



USERS MANUAL
Intelligent Diskette Controller
Model 1070

INTRODUCTION

Since the following document was written, version F 1.3 of the PerSci File Management Firmware has been released for use with the 1070 Controller.

The Kill Command Syntax of version F 1.3 is:

KK volume/drive seq.

The remaining commands are as in the previous versions. The double KK was adopted as a device to reduce erroneous deletion of diskette files.

F 1.3 is issued in two versions; F 1.3P for controllers that do not have the serial (RS232) option installed and F 1.3S for those that do.

Two versions have been coded because deletion of the code necessary to handle serial data resulted in significantly faster controller operations.

Version F 1.3S retains the capability of the previous F 1.0 and F 1.2 and will work with either serial or parallel data transfers.

EPROM IDENTIFICATIONS

EPROMs coded at PerSci are marked U21, 22, 23, and 24 to indicate the appropriate sockets on the controller PCB. Additional markings have been made to indicate the coding as follows:

F 1.0	U21, 22, 23, 24
F 1.2	U21A, 22B, 23C, 24D
F 1.3P	21P, 22P, 23P, 24P
F 1.3S	21S, 22S, 23S, 24S

PREFACE

This document describes the definitive production version of the PerSci Model 1070 Intelligent Diskette Controller using PerSci File Management Firmware version F1.2. It is also applicable to the predecessor version, F1.0, with certain exceptions as noted herein. (Version F1.1 was never released.)

Version F1.2 File Management Firmware offers several improvements over its predecessor. It implements two additional commands (Xecute and Zap), supports hexadecimal as well as decimal numeric command parameters, provides automatic switching between the serial and parallel interfaces, and has a number of internal performance improvements. F1.2 also takes advantage of new revisions in the circuitry of the PerSci controller and drives to eliminate the head load delays (and clatter) during disk-to-disk copying and other multi-disk operations.

NOTE: When F1.2 is used with earlier PerSci controllers and drives which do not in combination have the necessary logic circuitry to permit the firmware to load all heads simultaneously, the controller Mode command must be used at each power up to inform the Firmware of this fact. Controllers and drives which in combination support this simultaneous head load capability are:

Model 1070 Controllers with Assembly 200350
(PCB 200349A, B, or higher)

Model 277 Drives with Data and Interface
Assembly 200263-003 Rev. H, or higher
(PCB 200262C, or higher)

Model 70 Drives with PCB Assembly 200208

(PCB numbers are etched on the non-component side of printed circuit boards.)

NOTE: Predecessor Firmware version F1.0 must be used only with controllers and drives which DO NOT in combination have the simultaneous head load circuitry.

It is the policy of PerSci not to distribute program source listings of other details of the File Management Firmware beyond those described in this manual. From time to time, PerSci will issue new versions of the Firmware (for example, F1.2) to correct any known errors or to provide functional enhancements. Such Firmware revisions in ROMs or EPROMs will be made available to users of previous versions for a nominal charge. PerSci will also assist users in adapting the Model 1070 Controller to special applications (within reason) which require specialized controller software. The "Xecute" command (which permits special disk-resident software to be loaded into the controller and executed) was added to F1.2 to facilitate this. A charge will be made for any such software development work.

TABLE OF CONTENTS

1. GENERAL DESCRIPTION	1
1.1. Summary of Features	1
1.2. Hardware	1
1.3. Firmware	1
1.4. Interfaces	1
1.5. Diskette Format	2
1.6. Companion Diskette Drives	2
2. HARDWARE SPECIFICATIONS	3
2.1. Physical Specifications	3
2.2. Microcomputer Interface Specifications	3
2.3. Diskette Drive Interface Specifications	4
2.4. Power Requirements	5
2.5. RS232 Serial Interface Option	5
3. FIRMWARE SPECIFICATIONS	6
3.1. Theory of Operation	6
3.2. Controller Commands	9
3.3. Controller Interface Protocol	18
3.4. Diskette Format	21

APPENDICES

- APPENDIX A: Sample Driver Program Flowchart
 Sample 8080 or Z80 Driver Program
 Sample 6800 Driver Program
- APPENDIX B: Interface Schematic for S-100 Bus
 Interface Schematic for 6800
 Timing Data
- APPENDIX C: Brief History of the Model 1070 Controller
 Option Jumper Data
 Connector Data
 Schematic for Controller
 Connection of Additional Drives
 Sector Sequences
- APPENDIX D: Applications Note for Simultaneous Head Load
 Circuit Modifications

SECTION 1 - GENERAL DESCRIPTION

1.1. SUMMARY OF FEATURES

The PerSci Model 1070 is the first truly intelligent diskette controller. Can you imagine a controller which manipulates diskette files by name and provides the full functional capabilities of an advanced disk operating system, yet which requires no more support software in your microcomputer than does a paper tape reader or magnetic tape cassette drive? The Model 1070 accomplishes all of this on a single 4.5" by 7" circuit board through a combination of state-of-the-art LSI and microprocessor technology, advanced firmware techniques, and high-density packaging. The controller supports up to four PerSci Model 70 single diskette drives or up to two PerSci Model 277 dual diskette drives, providing a high-performance mass storage subsystem with an on-line capacity of more than one million bytes.

