MICRO APPLICATIONS

TRS-80 DISC INTERFACING GUIDE

Micro Applications
24232 Tahoe Court
Laguna Niguel, CA 92677
~~(714) 831-7004~~

Revision 1

MICRO APPLICATIONS

TRS-80 DISC INTERFACING GUIDE

TABLE OF CONTENTS

Chapter 1

Disc Basics

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

A floppy disc system is made up of the disc drive or drives themselves, a controller, and the microcomputer. In our case the microcomputer is the Radio Shack TRS-80, the controller is the Western Digital FD1771B-01, and the disc drive is the Shugart SA400. A block diagram of the TRS-80 disc 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 disc(s) interprets commands sent to it over the data lines and translates these commands into disc-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 design! Commands are sent to the disc drive by the controller chip to perform functions for head positioning and reading and writing, and the "status" of the drive is returned back to the controller chip. We'll be talking about the operation of each of these component parts in future chapters, but for the time being let's concern ourselves with how the data is stored on the "diskette" and some of the physical attributes of the diskette and disc drive.

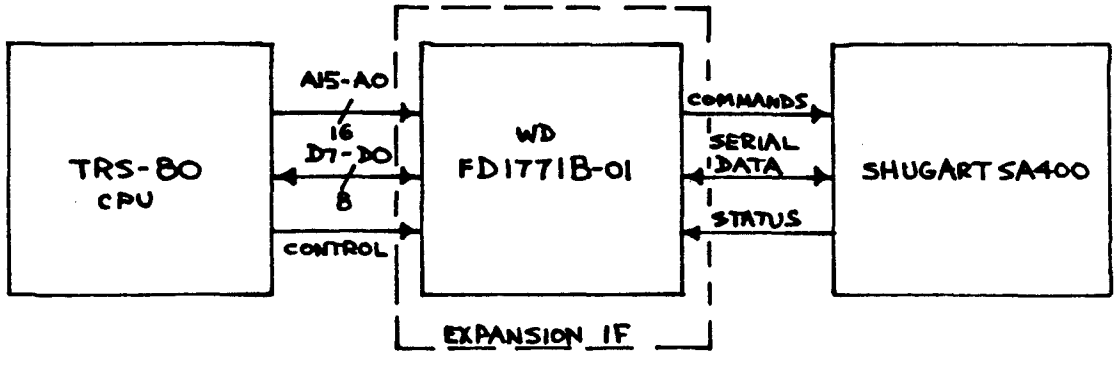


Figure 1. TRS-80 Disc

When the disc drive is mentioned in this text, we'll refer to the "disc" or "drive" or "disc 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 a 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 that spins the diskette and moves a recording head along a radius while the disc 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 disc 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 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 1/10 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 are the same, although the data is packed a little more tightly into the innermost tracks.

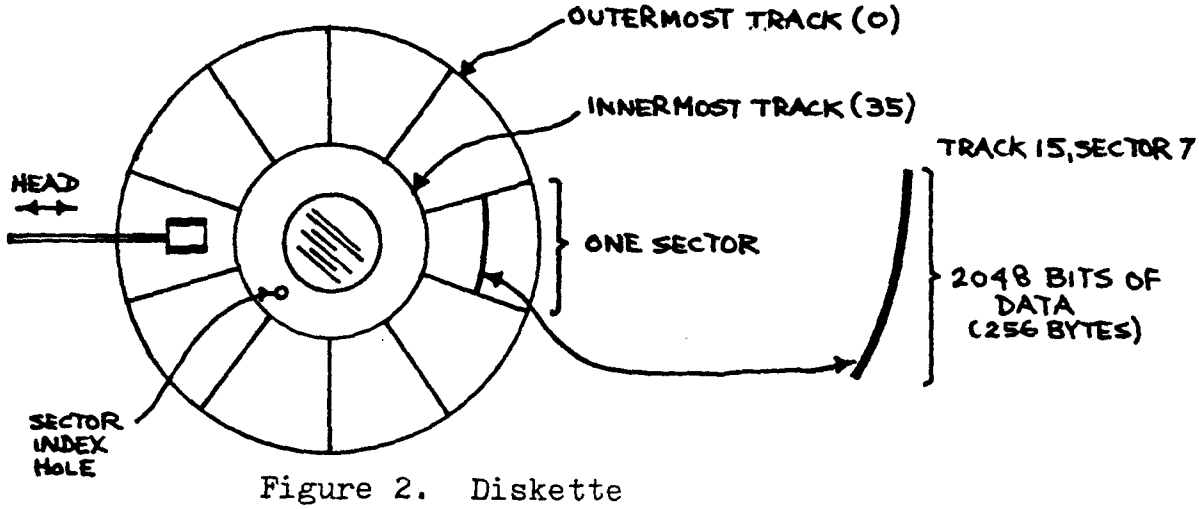


Figure 2. Diskette

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 disc when the index hole passes five times per second.

Each sector on the disc holds 256 data bytes. The entire track can hold 256*10 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*8 bits of data, or 20,480 bits along the circumference. Data ecorded 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 2047, 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 (1/40 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 ½ of 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 (o through 9) to the disc controller, just as it had previously passed a track address before the seek. The disc 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, than the time required to read or write the sector is about one revolution time plus 1/10th of a revolution time for the 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 ½ revolution or 1/10th 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 + average sector time + data time, or about 450 msec + 100 msec + 20msec = 570 msec per 256-byte sector, or roughly ½ second. The average access for data accessed "sequentially" in adjoining sectors and tracks would be about 100 msec (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 I/O operation is performed the disc 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 1 to 10 sectors worth with one command. Checks are provided for valid data, positioning errors, and other "disc 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 (in Radio Shack's infinite wisdom) as "1", "2", "3", and "4". Disc 1 always contains a diskette with the TRS Disc Operating System (or TRSDOS) and utility programs on the first 34K bytes of the disc, or so. The "default" drive is disc drive number 1 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 1.

