TRS-80 Model I Disk Interfacing Guide

William Barden, Jr

CONTENTS

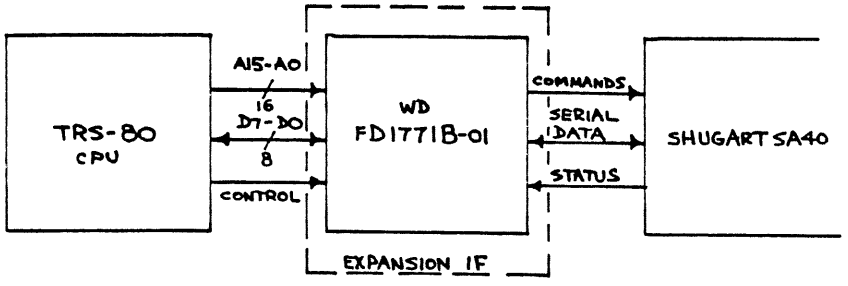
Chapter 1

Disk Basics

This text describes the operation of the Shugart SA400 Mini-floppy disk drive in the Radio Shack TRS-80 Model I Microcomputer system. It is divided into five chapters. The first chapter, Disk Basics, describes the general operation of minifloppy disks. Chapter 2, Shugart SA400 operation, describes the operation of the disk drive itself in terms of interface signals and functions. The next chapter is concerned with operation of the Western Digital FD1771B-01 Floppy Disk Formatter/Controller chip used in the TRS-80 Model I Expansion Interface. Chapter 4 shows how the expansion interface decodes disk addressing and commands. The last chapter shows how Radio Shack software communicates to the disk and how one may do machine language (assembly language) and limited Basic-level programming of disk systems. Appendices provide related material, such as controller commands and disk format.

A floppy disk system is made up of a disk drive or drives, a controller, and the microcomputer. In our case, the microcomputer is the Radio Shack TRS-80 Model I, the controller is the Western Digital FD1771B-01, and the disk drive is the Shugart SA400. A block diagram of the TRS-80 disk system is shown in Figure 1. As with other units in the TRS-80 system, the CPU communicates over 16 address lines, A15 through A0, eight data lines, D7 through D0, and a set of control lines that specify whether reading or writing and other functions are being performed. The controller for the disk(s) interprets commands sent to it over the data lines and translates these commands into disk-type commands that the Shugart SA400 can recognize. The single chip controller is a 40-pin chip that is effectively a microcomputer in itself, and replaces a hundred chips or so for a TTL (Transistor-Transistor Logic) design. Commands are sent to the disk drive by the controller chip to perform functions for head positioning and reading and writing, and the "status" of the drive is returned to the controller chip. We will be talking about the operation of each of these component parts in future chapters, but for the time being, let us concern ourselves with how the data is stored on the "diskette" and some of the physical attributes of the diskette and disk drive.

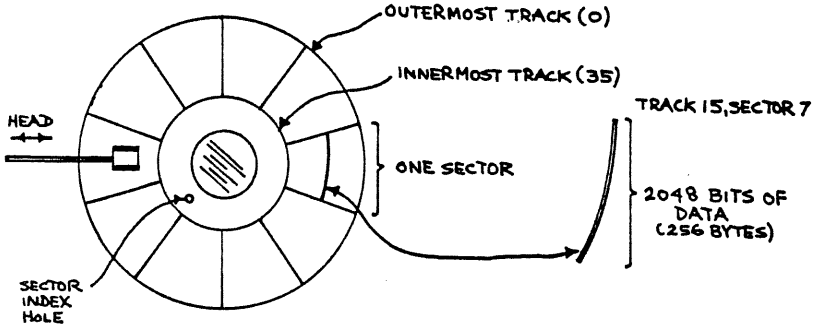
Figure 1. Block Diagram of the TRS-80 Disk System



When the disk drive is mentioned in this text, we will refer to the "disk" or "drive" or "disk drive". When the recording media is mentioned, the term "diskette" will be used. The diskette used in the Shugart SA400 is basically a 5¼ inch diameter flexible or "floppy diskette made up of plastic, coated with a magnetic oxide similar to that used for recording tapes. The diskette fits in a square holder for protection and ease of storage. The square holder fits inside the SA400, which is really only a device which spins the diskette and moves a recording head along a radius while the diskette is spinning, along with associated electronics to read and write data. The recording head reads flux changes or produces flux changes for writing, similar to a tape recording head. Other disk electronics control head positioning, protection of the diskette from writes, and other functions.

The diskette spins at 300 revolutions per minute. As the diskette revolves, the head can be moved along a radius towards the center or back again in small increments. Each discrete position over the diskette defines a "track", as shown in Figure 2. There are 35 tracks for a Shugart SA400, and therefore 35 valid positions along the radius. When the head is positioned along the radius over a track it can read the data along the concentric circle defined by the track. This circle is divided into ten "sectors", each occupying one-tenth of the circumference of the track. The circumference of the innermost circles or tracks are obviously less than the outermost tracks, but the content of the tracks is the same, although the data is packed a little more tightly into the innermost tracks. This may cause possible "data separation" problems when 40 track formatting is attempted.

Figure 2. The Organization of the Diskette



Once positioned over a track, sector 0 can be sensed by a small sector index hole in the diskette which causes a signal to be generated by the diskette when the index hole passes five times per second.

Each sector on the diskette holds 256 *data* bytes. The entire track can hold 256 times 20 bytes, or 2560 bytes of data. As there are 35 tracks, the entire diskette can hold 89,600 bytes of data. This data is recorded in *serial* fashion along the track, so that one track holds 2560 times 8 bits of data, or 20,480 bits along the circumference. Data recorded along every track then, can be viewed as a long string of data bits, starting from sector 0 of the track and ending at data bit 20479, the last data bit of sector 9. In addition to the *data* stored in a sector "record" however, there are other bit patterns that are not user data. This data identifies the sector address, defines a "gap", stores a checksum of user data, and contains other relevant data pertaining to reading and writing the sector. This data must be put on the diskette by a special "formatting" process prior to user data being stored on the diskette. One can look at the formatting process as supplying a skeletal set of records with proper sector gaps and identification data, and large unused areas awaiting user data. The formatting process and actual track format is described later in this text.

If data is to be read or written to a sector, the head is first positioned over the proper track, 0 through 34. Information about where programs or data are to be found on the diskette must be maintained by the system user in a software "directory" that contains a file name and track and sector address, along with other particulars. Track positioning is called a "seek" operation, and takes about 25 milliseconds (1/40th of a second) to go from track to track, a little under a second to go from track

0 to track 34 (worst case), and about 450 milliseconds for the average seek that must traverse about one-half the number of sectors.

Once the head is positioned over the proper track, the desired sector can be read. Prior to sector read (or write) the program passes a sector address (0 through 9) to the disk controller, just as it had previously passed a track address before the seek. The disk controller senses when the proper sector spins under the head by detecting the index hole and identification data from the diskette. If the sector has *just* passed the head when the sector read or write command is given, then the time required to read or write the sector is about one revolution time plus one-tenth of a revolution time for read or write. As the diskette is spinning at 5 revolutions per second, this worst case "sector latency" will be about 220 milliseconds. The average *access* for reading or writing to a sector is about one-half revolution or one-tenth of a second. Once positioned over the proper sector, data is transferred at 2560 bytes per revolution, or about 12,800 bytes per second for *user* data. (The actual data transfer rate is closer to 15,625 bytes per second because both user data and identification data is being read.)

The "average access" for records dispersed all over the diskette would be the average seek time plus average sector time plus data time, or about 450 milliseconds plus 100 milliseconds plus 20 milliseconds equals 570 milliseconds per 256 byte sector, or roughly one-half second. The average access for data accessed "sequentially" in adjoining sectors and tracks would be about 100 milliseconds (basically average sector access time) for processing of 256 byte records. Quite a change from 500 baud (50 characters per second) cassette tape!

Another factor to consider in computing access times is motor turn-on time. When an Input/Output (I/O) operation is performed the disk drive motor must first be turned on and brought up to speed. This takes less than a second. If a series of I/O operations are to be performed, the motor is kept on by continuously "selecting" the drive, so that the one second turn-on time occurs only at the beginning of the set of operations. If the operations occur greater than about three seconds apart, however, the motor must be turned on for each set of operations.

Data is normally read and written one sector at a time, although it is possible to read one to ten sectors worth with one command. Checks are provided for valid data, positioning errors, and other "disk status" as in any complicated I/O device.

Each diskette square holder has a small notch cutout that can be covered over with a label or tape. When this is done the diskette is "write-protected" and data can be read from, but not written to, the diskette.

The TRS-80 permits up to four drives to be connected to the expansion interface box with one cable. These are numbered as "0", "1", "2", and "3". Drive 0 always contains a diskette with the TRS Disk Operating System (or TRSDOS) and utility program on the first 34K bytes or so of the diskette. The "default" drive is disk drive number 0 for commands that do not specify a drive number, and the "bootstrap" program is also contained in sector 0, track 0 of the diskette in drive 0.

Now that we have seen in general how disk storage functions, we will continue by discussing the Shugart SA400 in the next chapter, followed by the controller chip, addressing, and disk programming.

Chapter 2

Shugart SA400

This chapter will describe the Shugart SA400 disk drive as used in the Radio Shack TRS-80. The SA400 drive is essentially unmodified internally by Radio Shack, the exception being minor addressing modifications and a terminator that is connected to a dip socket on the disk electronics board. This terminator is installed only in drive number 1, hence the difference in the two types of drives supplied. The SA400 requires power supplies of +12 volts and +5 volts DC. These are installed on the rear of the cabinet for the drives (the cabinet or cover is another optional item). Initially, Radio Shack had a problem with heat dissipation for the drive power supplies, but this has been alleviated in later drives. Cabling is also supplied by Radio Shack and is discussed later in this text. The cable is a simple ribbon cable.

Another point that should be mentioned here is that the Shugart SA400 has become something of a de facto standard for minifloppy disk drives. Several other manufacturers make drives that are presumably plug-to-plug compatible with the SA400 and could be used in place of the SA400.

Physical Data

The SA400 is very compact, so much so that many microcomputer manufacturers have installed the entire drive in existing cabinetry to provide a disk system. The drive measures 5¾ inches high by 3¼ inches wide by 8 inches deep (plus power supply). Weight is about three pounds. There are two basic assemblies in the SA400, the drive mechanism itself, and a printed circuit board associated with head electronics and interfacing. The drive mechanism uses a DC drive motor with a servo speed control and integral tachometer. The motor rotates a spindle through a belt drive system. The drive has a mechanical interlock on a door latch to insure that the diskette is properly inserted.

The read/write head is ceramic and is mounted on a carrier assembly. The head assembly is positioned through the use of a spiral cam. The cam is driven by a stepping motor that positions the cam and head by rotating the cam in discrete increments (if you are like me, you are falling asleep by now - pull off the disk cover, and you will immediately see th mechanism which steps the head assembly from track to track. I 's better than a long description, although I don't want to offend any mechanical engineers out there).

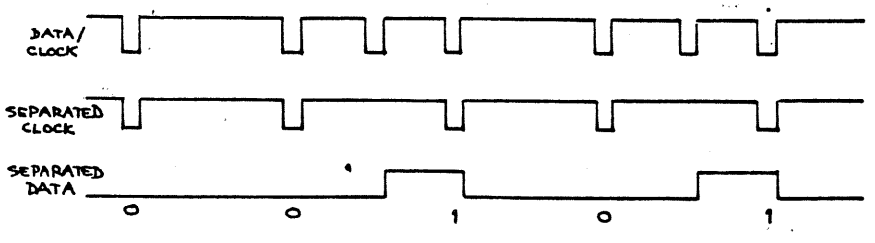
Printed Circuit Board Electronics

The printed circuit board assembly contains electronics to perform the following functions:

1. Detect the sector index mark in the diskette.
2. Position the head to the proper diskette track.
3. Load the head (press the diskette against the read/write head in preparation for reading and writing.
4. Generate signals in the head to write data or read flux changes from the diskette, including merging data and clock signals.
5. Detect the write protect condition.
6. Detect when the drive is selected.

The functions above are uncomplicated, with the exception of the read and write data function. Data written to the diskette is merged with a clock signal. When the combined data/clock signal is read from the diskette the clock and data must be separated by special circuitry. In this case the circuitry is contained in the Western Digital FD1771B-01 chip, along with circuitry to create the merged clock/data. The effect of merging the data and clock is to produce a pulse train that is *frequency modulated*. Whenever a one bit is generated, two pulses result during a clock cycle, while only one pulse is generated for a zero bit. A typical string of data read from a diskette is shown in Figure 1. The recording method is shown for information only, as one should never have to deal with data at this raw level, unless possibly troubleshooting an inoperative disk drive.

Figure 1. Data Recording



Control Signals

The standard Shugart SA400 interface signals are shown in Table 1. These are the signals that are present on the disk drive printed circuit board connector that attaches to the TRS-80 cabling. We will discuss these signals in general initially, without regard to the TRS-80, and

describe in chapter three how they relate to the FD1771B-01 controller chip (or vice versa).

Table 1. SA400 Interface Signals

Controller to Disk	Disk to Controller
Select 0	Index
Select 1	Track 0
Select 2	Write Protect
Select 3	Read Data
Drive Motor Enable	
Direction	
Step	
Write Data	
Write Enable	

Assume the disk is off. When power is supplied and the signal Drive Motor Enable goes low, the drive motor "comes up" to a speed of 300 revolutions per minute and stabilizes at this speed in less than one second. When this signal goes high, the drive motor stops in less than a second. In the TRS-80, the motor is normally off and is only turned on for reading and writing.

The Direction and Step signals are outputs to the disk that control the stepping of the drive head. If Direction is low and a Step pulse is issued, the head will step over towards the center of the disk by *one track*. If Direction is high and a Step pulse is issued, the head will step one track away from the center of the diskette. Obviously, to step from the outermost track (0) to the innermost track (34) requires 34 step pulses. Each Step pulse goes from high to low for a duration of 200 nanoseconds (200 billionths of a second) to 2 milliseconds (2 thousandths of a second).

The Index signal is an output from the disk drive which appears as a pulse every 200 milliseconds at the beginning of a track. It is generated by the appearance and detection of the index hole in each diskette five times per second.

Signal Track 0 is another output from the disk indicating that the head is positioned over the outermost track, track 0. This causes signal "Track 0" to go low.

Signal Write Protect is low whenever an opaque label is put over the diskette write-protect notch. This signal informs the controller that writing data to the disk is not possible.

Signals Select 0 through Select 3 are used to select one of four drives. The drive number within the SA400 is determined by a dip shunt. In the TRS-80, however, selection is determined by position along the cable, and if Select 0 is low, drive 1 is selected, if Select 1 is low drive 2 is selected, and so forth.

Reading and Writing

The normal sequence before reading or writing data is to turn on the drive motor and to then position the head by stepping until the head is positioned over the desired track. Now data can be read or written after the motor reaches full speed (one second) and the desired sector appears under the head. The controller chip issues the proper number of Step commands in the proper Direction to postion the head and may also control Drive Motor Enable, although this is not done in the TRS-80 (addressing the disk turns on the motor for a period of time).

When a write is to be performed, the Write Enable line must first go low to signal the drive that a write will be taking place. The current in the head is turned on by the Write Enable signal in preparation for the write. Writes cannot be performed unless Write Enable is low.

Data to be written is sent to the disk via the Write Data line. Each high to low transition on this line causes a magnetic flux change in the read/write head. The recording technique used is the previously described frequency-modulation type (double frequency) in which data and clock form a combined Write Data signal.

