Build This Economy Floppy Disk Interface

Dr Kenneth B Welles
General Electric, Nela Park
2623 Fenwick Rd
University Heights OH 44118

The floppy disk drive offers the advanced computer hobbyist tremendous potential for a high performance computer system. With one or more floppy disk drives, an interface, and the proper operating software, the hobbyist can store hundreds of different programs on a single disk. Each of the programs can be given a name such as STARTREK, BASIC or EDIT, and a program can be run simply by typing its name, for instance "RUN EDIT". With this interface, the program can be brought into the computer at speeds of up to 31,250 bytes per second (for programs less than 5000 bytes long in the proper format). Each disk will store over 300,000 bytes of programs, computer music, Dazzler graphics, ASCII text, synthesized speech thesaurus or data of any form, and any data on the disk can be accessed in at most one second, typically in less than one quarter second. In fact, the draft of this article was written and edited using mass storage on a disk drive in my personal home computer system. The entire article takes up less than seven percent of one floppy disk, and the time saved in the retyping of successive revisions of the article was tremendous. *[Groan! Do I wish I had a floppy disk, CRT display, HYPERTEXT software and input scanners in my office . . .* CH]

Floppy disks also allow the quick assembly of large programs, without having to start, stop and rewind cassette players. Proper software allows a single floppy disk drive to merge several data files into one ordered file (for the updating of mailing lists or financial records), an operation which would take several cassette recorders on a cassette based operating system.

All of the features mentioned are the potentials of a floppy disk computer system. For a personal computing user to realize these potentials, he or she needs both

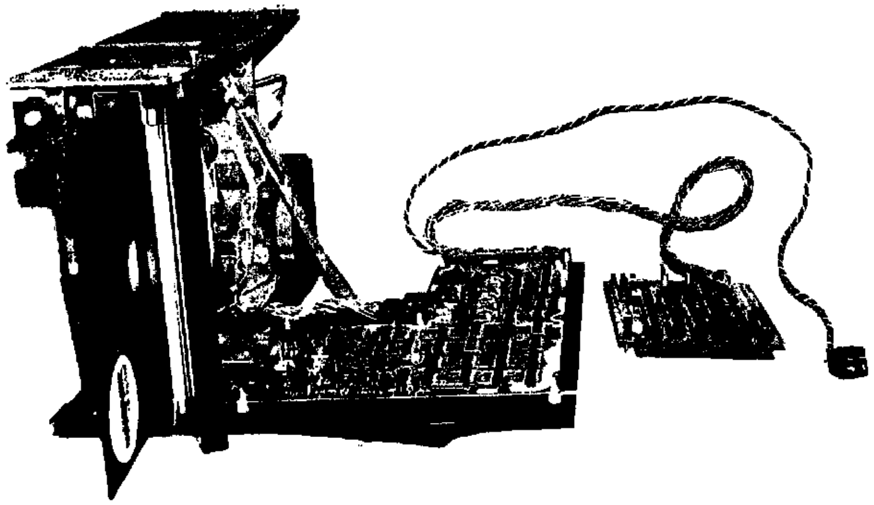


Photo 1: The author's disk drive and interface board shown removed from the system. The Innovex drive is at left, with a diskette partially inserted in the front door and the electronics board for the drive shown in an "open" position. The interface board is at the end of a multiconductor twisted pair cable, and a separate cable is used for drive power.

About the Author

Dr Welles is an enthusiastic personal computing user, with a fairly well developed system. At the time he wrote the current article, his system included an Altair processor, 14 K programmable memory, 5 K of 2708 ROM, and 2 Innovex floppy drives interfaced to the system. Miscellaneous peripherals include homebrew versions of a paper tape reader, television display, a modified office Selectric typewriter output, vector graphics, television camera input, and TV dazzler outputs among others. His main interests are image processing, pattern recognition, computer graphics and robotics. The entire text of his draft was typed and edited on his system, with hard copy output printed on the Selectric as the draft text submitted to BYTE. [At some yet to be determined future date, we'll eliminate the paper step and have authors such as Dr Welles simply send an appropriate machine readable representation of their articles . . . CH]

hardware and software. This article covers a hardware interface for floppy disk drive units.

Until recently, only the well financed hobbyist could afford a floppy disk drive for a personal system. In addition to the $650 to $1000 cost of the drive unit, one was also forced to spend from $300 to $1500 for a floppy disk drive controller. The high price of the controller buys a very intelligent electronic device, however. A single command from the computer causes the controller to seek a particular track on one of up to four disk drive units, load the head, find the desired sector, format and read or write the data, calculate the CRC (Cyclic Redundancy Check), determine if the transfer had been successful, and retry the transfer in the event of a read or write error. The design of such an intelligent controller is based on the old school, IBM/360 approach that processor time is too valuable to waste doing the housekeeping for a peripheral device. A personal computing user, on the other hand, has lots of processor time, limited funds, and consequently a different philosophy. One of the original reasons for the development of microprocessors was to perform in software all of those functions that would normally (and expensively) have to be designed in hardware. In this vein, in collaboration with W R Hemsath of Cornell University, I have designed and built a floppy disk drive interface which incorporates minimal hardware, and yet does not sacrifice the flexibility needed to read and write various data formats. This interface consists of only 17 integrated circuits, only one of which is a special purpose chip. The total cost of the chips is less than $25. The design shown here will interface up to eight floppy disk drives to an 8080 processor. In order to properly describe the design and function of the interface, let us first review briefly what steps are required to transfer data to or from a floppy disk.

