rather, they are divided by 2 in the first stage of the sector address counters. This means that although 32 physical sectors could be recorded per track on the diskette, this system will record only 16.

Each sector contains 256 bytes of data instead of 128. I did this to facilitate future double-density operation. The size of the buffer is 256 bytes of data plus some extra for preambles, checksums, and so on, for both single or double density operation. Only the number of sectors per track will be charged: they will increase from 16 for single density to 32 for double-density. For purposes of this article, single density operation is assumed.

The interface board knows nothing about the particular recording format involved, but merely collects a sector's worth of information (ie: everything appearing between two successive sector pulses) from the disk drive and gives it to the processor. The processor must know all about the organization of the data in the sector. The software completely controls the organizing and formatting of this data.

Although it is feasible to configure the buffer memory as an 8 bit wide parallel buffer, the serial configuration has several advantages including the lowest cost and chip count. It also permits efficient utilization of low cost memory chips such as the 2102A. Although it would seem logical to use serial shift registers for the buffer, doing so destroys the processor's ability to randomly access the buffer. Another reason for using serial data transmission is that conversion to parallel format, when receiving data from the disk, would necessitate some form of data synchronization information so that the packing of the serial data into the parallel buffer can be coordinated. Without this information, bit 1 might end up in bit 8's location. This synchronization function must be provided in any case. Postponing the problem until a time when the processor is processing the data, instead of the time when the disk drive and interface are exchanging data, allows the interface hardware to be format independent. This means that, with little

Figure 2: Sequence of main events when reading from or writing to the floppy disk.



or no change to the interface board, any hardware sectored format can be accommodated by the use of appropriate software. With the present amorphous state of the home computing industry, format independence has to be a significant advantage.

In figure 1, the three 2102As (ICs 4, 14 and 24) form a 3072 by 1 bit buffer memory. The three 74177 counters (ICs 5, 15 and 25), which are associated with the three 2102As, allow storage of the serially accessed disk data in the randomly accessed memories. These counters are incremented for each bit of disk data passed, and provide addresses for the 2102As during both the read and write operations. During reading, the disk provides clocking information in the form of SEP CLOCK on the Memorex disk drive. SEP CLOCK is used to clock the counters and address the memory. During writing, the interface card must provide clocking information to the disk. The crystal oscillator provides this write clock information for the disk drive as well as for the counters which address the memory.

The preceding has explained the functioning of the counters and the addressing of the 2102A buffer during data exchanges between the disk drive and the interface card. Movement of data between the interface card and the processor takes place before or after the interface's dialogue with the disk drive, as shown by the sequence of operations for read and write commands in figure 2. During this processor activity, the address counters take on a completely different function. The processor can randomly access any bit in the memory buffer. Therefore, the address for the buffer comes from the processor rather than from the address counters.

During this time, the addresses on the processor's bus, which are present at the preset inputs of the 74177s (ICs 5, 15, 25), are gated through to the 2102As. When the processor attempts to read from or write to the buffer it will be accessing only one bit at a time, since the buffer is only one bit wide. The 6800's read and write line is gated directly to the read and write line of the 2102As. Write data from the processor will be fed to the data input lines of the 2102As via a three state device enabled by the proper address decoding from the processor bus (IC1). When the processor reads from the buffer memory, address decoding (IC1B) enables a three state bus driver and places one bit of data from the buffer on the processor bus.

During a diskette read operation, it is desirable to use the received clock to strobe the received data into memory. In order to satisfy the minimum write pulse width requirements of some 2102 type programmable memories, it is necessary to use a oneshot (IC22) triggered by the trailing edge of the receive clock to generate the actual write strobe. The received data is latched in a 7474 (IC31B) and maintained until the end of the write strobe. Note that while it would normally be poor practice to use the write strobe to change both address and data, the architecture of this interface and of most 2102 type programmable memories permits this. The actual value of the address present at the edges of the write strobe appears to be irrelevant, provided that the address is stable. In this design, the address and the data change after the trailing edge. This arrangement has been tested with several types of programmable memories, from fast new ones to slow obsolete ones. There were no problems in any instance.

During a write operation, a 4 MHz crystal provides clock pulses to the disk drive and address counters for the 2102As. A 4 MHz crystal is used to facilitate the future double density option. A 2 MHz crystal could be used for standard density operation with the proper circuit modifications. The write clock and write data signals (outputs of IC17 and IC28B) are combined into a composite clock and data signal and fed to the disk drive via three paralleled sections of a 7416 high power inverter (ICs 10A, B, and C). Three sections are used to provide adequate signal current into the low impedance line receiver within the disk drive. Although line receivers are recommended by Memorex for the received clock and received data signals at the interface, standard TTL inputs (with the addition of the pull up resistors shown in the schematic) have proven to be adequate.