1.2. HARDWARE

The controller board incorporates a microprocessor and its associated support electronics, a LSI diskette drive controller chip, 4K bytes of ROM (optionally EPROM) containing the file management firmware, 1K bytes of RAM used for sector buffers and file tables, an eight-bit parallel microcomputer interface, and an optional RS-232 serial asynchronous interface. Required power for the controller (+5, +12, and -12 volts regulated) can be derived either from the microcomputer or diskette drive power supplies.

1.3. FIRMWARE

The controller firmware resides in ROM on the controller board and performs the file management functions normally associated with the most advanced microcomputer disk operating systems. Supported functions include: diskette format initialization with optional sector interleave; maintaining and searching and index of files on each diskette; allocation and deallocation of diskette space; sequential, random, stream, and direct file access methods; blocking and unblocking of both fixed-length and variable-length records; creating, deleting, renaming, and copying of files; error detection and error retry; and even diagnostic testing of the diskette drives. These file management functions are specified by means of a high-level controller command language. Only minimal support software is needed in the host microcomputer, making it exceptionally easy to use the controller with existing non-disk-oriented operating systems, language processors, and other software.

1.4. INTERFACES

Two alternative methods are provided for interfacing with the controller: parallel and serial. The parallel microcomputer interface includes a buffered eight-bit bidirectional data bus with handshake and address selection logic consistent with the interface requirements of most currently-available microprocessors including the 8080, 6800, Z-80, etc. The

optional EIA RS-232 serial asynchronous interface provides sixteen switch-selectable transmission speeds from 50 to 19,200 bits per second, interfacing directly with virtually any standard terminal, modem, or serial microcomputer interface port. When using a controller which includes the optional serial interface, switching between parallel and serial is performed automatically by controller firmware.

1.5. DISKETTE FORMAT

The diskette initialization function of the controller creates a soft-sectored diskette format which is IBM 3740 compatible. Each diskette contains 77 tracks with 26 sectors per track and 128 data bytes per sector. The first track is reserved by the controller for use as an index of files, while the remaining 76 tracks are available for data storage. Formatted capacity of each diskette is 252,928 bytes plus the index track.

1.6. COMPANION DISKETTE DRIVES

The PerSci Model 70 single diskette drive and Model 277 dual diskette drive incorporate many design features previously unique to large disk technology, resulting in unexcelled reliability and performance, small size, and fast access to data. The use of voice coil positioning provides access times which are five to seven times faster than other available diskette drives with stepping motor positioners. Automatic motor-driven diskette load and unload assures simple and accurate diskette insertion and eliminates the possibility of diskette damage. Power consumption is one fourth of the power required by competitive drives, no cooling fan is required, and operation is virtually noiseless. Compact design permits five single drives or four dual drives to be mounted within the width of a 19" rack. The PerSci Model 1070 intelligent diskette controller is especially designed to take maximum advantage of the high-performance capabilities of these drives.

SECTION 2 - HARDWARE SPECIFICATIONS

2.1. PHYSICAL SPECIFICATIONS

The controller consists of a single printed circuit board with dimensions 4.50" x 7.00" which mates with edge connectors along the two 4.50" sides of the board. One edge connector has 72 pins (dual 36) with .100" spacing, and carries the parallel interface, RS-232 serial interface, and controller power connections. The other edge connector has 50 pins (dual 25) with .100" spacing and provides the interface with the diskette drive(s). The controller board is physically compatible with Vector Electronics plugboards and card cages with 72 pin connectors.

2.2. MICROCOMPUTER INTERFACE SPECIFICATIONS

2.2.1. Mating Connectors

The microcomputer interface uses an edge connector with 72 pins (dual 36) and .100" spacing (Amphenol 225-23621-201 or equivalent). In the listing below, all signals are TTL active high, except those marked * are TTL active low and those marked ** are EIA RS-232 levels.

Pin ID	Signal Designation	Pin ID	Signal Designation
-----	-----	-----	-----
PARALLEL INTERFACE		RS-232 SERIAL INTERFACE	
1 thru 8	Data Bus 0 thru 7	LL	Transmit Data**
E thru T	Addr Bus 4 thru 15	32	Receive Data**
27	Select*	KK	Data Term. Ready**
18	Read Strobe*	31	Data Set Ready**
19	Write Strobe*	HH	Request to Send**
A	Status/Data	29	Clear to Send**
CONTROLLER RESET		CONTROLLER POWER	
17	Reset Controller*	RR,36	Ground
U	Reset Complete*	PP,35	+5v Regulated
		34	+12v Regulated
		NN	-12v Regulated

2.2.2. Signal Definitions

Address Bus 4 through 15:

When the controller is jumpered for internal address decode (M to N and N to P), the presence of a 12-bit address on these lines which matches the jumper-selected controller address causes the parallel interface to be enabled. These lines are generally connected to the 12 high-order bits of a microcomputer address bus.

Select*:

When the controller is jumpered for internal address decode (M to N and N to P), this line is an output which goes low whenever the parallel interface is enabled by the address decode logic. When the controller is jumpered for external address decode (N to P only), this line is an input which causes the parallel interface to be enabled when it is driven low.

Read Strobe*, Write Strobe*:

Whenever the parallel interface is enabled, a low level on the Read Strobe* or Write Strobe* line causes the controller to transfer a byte of data to or from the data bus, respectively.