Disk Drive Operation

In operation, a disk is inserted into the drive and the access door is closed. The act of closing the door engages the disk onto the spindle, and the disk is then rotated at 360 RPM. A stepper motor drives the magnetic data transfer head radially in and out to 77 discrete positions, the outermost called track 0 and the one nearest the center of the disk called track 76. Normally, the head does not touch the spinning disk, but is positioned a small distance away from it. When data is to be read or written, a modified relay is energized allowing a spring loaded pressure pad to press the flexible disk into contact with the head. Timing holes punched in the floppy disk pass by a photo detector and generate a series of pulses. These "sector pulses" are used to determine which one of 32 segments or sectors of the disk is currently passing the head. Use of such holes to define sectors is called "hard sectoring" in disk drive jargon. The pulses are used to signal the approximate starting point of each sector. Data is read from and written to the disk in a manner quite similar to the reading and writing of data on magnetic cassettes. In normal operation, each of these 32 sectors will store slightly over 1024 data bits, or 128 bytes. To write data onto a particular track and sector of the disk, the following operations must take place:

1. The head is moved in or out to the desired track.
2. The pressure pad is loaded, pressing the disk against the head.

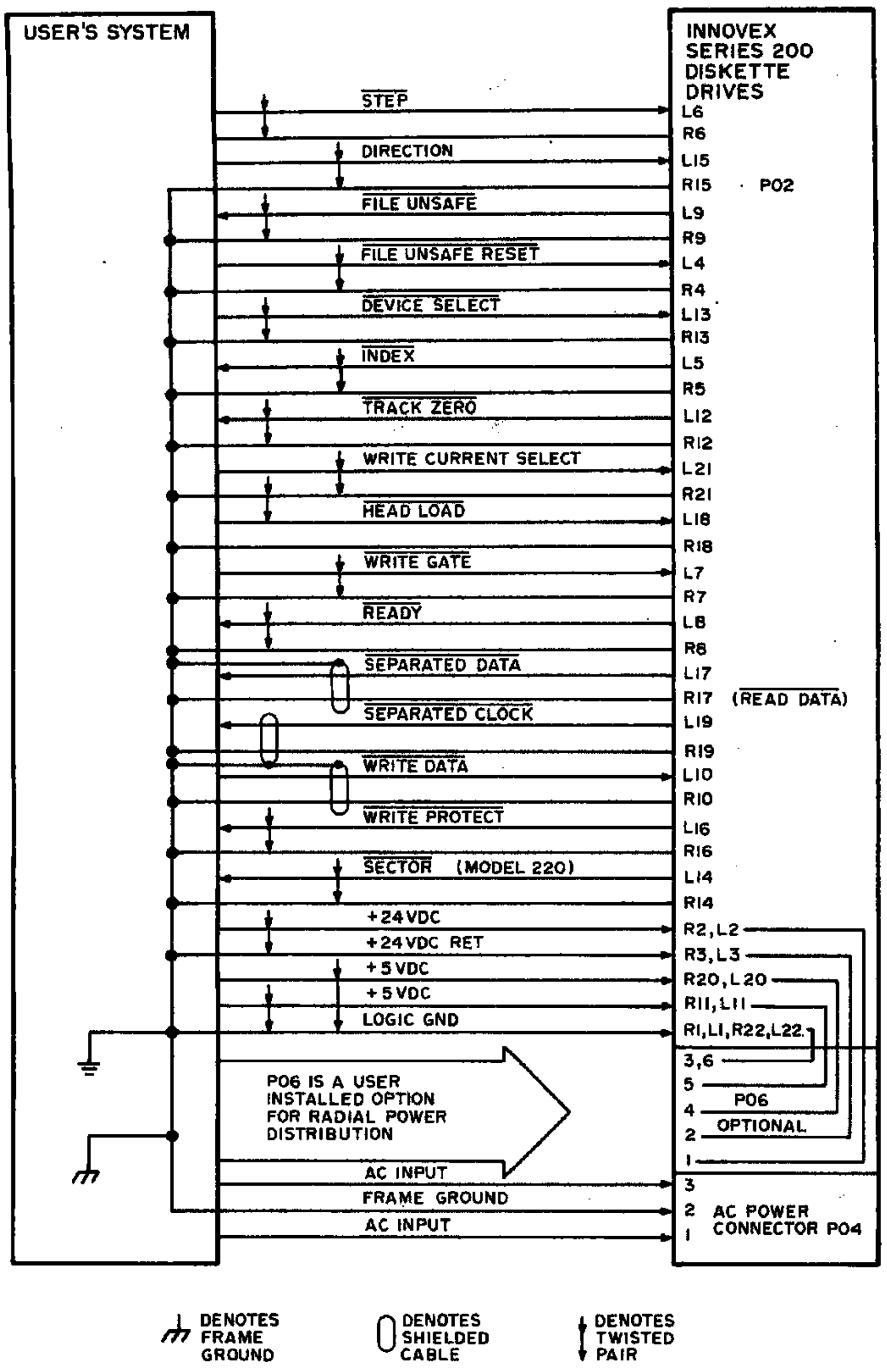


Figure 1: This diagram, redrawn from the Innovex Series 200-M Maintenance Manual, *shows all of the TTL level signal lines that must be passed between the disk drive and the controlling interface.*

The signals sent to the drive from the interface are:

Device Select: When this line is high, all commands from the interface are ignored by the drive, and all signals from this drive unit are put into a high impedance state. If several drives are used, all of the input and output signals may be tied together on a common bus with the exception of the device select lines. By pulling only one of the several device select lines low, the interface selects that particular disk drive to send commands to and receive data from.

Step: A low going pulse on this line causes the head positioning motor to move the data transfer head in or out one track.

Direction: During a step pulse, if this line is high then the head moves out one track (towards track 0). If this line is low, then the head will move in one track.

Head Load: When this line is low, the pressure pad brings the spinning disk in contact with the data transfer head.

Write Current Select: Because the surface velocity of the disk relative to the head varies from the outermost to the innermost track, the density of the data on the disk will also vary. To compensate for this variation, the write current select line varies the amount of current used to write data as a function of the track being written. This line must be low when writing data onto tracks 0 to 43, and high for tracks 44 to 76.

Write Gate: Pulling this line low enables the data on the write data line to be sent to the head and recorded onto the disk.