Software

The schematic of figure 1 does not represent a floppy disk controller, but only an interface. The processor intelligence is needed to make the system a controller. The bulk of the effort in establishing a system of this type occurs in the writing of the necessary software. This approach keeps the hardware costs low and allows the frugal experimenter to substitute his or her own software writing ability for the out of pocket hardware expenses.

Two phases in any read or write operation of the system were previously mentioned. A third phase is necessary prior to any actual reading or writing. The disk drive's data transfer head must be stepped

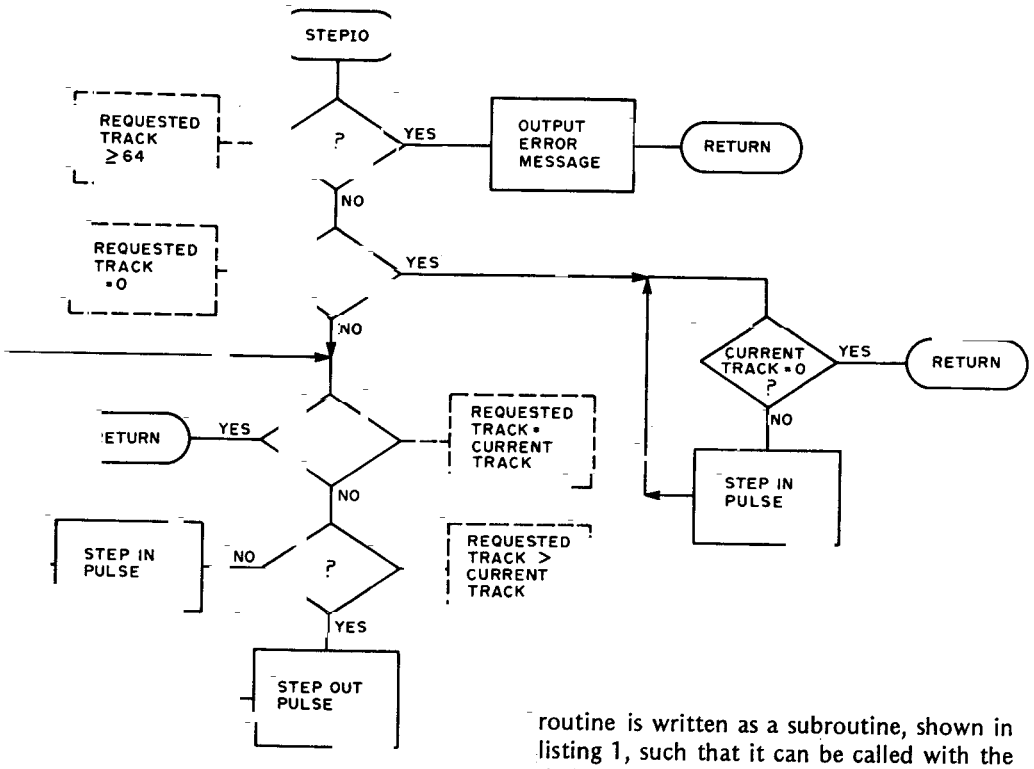


Figure 3: The stepping routine, which moves to any track that is requested.

to the proper track. Then the system must interrogate the hardware's sector address counters in order to know when it is permissible to issue a read or write command.

Stepping to the Proper Track

There are either 64 or 77 tracks on floppy disk systems (the Memorex 651 uses 64). When power is first applied to the system, the position of the data transfer head is unknown. It is necessary to find over which track the head is positioned. Unfortunately the disk drive tells us nothing about the location of the head unless it is sitting at the very first track of the system (track 0). Some disk drives have an additional sensor at the very last track. Consequently, during initialization of these systems it is necessary to step the data transfer head in toward track 0 until the track 0 sensor is tripped. If track 0 is requested (as shown in the flowchart of figure 3 for the stepping routine), the program will step the data transfer head in, while searching for a TRACK 00 signal. An 8 bit register remembers where the head is located. Once the head is stepped away from track 0, the software has no way of determining what track is currently accessed unless it knows how many step pulses have been issued. Every time a step pulse is issued, this counter will be incremented or decremented appropriately unless the track 0 signal appears, in which case the counter is cleared. This step routine is written as a subroutine, shown in listing 1, such that it can be called with the desired track parameter passed as a value in another register.