Data being read is sent from the disk by means of the Read Data line. A data pulse is sent for each flux transition on the diskette.

Both read and write data appears as a series of serial pulses. The controller performs a serial to parallel conversion in data read from the disk and a parallel to serial conversion for data written to the disk.

In essence then, there are really not many things that the disk can do. It can only step one track at a time in one direction, write a data pulse, read a data pulse, and report back on the status of the index mark, track 0 position, and write protect. All of the other functions such as reading a sector, formatting, writing a sector, stepping more than one sector to a given track, and finding a given sector must all be implemented in external (to the disk) logic. In the next chapter we will see how the floppy disk controller implements these functions and others.

The following table recaps disk signals, lists their pin numbers for the SA400 connector and the TRS-80 cabling, and cross references the Shugart name with TRS-80 terminology.

Table 2. SA400/TRS-80 Signals

Shugart Pin #, Signal Name		TRS-80 Cable Pin #, Signal Name	
1	spare	1	ground
2	spare	2	not used
3	spare	3	ground
4	spare	4	not used
5	spare	5	ground
6	spare	6	not used
7	ground	7	ground
8	Index	8	Index Pulse
9	ground	9	ground
10	Select 0	10	DS1
11	ground	11	ground
12	Select 1	12	DS2
13	ground	13	ground
14	Select 2	14	DS3
15	ground	15	ground
16	Drive Motor Enable	16	Motor On
17	ground	17	ground
18	Direction	18	Direction Select
19	ground	19	ground
20	Step	20	Step
21	ground	21	ground
22	Write Data	22	Write Data
23	ground	23	ground
24	Write Enable	24	Write Gate
25	ground	25	ground
26	Track 0	26	Track Zero
27	ground	27	ground
28	Write Protect	28	Write Protect
29	ground	29	ground
30	Read Data	30	Read Data
31	spare	31	ground
32	Select 3 (jumper)	32	DS4
33	spare	33	ground
34	spare	34	not used

Chapter 3

Western Digital FD1771B-01

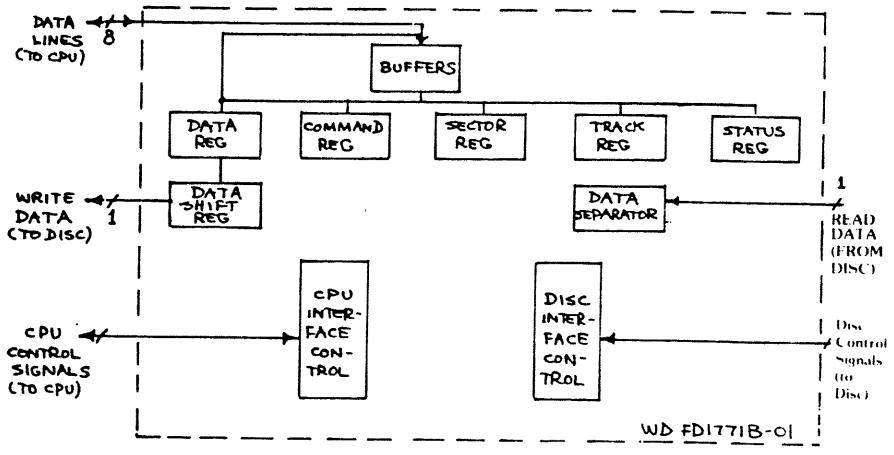
The Western Digital FD1771B-01 is a 40 pin floppy disk controller chip which enables the TRS-80 to issue simple commands for head positioning and reading and writing of data. The controller chip then performs the complicated timing functions for implementation of these commands. The commands that may be issued to the FD1771B-01 are:

1. Restore. Move the head to track zero.
2. Seek. Find the currently specified track.
3. Step. Step the head in the last direction.
4. Step In. Step the head one track in.
5. Step Out. Step the head one track out.
6. Read. Read one byte of user data.
7. Write. Write one byte of user data.
8. Read Address. Read identification field.
9. Read Track. Read entire track.
10. Write Track. Write entire track.
11. Force Interrupt. Terminate operation.

Data to be written to the disk is transferred from the TRS-80 to the FD1771B-01 in parallel, 8 bits at a time. Data to be read from the disk is assembled from bit serial data from the disk into 8 bit bytes and then sent to the TRS-80 in parallel. In addition to serial/parallel conversion for data, the FD1771B-01 also recieves parallel data related to head positioning for the disk. All parallel data, whether commands or user data, is sent to the FD1771B-01 over the eight-bit data bus from the TRS-80 CPU, D7 through D0. Every time a byte of data is sent over the data bus, the disk controller must first be addressed by performing a "load instruction" for a read from the controller, or a "store instruction" for a write to the controller. Addressing and transfer of data to the disk controller will be explained in more detail in the following two chapters.

The FD1771B-01 contains several registers for positioning and disk data. They are shown in Figure 1.

Figure 1. The FD1771B-01



Tracks on the SA400 are numbered from 0 (outermost) to 34 (innermost). The track register is an 8-bit register which holds the current track number. Every time the disk head is stepped, the track register is automatically incremented or decremented to reflect the current track position. The track register may be read or loaded by a "load" or a "store" instruction in the Z 80.

The sector register is an 8-bit register which holds the current sector number. As the ten sectors rotate under the head, the sector register is adjusted to hold the current sector number. The sector register may be read or loaded by a "load" or a "store" instruction in the Z-80.

The command register is an 8-bit register which holds one of the eleven possible commands that may be issued to the FD1771B-01. The status register is an 8-bit register which holds status information from the disk. The status in the register varies with the command, but represents such typical conditions as "track 0", "disk busy" and "disk protected".

An 8-bit data register holds data that is read from the disk or is to be written to the disk. This register interfaces to another data register, which converts the parallel TRS-80 data into serial form.

Other logic in the FD1771B-01 is concerned with separating the data from the clock signal (data separator), arithmetic within the disk controller (arithmetic and logical unit), and control logic.

Clock Signal

The clock signal for the FD1771B-01 is a 1.0 mhz square wave input to the disk controller. The clock is used to control internal device timing and is also used to generate clock and data pulses sent to the disk for writes.

Power Supply Signals

The FD1771B-01 requires +12 VDC, +5 VDC and -5 VDC.

FD1771B-01 to Disk Signals

The signals described under the SA400 are shown below, referenced to their FD1771B-01 signals. The FD1771B-01 signals perform the same functions as previously discussed. There is no chicken/egg debate here as the disk signals were defined before the controller; the controller is designed to easily implement the necessary logic for the disk. Pin numbers for both the SA400 and FD1771B-01 are provided in the table. Figure 2 shows a "pin-out" of the controller chip with all 40 pins of the controller chip and their corresponding signals.

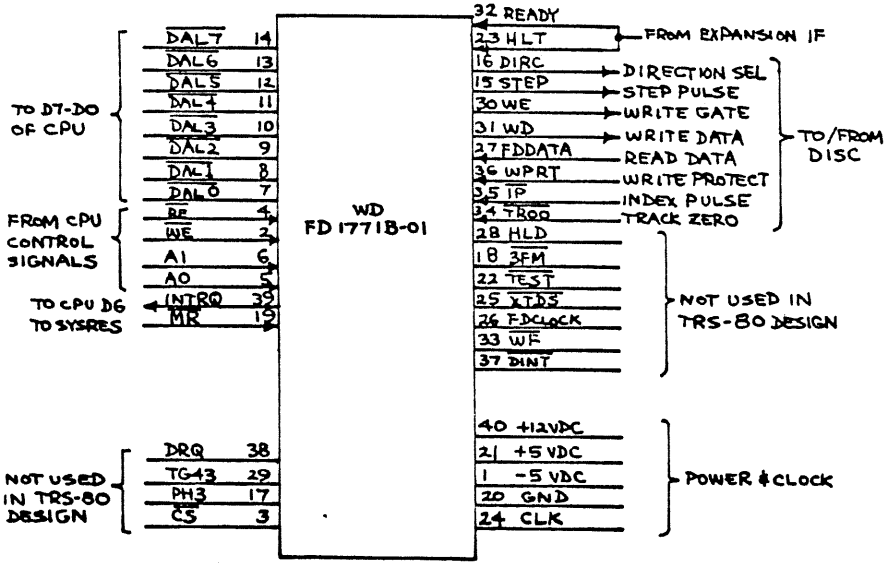
FD1771B-01 Signal (pin)		SA400 Signal (pin)
HLT (32)		
RDY/HLT (23)	←	from expansion interface
STEP (15)	→	STEP PULSE (20)
DIRC(16)	→	DIRECTION SEL (18)
WE (30)	→	WRITE GATE (24)
WD (31)	→	WRITE DATA (22)
FDDATA (27)	←	READ DATA (30)
WPRT (36)	←	WRITE PROTECT (28)
IP (35)	←	INDEX PULSE (8)
TRØØ (34)	←	TRACK ZERO (26)

A set of other signals for the FD1771B-01 are not used in the TRS-80 implementation. Examples of these are the factory test input (pin 22) and signals connected with an external data separator (pin 25).

FD1771B-01 Signal (pin)	Description
HLD (28)	Head Load
3FM (18)	Three phase motor select
TEST (22)	Test input
XTDS (25)	External data separator

FD1771B-01 Signal (pin)	Description
FDCLOCK (26)	Floppy disc clock (external separator)
$\overline{WE}$ (33)	Write fault
DINT (37)	Disc initialization

Figure 2. FD1771B-01 Pin-Out



TRS-80 to FD1771B-01 Signals

Both commands and data are passed between the TRS-80 and FD1771B-01 by DAL7' through DAL0' (most significant to least significant). This is a bidirectional bus that feeds the data, sector, track, command, and status registers depending upon the command sent out by the TRS-80. The DAL7' through DAL0' lines are connected (or gated) to the data bus to the TRS-80, D7 through D0, respectively.

Input signals RE' and WE' (pins 4 and 2) are read and write signals from the TRS-80. Each read and write operation to the FD1771B-01 transfers one byte of data, command, or status between the FD1771B-01 and Z-80 or CPU. A0 and A1 (pins 5 and 6) interface directly to A0 and A1 of the TRS-80 and control which type of data transfer is to be made. If a read or write is performed to the FD1771B-01, the following actions take place, dependent upon the state of A0 and A1. These transfers will be described in more detail later in this text.

A1	A0	Read Action	Write Action
0	0	Read staus	Write command
0	1	Read track	Write track
1	0	Read sector	Write sector
1	1	Read data	Write data

Signal INTRQ (Interrupt Request) would normally be used to generate an interrupt in many computers, but in the Radio Shack implementation is used to signal a ready status to the TRS-80 by being tied to bus line D6. Signal INTRQ goes high at the end of any operation performed by the FD1771B-01 and is reset when a new command to the FD1771B-01 is issued.

Signal MR' is "Master Reset" and is used to initialize the FD1771B-01 to an initial condition.

Another set of signals commonly used for interfacing are not connected or deactivated in this configuration. They are shown below:

Signal (pin)	Description
DRQ (38)	No connection. Data request.
TG43 (29)	Track greater than 43, no connection
PH3 (17)	Phase 3, no connection
CS' (3)	Chip Select. Ground.

Commands

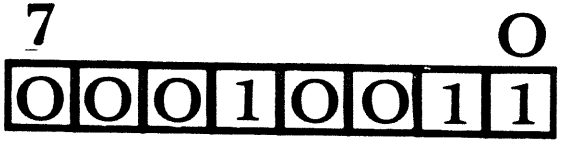
Eleven commands may be sent to the FD1771B-01 by the TRS-80. They were previously listed. The first group of commands are related to motor positioning. They are Restore, Seek, Step-In, and Step-Out. All are sent to the FD1771B-01 by a write A0=0, A1=0 output. In the TRS-80, this would be accomplished by loading a CPU register with a byte defining the command value and performing a "store" instruction to address 37ECH. The 37ECH location is actually the disk address, and

the least significant two bits of the "C" are 00, which specifies the command register. Signal WE is active because a "store" instruction is being executed, and the write causes the command byte to be transferred from the CPU register to the command register in the FD1771B-01.

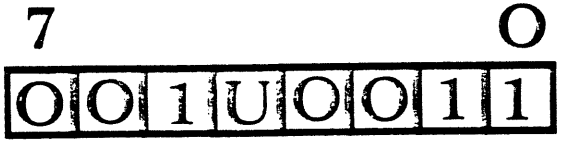
Restore is issued by outputting a data byte of:

The Restore moves the disk head to a position over track 0. There are three *fields* in the Restore command. The first field, H, may be a 1 or a 0 in the TRS-80 system as it causes no action (the head is loaded at motor on). In other implementations it unloads or loads the head before a command. The second field is a *verify* field and may be a 0 or a 1. It is used to verify that the track address read from the diskette is the same as the current track register contents. The third field (two bits) is used to vary the stepping rate of the head. The proper rate for TRS-80 operation with the SA400 is the stepping rate defined by binary 11. For discussion purposes, we will assume that the normal coding of these fields in the TRS-80 will be binary 0011. These three fields are also used in the other four commands in this group (Seek, Step, Step-In, and Step-Out) and we will assume a binary 0011 configuration here also.

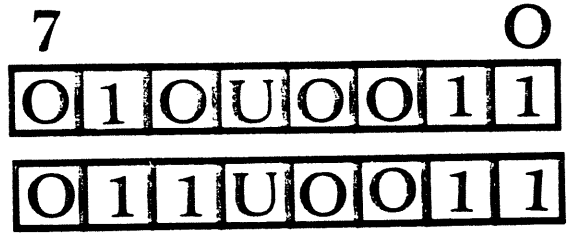
The Seek command must be preceded by an output to the data register to load the data register with the desired track number. (The output would be performed by a "store" to location 37EFH to store a track number in a CPU register.) When a Seek command is issued after the track number has been stored in the data register, the FD1771B-01 will automatically position the head over the proper track by comparing the contents of the data register with the current track register and issuing an appropriate series of Step Pulses in the proper Direction. The Seek command format follows:



The Step command causes one step of the head to occur. The direction of the step, in or out, is the same as the previous Step command. The Step command *does* have an active field, U. The U field determines whether the track register is updated after the step. If U=1, the track register is incremented or decremented by one, dependent upon the direction of the step. If U=0, no adjustment is made. The Step format is:



The Step-In and Step-Out commands are similar to the Step, except that the direction is explicit. The U field operates as in the Step.



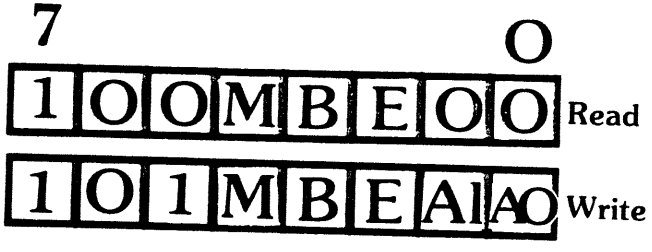
To reiterate the commands in the first group then, Restore finds track 0, Seek finds a specified track, Step, Step-In and Step-Out move one track. The Step commands have a field specifying whether the track register should be updated automatically. Generally this field will be set. The verify field may be optionally used on all commands to verify that the track # read from diskette matches that in the controller track register.

The second group of the eleven commands in the FD1771B-01 are related to reading and writing data. There are two of these - Read Command and Write Command.