Write Data: Data to be written on the disk must be serialized and sent out on the write data line as a series of low going clock pulses (one pulse every 4 μs) separating the presence (a 1 data bit) or absence (a 0 data bit) of a low going data pulse. Figure 2 shows the write data signal used to send the data bit string 10100.

File Unsafe Reset: This line is pulsed low just before a write operation is to take place. The pulse resets the file unsafe status to a safe (write enabled) condition, thereby allowing the write operation to be performed.

The signals sent to the interface by the disk drive are:

File Unsafe: A low signal on this line indicates that an error condition existed when a write operation was attempted. When file unsafe goes low, no writing can be done on the disk, preventing the loss of previously written data due to some error condition.

Track Zero: When the data transfer head is positioned at track 0, this line goes low, enabling the computer to calibrate the head position. When the head is at tracks 1 to 76, this line is high.

Index: A 500 μs low going pulse appears on this line to signify that the index hole has just come into position under the photodetector. This pulse is used by the computer to determine which sector is sector 0.

Sector: A 500 μs low going pulse appears on this line each time a sector hole (not an index hole) passes under the photodetector. 32 pulses occur every revolution, and these pulses are used to determine the approximate starting positions of the various data sectors.

Ready: When AC and logic power are present at the disk drive and a disk is loaded, the ready line goes low.

Separated Clock: When previously written data is being read from the disk, the clock is recovered from the data stream, and is presented on this line as a series of 200 ns low going pulses. The recovered clock pulses come approximately every 4 μs with variations due to the changes in drive motor speed.

Separated Data: The serial data coming from the disk during a read is indicated by the presence (a 1 data bit) or absence (a 0 data bit) of a 200 ns low going pulse on the separated data line, between adjacent separated clock pulses.

Write Protect is an optional signal that is not used in this interface. On a disk drive with this option added, the user can write protect the data on a disk by punching out or uncovering a write protect hole in the disk jacket. A write protected disk cannot be written onto.

3. Sufficient settling time is allowed for the head movement and pressure pad loading to fully stabilize.
4. Delay until the start of the sector pulse which corresponds to the desired sector.
5. Turn on the WRITE GATE of the disk drive to allow data to be written.
6. Write 64 0 bits (16 bytes of 0).
7. Write a single synchronizing byte (sync byte).
8. Write the desired data bytes.
9. Write 64 0 bits.
10. Turn off the WRITE GATE to prevent any more data from being written.
11. Unload the pressure pad.

Because the disk drive records data serially, steps 7 and 8 require that each byte being written must be sent out as a series of 8 bits, with one bit being sent out every 4 μs, and with no skipped bits between bytes.

Reading data from the disk requires a similar series of operations:

1. The head is moved to the desired track.
2. The pressure pad is loaded.
3. Settling time is allowed for movement and loading.
4. Wait for the start of the sector pulse corresponding to the desired sector.
5. Search for the first occurrence of the sync byte.
6. Read in the desired data.
7. Unload the pressure pad.

Searching for the sync byte entails shifting the incoming serial data into a 8 bit byte and comparing the result of each shift with what the sync byte should be, every time that a new bit is read (every 4 μs). When a match is found, then the data bit stream that follows is broken into bytes on every eighth bit, using the sync byte boundary to define the data byte boundaries that come after the sync byte.

From the proceeding lists of read and write procedures, two things become apparent: First, the speed required for shifting data in and out (1 bit every 4 μs) is too fast for most microprocessors to handle under software control (and searching for the sync byte is more time consuming still!). Second, all of the other operations (stepping the head from track to track, loading the head, searching for the proper sector pulse and turning the write gate on and off) are easily within the capabilities of microprocessor software control. Therefore a minimum hardware interface should control all of the functions which are not time-critical, through software and a simple input and latched output port. The remaining functions then determine the major portion of the design.

The disk drive we used for this interface is an Innovex 220 hard sectored flexible disk drive, and the signal lines required to operate the drive are typical of most floppy disk drives. There are 15 standard TTL level signals required to operate the model 220 drive, 8 from the interface to the drive, and 7 from the drive to the interface. The signal names and functions for the interface are summarized in figure 1.

Figures 3 and 4 show the circuitry of the floppy disk interface. The circuit has 6 major sections: processor IO instruction decode, instruction latch to disk drive, status load from disk drive, head load-unload, USRT transmit, and USRT receive.

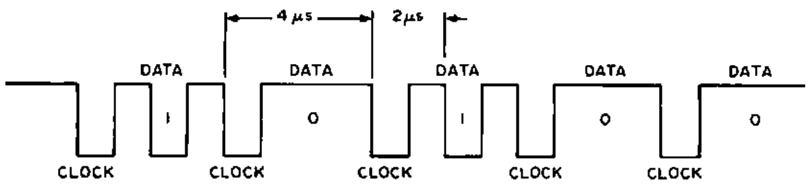


Figure 2: The timing of data cells on the disk. Each bit cell is framed by a clock pulse on either side. If the data is 1, a pulse appears in the middle of the 4 μs cell width; if the data is 0, no pulse appears in the middle of the cell. The waveform in this example has 5 cells with the pattern of data needed for the string 10100.

Table 1: Semantics of the OUT 243 instruction. This table lists each accumulator bit, along with its meaning when used to transfer data to the disk interface in the OUT 243 instruction of an 8080. (In a different wiring of the IO instruction decoder, or in a different computer, the same format could be used for the actual data transfer.)