Now that we've seen in general how disc storage functions, we'll continue by discussing the Shugart SA400 in the next chapter, followed by the controller chip, addressing, and disc programming.

Chapter 2

Shugart SA400

This chapter will describe the Shugart SA400 disc drive as used in the RS TRS-80. The SA400 drive is essentially unmodified internally by Radio Shack, the exception being minor addressing mods and a terminator that is connected to a dip socket on the disc electronics board. This terminator is connected only in drive number 1, hence the difference in the two types of drives supplied by RS. The SA400 requires power supplies of +12V and +5VDC. These are installed on the rear of the RS cabinet for the drives (the cabinet or cover is another optional item). Initially RS had a problem with heat disipation for the drive power supplies, but this has been alleviated somewhat in later discs, although the drives still run somewhat hot at this time of writing. Cabling is also supplied by RS 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 disc 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 disc system. The drive measures 5 3/4 inches high by 3 $\frac{1}{4}$ 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 ensure that the diskette is properly inserted.

The read/write head is a ceramic head which is mounted on a head 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're like me you're falling asleep by now - pull off the disc cover, and you will immediately see the mechanism by which the head assembly is stepped from track to track. It's better than a long description, although I don't want to offend any mechanical engineers out there).

PCB 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 WD 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 disc drive.

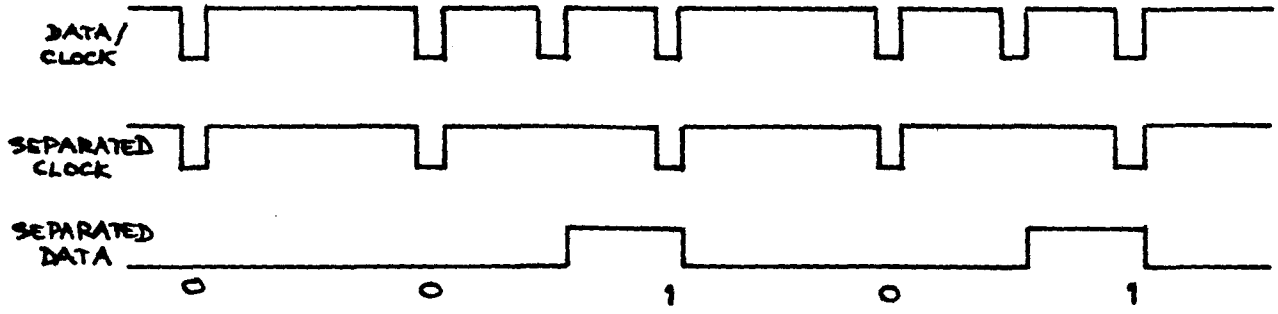


Figure 1. Data Recording

Control Signals

The standard Shugart SA400 interface signals are shown in table 1 below. These are the signals that are present on the disc drive pc 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).

Controller to Disc	Disc to Controller
SELECT 0	INDEX
SELECT 1	TRACK 0
SELECT 2	WRITE PROTECT
SELECT 3	READ DATA
DRIVE MTR ENABLE	
DIRECTION	
STEP	
WRITE DATA	
WRITE ENABLE	

Table 1. SA400 Interface Signals

Assume the disc is off. When power is supplied and 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 disc 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 disc 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 disc. 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 nsec (200 billionths of a second) to 2 millisecond (2 thousandths of a second).

The INDEX signal is an output from the disc drive that appears as a pulse every 200 msec 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 disc indicating that the head is positioned over the outermost track, track 0. Track 0 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 disc 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 connected, 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 position the head and may also control DRIVE MTR ENABLE, although this is not done in the TRS-80 (addressing the disc 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 disc 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 disc 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 transformation in data read from the disc and a parallel to serial transformation for data written to the disc.

In essence, then, there are really not many things that the disc 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 disc) logic. In the next chapter we'll see how the floppy disc controller implements these functions and others.

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

SHUGART PIN#,	SIGNAL NAME	TRS-80 CABLE PIN,	SIGNAL NAME
1	spare	1	gnd
2	spare	2	not used
3	spare	3	gnd
4	spare	4	not used
5	spare	5	gnd
6	spare	6	not used
7	gnd	7	gnd
8	INDEX	8	INDEX PULSE
9	gnd	9	gnd
10	SELECT 0	10	DS1
11	gnd	11	gnd
12	SELECT 1	12	DS2
13	gnd	13	gnd
14	SELECT 2	14	DS3
15	gnd	15	gnd
16	DRIVE MOTOR ENABLE	16	MOTOR ON
17	gnd	17	gnd
18	DIRECTION	18	DIRECTION SEL
19	gnd	19	gnd
20	STEP	20	STEP
21	gnd	21	gnd
22	WRITE DATA	22	WRITE DATA
23	gnd	23	gnd
24	WRITE ENABLE	24	WRITE GATE
25	gnd	25	gnd

SHUGART PIN#,	SIGNAL NAME	TRS-80 CABLE PIN,	SIGNAL NAME
26	TRACK 0	26	TRACK ZERO
27	gnd	27	gnd
28	WRITE PROTECT	28	WRITE PROTECT
29	gnd	29	gnd
30	READ DATA	30	READ DATA
31	spare	31	gnd
32	spare	32	DS4
33	spare	33	gnd
34	spare	34	not used

Table 2. SA400/TRS-80
Signals

Chapter 3

Western Digital FD1771B-01

The WD FD1771B-01 is a 40-pin floppy disc controller chip that 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 ID field.
9. Read Track. Read entire track.
10. Write Track. Write entire track.
11. Force Interrupt. Terminate operation.

Data to be written to the disc is transferred from the TRS-80 to the FD1771B-01 in parallel, 8 bits at a time. Data to be read from the disc is assembled from bit serial data from the disc 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 receives parallel data related to head positioning for the disc. 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 disc 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 disc controller will be explained in more detail in the following two chapters.

The FD1771B-01 contains several registers for positioning and disc data. They are shown in figure 1 below.