User data is generally transferred a sector's worth at a time, or 256 bytes. In general, data may be transferred under register I/O or Direct Memory Access (DMA) I/O. The latter operation allows an I/O device controller to transfer data between memory and the I/O device independentally of the CPU, and requires some fairly involved logic to sequence data transfer while "locking" out (suspending) CPU operation. DMA has the advantage of permitting the CPU to execute program instructions while the I/O transfers are taking place. The method used in

the TRS-80 is register I/O. This implementation works, but at the cost of CPU overhead. The CPU is continually "I/O bound" waiting in a "status loop" to transfer the next byte of data when the controller says it is ready (in this case the controller is the FD1771B-01). To "keep up with" the disk, the controller must be able to transfer one sector's worth in about one-tenth revolution, or 20 milliseconds, which is equivalent to transferring a byte every 78 microseconds. As normal instruction times are about 8 microseconds in the TRS-80, the Z-80 CPU can indeed keep up with data flowing from or to the disk. We will see the exact instruction sequence later in this book.

Prior to the read or write, the TRS-80 program must load the sector register with the sector number to be read or written. Also, of course, the head must be positioned over the proper track by a Restore, Seek, or series of Step commands. The sector register is loaded by a "store" instruction with an address of 37EEH (A1=1, A0=0) which stores a sector number from a CPU register into the FD1771B-01 sector register. The format of the Read and Write commands are shown below. As in the case of the positioning commands in the first group, the command is written to the FD1771B-01 command register by execution of a "store" instruction with an address of 37ECH (A1=0, A0=0).



There are three fields in the Read command and four fields in the Write command. The M field in both is used to enable the read or write of multiple sectors or records. If M=0 a single record will be transferred. If M=1, more than one sector will be transferred. The usual case for the TRS-80 is to transfer only one sector, however, two to ten sectors could also be transferred by setting this bit. The B field of the Read or Write is always set to a 1 to signify IBM-compatible disk format, which is simply a de facto standard established by IBM. Field E controls a 10 millisecond delay which enables the head to engage before the read or write. As the head is *always* engaged before the read or write from motor turn-on time (the HLD field is not used), the E field could be a one or zero. We'll use the 10 millisecond delay only because Radio Shack uses it.

The A1/A0 fields of the Write command control writing of the data

address mark on the diskette. The standard IBM format for this is a hexadecimal FB, which is specified by a field of binary 00.

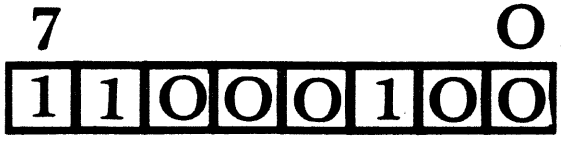
Data is written or read to the diskette by continuously monitoring (reading) the status from the FD1771B-01. One of the status bits is DRQ, or Data Request, which signifies either that the data register contains the next byte of read data or has been 'emptied" of the last byte of write data. Examples of programming for reads and writes will be shown later.

There are three commands in the next group, related to disk formatting. They are: Read Address, Read Track, and Write Track. As we mentioned previously, disk formatting initializes the diskette to a standardized (IBM) format with identification data for track and sector number, gaps between sectors, CRC (checksum) characters, and other data. Writing new data to the diskette is done in the 256-byte area reserved for user data. The complete format of the TRS-80 diskette is given in Appendix B.

Read Address is a command that reads six identification bytes from the diskette from the next encountered identification field. Each time one of the bytes is read the DRQ (Data Request) bit is set in the status, so that the TRS-80 program can read it from the FD1771B-01 by a "load" 37EFH instruction. The six bytes read are as follows:

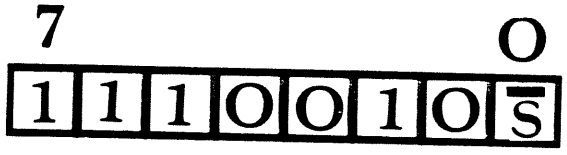
1. Track address, 0 through 34
2. Zeros
3. Sector address, 0 through 9
4. Sector length
5. CRC character 1
6. CRC character 2

These six bytes correspond to the bytes given in the appendix for track format. The Read Address command could be performed at any time, and not just during a formatting operation. The command for a Read Address is shown below:



The Read Track command reads an entire track of the current diskette. Not only user data, but identification data is read as well. As in the case of Reading user data, the Data Request bit is used to indicate when the next byte of data is ready to be transferred from the FD1771B01 data register to the Z-80 CPU. Gaps on the diskette are also read and transferred. The Read Track command has one field, the S' field. If S'=0, the accumulation of bytes is synchronized to each address

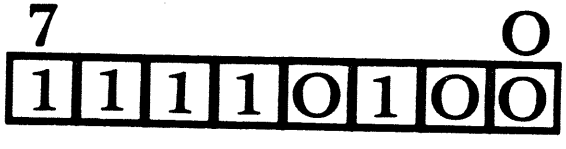
mark encountered, so that the controller does not "get lost" in reading the track full of data. If S=1, no synchronization is done. The Read Track will primarily be done during the formatting process to verify formatting data, although it could be done at any time. The format of the Read Track is:



Write Track is used to format the diskette accoring to the format shown in Appendix B. Data in the given format is presented to the FD1771B-01 one byte at a time by the program. The Data Request bit is continually checked to see when the controller chip is ready for the next data byte. Address marks are written to the diskette by the controller chip upon detection of certain data patterns sent by the CPU. A CRC (cyclic redundancy check type of checksum) is written to the disk in the same fashion. The codes for these actions are:

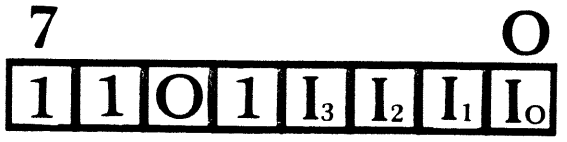
Data Pattern	Description	Clock Mark
F7H	Write CRC	FFH
FBH	Data Address Mark	C7H
FCH	Index Address Mark	D7H (not used)
FEH	ID Address Mark	C7H

Obviously, no user data is written to the diskette during the Write Track operation. The user area is filled with E5H or some other nonconflicting pattern (the bytes above would generate a CRC or address action). The format of the Write Track is:



There is one command in the last group of eleven commands. The Force Interrupt command is used to terminate the current command and to generate an "interrupt" in many systems. In the TRS-80 a Force Interrupt command also causes an interrupt (if interrupts are

"enabled"). Termination of the current command and an interrupt occur when one of four conditions occur as specified in a four-bit field in the Force Interrupt command.



The four-bit field is defined by I_3, I_2, I_1, I_0. Each specifies a different condition for the termination, as follows:

I_0=1, terminate on not ready to ready transition.
I_1=1, terminate on ready to not ready transistion.
I_2=1, terminate on next index pulse.
I_3=1, immediate terminate/interrupt.

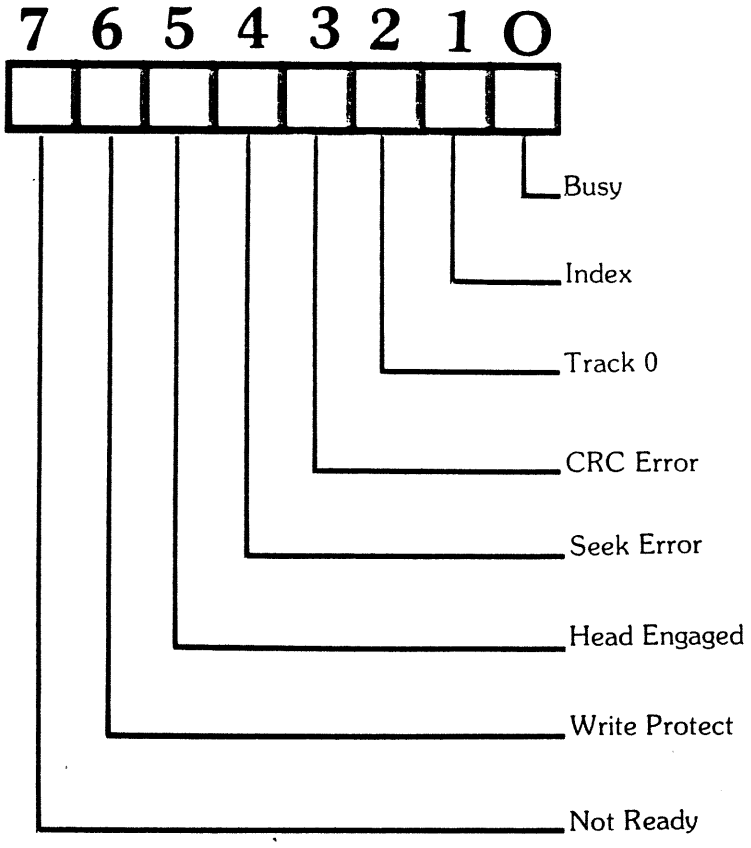
If none of the above conditions is specified, there is no interrupt but the command is terminated (I_0=I_1=I_2=I_3=0). Why have so many conditions? Why not? It is even possible that all might be useful. For practical purposes however, probably only a Force Interrupt with no interrupt (binary 11010000) would be used, and that only to terminate a read or write of multiple sectors after the desired number of sectors. The FD1771B-01 is designed to cover many contingencies, but only a portion of all possible commands or conditions will be used in the typical microcomputer installation.

Registers

We have seen how the FD1771B-01 registers are used in the course of positioning and transfer of data. A recap of their use follows:

Command Register	Used to hold command to be acted upon. Command is written into the register from a CPU register.
Track Register	May be automatically updated with each Step. May be read by CPU.
Sector Register	Setup with sector number prior to a read or write. May be read by CPU.
Data Register	Used to hold data passing between the CPU and disk during reads and writes. Used continuously as data is transferred one byte at a time.

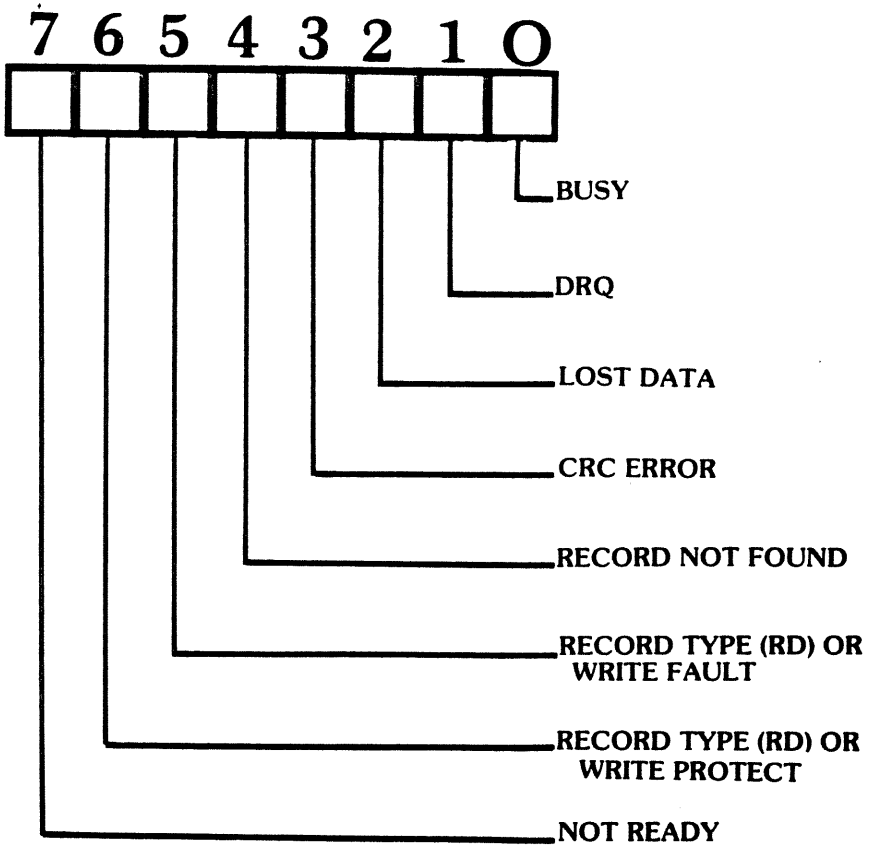
The controller register not described above is the Status Register. The Status Register hold various status conditions during different operations. For a positioning command (Restore, Seek, Step, Step-In, Step-Out), the Status Register holds status as follows:



Not ready (bit 7) is always false (ready) when the disk is being addressed. Write protect (bit 6) indicates that the diskette write protect notch is covered. Head engaged (bit 5) is always true when a disk operation is being performed. Seek error and CRC error are active if verification was used. But 4 is set on track not found. Track 0 (bit 2) is set when the head is over track 0. Index (bit 1) is true when the index mark is detected from the disk. Busy (bit 0) is set when a command is in process and reset when the command is completed. What status bits would normally be used in the TRS-80 for positioning commands? Certainly the

busy bit. The busy bit would be checked prior to issuing any new command to see if the controller was engaged in a previous activity. Any new action would be delayed until the busy bit was reset. Track 0 might also be checked after issuing a Restore operation, to see that the Restore was successfully completed.

During a Read or Write command, the status register holds status as follows:



For a Read or Write, the CRC refers to either identification data or to user data (status bit 3). This bit will be set to indicate invalid data. Lost data (bit 2) means that the CPU did not respond fast enough to keep up with the data being transferred between the CPU and disk. If this condition is true, the bit is set. This should never occur except in catastrophic cases. Record not found indicates that the desired track or sector or both was not found. This bit (4), would be set if this error

condition occurred (for example, loading an invalid sector number prior to the read). Data Request, DRQ bit 1, indicates that the buffer is empty on a write or full on a read. Busy (bit 0) is set during the time the Read or Write is active. Not Ready (bit 7) is always false (ready) when the disk is being addressed. On a Write, bit 6 is set for a write protect condition, while bit 5 indicates a write fault. On a Read these bits should be a 01 to indicate a 256 byte length. The chief bits used for status would be busy and DRQ. Busy would be tested by the program to insure that a previous operation was over. DRQ would continually be checked for the next data byte to be transferred. The other bits would be used to indicate fault conditions.

Status is also available during the formatting type commands of Read Address, Read Track, and Write Track. The status bits would have the same meaning as in the other commands. Status during one of these three commands is as follows:

Status Bit	Read Address	Read Track	Write Track
7	not ready	not ready	not ready
6	0	0	write protect
5	0	0	write fault
4	ID not found	0	0
3	CRC error	0	0
2	lost address	lost data	lost data
1	DRQ	DRQ	DRQ
0	busy	busy	busy

This chapter has discussed the operation of the Western Digital FD1771B-01, primarily in terms of what internal operations are performed, and how it interfaces to the Shugart SA400 disk drive. The next chapter will show how the TRS-80 communicates to the FD1771B01 to enable the program to perform positioning operations and reads and writes. The last chapter will show the sequence of operations to perform in a program to accomplish useful work with the disk.

Chapter 4

Expansion Interface

A schematic of the expansion interface as it pertains to the disk is shown in Figure 1. There are several sections to be considered in this implementation. They are:

1. Disk selection
2. Motor on circuitry
3. Addressing
4. FD1771B-01 functions
5. Interrupt circuitry

Disk Addressing

There are two general addresses associated with the disk(s). The address used to select the drive is 37EXH, where X represents 1 through 8. The 37E0H "device select" signal is decoded by two chips in the expansion interface, Z32, a dual 2-line to four-line decoder (74LS155) and Z43, a dual 2-line to four-line decoder (74LS139). These chips are also used to generate a "controller select" signal from address 37EXH, where X represents C, D, E, or F.