OUT 243 INSTRUCTION

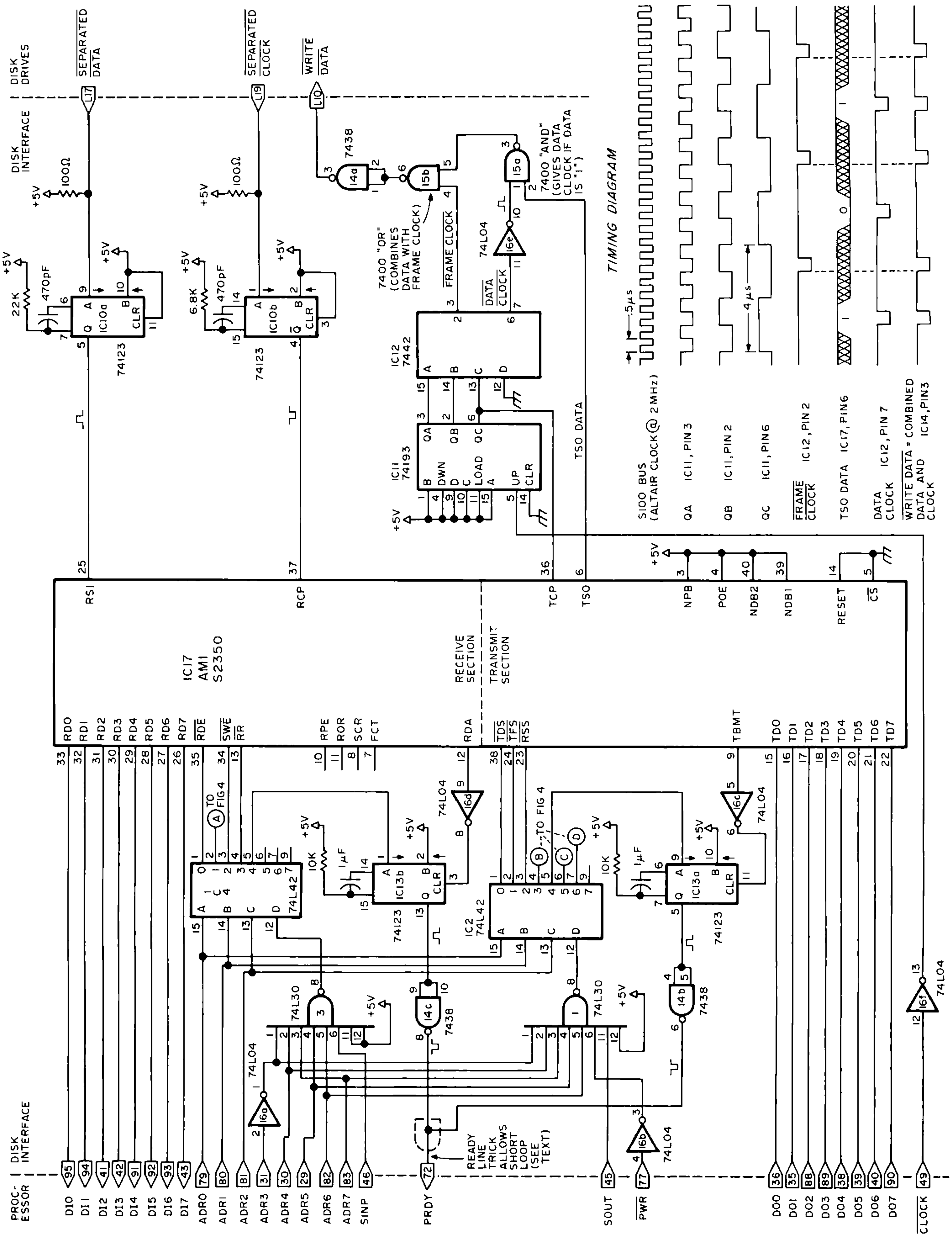
Bit	Signal Name	Polarity in Accumulator
0	Write Current Select	1 for tracks 0 to 43, 0 for track 44 to 76
1	File Unsafe Reset	0 to 1 to 0 transition causes reset
2	Direction	1 for step in, 0 for step out
3	Write Gate	1 enables the drive to write
4	Step Track	0 to 1 to 0 transition steps one track
5,6,7	Drive Select	000 selects drive 0, 111 for drive 7

Table 2: Semantics of the IN 241 instruction. This table lists the status bits read by the IN 241 instruction of an 8080 using this interface.

IN 241 INSTRUCTION

Bit	Signal Name	Polarity in Accumulator
0	Track Zero	0 means the head is at track 0
1	File Unsafe	0 means file unsafe condition exists
2	Ready	0 means disk drive is ready
3	Sector Hole	1 to 0 transition marks start of each sector
4	Index Hole	0 means that the next sector is sector 0
5	Head Loaded	1 means that the head is still loaded
6,7	Unused	Always 1

Figure 3: This diagram shows the major portion of the disk drive interface. IC1 and IC2 form the output command decoder. IC3 and IC4 form the input command decoder. IC10 sets up the data from the disk into a format acceptable to the USRT. IC11 and IC12 put the data from the USRT into the proper format for the disk drive. A list of all integrated circuits with power connections is found in table 3.