Stepping takes place in response to individual pulses from the interface. The pulses are formed in software by turning a hardware bit on, and then off. The width of the pulse will be determined by the processor's execution speed. Since two or more instructions are involved, the width will be several microseconds. Several NOPs can be inserted in the software as a safety measure.

Listing 1: Stepping in or out routine written for the Motorola 6800 processor.

```
FA3D B6 A055 STEPIO LDA A  REQTRK
FA40 81 3F          CMP A  #63
FA42 2F 03          BLE    *+5
FA44 7E FB8F        JMP    ADRERR
FA47 81 00          CMP A  #0
FA49 26 0F          BNE    STP2
FA4B C6 80    STP1  LDA B  #$80     TEST TRACK ZERO BIT
FA4D F5 9C00        BIT B  $9C00
FA50 27 04          BEQ    *+6
FA52 8D 1D          BSR    STEPIN
FA54 20 F5          BRA    STP1
FA56 7F A054        CLR    ACTTRK   TRACK ZERO FOUND
FA59 39             RTS
FA5A B1 A054 STP2   CMP A  ACTTRK
FA5D 26 01          BNE    *+3
FA5F 39             RTS
FA60 2B 04          BMI    *+6
FA62 8D 06          BSR    STPOUT
FA64 20 F4          BRA    STP2     LOOP TILL TRACK ZERO
FA66 8D 09          BSR    STEPIN
FA68 20 F0          BRA    STP2
FA6A 7C A054 STPOUT INC    ACTTRK
FA6D C6 01          LDA B  #1        STEP OUT PULSE
FA6F 20 05          BRA    STP4
FA71 7A A054 STEPIN DEC    ACTTRK
FA74 C6 02          LDA B  #2        STEP IN PULSE
FA76 F7 9C00 STP4   STA B  $9C00     BEGIN STEP PULSE
FA79 5F             CLR B
FA7A 01             NOP              DELAY
FA7B 01             NOP
FA7C F7 9C00        STA B  $9C00     END STEP PULSE
FA7F CE 0C00 DELAY  LDX    #$C00
FA82 09       STP3  DEX
FA83 26 FD          BNE    STP3
FA85 39             RTS
```