Data Bus 0 through 7:

These eight bidirectional data lines are tri-stated (floating) except when the parallel interface is enabled and Read Strobe* or Write Strobe* is active.

Status/Data:

Whenever the parallel interface is enabled, a high level on this line causes the controller status port to be selected, and a low level causes the data port to be selected. This line is generally connected to the low-order bit (A0) of a microcomputer address bus.

Reset Controller*:

A low level on this line causes the controller to be reset.

Reset Complete*:

This line goes high when Reset Controller* is made active or the controller reset button is depressed, and returns low after the reset signal has been removed.

Transmit Data, Receive Data**, Data Terminal Ready**, Data Set Ready**, Request to Send**, Clear to Send**:**

These lines have their standard RS-232 definitions. Transmit Data** from the controller is serial asynchronous with one start and one stop bit, eight data bits, and no parity.

2.3. DISKETTE DRIVE INTERFACE SPECIFICATIONS

2.3.1. Mating Connectors

The diskette drive interface uses an edge connector with 50 pins (dual 25) and .100" spacing (Scotchflex 3415-0000 or equivalent for flat ribbon cable, Viking Connector 3VH25/1JN-5 or TI Connector #312125 or equivalent for solder connections). All odd-numbered pins are connected to ground to facilitate the use of twisted-pair cable between the controller and diskette drive(s), which is strongly recommended.

Pin	Signal Designation	Pin	Signal Designation
4	Drive 3 Select	28	Drive 2 Select
10	Seek Complete	34	Direction
12	Restore	36	Step
14	Remote Eject	38	Write Data
16	Direct Head Load	40	Write Gate
18	Drive 1 Select	42	Track 00
20	Index	44	Write Protect
22	Ready	46	Separate Data
26	Drive 0 Select	50	Separate Clock

2.3.2. Signal Definitions

For signal definitions, refer to PerSci Product Specifications, Model 70 or Model 277 Diskette Drive.

2.4. POWER REQUIREMENTS

Power requirements for the Model 1070 controller are: +5 volts at 1.5 amp maximum, +12 volts at 150 ma maximum, -12 volts at 200 ma maximum, all voltages regulated within plus or minus five percent.

2.5. RS232 SERIAL INTERFACE OPTION

This is a factory-installed option which provides an EIA standard RS-232 serial asynchronous interface in addition to the standard parallel microcomputer interface. Switching between parallel and serial is performed automatically by controller firmware; when the controller receives a command over one of the interfaces, it responds using the same interface. The RS-232 Serial Interface Option includes an on-board speed selection switch with the following settings:

Switch Setting	Transmission Speed (BPS)	Switch Setting	Transmission Speed (BPS)
0	50	8	1,800
1	75	9	2,000
2	110	A	2,400
3	134.5	B	3,600
4	150	C	4,800
5	300	D	7,200
6	600	E	9,600
7	1,200	F	19,200

NOTE: The controller outputs serial characters with one start bit, eight data bits (no parity), and one stop bit for all transmission speeds.

SECTION 3 - FIRMWARE SPECIFICATIONS

3.1. THEORY OF OPERATION

3.1.1. File Allocation

A diskette volume contains 77 tracks with 26 sectors per track and 128 data bytes per sector. The outermost track is reserved by the controller for use as an index (i.e., a table of contents) for the volume, while the remaining 76 tracks are available for file storage.

When a new file is created on a diskette volume, it receives an allocation of contiguous sectors. The minimum file allocation is one sector, and the maximum allocation is 1,976 sectors (i.e., 76 tracks of 26 sectors, or 252,928 bytes). The first file created on a newly initialized diskette receives an allocation starting immediately above the index track. Subsequently created files receive an allocation starting immediately above the allocation of the previously created file. The allocation of each file is recorded on the index track.

When a file is deleted, its block of contiguous sectors is deallocated, and its index entry is marked as deleted. The controller provides a command ("Gap") to compress the allocations on a volume, eliminating the gaps caused by previous file deletions and making the space available for subsequent file creations.

3.1.2. File Access Methods

The controller provides four methods for accessing and updating data stored on diskette.

The stream access method permits an entire file to be read or written as a continuous stream of data bytes (as if the diskette file were a very high speed paper tape). Stream access is the simplest access method to use, requiring only a single controller command to read (load) or write (save) an entire file. It is ideally suited to the storage and retrieval of executable programs or any other use in which paper tape or cassette tape is conventionally used. Stream access is performed using the "Load" and "Save" controller commands.

The punctuated access method treats a file as a sequence of variable-length records separated by punctuation marks (the controller uses the ASCII record separator character "RS" for this). A punctuated file may be positioned at its beginning or end, and variable-length records may be read or written in sequence, one at a time. Records may span sector boundaries on the diskette but this is made transparent by the controller. Punctuated access is most appropriate for the storage of text files (e.g., source programs or word processing files) or for any application in which sequential access to variable-length records is desirable. Because of its dependency on a unique punctuation character ("RS") to delimit records, punctuated access is not well suited to the storage of arbitrary binary information.

The relative access method treats a file as a byte-addressable memory. A relative file may be positioned at its beginning, end, or to any desired byte position within the file. Any number of bytes may then be read or written. Relative read and write operations may span sector boundaries but this is made transparent by the controller. Relative access is ideal for data base oriented applications in which random access is required. Both punctuated and relative access are performed using the "File", "Position", "Read", and "Write" controller commands.