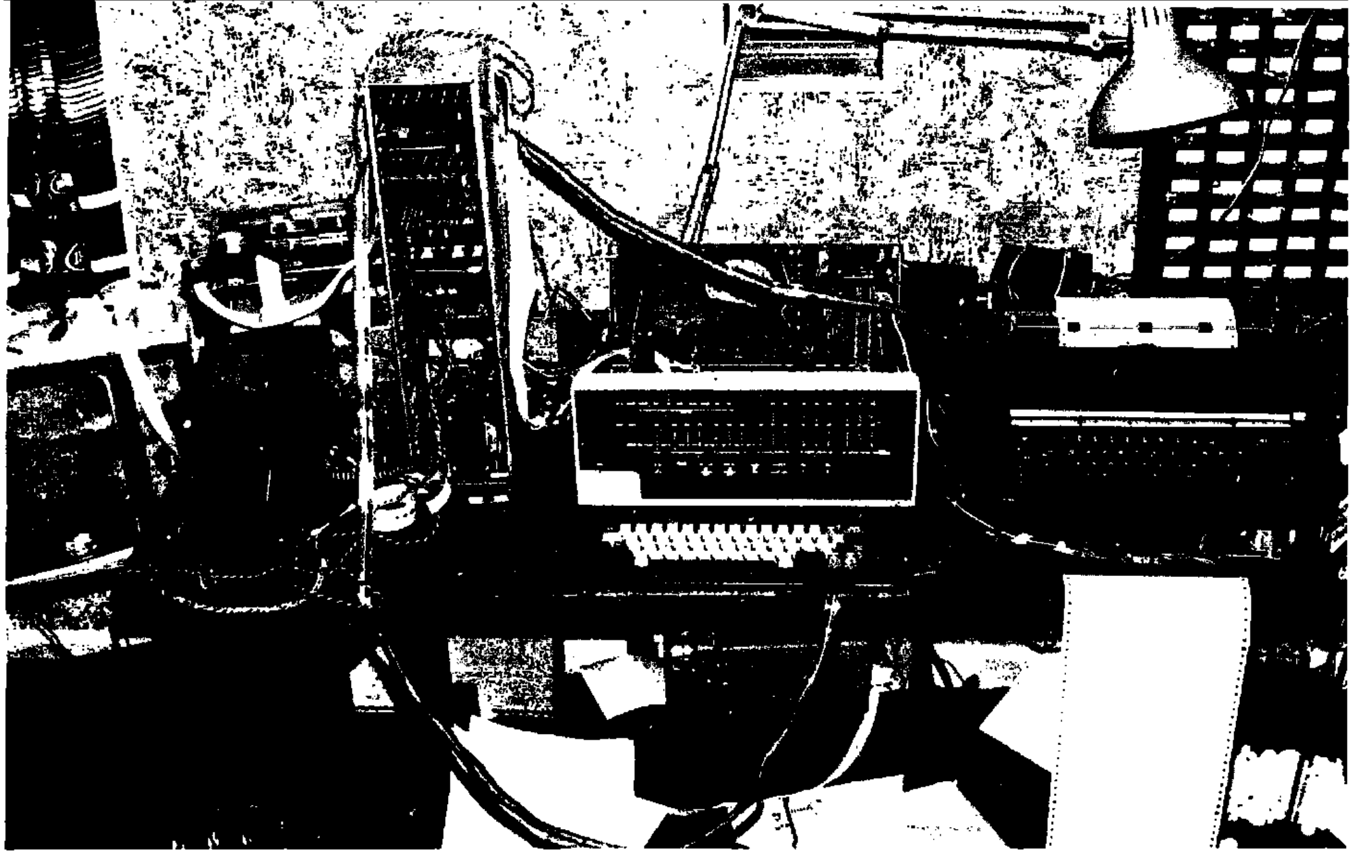


Photo 2: The author's system. The processor is an Altair, and other peripherals include a homebrew Selectric typewriter interface . . . .

Processor IO Instruction Decode

IC1 and IC2 decode output instructions to the interface. Executing the 8080 instructions OUT 240, OUT 241, . . . OUT 247 (240 to 247 decimal) cause 500 ns low pulses on the output lines 0 to 7 of IC2. These pulses can be used to latch data from the output data bus lines DO0 to DO7 into various registers, or to trigger specific functions (as will be shown later).

IC3 and IC4 form the input instruction decoder for the instructions IN 240 to IN 247 in a similar manner to the output decoder. The pulses on the output lines of IC4 are used to gate data onto the input data bus lines DI0 to DI7 and into the accumulator. Again, the pulses may be used to trigger specific functions that are not data input operations. [*In adapting this design to a non 8080 based computer, this decoding logic would have to be modified . . . .* CH]

Table 3: Integrated circuit power wiring list. This table lists each integrated circuit in the floppy disk interface, along with its power wiring pins.

Number	Type	+5 V	GND
IC1	74L30	14	7
IC2	74L42	16	8
IC3	74L30	14	7
IC4	74L42	14	7
IC5	74LS175	16	8
IC6	74LS175	16	8
IC7	7442	16	8
IC8	8097	16	8
IC9	74123	16	8
IC10	74123	16	8
IC11	74193	16	8
IC12	7442	16	8
IC13	74123	16	8
IC14	7438	14	7
IC15	7400	14	7
IC16	74L04	14	7
IC17	S2350	2	1

Note: 74LXX and 74LSXX types may be replaced by 74XX; 8097 may be replaced by 8T97.

Instruction Latch to Disk Drive

Execution of an OUT 243 causes the contents of the 8080's accumulator to be loaded into IC5 and IC6. The 5 least significant bits are used to send the low speed control signals to the disk drive. Table 1 shows the allocation and the polarity of these bits as they appear in the accumulator. The three most significant bits are used by IC7 to select one of up to eight different drives which may be attached to each interface.