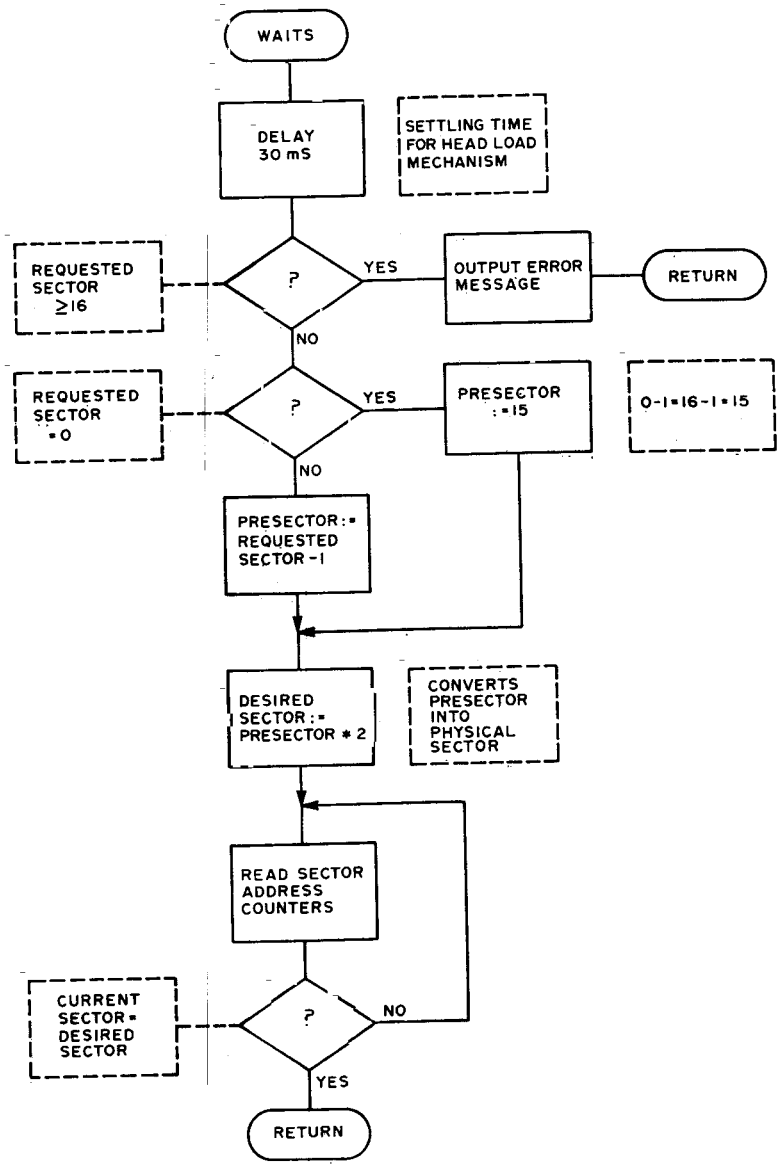


Figure 4: A routine which searches a single track until it finds the required presector.

```
FAA4 BD D9   WAITS  BSR    DELAY
FAA6 B6 A051        LDA A  REQSEC
FAA9 81 10          CMP A  #$10     IS REQ'D SECT. LEGIT.?
FAAB 2D 03          BLT    WS1
FAAD 7E FB8F        JMP    ADRERR
FAB0 81 00   WS1    CMP A  #0       SECTOR 0 REQ'D
FAB2 26 02          BNE    WS2
FAB4 86 10          LDA A  #$10     DUMMY SECTOR #
FAB6 4A      WS2    DEC A           SET UP PRESECTOR
FAB7 48             ASL A           SET UP HARD SECTOR NUMBER
FAB8 F6 9C00 WS3    LDA B  $9C00    GET SECTOR FROM INTFC
FABB C4 1F          AND B  #$1F
FABD 11             CBA             REQ:ACTUAL
FABE 26 F8          BNE    WS3
FAC0 39             RTS
```

Listing 2: Interrogation routine written for Motorola 6800. This routine checks each of the sector headings on a track until it finds the requested presector of the desired sector.

Since a stepping motor is involved (some of the newer drives incorporate linear actuators which are faster), a considerable time delay between step pulses must be allowed to prevent the stepper motor from overshooting the wrong track. The software stepping routine incorporates a 30 ms delay. Memorex specifies a 20 ms delay, and some other manufacturers call for 10 ms delays or less. The builder should adjust the delay parameter to generate the necessary delay.

Waiting for the Proper Sector

Since counters are built into the hardware, the sector address will be accurately known within one diskette revolution after application of power. Anytime the software reads the state of the sector counters, it will receive the binary number of the physical sector currently passing under the data transfer head. The use of this information now becomes somewhat involved.

At some time in the future, a read or write command will be issued. Where that occurs is a function of the time when the command is issued relative to the position of the diskette at the moment of issuance. If the user wants to write at logical sector 4 of the current track, for instance, the write command is issued prior to the appearance of physical sector 4 beneath the data transfer head. In other words, the command should take place while the data head is over sector 3 because the actual writing must begin at the leading edge of the sector pulse that introduces physical sector 4.

Things are actually more complex than this. The counters which address the on board buffer memory must be reset prior to the beginning of the actual read or write operation. Otherwise, the buffer addresses would begin in the middle of the buffer. In order to guarantee sufficient time for resetting the counters, some finite time interval must occur during the preceding physical sector. The only way to insure this is to locate the end of second preceding physical sector and issue the read or write command at the beginning of the immediately preceding physical sector.