Finally, the direct access method permits any specified sector of any specified track of a diskette to be read or written directly, bypassing the file management functions of the controller altogether. Direct access is performed using the "Input" and "Output" controller commands.

3.1.3. File References

A file reference identifies a particular file or group of files. File references may be either unique or ambiguous: a unique file reference identifies one file uniquely, while an ambiguous file reference may be satisfied by several different files.

File references consist of four components: a name of up to eight characters (NNNNNNNN), a version consisting of a period followed by up to three characters (.VVV), a type consisting of a colon followed by a single character (:T), and a drive number consisting of a slant followed by a digit between 0 and 3 (/D). The version, type, and drive components are optional and are set off from the name by means of their unique leading punctuation characters (NNNNNNNN.VVV:T/D). A missing name, version, or type is assumed to be blank, and a missing drive number is assumed to refer to the current default drive.

The following are examples of valid unambiguous file references:

MONITOR	MASTER/2	STARTREK.BAS/1
MONITOR.SRC	MASTER:\$	STARTREK.XQT
MONITOR.OBJ:A	MASTER.ONE	STARTREK:Z/0

The special characters "?" and "*" may be used to make a file reference ambiguous so that it may match a number of different files. The "?" is used as a "wild-card" character which matches any character in the corresponding position in a file reference. Thus the ambiguous file reference:

PER????.BA?

matches all of the following unambiguous file references:

PERFECT.BAL	PERSCI.BAS	PERQ.BAX
-------------	------------	----------

The character "*" is used to denote that all character positions to the right are wild-cards unless otherwise specified. The following examples illustrate the flexibility which this facility provides:

Reference	Equivalent to	Ambiguous Reference Matches
MONITOR.*	MONITOR.?????	all files with name MONITOR
*.BAS	?????????.BAS:?	all files with version .BAS
Z*	Z?????????.?????	all files starting with Z
*	?????????.?????	all files on the diskette

3.2. CONTROLLER COMMANDS

Controller commands consist of a single command letter followed (in most cases) by one or more command parameters. Parameters must not contain embedded spaces, must be set off from one another by spaces, and may optionally be set off from the command letter by spaces.

CONTROLLER COMMAND SUMMARY

Command	Command Syntax	Command Function Summary
-----	-----	-----
Allocate	A file sectors	Allocates an empty file "file" of "sectors" sectors.
Copy	C file1 file2 sectors	Copies files matching "file1" to same or different diskette, optionally renaming according to "file2" and reallocating according to "sectors".
Delete	D file	Deletes all files matching "file".
Eject	E /drive	Ejects diskette in drive "drive".
File	F unit file	Opens "file" and associates with "unit".
	F unit	Closes the open file associated with "unit".
	F	Closes all open files.
Gap	G /drive	Compresses allocations on "drive" to eliminate gaps.
Input	I track sector /drive	Reads specified sector.
*Kill	K volume/drive seq	Initializes diskette with interleave "seq".
	K volume/drive	Deletes all files on diskette without initializing.
Load	L file	Reads entire file "file" as a stream.
Mode	M date:options/drive	Sets current date, I/O options, and/or default drive.
Name	N file1 file2	Renames file "file1" in accordance with "file2".
Output	O track sector /drive	Writes specified sector.

Position	P	unit	sector	byte	Positions the open file associated with "unit".
	P	unit			Reports current position of file associated with "unit".
Query	Q	file			Reports index information for files matching "file".
Read	R	unit	bytes		Relative read of file associated with "unit".
	K	unit			Punctuated read of file associated with "unit".
Save	S	file			Creates new file "file" by writing as a stream.
Test	T	option/drive			Executes a diagnostic test on drive "drive".
Write	W	unit	bytes		Relative write to file associated with "unit".
	W	unit			Punctuated write to file associated with "unit".
Xecute	X	file	option		Loads file "file" into controller RAM and executes it.
Zap	Z	unit			writes end-of-data mark at present position of file associated with "unit".

NOTE: Numeric command parameters (byte, bytes, sector, sectors, seq, track) must be decimal for version F1.0 Firmware, but may be either decimal or hexadecimal for version F1.2. Hexadecimal parameters must be prefixed by a letter (such as "H" or "X"; for example, the commands:

```
A FNAME 32
A FNAME X20
```

will both allocate a file whose length is 32 (decimal) sectors.

NOTE: The commands Xecute and Zap are not in the F1.0 version.

3.2.1. Allocate Command (A file sectors)

The "Allocate" command creates a new, empty file with the specified allocation (decimal or hex number of sectors).

Example:

```
A BIGFILE 1000
```

3.2.2. Copy Command (C file1 file2 sectors)

The "Copy" command copies one or a collection of files from a diskette volume to the same or a different diskette volume. The copied files may have the same or different names as the original files, and may have the same or different allocations. The "Copy" command cannot be used if there are any open files.