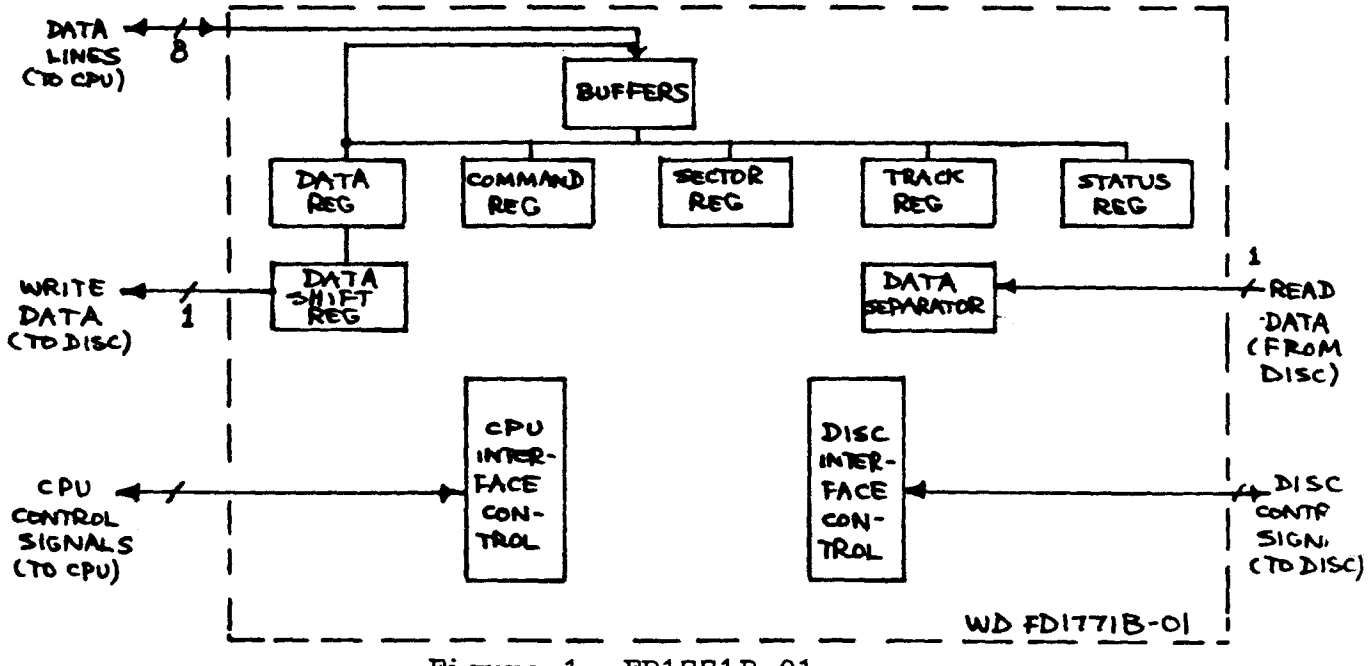


Figure 1. FD1771B-01

Tracks on the SA400 are numbered from 0 (outermost) to 34 (innermost). The track register is an 8-bit register that holds the current track number. Every time the disc 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 "store" instruction in the Z-80.

The sector register is an 8-bit register that 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 "store" instruction in the Z-80.

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

An 8-bit data register holds data that is read from the disc or is to be written to the disc. This register interfaces to another data register, the data shift register, that 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 disc 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 disc controller. The clock is used to control internal device timing and is also used to generate clock and data pulses sent to the disc for writes.

Power Supply Signals

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

FD1771B-01 to Disc 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 disc signals were defined before the controller; the controller is designed to easily implement the necessary logic for the disc. 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)

FD1771B-01 Signal (pin)		SA400 Signal (pin)
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)
TR00 (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
FDCLOCK (26)	Floppy disc clock (external separator)
WF (33)	Write fault
DINT (37)	Disc initialization

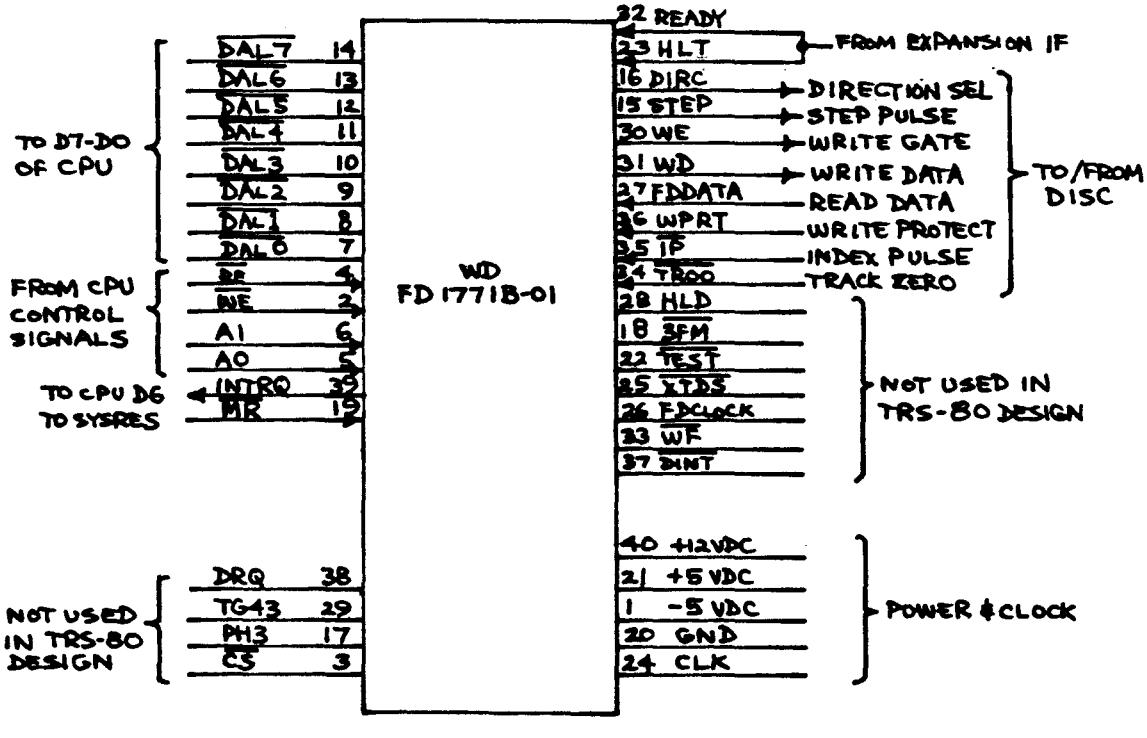


Figure 2. FD1771B-01 Pin-Out

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 sigificant). 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 to (or gated) the data bus of 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 of the 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 status	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 RS implementation is used to signal a ready status to the TRS-80 by being tied to data 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.N/C.
PH3 (17)	Phase 3. N/C.
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