The flowchart in figure 4 shows a subroutine which, when given the desired sector to be read from or written to, will loop while interrogating the sector counters until the presector is found. Although the software of listing 2 appears to ignore the need to guarantee some resetting time for the buffer address counters, this is not the case. Since there are two physical sectors for each logical sector (32 physical sectors per track versus 16 logical sectors) the software is sensing

```
F9BB FF A05E READSV STX   DBUFF    SAVE DATA DESTINATION ADDR
F9BE BD FA86        JSR   MSETUP   MOVE HEAD TO TRACK/SECTOR
F9C1 86 03          LDA A #3
F9C3 B7 9C01        STA A $9C01    REQUEST READ
F9C6 BD FA95        JSR   WAITFL   WAIT FOR 'FINISHED' FLAG
F9C9 BD FB08        JSR   ENPACK   CONVERT DATA TO 8-BIT BYTES
F9CC BD FB66        JSR   CHKSUM   COMPUTE NEW CHECKSUM
F9CF FE A05A        LDX   CKSM
F9D2 BC 5100        CPX   RBUFF+256  COMPARE NEW WITH RECORDED
F9D5 27 03          BEQ   *+5
F9D7 7E FB87        JMP   CKSMER
F9DA FE A056        LDX   REQADR   ADDR REQ'D BY CONTROLLER
F9DD BC 5102        CPX   RBUFF+258  COMPARE W/RECORDED ADDR
F9E0 27 03          BEQ   *+5
F9E2 7E FB99        JMP   TSIDERR
F9E5 CE 5000        LDX   #RBUFF
F9E8 FF A05C        STX   SBUFF
F9EB BD FA26        JSR   COPY
F9EE 39             RTS

F9EF FF A05C WRITSV STX   SBUFF    X-REG DESIGNATES SOURCE DATA
F9F2 C6 20          LDA B #$20     TEST WRITE PROTECT
F9F4 F5 9C00        BIT B $9C00
F9F7 26 03          BNE   *+5
F9F9 7E FB9E        JMP   WPROT
F9FC CE 5000        LDX   #RBUFF
F9FF FF A05E        STX   DBUFF
FA02 BD FA26        JSR   COPY
FA05 BD FB66        JSR   CHKSUM   COMPUTE CHECKSUM
FA08 FE A05A        LDX   CKSM
FA0B FF 5100        STX   RBUFF+256
FA0E FE A056        LDX   REQADR
FA11 FF 5102        STX   RBUFF+258
FA14 BD FAC1        JSR   UNPACK   STORE ASSEMBLED FILE
                          INTO SERIAL BUFFER
                    *
FA17 BD FA86        JSR   MSETUP   POSITION HEAD, ETC.
FA1A 86 02          LDA A #2       WRITE TRANSACTION
FA1C B7 9C01        STA A $9C01
FA1F BD FA95        JSR   WAITFL   WAIT TILL DONE
FA22 BD FB3D        JSR   CHKFIL   CHECK FILE UNSAFE
FA25 39             RTS
```

Listing 3: The WRITSV routine allows writing to the disk, and the READSV routine is used to read data from the disk. When either of these two routines is entered, the correct track and sector have been found and the data transfer head is already loaded and allowed to settle.

physical sectors and dividing the sensed addresses by 2. This will allow at least 5 ms (the time for one physical sector) for resetting the address counters.

Read and Write Commands

One bit of a status word is used to identify a read or write command. In the schematics, logical 1 means *read* and logic 0 means *write*. Another bit is used as the transaction request bit. Setting this byte to logical 1 enables the hardware to execute a data transaction when the next sector pulse occurs. During this period of time the processor must not read from or write to the on board buffer, since this will immediately set the address counters to the value present on the processor's address bus. The processor needs to know if the disk drive is talking to the buffer memory. Therefore, a busy or ready bit is incorporated into the hardware. Proponents of interrupt driven systems should note that a separate flip flop is set at the end of a read or write transaction. This should be reset in software prior to issuing a new command. Hardware will reset the flip flop automatically at the beginning of each transaction in the event that software does not do this. This flip flop can be used as a flag for a testing loop, or it can be tied to an interrupt line to generate an interrupt upon completion of the read or write transaction. The software examples used here incorporate a flag testing loop routine.

Operating Sequence

Sometime prior to the passing of the data to or from the disk drive, the head load bail must be energized to permit the data transfer head to come into contact with the diskette. The mechanical properties of this mechanism dictate that some time must be allowed for settling prior to the commencement of any data movement. Various philosophies exist on the subject of how long and how often the head should remain loaded. Some feel that the head should be loaded continuously during periods of usage, and that frequent loading and unloading of the head causes aggravation of the head as well as a media wear problem. The software I use assumes just the opposite: The less time the head and media are in contact, the less wear will occur. Memorex drives are particularly vulnerable to wear, since they use metal heads instead of newer ferrite and ceramic heads. A Memorex drive using this software and operating in a suitably clean environment for several months has had no detectable wear to either the data head or the diskette. I feel that this is due to the fact that the data head is never in contact with the media for more than a fraction of a second at a time. The accumulated contact time is kept very low.

In the software of listing 3, the data head is loaded just after the stepping routine, immediately following the arrival at the proper track. Recall that the system will begin its search for the pre-presector after the proper track has been accessed. The sector wait algorithm contains a 30 ms delay during which the head load mechanism can settle. An extra delay loop should be inserted at the entry of the sector wait routine if additional settling time is necessary. The accumulated system delay, including the sector wait routine, will probably be adequate.