Let's take a look at how these two select signals are generated. The outputs from Z43 are in two groups 1Y0, 1Y1, 1Y2 and 1Y3 and 2Y0, 2Y1, 2Y2 and 2Y3. Each group is controlled by an enable signal G1 or G2. These enables must be low for any of the outputs to be active. If the enable is active then *one* of the outputs is active dependent upon the configuration of the select bits A1/B1 or A2/B2. The select bits define binary values 00, 01, 10, or 11 selecting Y0, Y1, Y2 or Y3, in that order; the most significant bit is select bit B and the least significant is select bit A. With G1 low and select bits B1=0 and A1=1, for example, output 1Y1 would be active, or low. Note that outputs 1Y1, 1Y2 and 1Y3 are not used in disk addressing. 1Y1 is not connected, and the other two are used to denote input of a 32K or 48K address, respectively.

Output 1Y0 is fed back to the *enable* of the second decoder on the chip, G2. Now 1Y0 is active, or low, when RAS* is low and A15 and A14 are both zeros. RAS* comes from the CPU and is low when a memory request is made. The disk controller chip is "memory mapped" and addressed as a memory location, so RAS* will be low when we are addressing the controller chip or selecting a disk. Now with G2 low, the

outputs at 2Y0, 2Y1, 2Y2 and 2Y3 reflect the configuration of inputs B2 and A2. 2Y1, 2Y2 and 2Y3 are not used. 2Y0 will be active when A2 and B2 are both low. Since B2 is directly connected to A11, A11 must be a zero for 2Y0. A2 is low when address lines A5, A6, A7, A8, A9, A10, A12 and A13 are true, causing the output of NAND gate Z42 to go low. Output 2Y0 will be active (low) therefore, when the following conditions exist:

ADDRESS BITS

15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
0	0	1	1	0	1	1	1	1	1	1	X	X	X	X	X

RAS*=O **X=DON'T CARE**

Output 2Y0 feeds chip Z32 enabling G1 and G2. This chip acts much the same as Z43, except that there are two additional enables, C1 and C2. When C1 is high, then outputs 1Y0, 1Y1, 1Y2 and 1Y3 are enabled. When C2 is low, then outputs 2Y0, 2Y1, 2Y2 and 2Y3 are enabled. A common two select bits A and B determine which of the four outputs will be active. Now C1 and C2 are connected to two CPU signals RD* and WR*, respectively. These signals are active (low) when a read or write are being performed. The read and write are mutually exclusive, of course, and are used for addressing memory and I/O devices. When a read is being performed, signal RD* will be low and input C1 will be high (Z23 inverts the RD* signal) and one of four outputs 1Y0, 1Y1, 1Y2 or 1Y3 will be active or low. When a write is being done, C2 will be low and one of the outputs 2Y0, 2Y1, 2Y2 or 2Y3 will be active or low. Now B and A, the two select signals, are directly connected to address lines A3 and A2. Bearing this in mind, the following table shows how the eight outputs of Z32 decode:

Address	Read	Write	Pin	Signal
37E0	Y	N	1Y0	37E0 Read
37E4	Y	N	1Y1	no connection
37E8	Y	N	1Y2	37E8 Read (printer)
37EC	Y	N	1Y3	37EC Read
37E0	N	Y	2Y0	37E0 Write
37E4	N	Y	2Y1	CSW Cassette latch
37E8	N	Y	2Y2	37E8 Write (printer)
37EC	N	Y	2Y3	37EC Write

The table indicates the general address for activating each of the eight signals. In fact, since A1 and A0 are not used in generating the address select signals, each of the eight addresses above actually represents a block of four addresses. Address 37EC, for example, represents addresses 37EC, 37ED, 37EE and 37EFH. The manner in which these address selects are used will be discussed below.

Disk Selection

Four signals are shown on Figure 1, and represent the disk select signals for disk drives 1, 2, 3 and 4. Only one of these signals should be active at any time. These signals are output from a four bit latch, Z36 (74LS175). The address of this latch is 37E0H. When a write is done to address 37E0H, the four low-order bits on the data bus, D3 through D0, are clocked into the four bits of Z36 by the 37E0 write signal. Loading a register with 1, 2, 4 or 8, and performing a "store" to location 37E0H then, selects disk drive 1, 2, 3 or 4, respectively.

Motor On Circuitry

Whenever a disk drive is selected, the 37E0H signal also goes to chip Z29. Z29 is a "one-shot" which provides a short pulse for a predetermined period of time, in this case approximately 3 seconds. The pulse from Z29 is used to enable signal Motor On, which turns on the disk drive motor in preparation for disk activity. After three seconds or so, the motor will automatically turn off if no further 37E0 write is performed. Another related action performed by the 37E0 write is generation of a "ready" signal to the FD1771B-01. Inputs HLT and RDY are used by the FD1771B-01 to determine whether the head is loaded and the disk drive is ready for activity. In the TRS-80 implementation, these inputs are true only when one of the drives has been selected. If one drive has been selected, then one of the four latches in Z36 is set, and the corresponding "not Q" output is a zero, making the common HLT/RDY signal a high. This signal goes low when Z36 is cleared by Z29 at the end of the delay.

FD1771B-01 Functions

The address of the FD1771B-01 is 37ECH, as explained above. Whenever a read or write is performed to memory address 37EC, data flows between the CPU and the FD1771B-01. The register within the FD1771B-01 that is being addressed is determined by address bits 1 and 0.