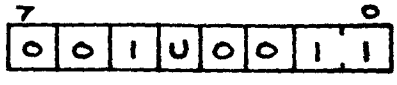
register with a byte defining the command value and performing a "store" instruction to address 37ECH. The 37ECH location is actually the disc address, and the least significant two bits of the "C" are 00, which specifies the command register. Signal WE is active because a "store" instruction is being 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

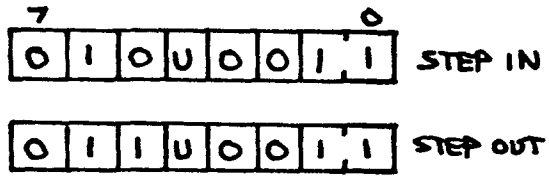
The RESTORE moves the disc head to a position over track 0. There are three fields in the RESTORE command. The first field, H, may be a 1 or 0 in the TRS-80 system as it causes no action (the head is loaded at motor on). In other implementations it unloads or loads the head before a command. The second field is a verify field and may be a 0 or 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 11_2. For discussion purposes, we'll assume that the normal coding of these fields in the TRS-80 will be 0011_2. These three fields are also used in the other four commands in this group (SEEK, STEP, STEP-IN, and STEP-OUT) and we'll assume a 0011_2 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.

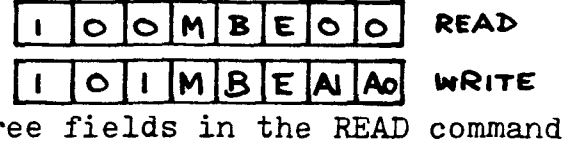


To reiterate the commands in the first group, then, RESTORE finds track 0, SEEK finds a specified track, STEP, STEP-IN, and STEP-OUT move one track. The STEP commands have a field specifying whether the track register should be updated automatically. Generally this field will be set. The verify field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

The second group of the eleven 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 disc, the controller must be able to transfer one sector's worth in about 1/10th revolution, or 20 msec., 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 disc. We will see the exact instruction sequence later in the 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 37EFH (A1=0,A0=1) 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 tranferred. If M=1 more than one sector will be transferred. The usual case for the TRS-80 is to transfer only one sector, however, 2 to 10 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 disc format, which is simply a de facto standard established by IBM. Field E controls a 10 msec 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 msec 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 00_2.

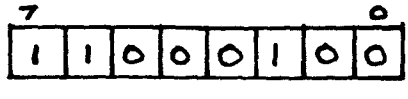
Data is written or read to the diskette by continuously monitoring (reading) the status from the FD1771B-01. One of the status bits is DRQ, or Data Request, which signifies either that the data register contains the next byte of read data or has been "emptied" of the last byte of write data. Examples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

There are three commands in the next group, related to disc formatting. They are: READ ADDRESS, READ TRACK, and WRITE TRACK. As we mentioned previously, disc 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 TRS-80 diskettes is given in Appendix B.

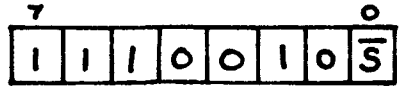
READ ADDRESS is a command that reads six identification bytes from the diskette from the next encountered id field. Each time one of the bytes is read the DRQ (Data Request) bit is set in the status, so that the TRS-80 program can read it from the FD1771B-01 by a "load" 37EFH instruction. The six bytes 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 format for a READ ADDRESS is shown below:



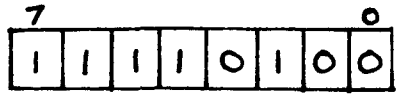
The READ TRACK command reads an entire track of the current diskette. Not only user data, but identification data is read as well. As in the case of READing user data, the Data Request bit is used to indicate when the next byte of data is ready to be transferred from the FD1771B-01 data register to the Z-80 cpu. Gaps on the diskette are also read and transferred. The READ TRACK command has one field, the $\overline{S}$ 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:



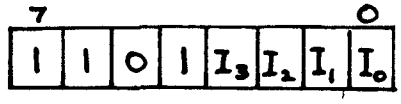
WRITE TRACK is used to format the diskette according to the format shown in Appendix B. Data in the given format is presented to the FD1771B-01 one byte at a time by the program. The Data Request bit is continually checked to see when the controller chip is ready for the next data byte. Address marks are written to the diskette by the controller chip upon detection of certain data patterns sent by the cpu. A CRC (cyclical redundancy check type of checksum) is written to the disc 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 (not used)	D7H
FEH	ID Address Mark	C7

Obviously no user data is written to the diskette during the WRITE TRACK OPERATION. The user area is filled with E5H or some other non-conflicting 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 curent 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 curent 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, and 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 transition