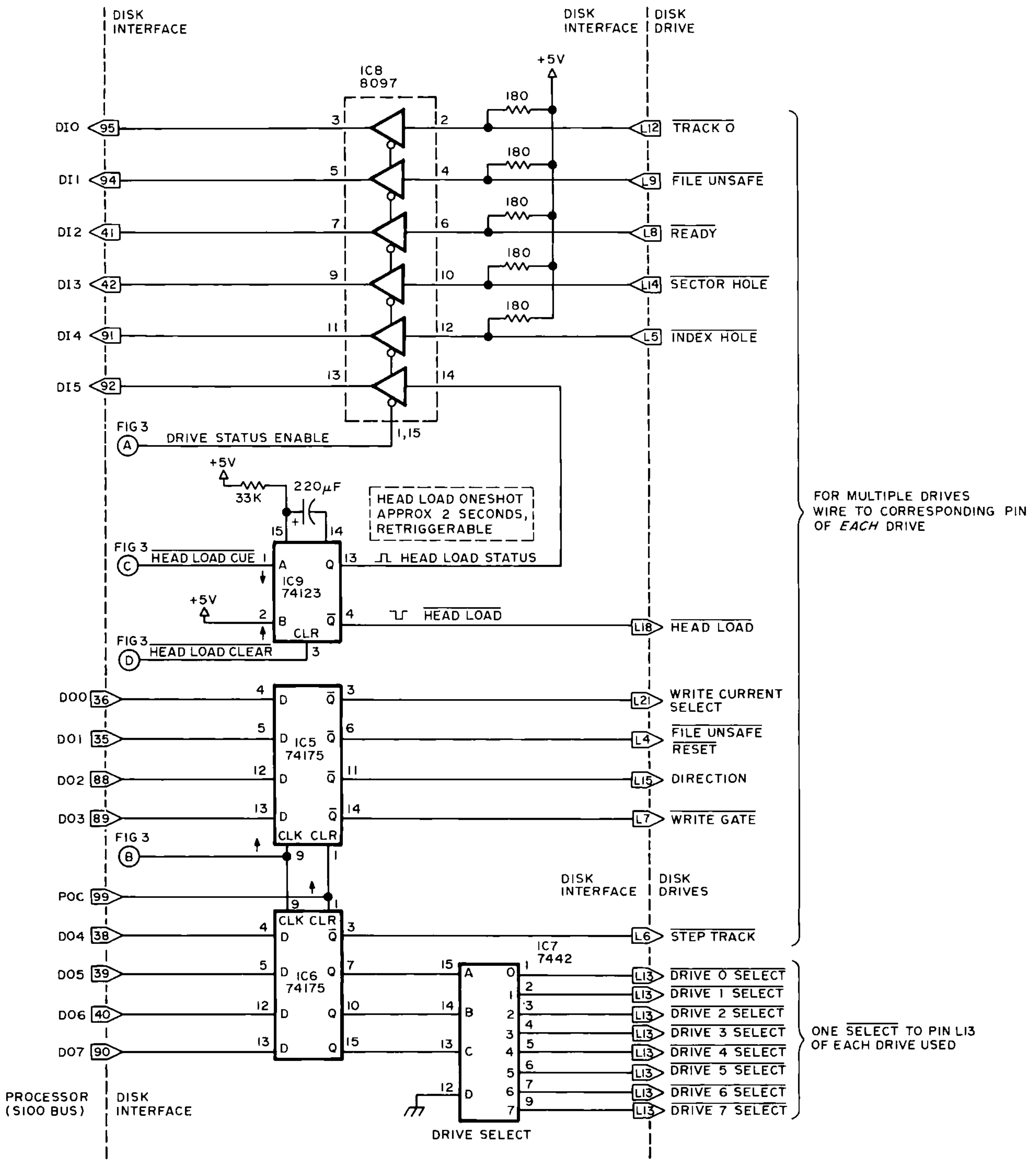
Status Load from Disk Drive

Execution of an IN 241 instruction enables IC8 to load the current status of the selected disk drive onto processor input data lines DI0 to DI5. Table 2 shows the allocation and polarity of these bits as they are loaded in the accumulator. The two most significant bits are unused, and will always show 1s.

Head Load-Unload

IC9 is a retriggerable one shot with a 2 second pulse width. Executing an OUT 245 instruction initiates this pulse and loads the disk drive head, regardless of the contents of the accumulator. If another OUT 245 instruction is executed within 2 seconds of the first OUT 245, then the head will remain loaded for a further 2 seconds. The head will unload 2 seconds after the last OUT 245

Figure 4: This diagram shows the circuitry used to perform all of the low speed functions of the disk drive. IC8 is a 6 bit input port, and IC5 and IC6 are an 8 bit latched output port. IC7 selects one of up to 8 disk drives on the system, and IC9 controls the loading of the disk's data transfer head for a read or write operation.



(load head) instruction. This 2 second pause allows the head to stay loaded during successive reads and writes to the disk, but will automatically unload the head after 2 seconds without any disk activity. Alternatively, an OUT 246 instruction will cause the head to be unloaded immediately if and when that is desired. This automatic head unload feature minimizes wear on the floppy disk. If it were not present in some hardware or software form, the head would be continuously in contact, wearing out disks quite quickly if your machine ran 24 hours a day.

The USRT

The abbreviation USRT stands for Universal Synchronous Receiver Transmitter; this chip really is quite universal. Although it was originally developed for data transmission over phone link, wire link, and some types of tape drive, the S2350 USRT performs all of the needed high speed data transfers to and from the disk with almost no modification. Before discussing the operation of the USRT transmit and receive sections of the interface as a whole, take a look at the functions of the USRT itself, as

TDS Transmit Data Strobe

An OUT 240 instruction of this interface puts a pulse on the TDS line which loads the accumulator into the USRT transmitter buffer through processor data output lines TD0 to TD7. The USRT then shifts this data byte out onto TSO (Transmit Serial Out). One bit is shifted onto TSO for each pulse on TCP (Transmit Clock Pulse).

TBMT Transmit Buffer Empty

Whenever the transmitter buffer is ready to receive another byte (from an OUT 240 instruction), the TBMT line goes high.

TFS Transmit Fill Strobe

An OUT 241 puts a pulse on the TFS line which loads the accumulator into the USRT fill buffer. If new data is not sent to the transmit data buffer by an OUT 240 soon after a TBMT signal, then the USRT has no data to send out on the TSO line. In this case, data from the transmit fill buffer is sent out in place of the missing data.

RSS Receiver Sync Byte Strobe

An OUT 242 pulses the RSS line which loads the accumulator into the USRT sync byte buffer, for use at the beginning of a data read operation.

RR Receiver Reset

An IN 243 causes the receiver section of the USRT to be reset into the "Search for Sync Byte" mode. The received serial data stream enters on RSI (Receive Serial Input), and is clocked into the received data buffer by the RCP (Receive Clock Pulse) line. When the data byte in the received data buffer matches the byte in the sync byte buffer, the RDA (Received Data Available) line goes high. After this happens, a new byte is put into the received data buffer after every eight clock pulses on RCP.