Figure 1 - TRS-80 DISC LOGIC

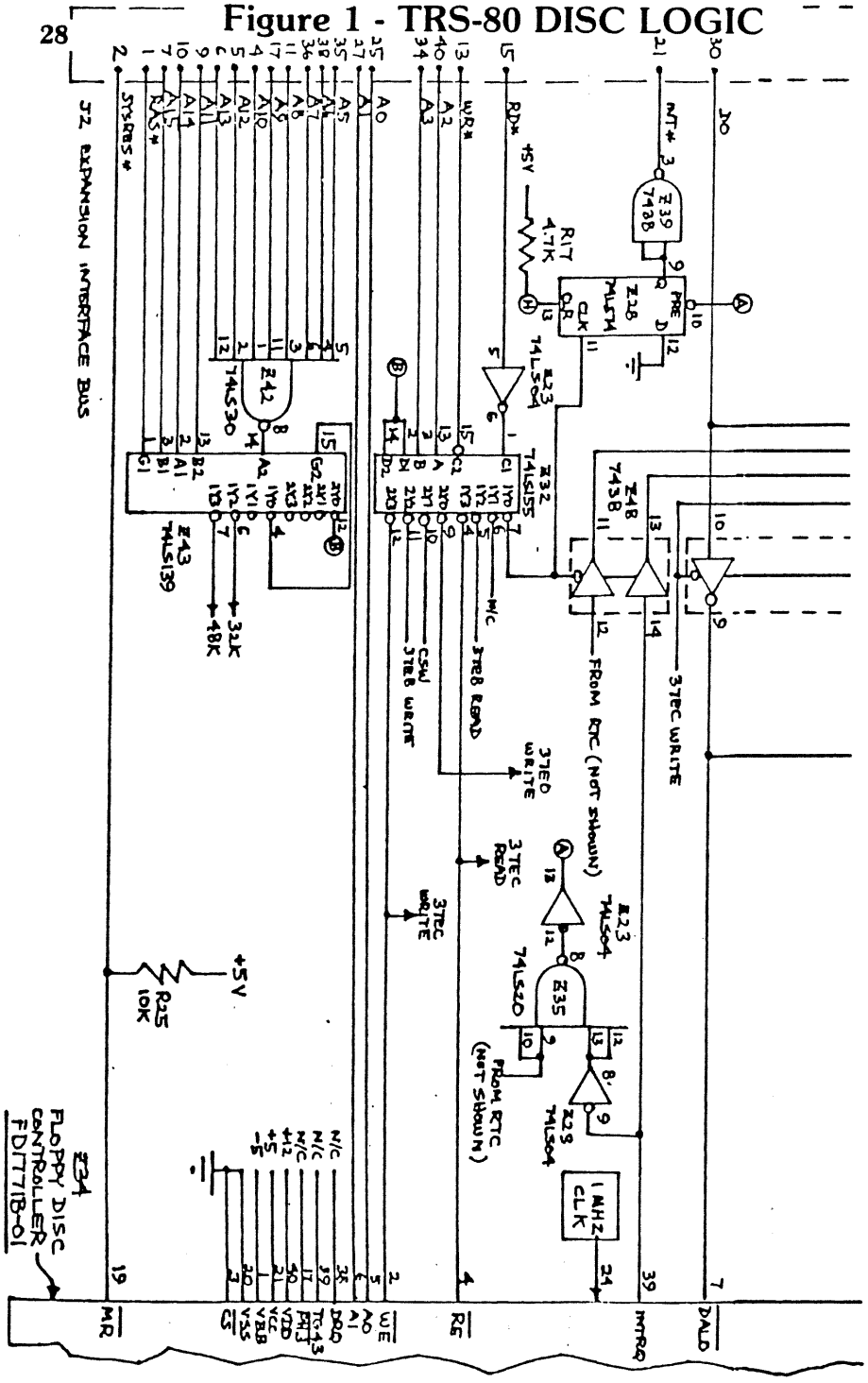


Figure 1 TRS-80 DISC LOGIC

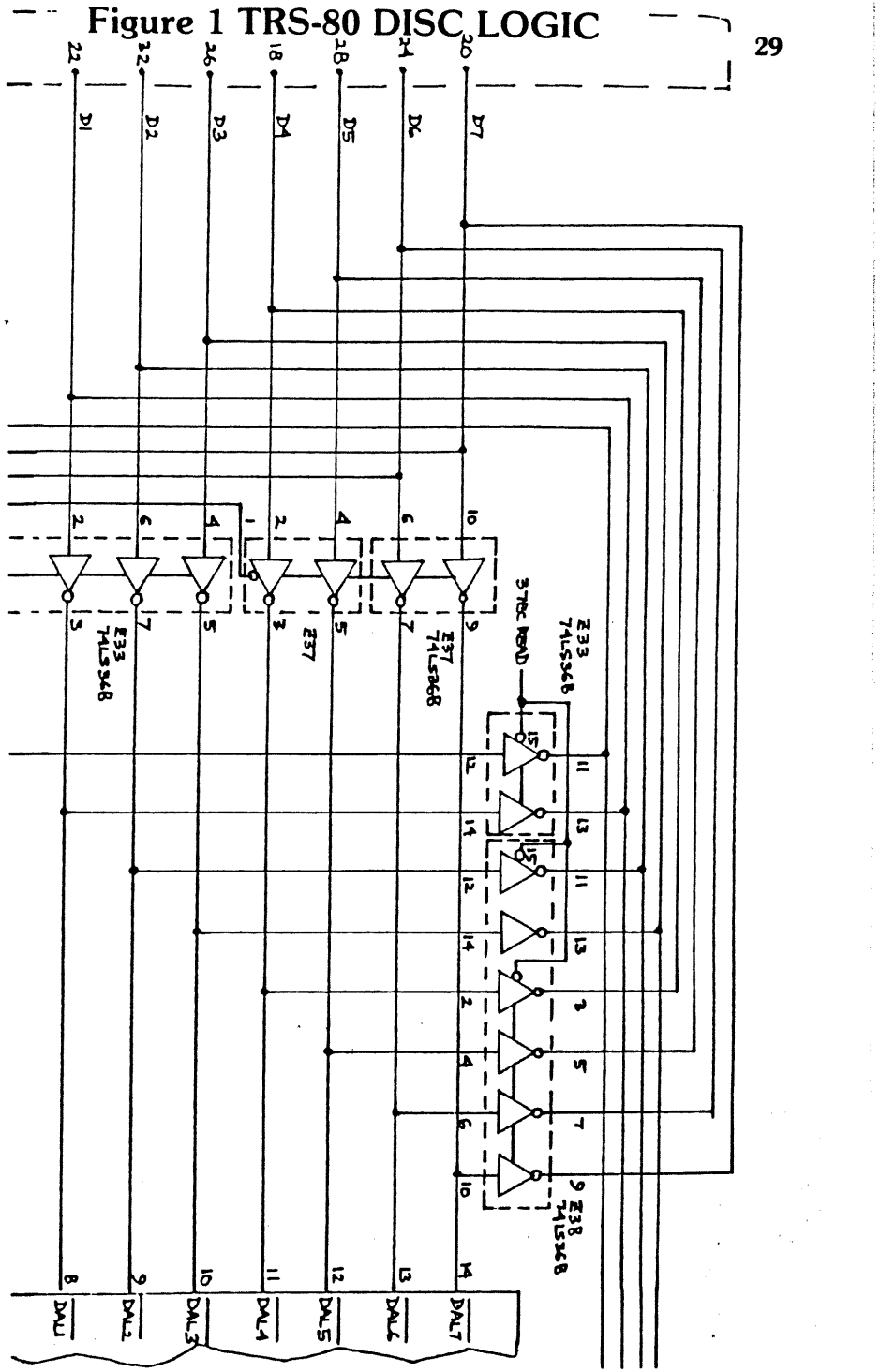


Figure 1 TRS-80 DISC LOGIC

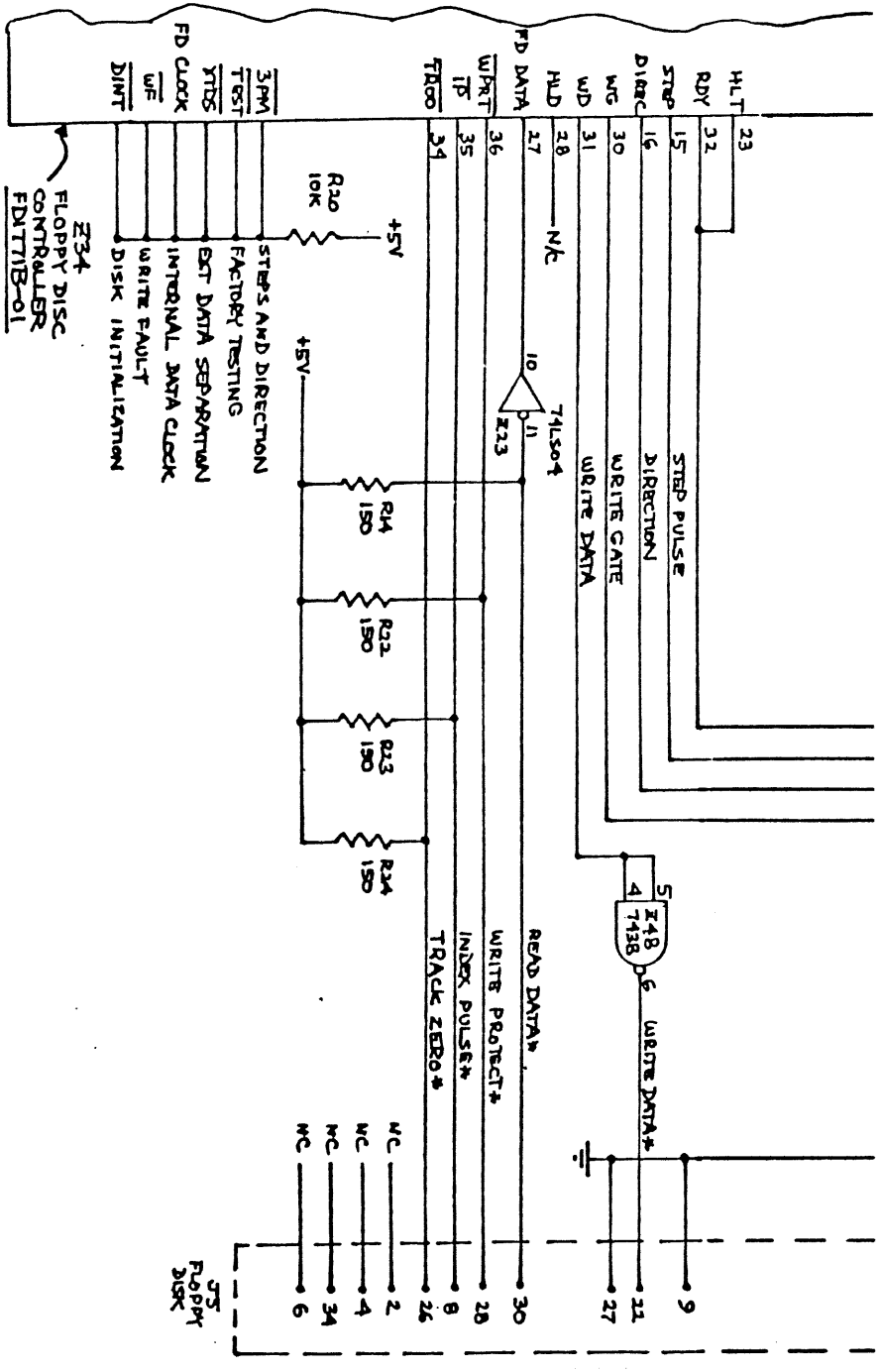
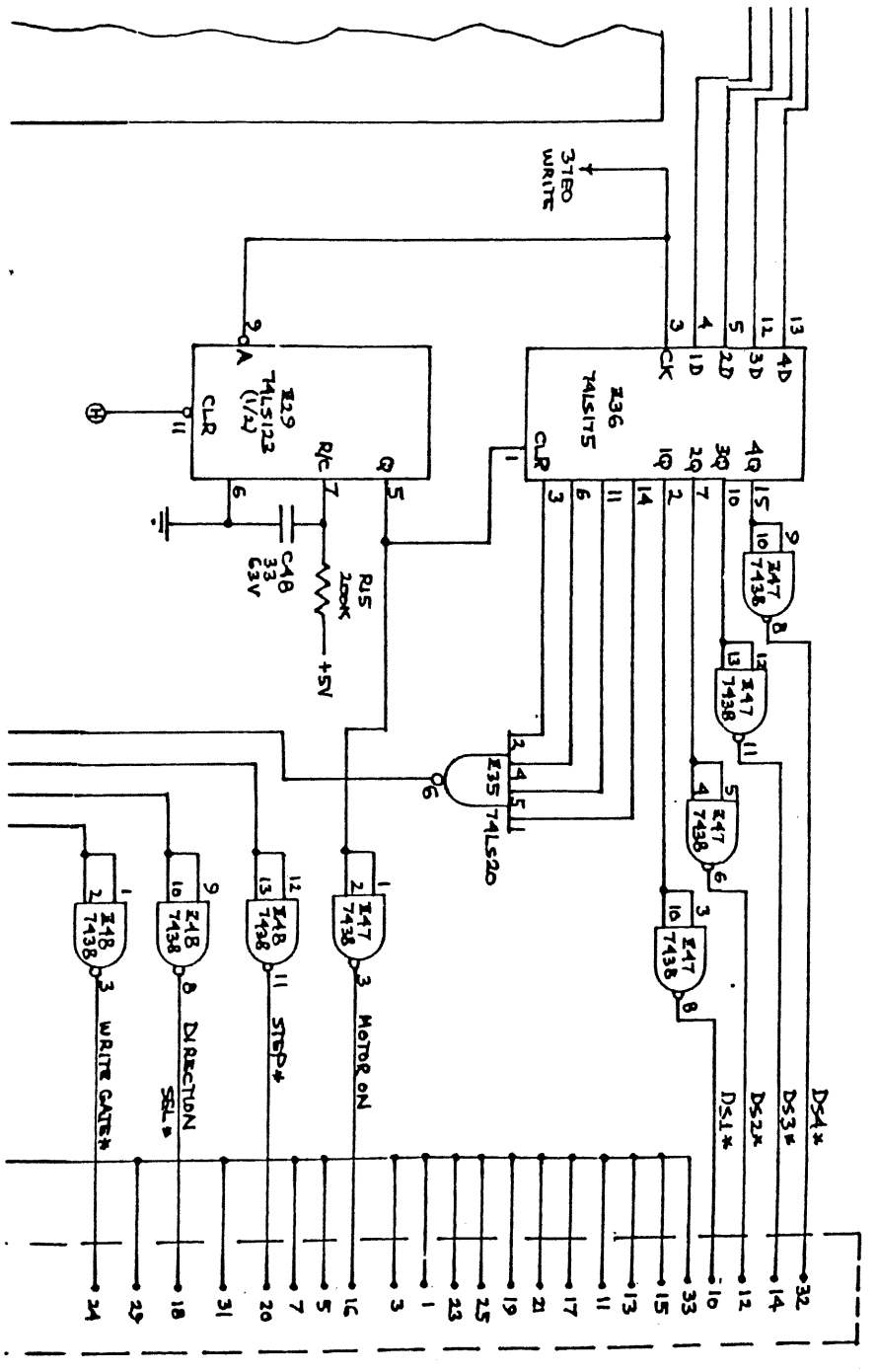
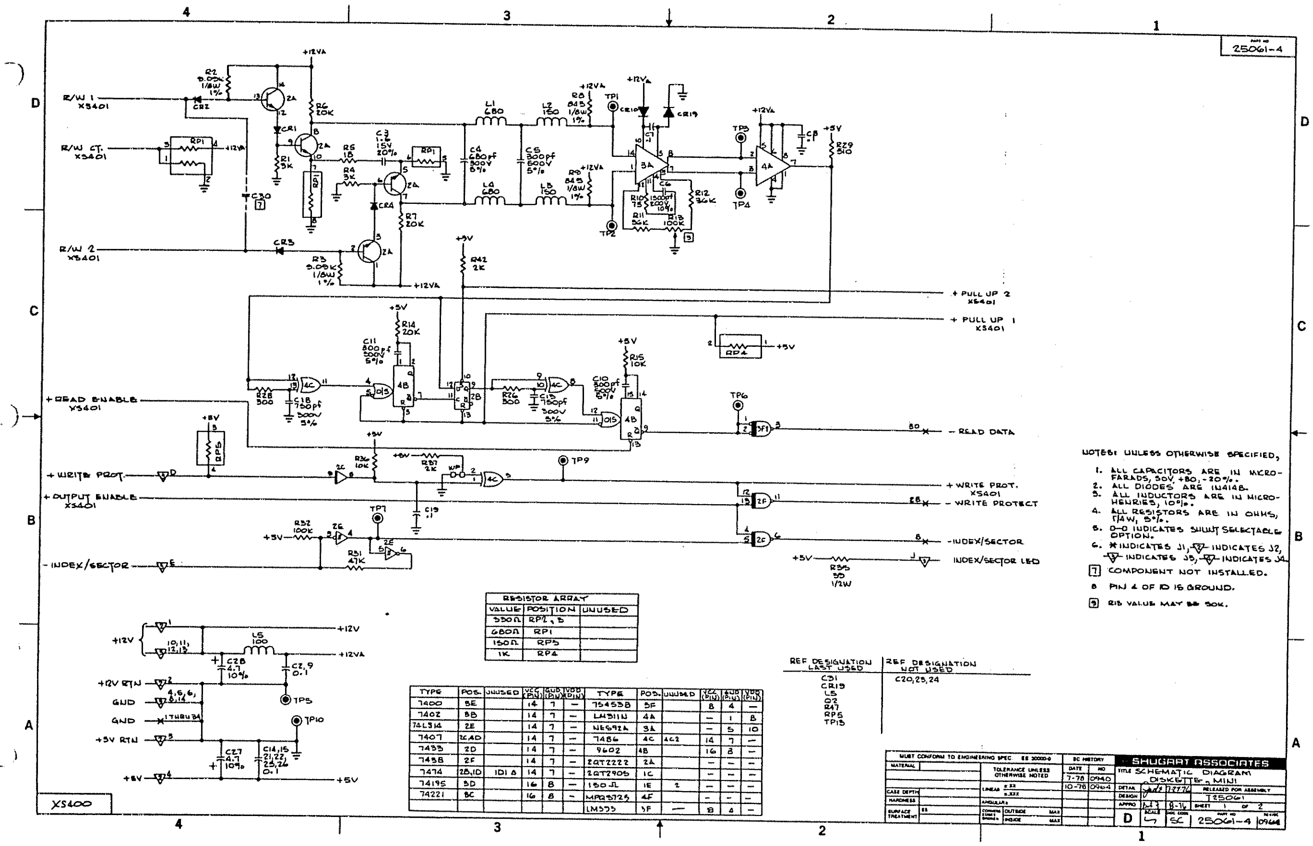
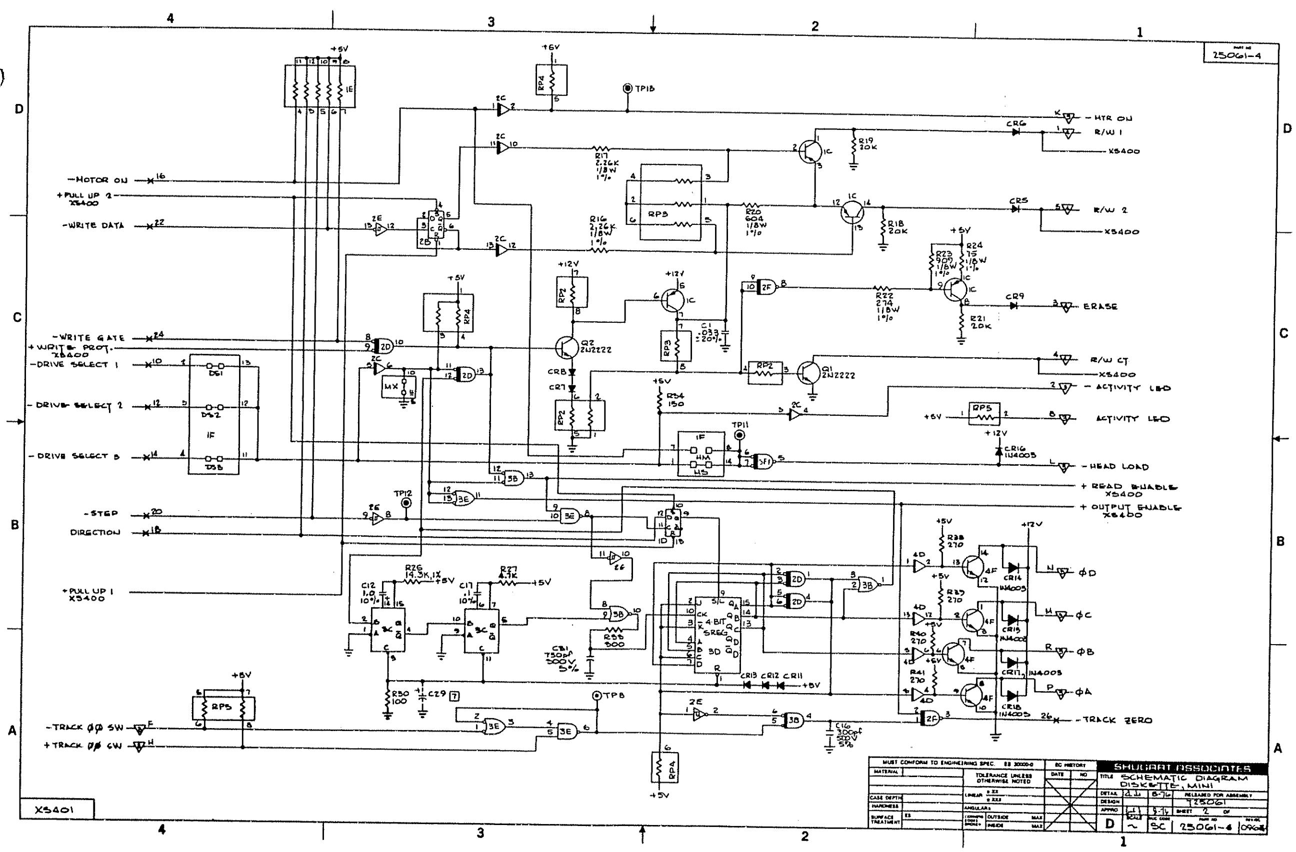


Figure 1 TRS-80 DISC LOGIC




25061-4
NOTES: UNLESS OTHERWISE SPECIFIED,
1. ALL CAPACITORS ARE IN MICRO-FARADS, 50V, +80, -20%.
2. ALL DIODES ARE 1N914B.
3. ALL INDUCTORS ARE IN MICRO-HENRIES, 10%.
4. ALL RESISTORS ARE IN OHMS, 1/4W, 5%.
5. O-O INDICATES SHUNT SELECTABLE OPTION.
6. ✕ INDICATES J1, ▽ INDICATES J2, ▽ INDICATES J5, ▽ INDICATES J4.
7 COMPONENT NOT INSTALLED.
8 PIN 4 OF 1D IS GROUND.
9 R18 VALUE MAY BE 30K.
RESISTOR ARRAY
REF DESIGNATION LAST USED
REF DESIGNATION NOT USED
SHUGART ASSOCIATES
SCHEMATIC DIAGRAM DISKETTE, MINI
25061-4
X5400

25061-4
-MOTOR ON
+PULL UP 2 X5400
-WRITE DATA
-WRITE GATE
+WRITE PROT. X5400
-DRIVE SELECT 1
-DRIVE SELECT 2
-DRIVE SELECT 3
-STEP
DIRECTION
+PULL UP 1 X5400
-TRACK ØØ SW
+TRACK ØØ CW
-HTR ON
R/W 1
R/W 2
ERASE
R/W CT
-ACTIVITY LED
ACTIVITY LED
-HEAD LOAD
+READ ENABLE X5400
+OUTPUT ENABLE X5400
-TRACK ZERO
SHUGART ASSOCIATES
SCHEMATIC DIAGRAM DISKETTE, MINI
25061-4
X5401

These two address bits control the operation, as described under "TRS-80 to FD1771B-01 Signals" in the previous chapter. To read status from the FD1771B-01, for example, a read to address 37EC would be performed. A write to the sector register would be accomplished by a write to address 37EEH (A1=1, A0=0). The read and write signals to the FD1771B-01 are signals 37EC read and 37EC write from the decode chip, which of course, are true for reads and writes in addresses 37EC through 37EFH.

Data is sent between the FD1771B-01 and CPU along the data bus, D7 through D0. Signal 37EC read is used to gate data from the controller to the CPU by enabling devices Z33 and Z38; signal 37EC write is used to gate data from the CPU to controller by enabling chips Z33 and Z37.

Interrupt Circuitry

At the completion of any FD1771B-01 operation, the controller generates an interrupt signal called INTRQ. In the TRS-80, this signal is gated onto the data bus by signal 37E0 read, along with the real time clock signal (INTRQ goes to D7 while RTC goes to D6). INTRQ also is routed to Z35 as one of two inputs that eventually set Z28 to generate an interrupt signal to the CPU. Since INTRQ is not only set to signal the end of a normal disk operation, but is also set to denote an unsuccessful disk operation, INTRQ could cause an interrupt in the CPU for abnormal conditions, providing that the interrupts were enabled (EI instruction executed). At this time of writing, not enough is known about the internal workings of TRSDOS to report on how disk interrupts are handled, if at all. Disk operations do *not* require interrupts, as we shall see in the next chapter, and for the time being we shall leave this question unresolved.