Figure 5 shows the sequence of operations for read and write commands and includes the movements of data relative to the mechanical operations. For a clearer description, see photo 1. If a wire operation is to take place, the first step in the process should be setting up the data in the interface's onboard memory buffer. Some sort of structure must be chosen for the various

zones which must be recorded onto each sector. Memorex recommends that a preamble of 128 logical 0s be written to the disk starting at the beginning of the sector. What goes onto the disk after that is dependent on the whims and opinions of the programmer. In the prototype system, a single logical 1 follows the preamble and serves to indicate that the next following bit is the first bit of the first word of data. 256 8 bit words are then written with the least significant bit first. This is followed by a 16 bit checksum, a 16 bit binary number to indicate the sector and track numbers, and zeroes for a postamble, which will end when the hardware encounters the sector pulse for the next sector. I make no claims for the elegance or efficiency of this format. It was expedient and has worked for months in a small developmental system. To set up the on board buffer with this information, a routine converts the source data from 8 bit parallel to serial (1 bit per address) in the buffer. After the buffer is set up, the mechanical movements and commands can be issued.

Operating

If the interface is wired correctly with properly working parts and the software is bug free, the system should work without any adjustments. However, if you build the hardware and write the software from scratch, there are many chances for error. I highly recommend that you build and test one section at a time. When testing the data circuits, a good quality oscilloscope, preferably dual trace with delayed sweep, is desirable. The read and write operations are conducted independent of the processor. This simplifies troubleshooting because bad software will not complicate the situation.

I suggest that you start by wiring the sector counters and verifying their proper operation by reading sector addresses into the processor. Once the sector counters are working, the sector wait software can be written and tested by observing the varying time that the software is tied up (by looking for the desired sector). This varying time is formally known as the rotational latency, and will be between 5 and 160 ms.

Next, wire the step in, step out, and track 00 signals. The stepper motor can be manually tested by shorting the stepping lines to ground. It is much more dramatic to write some simple software to exercise this function. At this time the stepping speed can be set and the malfunctioning of the stepper motor can be observed at excessively high step rates.

The first step in testing the operation of the on board buffer memory is to transmit data back and forth between the interface and the processor. Doing so will bypass the oneshot, address counters, etc, and make testing much simpler. Once the memory is found to be functional, communication with the disk can be tested. Grounding pin 11 of IC30B will force the hardware to execute a continuous string of read or write commands depending on the state of flip flop 31A. Flip flop 31A can be used to choose either read or write commands. When writing, the normally present reset signal at the timer counter (pin 13 of IC26) will be lifted. This allows the counter to count pulses from the oscillator. Pulses from the clock gate, pin 6

Figure 5: Timing diagram for read and write operations. Also shown is the physical and logical layout of the data on the floppy disk.

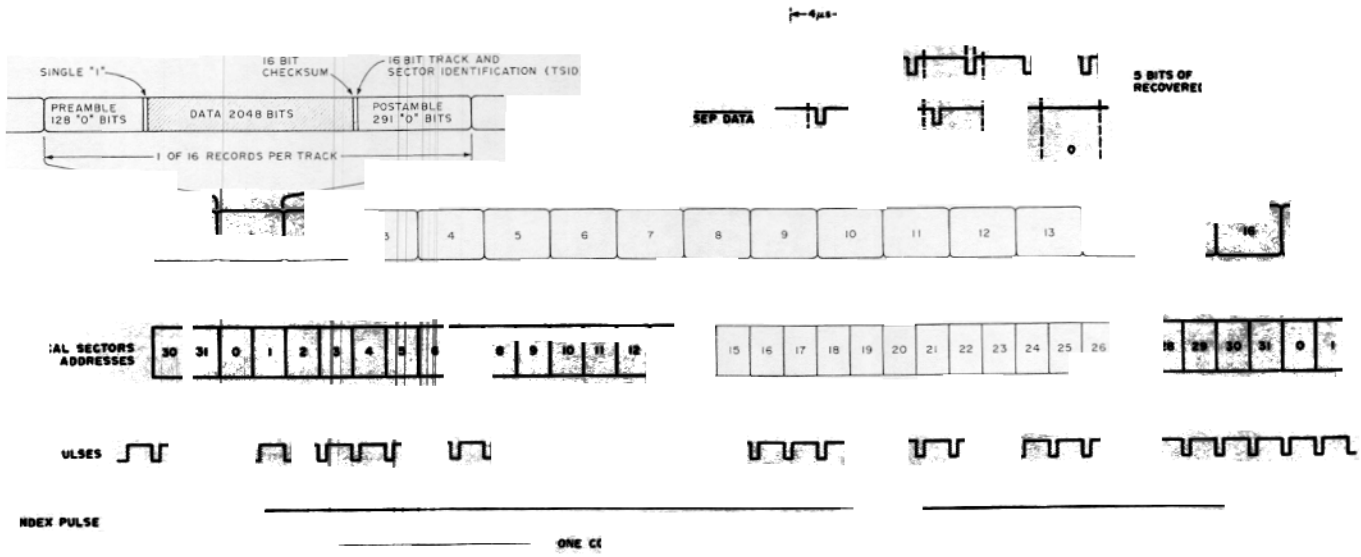


Photo 1a: Data as it appears when first received from the disk drive for eventual storage in the buffer memory.