I_2 = 1, terminate on next index pulse

I_3 = 1, immediate terminate/interrupt

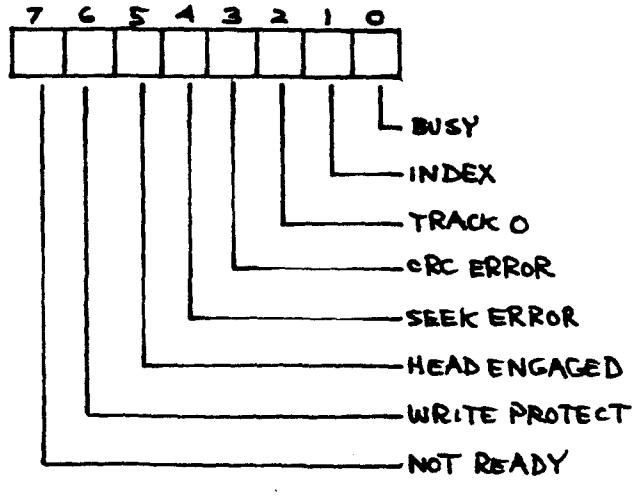
If none of the above conditions is specified, there is no interrupt but the command is terminated (I_0=I_1=I_2=I_3=0). Why have so many conditions? Why not? It is even possible that all might even be useful. For practical purposes, however, probably only a FORCE INTERRUPT with no interrupt (11010000_2) 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:

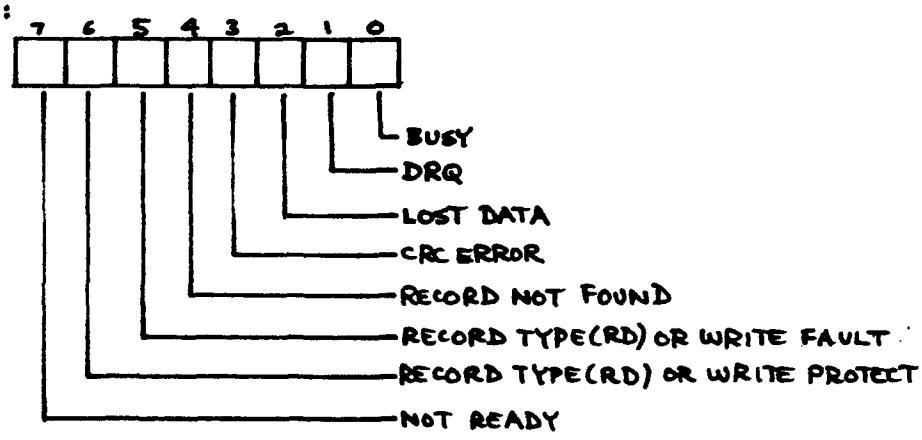
Command register:	Used to hold command to be acted upon. Command is written into the register from a cpu register,
Track register:	May be automatically updated with each STEP. May be read by cpu.
Sector register:	Setup with sector number prior to a read or write. May be read by cpu.
Data register:	Used to hold data passing between the cpu and disc during reads or 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 holds 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 disc 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 disc operation is being performed. Seek error and CRC error are active if verification was used. Bit 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 disc. 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 ID data or to user data (status bit 2). This bit will be set to indicate invalid data. Lost data (3) means that the cpu did not respond fast enough to keep up with the data being transferred between the cpu and disc. 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 disc 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_2 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 ensure 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 fnd	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 WD FD1771B-01, primarily in terms of what internal operations are performed, and how it interfaces to the Shugart SA400. The next chapter will show how the TRS-80 communicates to the FD1771B-01 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 disc.

Chapter 4

Expansion Interface

A schematic of the expansion interface pertaining to disc is shown in figure 1. There are several sections to be considered in the implementation. They are

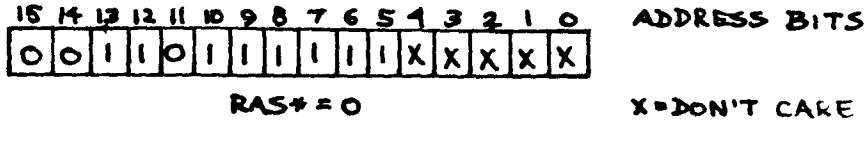
1. Disc selection
2. Motor on circuitry
3. Addressing
4. FD1771B-01 functions
5. Interrupt circuitry

Disc Addressing

There are two general addresses associated with the disc(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 FH.

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 disc 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 zeroes. RAS* comes from the cpu and is low when a memory request is made. The disc 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 disc. 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:



Output 2Y0 feeds chip Z32 enables 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 above 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.

Disc Selection

Four signals are shown on figure 1, and represent the disc select signals for disc 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 flip-flop Z36 (74LS175). The address of this flip-flop 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 disc drive 1, 2, 3, or 4, respectively.

Motor On Circuitry

Whenever a disc drive is selected, the 37E0H signal also goes to chip Z29. Z29 is a "one-shot" that 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