Examples:

```
C ALPHA BETA
C ALPHA/0 */1
C ALPHA/0 BETA/1 100
C */0 */1
C A*/0 B*/1
```

The first example makes a duplicate of the file ALPHA on the same diskette (default drive), calling the duplicate BETA. The second example copies the file ALPHA from drive 0 to drive 1, leaving the name and allocation unchanged. The third example also copies ALPHA from drive 0 to drive 1, but changes the name to BETA and gives the new file an allocation of 100 sectors (which may be larger or smaller than ALPHA). The fourth example copies all files from drive 0 to drive 1, preserving all file names and allocations. The last example copies only files with names starting with "A" from drive 0 to drive 1, changing the first character of each file name from "A" to "B".

3.2.3. Delete Command (D file)

The "Delete" command deletes a file or a collection of files from a diskette.

Examples:

```
D GEORGE
D *.OBJ/1
D XZ??/2
```

The first example deletes a single file GEORGE from the default drive. The second example deletes all files on drive 1 which have version .OBJ. The last example deletes all files on drive 2 which have two to four character names starting with "XZ".

3.2.4. Eject Command (E /drive)

The "Eject" command causes the diskette to be ejected from the specified drive. Note that this command is effective only if the diskette drive is equipped with the Remote Eject feature.

Examples:

```
E /2  
E
```

3.2.5. File Command (F unit file)

The "File" command opens and closes diskette files. A file must be open before punctuated or relative access is permitted by the controller. An open file is associated with a logical unit number between 1 and 5 (a maximum of five files may be open at one time).

Examples:

```
F 2 MASTER/1  
F 2  
F
```

The first example opens the file MASTER on drive 1 and associates it with logical unit 2. The second example closes the open file associated with logical unit 2. The third example closes all open files.

3.2.6. Gap Command (G /drive)

The "Gap" command compresses the allocations on a diskette volume to eliminate any gaps in the allocations caused by prior file deletions. The "Gap" command cannot be used if there are any open files.

Examples:

```
G /3  
G
```

3.2.7. Input Command (I track sector /drive)

The "Input" command reads a single specified sector of a diskette volume. The sector is specified by decimal track number (0-76), decimal sector number (1-26), and drive number. (In F1.2, the track and sector number may also be hexadecimal.)

Examples:

```
I 43 10 /1  
I 1 1
```

3.2.8. Kill Command (K volume/drive seq)^{*}

The "Kill" command deletes all files on a diskette volume. Optionally, the command also initializes (formats) the entire diskette, erasing all previously recorded information thereon and writing new sector headers on each track. The diskette may be initialized with any one of thirteen sector interleave sequences to enhance read/write performance. Further discussion of interleave sequences appears in section 3.4.3 of this document.

Examples:

```
K SCRATCH/3
K BACKUP 1
K MASTER 9
```

The first example deletes all files on drive 3, labels the volume SCRATCH, but does not initialize each track. The second example initializes the diskette on the default drive without interleave. The last example initializes with interleave sequence 9.

3.2.9. Load Command (L file)

The "Load" command reads a diskette file in its entirety as a stream.

Examples:

```
L BASIC
L EDITOR/3
```

3.2.10. Mode Command (M date:options/drive)

The "Mode" command may be used to set the current date, the default diskette drive, and/or various controller options. The current date is entered as a six character value (the format YYMMDD is suggested but not required by the controller). The default diskette drive is entered as the character "/" followed by a drive number (0-3); this becomes the drive which is used for all subsequent file references and commands which do not include an explicit drive number. The options are entered as the character ":" followed by a single hexadecimal digit (0 through F) whose bits are microcoded as follows (this applies to Fl.2 only):

Option	Meaning
-----	-----
:8	Supress non-fatal error messages
:4	Simultaneous head load NOT available
:2	Keep heads loaded continuously
:1	Model 70 drives in use

NOTE: At initial power up, the controller assumes by default that Model 277 drives with the simultaneous head load feature are in use.

*Fl.3 syntax KK volume/drive seq.

Examples:

```
M 770819
M /1
M :C
```

The last example above informs the controller that the controller and/or drive do not support simultaneous head load, and that non-fatal error messages are to be suppressed.

3.2.11. Name Command (N file1 file2)

The "Name" command modifies the name, version, and/or type of a file. The wild-card characters "?" and "*" are used to indicate that selected portions of the file reference are to be left unchanged, as illustrated in the examples.

Examples:

```
N ALPHA BETA
N BACKUP.2 *.3
N XRATED R*
```

The first example changes the file ALPHA to BETA. The second example changes BACKUP.2 to BACKUP.3, while the third changes XRATED to RRATED.

3.2.12. Output Command (O track sector /drive)

The "Output" command writes a single specified sector of a diskette volume. Its parameters are identical to those for the "Input" command.

Examples:

```
O 43 10 /1
O 1 1
```

3.2.13. Position Command (P unit sector byte)

The "Position" command permits open files to be positioned at the beginning, end, or at any specified byte position. The command may also be used to report the current position of an open file.

Examples:

```
P 2 213 88
P 2 213
P 2 0
P 2 9999
P 2
```

The first example positions the open file associated with logical unit 2 to byte 88 in sector 213 of the file. The second example positions the file to byte 0 of sector 213. The third example positions the file at its beginning, and the fourth example positions the file at its end-of-data (note that the controller does not permit a file to be positioned beyond its end-of-data).

Finally, the last example simply reports the current position of the file.