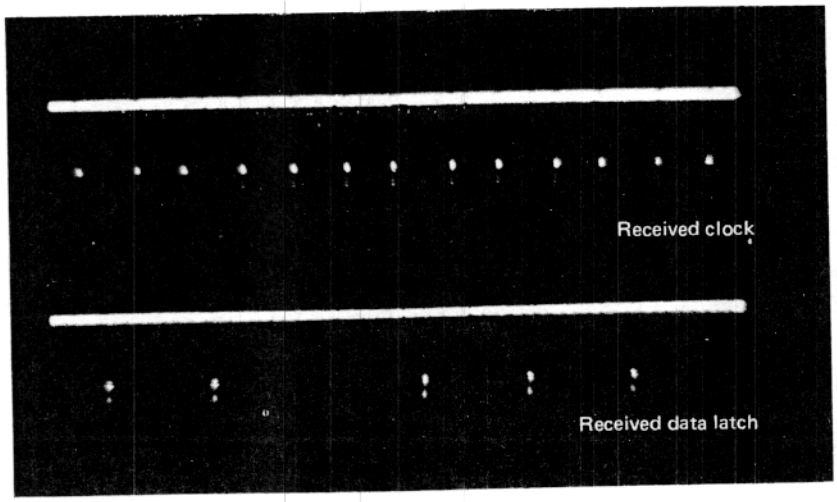
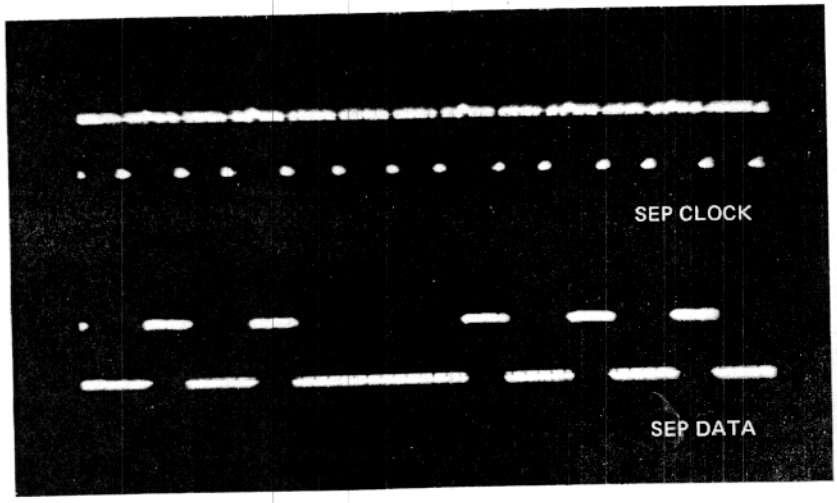


Photo 1b: An example of data which has been received from the disk drive and rearranged for storage in the buffer memory.



of IC28A, will be a continuous stream of pulses separated by 4 μs. Pulses from the data gate, pin 8 of IC17, will be dependent on the raw data coming from the 2102As. If the raw data is a steady high, the signal from the data gate will be similar to that from the the clock gate with the exception that the two signals will be displaced in time. When the two signals are combined at pin 8 of IC28B, they are interleaved. While the interface is in forced write mode, the clock pulses from the clock gate are applied to the address counters, pin 8 of IC25. These counters should be counting properly. Whatever data is stored in the 2102As should be coming out serially on the raw data line.

To test the read function, you will need a working disk drive and a prerecorded diskette. With the head loaded, a stream of data and clock pulses similar to those observed in the write circuits should be present at SEP CLOCK and SEP DATA. The one-shot should fire (pin 1 of IC22) on the trailing edge of each clock pulse. The one-shot's output should appear at the address counter's input and at the read and write line of the 2102As, pin 3 of ICs 4, 14 and 24. The 1 bit SEP DATA latch IC is set at the leading edge of any SEP DATA pulse and reset by the trailing edge of the one-shot's output.