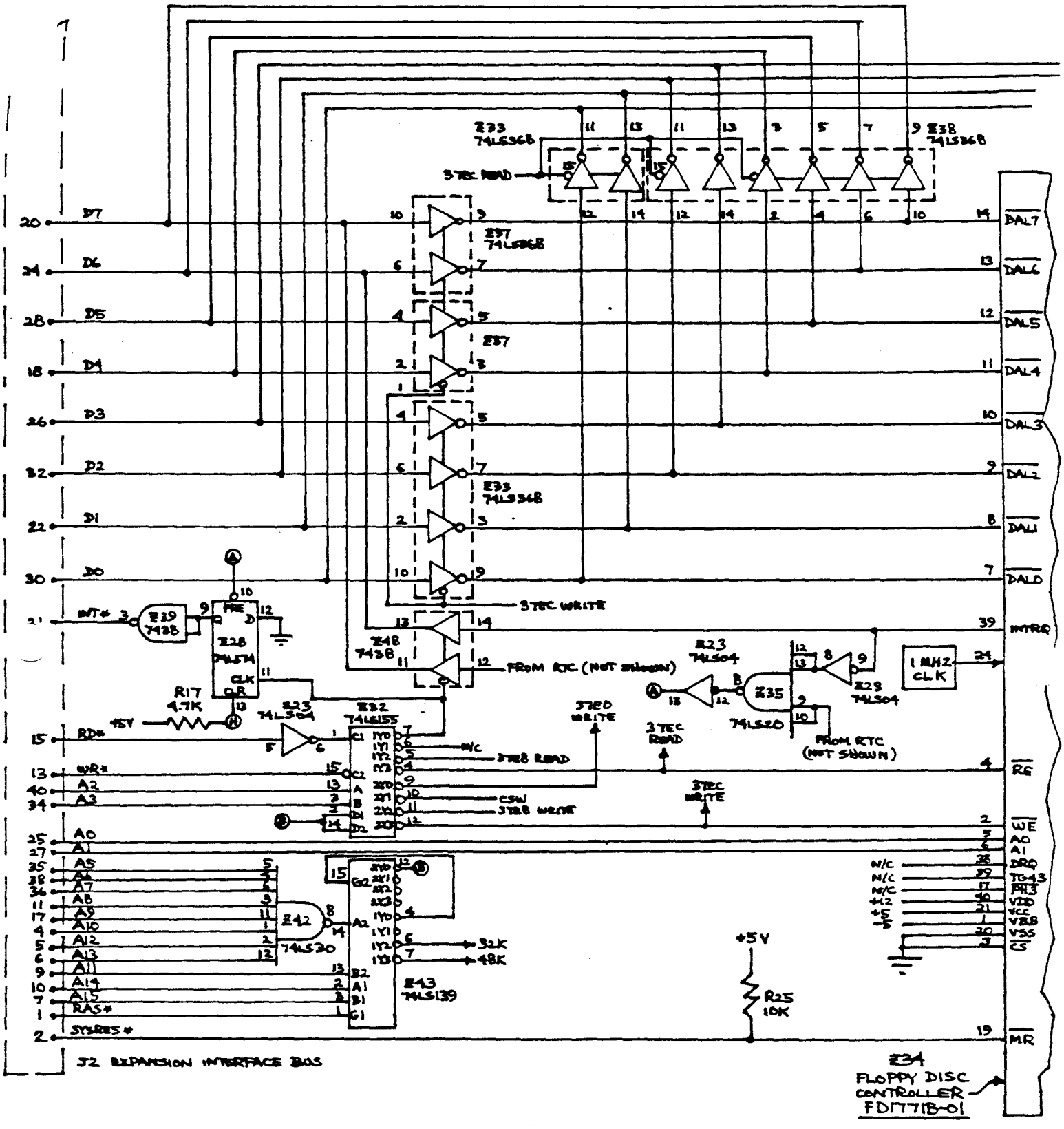


FIGURE 1

TRS-80 DISC LOGIC (sht 1)

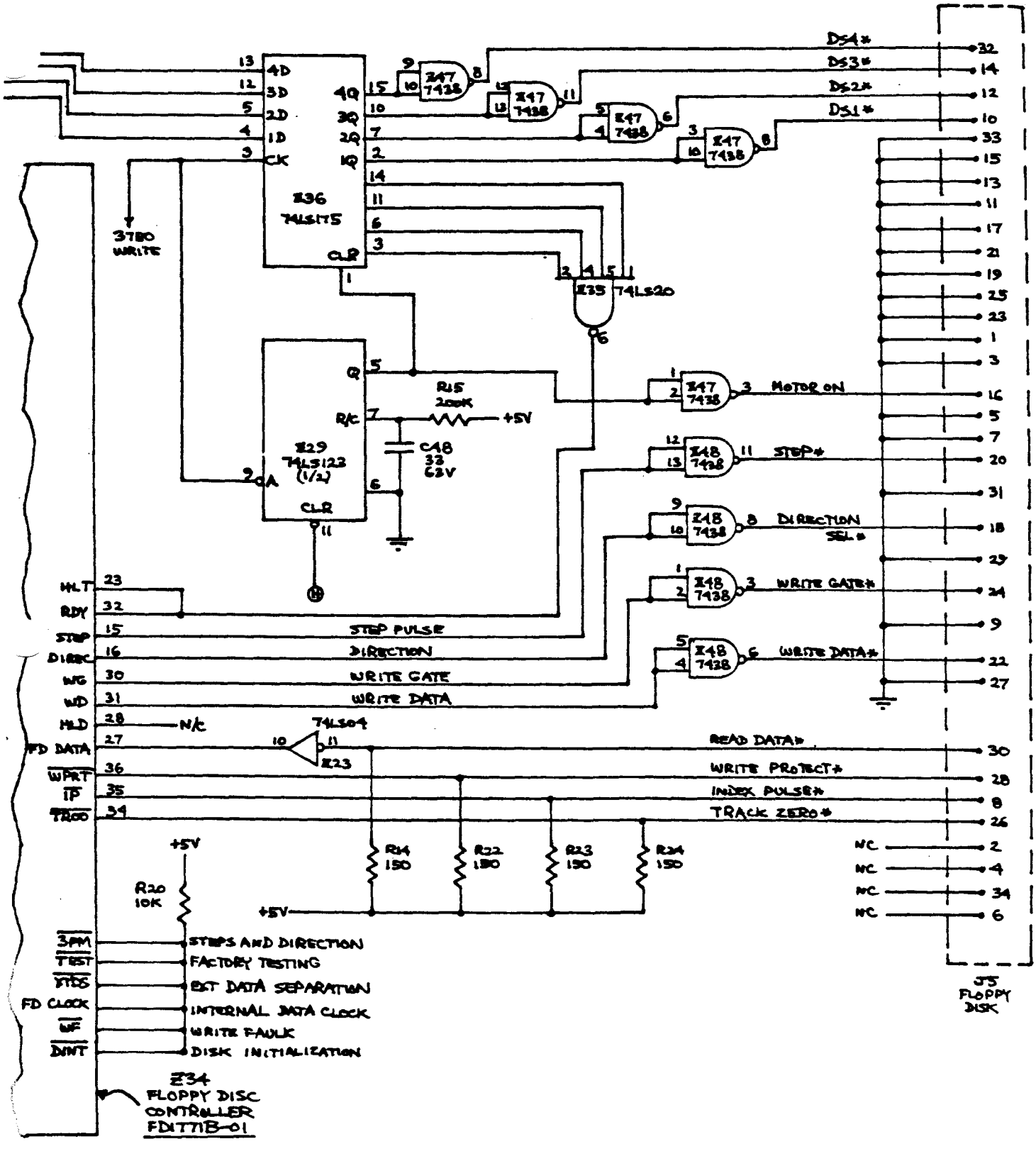


FIGURE 1

TRS-80 DISC LOGIC (sht2)