3.2.14. Query Command (Q file)

The "Query" command lists the following index information for one, some, or all files on a diskette volume:

- . Name, version, and type
- . Start of allocation (decimal track and sector)
- . Length of allocation (decimal number of sectors)
- . End of data (decimal sector and byte offset)
- . Date of creation
- . Date of last update

This information is preceded by a heading which lists the volume name, next available track and sector, volume interleave, and date initialized.

Examples:

```
Q ALPHA/2
Q *.SRC
Q *
```

Sample Query Listing:

```
SCRATCH.DSK 06-07 09 770215
PMF11.OBJ:3      01-01 0032 0031 082 770430
TEXTED          02-07 0025 0024 000 770503
DOCUMENT.TXT    03-06 0079 0078 001 770503 770618
```

3.2.15. Read Command (R unit bytes)

The "Read" command reads an open file by means of either the relative or punctuated access method (i.e., fixed-length or variable-length records).

Examples:

```
R 2 80
R 2
```

The first example reads a fixed-length record of 80 bytes from the current position of the open file associated with logical unit 2. The second example reads a variable-length record delimited by a record separator character ("RS").

NOTE: The maximum length of a fixed-length read is 65535 bytes (HFFFF).

3.2.16. Save Command (S file)

The "Save" command creates a new file by writing a stream of data onto the diskette. The resulting file receives an allocation of the minimum number of sectors needed to accommodate the length of the stream.

Examples:

```
S BASIC
S EDITOR/3
```

3.2.17. Test Command (T option/drive)

The "Test" command performs one of several diagnostic tests on the specified drive. Available tests are: V (random seek-verify test), R (random seek-read test), and I (incremental seek-read test).

Test V is a high-speed random-seek test. It performs a sequence of seeks to a randomly-selected track, reads the first encountered sector header on that track, and verifies that the correct track has been reached.

Test R is a random-seek-read test. It performs a seek to a randomly-selected track, then reads a particular randomly-selected sector on that track, and verifies that both the sector header and sector data are correct (using the CRC in each case).

Test I is an incremental-read test. It reads and verifies both the sector header and sector data of each sector on the diskette, starting at track 0 sector 1 and proceeding incrementally through track 76 sector 26.

Once initiated, tests V and R run indefinitely until the controller is reset or until a hard disk error is encountered which persists for five successive retries. Test I makes a single pass over the diskette, reading each sector once, and then terminates.

Examples:

```
T V/1
T R/0
T I
```

3.2.18. write Command (W unit bytes)

The "Write" command writes an open file by means of either the relative or punctuated access method (i.e., fixed-length or variable-length records). If data is written beyond the end-of-data of the file, the end-of-data is moved accordingly. The controller will not permit data to be written beyond the last sector allocated to the file.

Examples:

```
W 2 80
W 2
```

The first example writes a fixed-length record of 80 bytes to the open file associated with logical unit 2, starting at the current position of the file. The second example writes a variable-length record to the file, followed by a record

separator character ("RS").

NOTE: The maximum length of a fixed-length write is 65535 bytes (HFFFF hex).

3.2.19. Xecute Command (X file option)

The "Xecute" command loads an executable diskette file into controller RAM and executes it. This permits diskette-resident routines to extend the effective command repertoire of the controller. The option is a decimal or hex parameter which is passed to the routine. The "Xecute" command is not available in Fl.0.

Note that the "Xecute" command is not required for normal use of the controller, but was included to facilitate special applications of the controller. For further details, contact PerSci.

Examples:

```
X DRIVTEST 1  
X CONVERT
```

3.2.20. Zap Command (Z unit)

The "Zap" command truncates an open file by establishing the end-of-data at the current position of the file. Note that this command does not affect the allocation of the file, only its end-of-data position. The "Zap" command is not available in Fl.0.

Example:

```
Z 2
```

3.3. CONTROLLER INTERFACE PROTOCOL

3.3.1. Protocol Definition

The interface protocol between the microcomputer and the controller consists of sequences of ASCII characters and makes use of standard ASCII communications controls. The protocol for the simplest controller commands (Allocate, Eject, File, Kill, Mode, Name, Test, Xecute, Zap) is the following:

Microcomputer sends: command-text EOT
Controller sends: ACK EOT

The protocol for controller commands which return informational text (Copy, Delete, Gap, Position, Query) is the following:

Microcomputer sends: command-text EOT
Controller sends: informational-text CR LF ACK EOT

The protocol for controller commands which read data from diskette (Input, Load, Read) is the following:

Microcomputer sends: command-text EOT
Controller sends: SOH diskette-data ACK EOT

The protocol for controller commands which write data to diskette (Output, Save, Write) is the following:

Microcomputer sends: command-text EOT
Controller sends: ENQ EOT
Microcomputer sends: diskette-data EOT
Controller sends: ACK EOT

Finally, the controller may terminate any command at any time with a fatal error diagnostic message, using the following protocol:

Controller sends: NAK fatal-error-msg CR LF EOT

Note that no ACK will be transmitted by the controller in this case.

3.3.2. Error Diagnostic Messages

The controller issues two classes of error diagnostic messages: fatal and non-fatal. Fatal error diagnostic messages are always preceded by a NAK and followed by an EOT. They indicate the premature and unsuccessful termination of a controller command. The various fatal error diagnostic messages are listed below:

COMMAND ERROR ON DRIVE #n

Indicates that the controller received an invalid command or command parameter.