The only other circuit of any complexity is that of the interrupt flag. If a read or write operation is terminated normally by the occurrence of the second sector pulse at the "busy" flip flop (pin 11 of IC30B), then resetting the flip flop will clock the interrupt flip flop on (pin 9 of IC27B) and either generate an interrupt or be used as a flag. This interrupt flip flop will remain on until software clears it by setting the interrupt clear flip flop, or until hardware clears it as soon as the transaction request flip flop, pin 6 of IC30A, is set in preparation for another read or write operation.

All other circuits are address decoding, latching, or bus driving functions and do not bear explanation here.

Drive Power

The lowly power supply is usually the least considered element in a system until it has to be paid for or fixed. Most floppy disk drives are extremely inefficient, due principally to the stepper motor. In the Memorex design the stepper motor consumes 24 V at 2 A. I have had only one troublesome power supply problem. It turned out to be due to an induced transient caused by a snapping action in the rectifier diodes during recovery and turn off. The resultant noise caused problems everywhere. I solved this by simply shunting the diodes with .05 or .1 μF capacitors.

Intercabling

Memorex offers a version of its drive for use with twisted pair signal lines and a printed circuit edge connector in place of the 93 Ω coaxial cable and captive pin connector of the original. This newer version should definitely be specified when buying from Memorex. For intercabling of six to eight feet between the interface card and the drive, twisted pairs will be adequate and virtually any connector will suffice. Each twisted pair should consist of a signal wire paired with a ground wire and have one or two twists per inch. If you have a high

density application you should consult the manufacturer of the drive for recommended cabling, since comparatively high speed signals are involved.

Error Conditions

Three fundamental types of errors should be allowed for in the system:

- Desired data not accessible due to improper recording, dirt, or noise during reading.
- Wrong data accessed due to wrong track or sector.
- FILE UNSAFE error flagged by disk drive.

All three types of errors must be discovered by the software in order to establish reliable operation.

The FILE UNSAFE error is probably the most common type encountered. It typically results from attempting to write to a write protected diskette or attempting to write while the door of the drive is open. Malfunctions of the electronics within the disk drive will also cause a FILE UNSAFE error. These are quite rare, at least in Memorex drives. Since this type of error is typically due to the operator, Memorex recommends that manual intervention be required to reset the error condition.

Accessing wrong data because of being on the wrong track or sector will occur if the step rate of the pulses formed by the stepping software is too fast for the stepper motor or if electrical noise causes the sector counters to miscount. This type of error can only be detected during a read operation, and only if the sectors being read are formatted with information describing their intended physical location on the diskette. This is the purpose of the track sector identification (TSID) word included with each record written to the disk. When the record is read, the track sector identification is checked to see if it agrees with what was supposed to have been read. If it does not, an error has occurred. In practice this type of error should never occur unless there is some serious malfunction due to faulty design, improper operation, or bad software. In my prototype software, all operations halt when this type of error occurs and the operator is required to restart it.

The first of the three errors listed above is supposedly the most common. In my experience, though, it is quite rare *if* the drives and diskettes are kept clean and operated properly. The error is detectable by virtue of the checksum, a binary sum of all bytes of recorded data which is recorded at the end of the data field in the record. When the operating software reads a sector, it computes its own checksum and compares it to the recorded checksum. If the two sums disagree, an error has occurred.

Improvements

Since one step has already been taken in making the interface useful in an interrupt driven system, it might be sensible to make the interface more completely interrupt driven. In other words, eliminate the requirement that the processor wait during the rotational latency. This could be accomplished by simply adding latches and comparators to the sector counters' circuitry such that the processor could issue a desired sector number and a read or write command, then go about its business while the comparators search for the proper sector and initialize the read or write operation. This feature would probably only increase the chip count slightly and might be a worthwhile addition to units used with sophisticated disk operating systems.

The design I've discussed has been furnished to Midwest Scientific Instruments Inc for manufacture under license. The addressing structure of the Midwest Scientific version differs from that shown in the schematic of figures 1 and 2 in that they added peripheral interface adapter (PIA) chips, through which all communication between the processor and the interface must pass. This does not change the fundamental operation of the system, but it does increase the chip count. Budget minded users with the ability to build their own hardware are advised to build their own system, since it is basically not very complicated.

As an aid to persons building their own systems, the complete software, excerpts of which are used in this article, consisting of a 1 K byte microdisk operating system, is available for $10 in source form for the 6800 processor. Send to David M Allen, 1317 Central Av, Kansas City KS 66102. This software is written to utilize the routines of MIKBUG™ and provides named files of assignable length. Persons interested in hardware and software systems based on the interface discussed here should contact Midwest Scientific Instruments Inc, 220 W Cedar, Olathe KS 66061, (913) 764-3273.■