the disc drive motor in preparation for disc 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 disc 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 flip-flops 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 FD1771B-01. The register within the FD1771B-01 that is being addressed is determined by address bits 1 and 0. 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 37E0 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 decoder 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 disc operation, but is also set to denote an unsuccessful disc 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 TRS DOS to report on how disc interrupts are handled, if at all. Disc operations do <u>not</u> require interrupts, as we shall see in the next chapter, and for the time being we shall leave this question unresolved. An addendum describing disc interrupt action will be provided later upon request.

Chapter 5

Disc Programming

The rudimentary set of commands that can be used on the disc(s) in the TRS-80 are the commands described in chapter 3. No other basic disc 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 disc commands may be combined to implement a sophisticated disc operating system or disc file manage. 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 disc accesses may be optimized for access speed. It is beyond the scope of this text to describe the procedures for implementing a disc operating system or file manage, just as a text on Z-80 programming would not show the implementation of a BASIC interpreter. What will be shown, 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 disc operations using BASIC? The answer is that the head positioning operations such as restore, steps and seeks may indeed be 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 disc is handled in a cpu register. The program must continually test the disc status to see whether the disc is ready for the next data byte or whether the disc 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 or 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 disc 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'll do a simple write to address 37E0H. This operation should select a drive and turn on the disc 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 drive 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 1, 2, 3, or 4 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_{10} will be printed, indicating a "not ready" condition. Status during the selected time will be 68_{10} or 64_{10}, 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 disc not protected (0 or 4).

Now we'll 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 disc 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=0 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, secified by a 11_2 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 610.

```
100  POKE 14304,1               select
200  POKE 14316,3               restore
300  A=PEEK(14316)              get status
400  IF (A AND 1)<>GOTO 300     test busy
600  POKE 14304,1               select
610  POKE 14316,80              step in
700  A=PEEK(14316)              get status
770  A=PEEK(14317)              get track
800  PRINT A                    print
900  IF A< 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 til end
700  V=115                           step out
800  GOSUB 2000                      output and test
900  IF A<>0 GOTO 800                step out til 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 disc operation takes longer than that, the drive is automatically deselected. It's a good idea, therefore, to select before each new disc operation (command) to keep the disc in a ready condition. This applies not only to BASIC code, but to assembly language code as well.

We've seen the step in and step out command use. Now let's 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 langauge 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'll use code from a popular microcomputer. The code is unsophisticated and can be accessed using a symbolic disassembler such as Small Systems Software RSM-1S.

One of the first things that Level II BASIC does is to test whether a disc is present. If a disc <u>is</u> present, than 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 disc is not present,

of course, or if BREAK is pressed on power up, Level II goes instead to Level II BASIC, bypassing the disc.

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 location 0674H.

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

1. Read disc status (0696H)
2. Test for status bits from disc (069AH)
3. If no status for disc 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 disc 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 of status (06B2H)
8. Loop to 7 if busy
9. Else: Send 0 to sector register by outputting to location 37EEH. This prepares the disc to read sector 0 of track 0.
10. Send 8CH read command by outputting 8C to location 37E0 (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 checking the DRQ bit to see if a new byte of data has been assembled from the serial bit

```
0696  3A EC 37          LD  A,(37EC)      get disc status
0699  3C                INC A
069A  FE 02             CP  02
069C  DA 75 00          JP  C,0075        go if no disc
069F  3E 01             LD  A,01          for drive 1
06A1  32 E1 37          LD  (37E1),A      select drive 1
06A4  21 EC 37          LD  HL,37EC       1771 address comnds
06A7  11 EF 37          LD  DE,37EF       data reg address
06AA  36 03             LD  (HL),03       restore command
06AC  01 00 00          LD  BC,0000       delay 64K counts
06AF  CD 60 00          CALL 0060
06B2  CB 46     LOOP1   BIT 0,(HL)        test busy
06B4  20 FC             JR  NZ,LOOP1      go if still busy
06B6  AF                XOR A             zero to A
06B7  32 EE 37          LD  (37EE),A      0 to sector reg
06BA  01 00 42          LD  BC,4200
06BD  3E 8C             LD  A,8C          read command
06BF  77                LD  (HL),A        read sector 0
06C0  CB 4E     LOOP2   BIT 1,(HL)        test DRQ
06C2  28 FC             JR  Z,LOOP2       go if no data avail
06C4  1A                LD  A,(DE)        get next byte
06C5  02                LD  (BC),A        transfer data
06C6  0C                INC C             bump buffer pointer
06C7  20 F7             JR  NZ,LOOP2      go if not 256 bytes
06C9  C3 00 42          JP  4200          transfer to DOS loader
```

Figure 1. Disc Bootstrap Routine