DUP FILE ERROR ON DRIVE #n

Indicates that an attempt was made to create a new file with the same name as an existing file on the same diskette.

HARD DISK ERROR ON DRIVE #n

Indicates that a seek, read, or write error occurred which could not be successfully resolved in five retries.

NOT FOUND ERROR ON DRIVE #n

Indicates that the specified file could not be found in the index of the specified diskette.

OUT OF SPACE ERROR ON DRIVE #n

Indicates that an attempt was made to exceed the capacity of a diskette, an index track, or a file allocation.

READY ERROR ON DRIVE #n

Indicates that an attempt was made to access a diskette drive which is not in ready status.

UNIT ERROR ON DRIVE #n

Indicates that an attempt was made to read, write, or position a logical unit number with which no open file is associated, or that an attempt was made to use the "Copy" or "Gap" commands with one or more files open.

The clause "ON DRIVE #n" is omitted in the case of errors not associated with a particular drive, and is not provided at all in Fl.0.

Note that each fatal message begins with a unique letter, so that an interfacing program need only analyze the first character following a NAK to determine the type of fatal error.

Non-fatal error diagnostic messages are issued for soft disk errors. They are not preceded by a NAK, and they contain the following information:

- . type of disk operation (seek, read, or write)
- . error retry number (1 to 5)
- . track and sector at which error occurred
- . type of error (protect, fault, verify, CRC, or lost)

Multiple error-type indications may be received on a single non-fatal error message, and their meanings are as follows:

- . protect: a write was attempted on a write-protected disk
- . fault: a write fault was received from the drive
- . verify: the desired sector header could not be found
- . CRC: the sector header and/or data failed the CRC test
- . lost: one or more bytes were lost during a data transfer

During the transmission of diskette data (Load, Save, Read, Write, Input, and Output commands), non-fatal error messages are suppressed. They may also be suppressed under all circumstances by means of the Mode command.

3.3.3. Parallel Interface Considerations

The parallel interface offers a number of advantages in interfacing the controller to a microcomputer system: (1) its transfer rate is very fast, (2) it provides complete handshaking to coordinate data transfers in both directions, and (3) it provides a means for uniquely distinguishing communications control characters (EOT, ACK, NAK, SOH, ENQ) from data characters. The last two of these functions are accomplished by means of the controller status byte, whose format is:

- bit 7 - receive data available, control character
- bit 6 - receive data available, data character
- bits 5,4,3,2 - always "1"
- bit 1 - transmit buffer full, data character
- bit 0 - transmit buffer full, control character

When the microcomputer reads the controller data port, bits 7 and 6 of the controller status byte are reset and remain so until the controller sends another character to the parallel interface. When the microcomputer writes the controller data or status port, bit 1 or bit 0 (respectively) is set and remains so until the controller has processed the character from the parallel interface. Since communications control characters cannot be confused with data characters, arbitrary binary information may be read or written freely when using the parallel interface.

Before attempting to write to the controller, the status byte should be read and tested to ensure that no receive data is available and that the transmit buffer is empty. In particular, when the controller is powered up or reset, it outputs a control EOT (in Fl.0, a control ACK followed by a control EOT); these must be read before any command is sent to the controller.

The design of the parallel interface requires two write operations to transmit a control character (e.g., EOT). The first write should address the status register (this will set status bit 0) followed immediately by the second write to the data register (which will set status bit 1). Thus, the controller will see both status bits 0 and 1 set when reading a control character. However, when reading data from the controller, either status bit 6 or status bit 7 will be set by the controller, but never both.

As previously described, reading the controller data register will reset bits 6 and 7 of the status register, but reading the status register does not affect the contents of either register.

3.3.4. RS-232 Serial Interface Considerations

Since the speed of the optional RS-232 serial interface is regulated by a bit-rate clock rather than by cooperative handshaking, another means must be provided for preventing data from being sent to the controller when it is not ready to accept it. (This condition may occur when crossing sector boundaries during the "Save" or "write" commands.) When it is receiving data over the RS-232 interface, the controller normally keeps its RS-232 transmit data in a mark hold ("1") condition. When it is

momentarily unable to accept more data, it places its transmit data in a space hold ("0") condition until it is again able to accept data, then returns it to mark hold.

Since the RS-232 interface provides no means for distinguishing between communications control and data characters, the user must ensure that the significant communications control characters (EOT, ACK, NAK, SOH, ENQ) are not embedded in data sent to or from the controller. If arbitrary binary information is to be read or written, the user must provide a suitable escape convention for these characters.

3.3.5. Sample Driver Program

In order to provide additional guidance in the interfacing of the controller to a microcomputer system, flowcharts and an annotated assembly listing of a sample driver program are provided at the end of this document. The sample driver program makes use of the parallel interface and is coded for an 8080-based microcomputer system.

3.4. DISKETTE FORMAT

3.4.1. General Format

The diskette initialization function of the controller ("Kill" command) creates a diskette format which is IBM 3740 compatible. Each diskette contains 77 tracks with 26 sectors per track and 128 data bytes per sector. Tracks are numbered from 0 to 76 (outer to inner) and sectors are numbered from 1 to 26 on each track. Each sector has a header which defines the track and sector number (soft sectoring). Both the sector header and the data itself are provided with a 16-bit polynomial cyclic redundancy check (CRC) word.