Chapter 5

Disk Programming

The rudimentary set of commands that can be used on the disk(s) in the TRS-80 are the commands described in chapter 3. No other basic disk operations are possible. At this level, then, the user can only perform a restore, seek, step head, read sector, write sector, read address, read track, write track and force interrupt. Normally the read address, read track, and write track are used only for formatting the diskette, so the general purpose commands are only restore, seek step head, read sector, write sector, and an infrequent force interrupt (which resets the controller).

Just as Z-80 instructions may be combined to implement Level I or Level II Basic, rudimentary disk commands may be combined to implement a sophisticated disk operating system or disk file manager. Sequential or random access methods may be implemented, files may be defined in various types of directories, sorts and merges may be performed on records, and disk accesses may be optimized for access speed. It is beyond the scope of this text to describe the procedures for implementing a disk operating system or file manager, just as a text on Z-80 programming would not show the implementation of a Basic interpreter. What will be show however, are methods for performing individual operations such as restores and reads which may then be combined into user programs as building blocks for more advanced operations.

Is it possible to perform disk operations using Basic? The answer is that the head positioning operations such as restore, steps and seeks may indeed by implemented using Basic, but that reads and writes may not. Reading and writing sectors (and tracks) are implemented in the TRS-80 under programmed I/O operations. Each individual byte of data transferred between the CPU and disk is handled in a CPU register. The program must continually test the disk status to see whether the disk is ready for the next data byte or whether the disk has the next data byte available. Since bytes become available at the rate of one every 64 microseconds or so, the program must be fast enough to keep pace with this data rate. Fast loops within a Basic program might require 3 milliseconds, which is many times slower than the speed required to keep pace with reads and writes. Read or write operations then *must* be performed under assembly language software routines.

Head Positioning

To become familiar with the sequence of operations for disk functions, let's look at some Basic head positioning routines. These routines may be converted to assembly language routines quite simply, or left as Basic routines.

First of all we will do a simple write to address 37E0H. This opeation should select a drive and turn on the disk drive motor for about three seconds.

```
POKE 14304,0        37E0H address
```

The above command stores a zero value in the four bits of Z36, selecting no drive, and simultaneously turns the motor of the dirve on for about three seconds. A value other than zero could be used to select the drive; a 1, 2, 4 or 8 would select drive 0, 1, 2 or 3 respectively. The only effect of the select would be to bring down the select line for the appropriate drive.

Now put a *protected* diskette in the drive. The following routine will select drive 1 and then read the status from the drive. As long as the drive is selected (the select bits are cleared when Z29 and the motor goes off), we will get back status from drive 1. The status should tell us that the diskette is protected (see chapter 3), and should also tell us if sector 0 has just passed the index pulse detector. Since sector zero comes around every 200 milliseconds or so, we'll not see the sector zero status every time due to the timing of the Basic loop and the short "window" for sector 0. When the drive is "deselected" after three seconds, a status of 128 will be printed, indicating a "not ready" condition. Status during the selected time will be 68 or 64, indicating write protect and possibly that the head is over track zero.

```
100  POKE 14304,1                          select
200  A=PEEK(14316)                         get status
250  IF (A AND 2)=2 PRINT"SECTOR 0)"       test sector 0
300  PRINT A                               print
400  GOTO 200                              loop on status
```

Now take off the protect label or put an unprotected diskette in the drive and repeat the above program. Status during the select time should indicate disk not protected (0 or 4).

Now we will try a more advanced operation. The following program selects the drive, does a restore to move the head to track 0, reads back the track register and prints the value (should be zero), steps in the head one track, reads the track register and prints it (should be 1), and loops

back to the track register read. The command in 200 does a restore at a slow stepping rate and the command in 600 does a step-in with automatic update and slow stepping rate. The status check at 300 and 400 tests to see whether the disk is busy or whether it is done performing the restore. This busy check will be found on virtually every command. Value 14317 addresses the track register in the PEEK (a read).

```
100  POKE 14304,1                          select
200  POKE 14316,3                          restore
300  A=PEEK(14316)                         get status
400  IF (A AND 1)<>0 GOTO 300              test busy
500  A=PEEK(14317)                         get track register
550  PRINT A                               print
570  FOR I =0 TO 300:NEXT                  wait for display
600  POKE 14316,83                         step-in
700  A=PEEK(14317)                         get track register
800  PRINT A                               print
900  GOTO 700                              loop
```

In the above program the FD1771B-01 automatically incremented the track register when the step-in was performed. Unless the update bit is specified, that update will not be done. The following program steps in from track 0 after a restore and prints the track register each time. A delay is put in at 760 for display purposes.

```
100  POKE 14304,1                          select
200  POKE 14316,3                          restore command
300  A=PEEK(14316)                         get status
400  IF (A AND 1)<>0 GOTO 300              test busy
500  A=PEEK(14317)                         get track register
550  PRINT A                               print
570  FOR I=O TO 300:NEXT                   delay for display
600  POKE 14304,1                          select
610  POKE 14316,83                         step-in
700  A=PEEK(14316)                         get status
750  IF (A AND 1)<>0 GOTO 700              test busy
760  FOR I=0 TO 300:NEXT                   delay for display
770  A=PEEK(14317)                         get track register
800  PRINT A                               print
900  IF A < 35 GOTO 600                    keep on steppin'
```

To observe what happens when the update bit is not set in the step-in command, substitute a 67 for the 83 in 610. Don't let this continue too long, however, as some types of drives have been known to step off the end of the earth (or at least the step mechanism); a few steps are fine however, and will illustrate the update of the track.

The stepping rate used for the SA400 is 40 milliseconds, specified by a binary 11 in the step rate field in the positioning commands. This is the rate that should be used normally. For illustration of this field's use however, the following program may be used to step in at a faster rate. Use 80 for a fast rate and 83 for a slow rate in line 610.

```
100  POKE 14304,1                        select
200  POKE 14316,3                        restore
300  A=PEEK(14316)                       get status
400  IF (A AND 1)< >0 GOTO 300           test busy
600  POKE 14304,1                        select
610  POKE 14316,80                       step-in
700  B=PEEK(14316)                       get status
770  A=PEEK(14317)                       get track
800  PRINT A                             print
900  IF B < 35 GOTO 600                  loop
```

The above programs have used a step in command. Let's use the step out command and clean up some of the code. The following program uses subroutine 1000 as a busy check. A return is made only when the previous command is done. Subroutine 2000 performs an operation determined by V, waits for done, and then prints the track register. The program steps into track 35 and then steps out to track 0.

```
100  POKE 14304,1                        select
200  POKE 14316,3                        restore
300  GOSUB 1000                          test done
400  V=83                                step-in
500  GOSUB 2000                          output and test
600  IF A < 35 GOTO 500                  step in till end
700  V=115                               step-out
800  GOSUB 2000                          output and test
900  IF A< >0 GOTO 800                   step out till end
950  END                                 done
1000 A=PEEK(14316)                       get status
1100 IF (A AND 1) <>0 GOTO 1000          go if busy
1200 RETURN                              return
2000 POKE 14304,1                        select
2050 POKE 14316,V                        output command
2100 GOSUB 1000                          test done
2200 A=PEEK(14317)                       get track register
2300 PRINT A                             print
2400 RETURN                              return
```


One question the reader may be asking - why select each time a step is done? Don't forget that the motor stays on only for about three seconds. If the entire disk operation takes longer than that, the drive is automatically deselected. It is a good idea therefore, to select before each new disk operation (command) to keep the disk in a ready condition. This applies not only to Basic code, but to assembly language code as well.

We have seen the step in and step out command used. Now let us use the step from last direction. As you will recall, this command will cause the FD1771B-01 to step in the same direction as the previous command.

```
100  POKE 14304,1                        select
200  POKE 14316,3                        restore
300  GOSUB 1000                          test done
400  POKE 14316,83                       step-in
500  GOSUB 1000                          test done
600  FOR I=1 TO 34                       setup loop
650  POKE 14304,1                        select
700  POKE 14316,51                       step from last dir.
800  GOSUB 1000                          test done
850  A=PEEK(14317)                       get track
860  PRINT A                             print
900  NEXT                                loop
950  END                                 done
1000 A=PEEK(14316)                       get status
1100 IF (A AND 1)< >0 GOTO 1000          test done
1200 RETURN                              return
```

The only other positioning command we have not used is seek. The following program seeks from any input value. Obviously valid values are 0 through 34 for the track. The seek assumes that the data register of the FD1771B-01 has been loaded with the value representing the desired track. This is done at 500. When the seek command is executed at 700 the FD1771B-01 automatically steps in or out to position the head over the desired track. The track register must of course, have the correct current track number for a proper seek.

```
100  POKE 14304,1                        select
200  POKE 14316,3                        restore
300  GOSUB 2000                          test done
350  INPUT V                             input track #
400  POKE 14304,1                        select
500  POKE 14319,V                        output track #
```

```
 600  GOSUB 2000                          test done
 700  POKE 14316,19                       seek command
 800  GOSUB 2000                          test done
 900  A=PEEK(14317)                       get track
1000  PRINT A                             print
1050  A=PEEK(14316)                       get status
1060  PRINT A                             print
1100  GOTO 350                            loop on input
2000  A=PEEK(14316)                       get status
2100  IF (A AND 1)<>0 GOTO 2000           test done
2200  RETURN                              return if done
```

The above routines illustrate the use of the positioning commands. Assembly language implementation would follow the above approaches (we shall see assembly language routines later in this chapter). The important thing in the use of the positioning commands is to know where you are. The reference point is track zero to which the head may always be positioned. Although the FD1771B-01 should not fail to update the track register properly, that gaelic law applies.

Reading/Writing

To illustrate reading and writing of sectors, we will use code from a popular microcomputer. The code is unsophisticated and can be accessed using a symbolic disassembler such as Small Systems Software's RSM-1S.

One of the first things that Level II Basic does is to test whether a disk is present. If a disk *is* present, then Level II assumes that a program is present on sector 0, track 0, and reads that program in. The program, of course, is a portion of TRSDOS. The process of reading in the program is called "bootstrapping" or "booting in", and is a way for the system to pull itself up by it's own bootstraps after power up or RESET. If a disk is not present, of course, or if BREAK is pressed on power up, Level II goes instead to Level II Basic, bypassing the disk.

If we examine the first three instructions in Level II Basic we find:

```
0000  F3          DI          disable interrupts
0001  AF          XOR  A      0 to A
0002  C3 74 06     JP  0674    jump to loc 0674H
```

This sequence is always entered on power up, as a power up causes execution to start at location 0. The first instruction disables all external interrupts except for the non-maskable interrupt. The A register is loaded with 0 and a jump is then made to loaction 0674H.

At location 0674H we find the "disk boot" routine shown below in figure 1. The disk boot starts at 0696H after other initialization. The disk boot performs the following functions:

1. Read disk status (0696H)
2. Test for status bits from disk (069AH)
3. If no status for disk, go to location 0075H (069CH)
4. Else: Select drive 1 by outputting 1 to select latch address 37E1H (06A1H).
5. Send restore command (3) to disk command register (address 37EC - 06AAH).
6. Delay 65536 counts (0060 is a subroutine to delay by count in BC register pair). This is done to let the motor come up to speed and for head movement.
7. Test busy bit for status (06B2H).
8. Loop to 7 if busy.
9. Else: Send 0 to sector register by outputting to location 37EEH. This prepares the disk to read sector 0 of track 0.
10. Send 8CH read command by outputting 8C to location 37EC (06BFH). Read sector 0, track 0, byte 0.
11. Test bit 1 of the status register (DRQ). If DRQ is not zero, the next byte of data is not present in the FD1771B-01 data register and step 11 is again executed. Else: Data is present and the data is read into the A register (06C4H) and transferred to the 4200H area (06C5H).
12. The BC pointer is bumped by one (06C4H) to point to the next destination in the 4200 area (06C6H).
13. If C=0, steps 11 through 13 are repeated to transfer 256 bytes of data from sector 0 to 4200H through 42FFH.
14. A jump is made to TRSDOS start at 4200H (actually a TRSDOS loader).

Note that in the above routine all I/O is done on a "programmed I/O" basis. Reading a byte of data is done by checkding the DRQ bit to see if a new byte of data has been assembled from the serial bit stream. If so, a byte is read into A and then transferred to the 4200H area (200H bytes above the start of RAM. The read *resets* the DRQ bit and the process is repeated 256 times to read the sector of data.

This simple bootstrap is a common brute force approach to initializing a system. No checks are made on the validity of the data in this routine, but chances are good the operation will go off without a hitch. If not, another power up will repeat the entire process.

If a RESET is performed, the same test for disk present is made at location 66H and a jump made to location 0000H if the disk is there, where a transfer to the bootstrap routine is made.

```
0066  31 00 06     LD SP,0600          reset
0069  3A EC 37     LD A,(37EC)         get status
006C  3C           INC A
006D  FE 02        CP 02               test for disk status
006F  D2 00 00     JP NC,0000          go if disk is there
```

The read command in the above code is an 8CH. As previously described, the read fields should be set for a single sector (bit 4=0, IBM format (bit 3=1), and enable head load and 10 millisecond delay (bit 2=1). This command code should always be used, with the exception of reading multiple sectors. Note also in the above code that it does take time for the motor to get up to speed (less than a second) and that this should be a time-out in the program. Continuing disk operations should always perform a select to keep the motor on.

Figure 1. Disk Bootstrap Routine

```
0696 3AEC37          LD      A,(37ECH)  ;GET DISK STATUS
0699 3C              INC     A
069A FE02            CP      02
069C DA7500          JP      C,0075H    ;GO IF NO DISK
069F 3E01            LD      A,01       ;FOR DRIVE 1
06A1 32E137          LD      (37E1H),A  ;SELECT DRIVE 1
06A4 21EC37          LD      HL,37ECH   ;1771 ADDRESS CMDS
06A7 11EF37          LD      DE,37EFH   ;DATA REG ADDRESS
06AA 3603            LD      (HL),03    ;RESTORE COMMAND
06AC 010000          LD      BC,0       ;DELAY 64K COUNTS
06AF CD6000          CALL    0060H
06B2 CB46   LOOP1    BIT     0,(HL)     ;TEST BUSY
06B4 20FC            JR      NZ,LOOP1   ;GO IF STILL BUSY
06B6 AF              XOR     A          ;ZERO TO A
06B7 32EE37          LD      (37EEH),A  ;0 TO SECTOR REG
06BA 010042          LD      BC,4200H
06BD 3E8C            LD      A,8CH      ;READ COMMAND
06BF 77              LD      (HL),A     ;READ SECTOR 0
06C0 CB4E   LOOP2    BIT     1,(HL)     ;TEST DRQ
06C2 28FC            JR      Z,LOOP2    ;GO IF DATA NOT AVAIL.
06C4 1A              LD      A,(DE)     ;GET NEXT BYTE
06C5 02              LD      (BC),A     ;TRANSFER DATA
06C6 0C              INC     C          ;BUMP BUFFER POINTER
06C7 20F7            JR      NZ,LOOP2   ;GO IF NOT 256 BYTES
06C9 C30042          JP      4200H      ;TRANSFER TO DOS LOADER
```

Figure 2 shows the busy loop of another read. This read tests the DRQ as the previous one, but prior to the test of DRQ also tests the busy bit. The busy bit will be reset when the read operation is completed. For a single sector this will be after 256 bytes, while for multiple sectors this will be at the end of the track. If the busy bit is set, the next byte of data is obtained and stored away. If the busy bit is not set, a final status check is

made and the value of 5C is used to test for type, not found, CRC or lost data status. Any of these bits means that an error condition exists and the read was terminated improperly. In this case a force interrupt is used to reset the controller before either return is made. The type bit means that a data address mark other than FB or F9 was encountered, not found indicates track and sector were not found, CRC indicates invalid data, and lost data means that the CPU did not respond to the next byte of data.