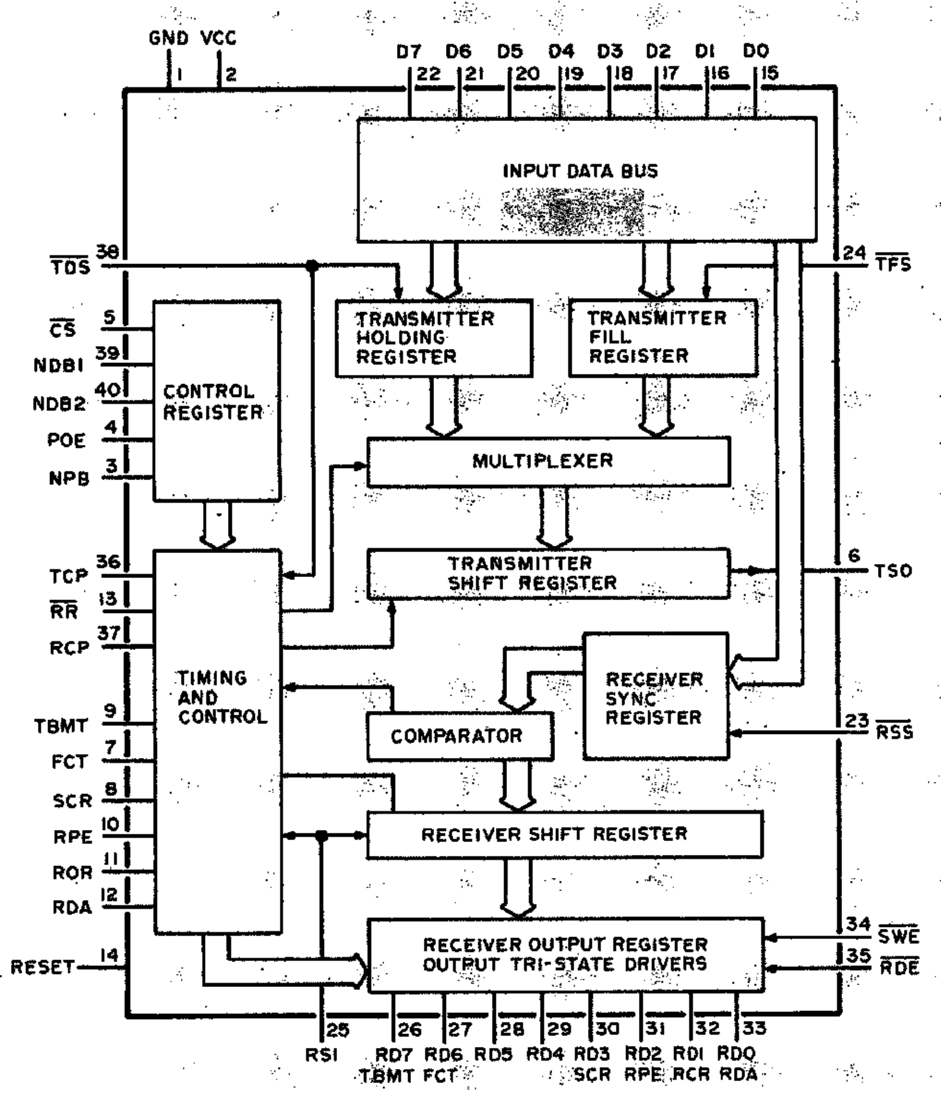
RDE Received Data Enable

An IN 240 instruction pulses the RDE line. This puts the data in the USRT received data buffer onto data lines RD0 to RD7, and it is loaded into the accumulator. In this manner, the 8080 brings in the data read from the disk.

SWE Status Word Enable

An IN 242 pulses the SWE line which loads the USRT status word into the accumulator to examine for data ready, or to find possible errors.

Figure 5: This is a block diagram of the USRT integrated circuit, the AMI S2350. The information here is redrawn from the original contained in AMI's data sheet on the device. The USRT integrated circuit is the heart of this inexpensive floppy disk interface, performing all of the high speed data manipulations needed to read and write data from and to the disk drive. The USRT was not intended to be used as a floppy disk interface when it was originally designed. But as demonstrated by this article, a little ingenuity can often come up with surprisingly versatile applications of standard integrated circuits for use in high speed data communications.



denoted by the various signal lines. Figure 5 shows a block diagram of the S2350, along with captions detailing these lines and their relation to the interface as a whole.

USRT Transmit

After the disk drive head has been loaded and the desired track and sector found, the write gate is turned on and data from the processor may be sent to the transmit section of the USRT through an OUT 240 instruction. IC11 divides the Altair 2 MHz clock by 8 to give the 250 kHz clock required by the disk drive. This clock is fed into TCP, and IC12 combines the data from the transmitter serial output line and another clock phase into the proper write data format required by the disk drive as seen in figure 2.

USRT Receive

IC10 is simply used as a pulse stretcher for the separated data and separated clock from the disk drive. The data pulse is expanded to overlap the falling edge of the clock pulse. This overlap allows the data to be read properly by the USRT. When a byte of data has been received (as denoted by the receiver data available line), an IN 240 instruction will load the received data into the accumulator.

Software Timing

The article to this point has shown how data can be transferred between the processor and the disk drive in the correct format, but nothing has been said about the ability of the 8080 to send or receive data at the proper rate. A 250 kHz bit rate is one byte of data in or out every 32 μs under ideal conditions. If the drive motor speed variations are taken into account, this figure can be as low as 30 μs per byte on a read operation. Since 8080 instructions take from 2 to 7 μs to execute (assuming a 2 MHz clock and fast memory), this restricts the read loop to very few instructions. If it is desired to transfer more than 256 bytes in or out at any one time, the read loop might look like:

	Symbolic Instruction	Execution Time
LOOPA:	IN STATUS	5.0 μs
	ANI DATAREADY	3.5 μs
	JZ LOOPA	5.0 μs
	IN DATA	5.0 μs
	MOV M,A	3.5 μs
	INX H	2.5 μs
	DCX B	2.5 μs
	MOV A,B	2.5 μs
	ORA C	2.0 μs
	JNZ LOOPA	5.0 μs
		36.5 μs

In the above example the HL register is used to point to the data buffer, and the BC register is the number of bytes to be read. The total time of the loop, 36.5 μs, is 6.5 μs too long for the worst case data read. Obviously this program will not read data in properly.