```
(At start (HL)=37ECH and (BC)=buffer pointer)
52C6  11 EF 37          LD  DE,37EF       data register address
52C9  C5                PUSH BC           waste time
52CA  C1                POP BC            waste time
52CB  18 0B             JR  52D8          start read
52CD  0F        BSYTS   RRCA              busy bit to C
52CE  30 0A             JR  NC,52DA       go if not busy
52D0  7E        LOOP    LD  A,(HL)        get status
52D1  CB 4F             BIT 1,A           test DRQ
52D3  28 F8             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 pntr
52D8  18 F6             JR  LOOP          loop for next byte
52DA  7E                LD  A,(HL)        get status
52DB  E6 5C             AND 5C            test type, nfnd,CRC,lost
52DD  D1                RET Z             go if no errors
52DE  36 D0             LD  (HL),D0       force int (reset)
52E0  C9                RET               return
```

Figure 2. Generalized Disc Read

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 process.

If a RESET is performed, the same test for disc present is made at location 66H and a jump made to location 0000H if the disc is there, where a transfer to the boot strap 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 disc status
006F  D2 00 00  JP  NC,0000     go if disc 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 msec 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 disc operations should always perform a select to keep the motor on.

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 the 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 in time to the next byte of data.

At this point it may be wise to talk about some of the idiosyncracies of the TRS-80 disc implementation. Although nothing is mentioned in the FD1771B-01 documentation, code in the RS system uses "time wasting instructions" after certain disc 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 be 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 disc command. Additional time wasters may be required after steps or read or write commands. Any further information on this problem will be included in an addendum to this manual. Another problem associated with the TRS-80 system is that some of the code associated with the disc is presumably deliberately vague. Therefore, do not be too surprised to find confusing routines or structure if disassembling or investigating parts of TRSDOS.

The operations for disc 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 checked 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 in place of record type in the case of the read.

```
                        (BC contains buffer pointer)
                        (HL contains status, command address 37E0)
FE61  7E        SLOOP   LD  A,(HL)          get status
FE62  0F                RRCA                busy bit to C
FE63  DA 60 FE          JP  C,SLOOP         go if busy
FE66  36 A8             LD  (HL),A8         write command
FE68  11 EF 37          LD  DE,37EF         data reg address
FE6B  C5                PUSH BC             waste time
FE6C  C1                POP  BC               "    "
FE6D  C5                PUSH BC               "    "
FE6E  C1                POP  BC               "    "
FE6F  C3 76 FE          JP  DRQCK           bypass busy check
FE72  0F        BSYCK   RRCA                busy to C
FE73  D2 82 FE          JP  NC,DONE         go if not busy
FE76  7E        DRQCK   LD  A,(HL)          get status
FE77  CB 4F             BIT 1,A             test DRQ
FE79  CA 72 FE          JP  Z,BSYCK         go if new data not req'd
FE7C  0A                LD  A,(BC)          get next data byte
FE7D  12                LD  (DE),A          output to data reg
FE7E  03                INC BC              bump buffer pntr
FE7F  C3 76 FE          JP  DRQCK           continue
FE82  7E        DONE    LD  A,(HL)          get status
FE83  E6 5C             AND 5C              test w pro, n fnd, CRC,
FE85  C8                RET Z                  lost
FE86  36 D0             LD  (HL),D0         output force int (reset)
FE88  C9                RET                 return
```

Figure 3. Generalized Disc Write

Formatting

Formatting a diskette can be performed by one of two methods. The TRS-80 software contains a disc formatting program that automatically formats a diskette to "standard" format as defined in Appendix B. However, TRS-80 software evidentally changes this standard format to a non-standard format for protection of proprietary files. The probable method for doing this is to essentiallv reformat a track dynamically (during TRSDOS execution for new file storage). User-formatted diskettes, on the other hand, do not contain any directories or applications programs and cannot be used by TRSDOS. A user-formatted disc, of course, can be used for any user file manage software that 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 (if laborious) of investigating data on any diskette track. A program to read any given track is shown below in figure 4 (track # 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.

Conclusion

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

The author apologizes for some of the unanswered questions in the text above. As more information becomes available it will be compiled in an addendum. Readers are urged to contact the author for questions or comments on disc operations.

```
5000  F3              DI              disable interrupts
5001  3E 01           LD  A,01        for select
5003  32 E1 37        LD  (37E1),A    select drive 1
5006  21 00 00        LD  HL,0000     delay count
5009  2D      LOOP1   DEC L
500A  C2 09 50        JP  NZ,LOOP1
500D  25        LOOP2 DEC H
500E  C2 0D 50        JP  NZ,LOOP2
5011  32 E1 37        LD  (37E1),A    select again
5014  3A 43 50        LD  A,TRACK     load track #
5017  32 EF 37        LD  (37EF),A    output to data reg
501A  21 EC 37        LD  HL,37EC     status, comnd address
501D  36 1B           LD  (HL),1B     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 trk regstr
5023  7E              LD  A,(HL)      get status
5024  0F              RRCA            busy to C
5025  DA 23 50        JP  C,5023      loop if busy
5028  01 00 60        LD  BC,6000     buffer area
502B  36 E4           LD  (HL),E4     read track command
502D  C5              PUSH BC         waste time
502E  C1              POP BC
502F  C5              PUSH BC
5030  C1              POP BC
5031  7E        LOOP3 LD  A,(HL)      get status
5032  0F              RRCA            busy to C
5033  D2 03 42        JP  NC,4203     return to monitor
5036  CB 47           BIT 0,A           on done
5038  CA 31 50        JP  Z,LOOP3     loop if not data
503B  3A EF 37        LD  A,(37EF)    read data
503E  02              LD  (BC),A      store in buffer
503F  03              INC BC          bump buffer pntr
5040  C3 31 50        JP  LOOP3       rtn for next byte
5043  XX      TRACK   DB  0           track #
```

Figure 4. Read Track Program

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 update 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	C2	
III	Read Track	1110010$\overline{S}$	E4	$\overline{S}$: 0 synchronize to address mark, 1 no synchronize
III	Write Track	11110100	F4	
IV	Force Inter	11010000	D0	*Usual value used

Status

BIT	TYPE I	READ	WRITE	READ ADDR	READ TRK	WRITE TRK
7	NOT RDY	NOT RDY	NOT RDY	NOT RDY	NOT RDY	NOT RDY
6	WRITE PTECT	REC TYPE	WRITE PTECT	0	0	WRITE PTECT
5	HEAD ENGAGED	REC TYPE	WRITE FAULT	0	0	WRITE FAULT
4	SEEK ERROR	REC NOT FOUND	REC NOT FND	ID NOT FND	0	0
3	CRC ERROR	CRC ERROR	CRC ERROR	CRC ERROR	0	0
2	TRK 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