Figure 2. Generalized Disk Read

(At start, (HL)=37ECH and (BC)=buffer pointer)

```
52C6 11EF37          LD      DE,37EFH   ;DATA REGISTER ADDRESS
52C9 C5              PUSH    BC         ;WASTE TIME
52CA C1              POP     BC         ;...AND SOME MORE
52CB 180B            JR      52D8H      ;START READ
52CD 0F      BSYTS   RRCA               ;BUSY BIT TO C
52CE 300A            JR      NC,52DAH   ;GO IF NOT BUSY
52D0 7E      LOOP    LD      A,(HL)     ;GET STATUS
52D1 CB4F            BIT     1,A        ;TEST DRQ
52D3 28F8            JR      Z,BSYTS    ;GO IF NO DATA
52D5 1A              LD      A,(DE)     ;GET ONE BYTE
52D6 02              LD      (BC),A     ;STORE IN BUFFER
52D7 03              INC     BC         ;BUMP BUFFER POINTER
52D8 18F6            JR      LOOP       ;LOOP FOR NEXT BYTE
52DA 7E              LD      A,(HL)     ;GET STATUS
52DB E65C            AND     5CH        ;TEST TYPE,NFND,CRC,LOST
52DD C8              RET     Z          ;GO IF NO ERRORS
52DE 36D0            LD      (HL),0D0H  ;FORCE INT (RESET)
52E0 C9              RET                ;RETURN
```

At this point it may be wise to talk about some of the idiosyncracies of the TRS-80 disk implementation. Although nothing is mentioned in the FD1771B-01 documentation, code in the Radio Shack system uses "time wasting instructions" after certain disk I/O commands. Evidentally some settling time is required after execution of FD1771B-01 commands. This is a type of thing usually found out after debugging, and may reflect specs not stated in FD1771B-01 documentation, system design problems, or code put in "just to play it safe". A sequence such as:

```
PUSH   AF
POP    AF
PUSH   AF
POP    AF
```

is typically used after a seek command. These instructions are essentially no operations and prevent another command from directly following an initial disk command. Additional time wasters may be

required after steps, or read or write commands. Another problem associated with the TRS-80 system is that some of the code associated with the disk is presumably deliberately vague. Therefore, do not be too surprised to find confusing routines or structure while disassembling or investigating parts of TRSDOS.

The operations for disk sector writes is handled much the same way as a read, except that the data register is now *loaded* with a data byte, and then the DRQ is checked to determine if the controller is ready for the next data byte. In the routine in figure 3, the busy bit is check as previously; it signifies the end of the write. After the write has been completed, the same status bits are checked. Bit 6 in the write case signifies write protect instead of record type as in the case of read.

Figure 3. Generalized Disk Write
(BC contains buffer pointer)
(HL contains status, command address 37EC)

```
FE61 7E       SLOOP   LD      A,(HL)      ;GET STATUS
FE62 0F               RRCA                ;BUSY BIT TO C
FE63 DA61FE           JP      C,SLOOP     ;GO IF BUSY
FE66 36A8             LD      (HL),0A8H   ;WRITE COMMAND
FE68 11EF37           LD      DE,37EFH    ;DATA REG ADDRESS
FE6B C5               PUSH    BC          ;WASTE TIME
FE6C C1               POP     BC          ;...AND SOME MORE
FE6D C5               PUSH    BC          ;...MORE YET
FE6E C1               POP     BC          ;...DONE WASTIN' TIME
FE6F C376FE           JP      DRQCK       ;BYPASS BUSY CHECK
FE72 0F       BSYCK   RRCA                ;BUSY TO C
FE73 D282FE           JP      NC,DONE     ;GO IF NOT BUSY
FE76 7E       DRQCK   LD      A,(HL)      ;GET STATUS
FE77 CB4F             BIT     1,A         ;TEST DRQ
FE79 CA72FE           JP      Z,BSYCK     ;GO IF NEW DATA NOT RE(
FE7C 0A               LD      A,(BC)      ;GET NEXT DATA BYTE
FE7D 12               LD      (DE),A      ;OUTPUT TO DATA REG
FE7E 03               INC     BC          ;BUMP BUFFER POINTER
FE7F C376FE           JP      DRQCK       ;CONTINUE
FE82 7E       DONE    LD      A,(HL)      ;GET STATUS
FE83 E65C             AND     5CH         ;TEST W PRO, N FND, CRC
FE85 C8               RET     Z           ;  LOST
FE86 36D0             LD      (HL),0D0H   ;OUTPUT FORCE INT (RESE
FE88 C9               RET                 ;RETURN
```


Formatting

Formatting a diskette can be performed by one of two methods. The TRS-80 software contains a disk formatting program that automatically formats a diskette to "standard" format as defined in Appendix B. User-formatted diskettes, on the other hand, do not contain any directories or applications programs and cannot be used by TRSDOS. A user-formatted disk, of course, can be used for any user file management software which operates independently of TRSDOS.

If the user wishes to format a diskette, he may do so by constructing a track's worth of data arranged in the proper format and then executing a write track command. The implementation of the write track will be almost identical to a write sector sequence, except that an entire track is written.

If the user wishes to read a track, the read track command can be implemented in the same fashion as read sector. In this case however, the entire track will have to be read into a large buffer before the busy bit is reset. All data on the track including gaps, data marks, and data is read in, so this is a convenient way (though laborious) of investigating data on any diskette track. A program to read any given track is shown below in figure 4 (track number in location 5043).

The read address command operates similarly to a read sector except that the six data bytes of an ID field are read from a track. The read address would primarily be used for verification of a formatting operation.

Figure 4. Read Track Program

```
5000 F3               DI                  ;DISABLE INTERRUPTS
5001 3E01             LD      A,01        ;FOR SELECT
5003 32E137           LD      (37E1H),A   ;SELECT DRIVE 1
5006 210000           LD      HL,0        ;DELAY COUNT
5009 2D       LOOP1   DEC     L
500A C20950           JP      NZ,LOOP1
500D 25       LOOP2   DEC     H
500E C20D50           JP      NZ,LOOP2
5011 32E137           LD      (37E1H),A   ;SELECT AGAIN
5014 3A4350           LD      A,(TRACK)   ;LOAD TRACK #
5017 32EF37           LD      (37EFH),A   ;OUTPUT TO DATA REG
501A 21EC37           LD      HL,37ECH    ;STATUS, CMD ADDRESS
501D 361B             LD      (HL),1BH    ;SEEK COMMAND
501F F5               PUSH    AF          ;WASTE TIME
5020 F1               POP     AF          ;NOTE: DRIVE SELECT
5021 F5               PUSH    AF          ;POSITIONS TRACK TO
5022 F1               POP     AF          ;0, RESETS TRACK REG
```

```
5023 7E          LD      A,(HL)      ;GET STATUS
5024 0F          RRCA                ;BUST TO C
5025 DA2350      JP      C,5023H     ;LOOP IF BUSY
5028 010060      LD      BC,6000H    ;BUFFER AREA
502B 36E4        LD      (HL),0E4H   ;READ TRACK COMMAND
502D C5          PUSH    BC          ;WASTE TIME
502E C1          POP     BC          ;...AND SOME
502F C5          PUSH    BC          ;...MORE YET
5030 C1          POP     BC          ;...DONE WASTIN' TIME
5031 7E    LOOP3 LD      A,(HL)      ;GET STATUS
5032 0F          RRCA                ;BUSY TO C
5033 D20342      JP      NC,4203H    ;RETURN TO MONITOR
5036 CB47        BIT     0,A         ;  ON DONE
5038 CA3150      JP      Z,LOOP3     ;LOOP IF NOT DATA
503B 3AEF37      LD      A,(37EFH)   ;READ DATA
503E 02          LD      (BC),A      ;STORE IN BUFFER
503F 03          INC     BC          ;BUMP BUFFER POINTER
5040 C33150      JP      LOOP3       ;RTN FOR NEXT BYTE
5043 00    TRACK DEFB    0           ;TRACK #
```

Conclusion

While the above examples do not constitute a complete description of every disk operation, it is hoped that they will provide the reader with the basic knowledge to write his own assembly language software routines for the disk if he desires. By bypassing TRSDOS it is possible to optimize disk storage space and access times, and to create whatever file management or disk operating system softweare the user requires, subject to his time and patience.

Appendix A

FD1771B-01 Commands for TRS-80

TYPE	COMMAND	BINARY	HEX*	FLAGS
I	Restore	00000V11	03	V: 0 no verify, 1 verify
I	Seek	00010V11	13	V: 0 no verify, 1 verify
I	Step	001U0V11	33	V: 0 no verify, 1 verify U: 0 no update, 1 update track register
I	Step In	010U0V11	53	V: 0 no verify, 1 verify U: 0 no update, 1 update track register
I	Step Out	011U0V11	73	V: 0 no verify, 1 verify U: 0 no update, 1 update track register
II	Read	100M1100	8C	M: 0 single sector, 1 multiple
II	Write	101M1100	AC	M: 0 single sector, 1 multiple
III	Read Address	11000100	C4	
III	Read Track	1110010S	E4	S: 0 synchronize to address mark, 1 no synchronize
III	Write Track	11110100	F4	
IV	Force Interrupt	11010000	D0	

* Usual value used.

Status

Bit	Typel	Read	Write	Read Address	Read Track	Write Track
7	Not Ready	Not Ready	Not Ready	Not Ready	Not Ready	Not Ready
6	Write protect	Record type	Write protect	0	0	Write protect
5	Head engag'd	Record type	Write fault	0	0	Write fault
4	Seek error	Record not found	Record not found	ID not found	0	0
3	CRC error	CRC error	CRC error	CRC error	0	0
2	Track 0	Lost data	Lost data	Lost data	Lost data	Lost data
1	Index	DRQ	DRQ	DRQ	DRQ	DRQ
0	Busy	Busy	Busy	Busy	Busy	Busy

Appendix B

Disc Format for TRS-80 [3] [4]

	WRITING			READING		
	#	Pattern	Description	#	Pattern	Description
Precedes First Sector	18	FF	Filler	18	FF	Filler
	1	FE	ID address mark	1	FE	ID address mark
	1	xx	Track # (0-9)	1	xx	Track #
	1	00		1	00	
	1	xx[2]	Sector # (0-9)	1	xx[2]	Sector #
First Sector	1	01		1	01	
	1	F7[1]	Generates 2 CRC bytes	2	xx	CRC chars (ID)
	12	FF	Filler	12	FF	Filler
	6	00	Filler	6	00	Filler
	1	FB	Data address mark	1	FB	Data addr. mark
	256	E5	For user data area	256	E5	Dummy user area
	1	F7[1]	Generates 2 CRC bytes	2	xx	CRC chars (data)
	12	FF	Filler	12	FF	Filler
	6	00	Filler	6	00	Filler
Physical Sectors 2-9	Repeat sector data 8 more times			8 more sectors read		
	1	FE		1	FE	
	1	xx		1	xx	
	1	00		1	00	
	1	xx[2]		1	xx[2]	
Last Sector	1	01	Same as above	1	01	Same as above
	1	F7[1]		2	xx	
	12	FF		12	FF	
	6	00		6	00	
	1	FB		1	FB	
	256	E5		256	E5	
	1	F7[1]		2	xx	
	106[5]	FF	Write until not busy	106[5]	FF	Read 'til not busy

NOTES:

[1] F7 character generates two CRC check characters in hardware.
[2] Physical sector numbers on diskette go in this order: 0, 5, 1, 6, 2, 7, 3, 8, 4, 9 (i.e. third sector of track is 1)
[3] Above defines one track.
[4] Above defines track without files. Data files are rewritten for 256 data bytes per sector.
[5] Approximate number of bytes before controller terminates operation.

Appendix C

DISKIO

Description

DISKIO is an assembly language program that is used to read and write data from one to four disks in a TRS-80 program. A software routine such as this is commonly called a "driver" and DISKIO is essentially a "disk I/O driver". The user may call DISKIO from another assembly language program, or from a Basic program using the USR function. DISKIO is oriented towards reads and writes of single sectors or tracks, but also provides the capability of head positioning, status check, or address read. DISKIO is entirely relocatable. That is, the machine language code from "DISKIO" to "end" may be moved bodily anywhere in memory without reassembly or modification of instruction address fields. Alternatively, of course, the routine may be reassembled at any convenient origin.

Calling Sequence

DISKIO is called by loading the index register IX with the address of a parameter block and by performing a CALL to DISKIO. Return is made to the location immediately following the CALL.

```
(IX) = parameter block pointer
CALL   DISKIO
(return)
```

The paramter block consists of eight bytes, specifying eight parameters for the DISKIO routine.

The first byte contains a function code of 0 through 6. Functions are:

0 Read status
1 Position head over given track
2 Read sector specified into buffer
3 Write sector specified from buffer
4 Read address (ID) data from given track
5 Read entire specified track
6 Write entire specified track

Byte 1 contains a sector number from 0 to 9 for functions 2 and 3 (read and write sector).

Byte 2 contains a track number from 0 through 34 for all functions except the read status function.

Bytes 3 and 4 contain a 16 bit address for functions 2 through 6. This address is the address of the data source for writing *to* the disk, or the data destination for reading *from* the disk. Note that the Z-80 uses the

low-order 8 bits in the first byte and the high-order 8 bits in the next byte. The assembler takes care of this automatically, and this information is meant for those users who may be "patching" machine code.

Byte 5 is not set up by the user, but does contain a "type of completion" after DISKIO has returned to the calling code. Upon return, a 01H value specifies that the function was performed without error. An FEH value ((-2) specifies that one of the calling paramaters was invalid - sector #, track #, function # or drive #. An FFH(-1) specifies that an error occurred during the disk operation. The status code (byte 6) may be checked to determine the type of error in this case. An FDH (-3) specifies that a position error occurred. The position error may have been prior to the main function, as in the case of functions 2 through 6, or a result of the position head function itself. In either case, the status byte (6) will show the nature of the error.

Byte 6 is not set up by the user and contains the status after the I/O operation has been performed. Status will be present after a successful I/O operation or in the case of an FFH or FDH type completion. Status is as follows:

Bit	Description
7	None
6	Write protect (functions 1, 3, 6)
5	Write fault (functions 3, 6)
4	Seek error (1), ID not found (4), record not found (2, 3)
3	CRC (checksum) error (functions 1 - 4)
2	Track 0 (function 1), lost data (others)
1	Index mark (function 1)
0	None

Status for an FDH type of completion will be identical to that returned for function 1.

Byte 7 has two fields. Bit 7 is a user-specified "wait" bit. If the W bit is a one, DISKIO will wait approximately 1 second for the disk motor to come up to speed before performing the disk operation. If the W bit is a zero, the operation will be performed immediately without a wait. *The W bit must be set for the first disk operation.* It need not be set for subsequent operations provided that these operations occur less than two seconds apart. The reason for this constraint is that each disk select (address) sets the motor on signal for about three seconds. After this three second period the motor turns off again, unless there has been a new select in the three second period. Use the wait option for initial disk

operations and for those operations that do not occur within two seconds of the last disk I/O.