By eliminating two lines of code the loop is reduced to a total time of 28 μs as shown in the following example. This is quite ample for the interface and allows additional leeway for the possibility of dynamic memory's introducing a wait state during the loop.

	Symbolic Instruction	Execution Time
LOOPB:	IN DATAWAIT (IN 244)	5.0 μs
	IN DATA (IN 240)	5.0 μs
	MOV M,A	3.5 μs
	INX H	2.5 μs
	DCX B	2.5 μs
	MOV A,B	2.5 μs
	ORA C	2.0 μs
	JNZ LOOPB	5.0 μs
		28.0 μs

Obviously this version of the routine will not work without some special "trick." In this case, the trick is that the first three lines of LOOPA have been replaced with the first line of LOOPB and some special hardware. The first three lines of LOOPA prevented the IN DATA statement from reading data before data was available. In LOOPB, the IN DATAWAIT is an IN 244 instruction. This triggers IC13b, a one shot, which puts the 8080 into a slow memory wait state by pulling the Altair's PRDY line low. When data is ready for input, the RDA line of the USRT resets IC13b and allows the LOOPB routine to continue. During normal execution of a read operation, the 8080 does a 4 μs wait between lines 1 and 2 of LOOPB. This wait state serves to synchronize the reading of the disk data with its availability. Any amount of data from a partial segment to an entire track may be input with this routine.

If some hardware failure should occur, and data stops coming into the USRT, then RDA will never go high. If no data arrives after 3 ms, then IC13b completes the one shot cycle and releases the 8080 wait state. This feature prevents a hardware failure in the disk drive or interface from hanging the

A printed circuit board is available for the advanced hobbyist to construct his or her own interface. The printed circuit board fits into a single Altair (or generic equivalent) slot, and supports the circuit described in this article with two additions:

1. Eight head load circuits allow multiple drives to load heads simultaneously.
2. Space is provided for a 1702 type PROM, to allow the user to load the operating system from the disk without toggling in any data.

The printed circuit and documentation only (no ICs or sockets) are available for $35 from K B Welles, 2623 Fenwick Rd, University Heights OH 44118.

processor up in an endless wait state. Whether a read operation is successful or not, the end of the loop is reached when the BC register pair's count is decremented to zero and the JNZ condition no longer pertains.

In order to write data, a software output loop similar to LOOPB is employed:

```
         Symbolic                 Execution
        Instruction                  Time

LOOPC:  OUT DATAWAIT
          (OUT 244)                 5.0 μs
        MOV A,M                     3.5 μs
        OUT DATA
          (OUT 240)                 5.0 μs
        INX H                       2.5 μs
        DCX B                       2.5 μs
        MOV A,B                     2.5 μs
        ORA C                       2.0 μs
        JNZ LOOPC                   5.0 μs
                                  ________
                                   28.0 μs
```

With this output loop, the 8080 can maintain the data rate required to transmit data to the disk properly. A similar hardware synchronization trick is also used in this case.

Final Hardware Notes

The circuit shown in figures 3 and 4 was developed for use with an Innovex 220 drive. The 220 has multiple options which can be selected by jumpers on the circuit board. The options required for use with this interface are:

1. Radial Interrupt Disabled (Link E installed)
2. Radial Rotation Sensing Disabled (Two Link Es installed)
3. Read Data Option Disabled (Link A installed)
4. Write Protect Option Disabled (Link H installed)
5. Stepper Power Option (Link E installed)
6. Radial Head Load Disabled (Link E installed)

The selected options allow multiple drives to be used with the interface. While up to eight disk drives can be connected in parallel (with the exception of the device select lines), the shorting clip on the P07 line must be removed from all but the last disk drive on the bus (P07 connects the bus termination resistors to +5 V). In addition, the user must provide power supplies for the following voltages and currents:

+5 V, 800 mA for each drive
−5 V, 75 mA for each drive
+24 V +/−2 V, 1.4 A for the first drive, 0.1 A more for each additional drive

Conclusion

The small number of ICs in this circuit (17) and their low cost and easy availability puts the construction of this circuit within the abilities of many intermediate and advanced computer hobbyists and experimenters. The addition of a disk drive to the average home system will increase the overall system usefulness many times. By reducing the time required for software generation to a fraction of that on a cassette or paper tape system, software throughput and sophistication of the typical personal computing user (and professional) will typically double or triple.

I currently have two drives running on an Altair system, and a complete disk operating system existing in 2 K of PROM that allows operating with up to 240 different named files on each disk. Loading BASIC takes only 6 seconds, and loading STARTREK using CLOAD takes only 3 more seconds. The disk drive and operating system has increased software generation at least fourfold, and made the system much more enjoyable to use.■

BABY! I MICROCOMPUTER

ARE YOU BUYING A SYSTEM?

BEFORE YOU DO
SEND FOR OUR LITERATURE. COMPARE OUR SPECIFICATIONS WITH OUR COMPETITORS' KITS AND ASSEMBLED SYSTEMS!

YOU WILL SEE

THE BABY! I SYSTEM IS NOT A BABY WHEN IT COMES TO HARDWARE AND PERFORMANCE
BUT IT IS WHEN IT COMES TO PRICE!

COMPARE TODAY
THEN BUY A
BABY! I MICROCOMPUTER

STM SYSTEMS, INC.
P.O. BOX 248
MONT VERNON, N.H. 03057
603-673-2581