The second field in byte 7 is the drive number (0 through 3). Drive 0 corresponds to Radio Shack drive # 1, and so forth.

Functions

Function 0, read status, reads back the current status from the specified disk. Drive number must be specified.

Function 1, position head, positions the disk head over the specified track (seek or restore). Track number and drive number must be specified. *Initially a position head to track 0 (restore) should be done before other disk I/O operations.* A restore effectively resets the internal track register in the controller and can be done at any time to ensure the controller "knows where it is".

Function 2, read sector, positions the head to the indicated track number and then reads in one 256 byte sector into the specified buffer area. Sector number, track number, buffer address and drive number must be specified.

Function 3, write sector, operates similarly to read sector, except the data is transferred from the buffer are to the specified track and sector. Sector number, track number, buffer address and drive number must be specified.

Function 4, read address ID, reads the six identification bytes from the *first available* sector of the given track into the specified buffer area. The format is track number, zeros, sector number, 01H, and two bytes of CRC (checksum) data. Sector number is not specified. Track number, buffer address and drive number must be specified.

Function 5, read track, reads in the entire specified track into the buffer area. Approximately 3106 bytes must be made available as the buffer area. Gaps, ID data, address marks, data marks, and user data is read. Track number, buffer address and drive number must be specified.

Function 6, write track, writes an entire track from the specified buffer area. The track data must be properly formatted because of hardware limitations in the controller. This operation is a "formatting" operation for initialization of the disk. Track number, buffer address and drive number must be specified.

Examples

Position Head to Track 0

```
        LD      IX,PARAM    ;parameter block address
        CALL    DISKIO      ;call disk I/O
        return              ;(other code)

PARAM   DB      1           ;position function
        DB      0           ;sector # not required
        DB      0           ;track 0
        DB      0           ;buffer address not required
        DW      1
        DB      0           ;zero TYC for return*
        DB      0           ;zero status for return*
        DB      80H         ;wait, drive 0
```

Read Sector 8, Track 13 (decimal)

```
        LD      IX,PARAM    ;parameter block address
        CALL    DISKIO      ;call disk I/O
        return              ;(other code)

PARAM   DB      2           ;read sector function
        DB      8           ;sector 8
        DB      0DH         ;track 13
        DW      BUFFER      ;buffer address
        DB      0           ;zero TYC for return*
        BD      0           ;zero status for return*
        DB      81H         ;wait, drive 1
```

* Not required

Notes:

1. Common functions used are position, read sector, and write sector.
2. Additional information on disk operations appears earlier in this book.
3. All registers are saved for return.
4. Should you experience trouble with DISKIO, the problem may be incorrect code. If all code in DISKIO is correct, the checksum obtained by adding the contents of byte 0 through 139H in the program should be 6CH. If the checksum is not 6CH, code has been incorrectly entered manually or an error has been made in the source code for assembly. (The checksum is obtained by zeroing an 8 bit total, and then adding byte 0, byte 1....through byte 139H. The resulting 8 bit value is the check sum, and should be 6CH.)

```
5000            00100           ORG     5000H
                00110 ;*****************************************
                00120 ;*                                       *
                00130 ;*               DISKIO-00-01            *
                00140 ;*                                       *
                00150 ;*****************************************
0000            00160 CODE0     EQU     0000H
0100            00170 CODE1     EQU     0100H
078C            00180 CODE2     EQU     078CH
0FAC            00190 CODE3     EQU     0FACH
05C4            00200 CODE4     EQU     05C4H
05E4            00210 CODE5     EQU     05E4H
0DF4            00220 CODE6     EQU     0DF4H
0000            00230 STAT0     EQU     0
0018            00240 STAT1     EQU     18H
001C            00250 STAT2     EQU     1CH
007C            00260 STAT3     EQU     7CH
001C            00270 STAT4     EQU     1CH
0004            00280 STAT5     EQU     04H
0044            00290 STAT6     EQU     44H
5000 F5         00300 DISKIO    PUSH    AF              ;SAVE REGS
5001 C5         00310           PUSH    BC
5002 D5         00320           PUSH    DE
5003 E5         00330           PUSH    HL
5004 AF         00340           XOR     A               ;ZERO A
5005 DD7705     00350           LD      (IX+5),A        ;ZERO TYC
5008 DD7706     00360           LD      (IX+6),A        ;ZERO STATUS
500B DD7E00     00370           LD      A,(IX)          ;GET FUNCTION
500E E6F8       00380           AND     0F8H
5010 2073       00390           JR      NZ,INVAL2       ;GO IF INVALID FUNCTIC
5012 DD7E00     00400           LD      A,(IX)
5015 FE07       00410           CP      7
5017 286C       00420           JR      Z,INVAL2        ;GO IF FUNCTION 7
5019 DD7E07     00430           LD      A,(IX+7)
501C E67C       00440           AND     7CH             ;GET DRIVE
501E 2065       00450           JR      NZ,INVAL2       ;GO IF GT 3
5020 DD7E07     00460           LD      A,(IX+7)
5023 E603       00470           AND     3
5025 3C         00480           INC     A
5026 47         00490           LD      B,A
5027 3E80       00500           LD      A,80H
5029 07         00510 DISK0A    RLCA
502A 10FD       00520           DJNZ    DISK0A          ;CONVERT TO POSITION
502C 32E037     00530           LD      (37E0H),A       ;SELECT
502F DDCB077E   00540           BIT     7,(IX+7)        ;TEST W
5033 2809       00550           JR      Z,DISK01        ;GO IF NO WAIT
5035 210000     00560           LD      HL,0000
5038 25         00570 DISK0B    DEC     H
5039 20FD       00580           JR      NZ,DISK0B
503B 2D         00590           DEC     L
503C 20FA       00600           JR      NZ,DISK0B       ;DELAY 64K COUNTS
503E 3AEC37     00610 DISK01    LD      A,(37ECH)       ;GET STATUS
5041 CB47       00620           BIT     0,A             ;TEST BUSY
5043 20F9       00630           JR      NZ,DISK01       ;GO IF BUSY
5045 DD7E00     00640 DISK02    LD      A,(IX)          ;GET FUNCTION 0-6
5048 010000     00650           LD      BC,CODE0        ;LOAD SEQ COMMAND
504B 1E00       00660           LD      E,STAT0         ;STATUS CHECK BITS
504D FE00       00670           CP      0
504F 2836       00680           JR      Z,DISK03
5051 010001     00690           LD      BC,CODE1
5054 1E18       00700           LD      E,STAT1
5056 FE01       00710           CP      1
5058 282D       00720           JR      Z,DISK03
505A 018C07     00730           LD      BC,CODE2
505D 1E1C       00740           LD      E,STAT2
505F FE02       00750           CP      2
5061 2824       00760           JR      Z,DISK03
5063 01AC0F     00770           LD      BC,CODE3
5066 1E7C       00780           LD      E,STAT3
5068 FE03       00790           CP      3
```



```
506A 281B       00800           JR      Z,DISK03
506C 01C405     00810           LD      BC,CODE4
506F 1E1C       00820           LD      E,STAT4
5071 FE04       00830           CP      4
5073 2812       00840           JR      Z,DISK03
5075 01E405     00850           LD      BC,CODE5
5078 1E04       00860           LD      E,STAT5
507A FE05       00870           CP      5
507C 2809       00880           JR      Z,DISK03
507E 01F40D     00890           LD      BC,CODE6
5081 1E44       00900           LD      E,STAT6
5083 1802       00910           JR      DISK03
5085 1834       00920 INVAL2    JR      INVAL1
5087 CB40       00930 DISK03    BIT     0,B             ;GET TRACK BIT
5089 2847       00940           JR      Z,DISK08        ;GO IF ZERO
508B DD7E02     00950           LD      A,(IX+2)        ;GET TRACK #
508E E6E0       00960           AND     0E0H
5090 280F       00970           JR      Z,DISK04        ;GO IF 0-31
5092 DD7E02     00980           LD      A,(IX+2)
5095 FE20       00990           CP      32
5097 2808       01000           JR      Z,DISK04        ;GO IF 32
5099 FE21       01010           CP      33
509B 2804       01020           JR      Z,DISK04        ;GO IF 33
509D FE22       01030           CP      34
509F 201A       01040           JR      NZ,INVAL1       ;GO IF GT 34
50A1 DD7E02     01050 DISK04    LD      A,(IX+2)        ;TRACK #
50A4 B7         01060           OR      A
50A5 2004       01070           JR      NZ,DISK05       ;GO IF NOT RESTORE
50A7 3E03       01080           LD      A,3             ;RESTORE COMMAND
50A9 1807       01090           JR      DISK06
50AB 32EF37     01100 DISK05    LD      (37EFH),A       ;OUTPUT TRK #
50AE C5         01110           PUSH    BC              ;WASTE TIME
50AF C1         01120           POP     BC
50B0 3E17       01130           LD      A,17H           ;SEEK COMMAND
50B2 32EF37     01140 DISK06    LD      (37EFH),A       ;OUTPUT RESTORE OR SEEK
50B5 C5         01150           PUSH    BC              ;WASTE TIME
50B6 C1         01160           POP     BC
50B7 C5         01170           PUSH    BC
50B8 C1         01180           POP     BC
50B9 1802       01190           JR      DISK07
50BB 1879       01200 INVAL1    JR      INVAL
50BD 3AEC37     01210 DISK07    LD      A,(37ECH)       ;GET STATUS
50C0 CB47       01220           BIT     0,A             ;TEST BUSY
50C2 20F9       01230           JR      NZ,DISK07       ;GO IF BUSY
50C4 DD7706     01240           LD      (IX+6),A        ;STORE STATUS
50C7 E698       01250           AND     98H             ;TEST STATUS
50C9 2807       01260           JR      Z,DISK08        ;GO IF OK
50CB 3EFD       01270           LD      A,0FDH
50CD DD7705     01280           LD      (IX+5),A        ;SET POSITION ERROR
50D0 185F       01290           JR      DISK27          ;TERMINATE
50D2 CB48       01300 DISK08    BIT     1,B             ;GET SECTOR BIT
50D4 2817       01310           JR      Z,DISK10        ;GO IF 0
50D6 DD7E01     01320           LD      A,(IX+1)        ;GET SECTOR
50D9 E6F8       01330           AND     0F8H
50DB DD7E01     01340           LD      A,(IX+1)
50DE 2808       01350           JR      Z,DISK09        ;GO IF 0-7
50E0 FE08       01360           CP      8
50E2 2804       01370           JR      Z,DISK09        ;GO IF 8
50E4 FE09       01380           CP      9
50E6 204E       01390           JR      NZ,INVAL        ;GO IF GT9
50E8 32EE37     01400 DISK09    LD      (37EEH),A       ;OUTPUT TO SECTOR REG
50EB C5         01410           PUSH    BC
50EC C1         01420           POP     BC
50ED CB50       01430 DISK10    BIT     2,B             ;GET READ/WR BIT
50EF D5         01440           PUSH    DE              ;SAVE STATUS CHK BITS
50F0 282E       01450 DISK12    JR      Z,DISK20        ;GO IF NO I/O
50F2 79         01460           LD      A,C             ;GET COMMAND
50F3 DD5E03     01470           LD      E,(IX+3)        ;BUFFER ADDRESS LS
50F6 DD5604     01480           LD      D,(IX+4)        ;MS
50F9 21EC37     01490           LD      HL,37ECH        ;STATUS REG ADDRS
```

```
50FC 77       01500          LD      (HL),A          ;OUTPUT COMMAND
50FD CB58     01510          BIT     3,B             ;TEST RD/WR
50FF C5       01520          PUSH    BC
5100 C1       01530          POP     BC              ;WASTE TIME
5101 C5       01540          PUSH    BC
5102 C1       01550          POP     BC
5103 01EF37   01560          LD      BC,37EFH        ;DATA REG ADDRS
5106 280C     01570          JR      Z,DISK18        ;GO IF READ
5108 7E       01580 DISK15   LD      A,(HL)          ;GET STATUS
5109 0F       01590          RRCA                    ;BUSY TO C
510A 3014     01600          JR      NC,DISK20       ;GO IF DONE
510C 0F       01610          RRCA                    ;DRQ TO C
510D 30F9     01620          JR      NC,DISK15       ;GO IF NOT RDY
510F 1A       01630          LD      A,(DE)          ;GET BYTE
5110 02       01640          LD      (BC),A          ;OUTPUT TO DISK
5111 13       01650          INC     DE
5112 18F4     01660          JR      DISK15          ;CONTINUE
5114 7E       01670 DISK18   LD      A,(HL)          ;GET STATUS
5115 0F       01680          RRCA                    ;BUSY TO C
5116 3008     01690          JR      NC,DISK20       ;GO IF DONE
5118 0F       01700          RRCA                    ;DRQ TO C
5119 30F9     01710          JR      NC,DISK18       ;GO IF NOT RDY
511B 0A       01720          LD      A,(BC)          ;GET BYTE
511C 12       01730          LD      (DE),A          ;STORE
511D 13       01740          INC     DE
511E 18F4     01750          JR      DISK18          ;CONTINUE
5120 3AEC37   01760 DISK20   LD      A,(37ECH)       ;GET STATUS
5123 DD7706   01770          LD      (IX+6),A        ;SAVE
5126 C1       01780          POP     BC              ;RESTORE STATUS CHK B
5127 A1       01790          AND     C               ;TEST
5128 3E01     01800          LD      A,1
512A 2802     01810          JR      Z,DISK25        ;GO IF OK
512C 3EFF     01820          LD      A,0FFH
512E DD7705   01830 DISK25   LD      (IX+5),A        ;SET ERROR TYC
5131 E1       01840 DISK27   POP     HL
5132 D1       01850          POP     DE
5133 C1       01860          POP     BC
5134 F1       01870          POP     AF              ;RESTORE REGS
5135 C9       01880          RET
5136 3EFE     01890 INVAL    LD      A,0FEH
5138 18F4     01900          JR      DISK25
0000          01910          END
00000 TOTAL ERRORS
```

```
CODE0  0000 00160   00650
CODE1  0100 00170   00690
CODE2  078C 00180   00730
CODE3  0FAC 00190   00770
CODE4  05C4 00200   00810
CODE5  05E4 00210   00850
CODE6  0DF4 00220   00890
DISK01 503E 00610   00550 00630
DISK02 5045 00640
DISK03 5087 00930   00680 00720 00760 00800 00840 00880 00910
DISK04 50A1 01050   00970 01000 01020
DISK05 50AB 01100   01070
DISK06 50B2 01140   01090
DISK07 50BD 01210   01190 01230
DISK08 50D2 01300   00940 01260
DISK09 50E8 01400   01350 01370
DISK0A 5029 00510   00520
DISK0B 5038 00570   00580 00600
DISK10 50ED 01430   01310
DISK12 50F0 01450
DISK15 5108 01580   01620 01660
DISK18 5114 01670   01570 01710 01750
DISK20 5120 01760   01450 01600 01690
DISK25 512E 01830   01810 01900
DISK27 5131 01840   01290
```



```
DISKIO 5000 00300
INVAL  5136 01890   01200 01390
INVAL1 50BB 01200   00920 01040
INVAL2 5085 00920   00390 00420 00450
STAT0  0000 00230   00660
STAT1  0018 00240   00700
STAT2  001C 00250   00740
STAT3  007C 00260   00780
STAT4  001C 00270   00820
STAT5  0004 00280   00860
STAT6  0044 00290   00900
```