3.4.2. Index Track Format

Track 0 is reserved by the controller for use as an index (i.e., table of contents) for the diskette volume. The controller makes use of an index track format which permits up to 100 files on each volume and which is not IBM 3740 compatible (the IBM 3740 index track format allows only 19 files). Sector 1 of the index track serves as a volume label. Sectors 2 through 26 each contain room for four 32-byte file entries:

bytes 1-8	file name
bytes 9-11	version
byte 12	type
byte 13	(reserved)
bytes 14-15	start of allocation
bytes 16-17	end of allocation
bytes 18-19	end of data
byte 20	end of data (byte offset)
bytes 21-26	date of creation
bytes 27-32	date of last update

3.4.3. Interleaved Sector Sequences

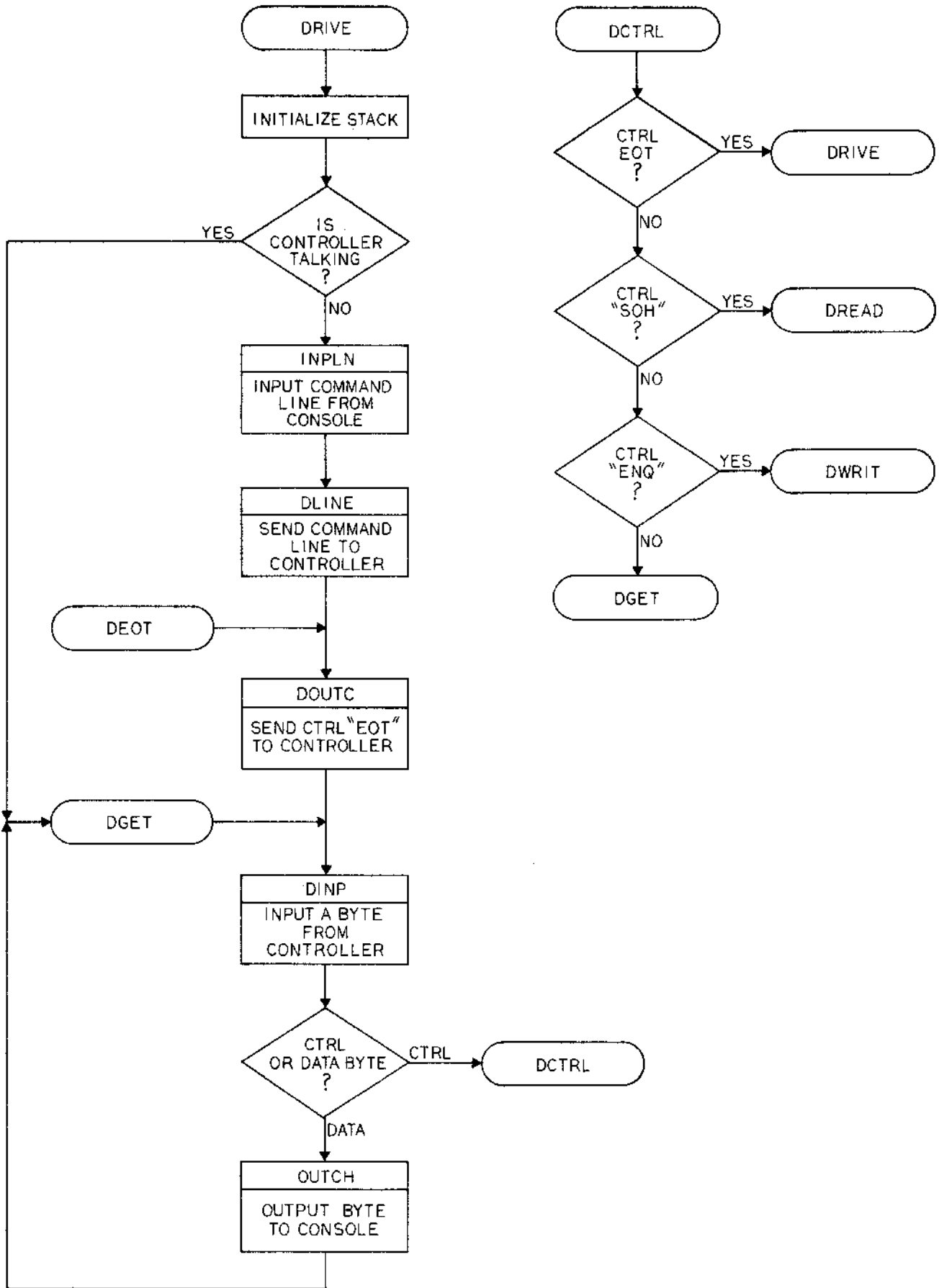
In order to enable users to optimize diskette subsystem performance in a variety of situations, the diskette initialization function of the controller ("Kill" command) supports twelve optional interleaved sector sequences in addition to the ordinary non-interleaved sequence. This function is controlled by the value (1 to 13) of the second parameter of the "Kill" command. The effect of the interleaved sector sequences is to provide additional time to process the data for a sector "N" before sector "N+1" is encountered in the course of diskette rotation. Sequence 1 (non-interleaved) provides the shortest time interval between successively-numbered sectors, and sequences 13 through 2 provide successively longer intervals.

NOTE: Sequences 6 through 9 generally provide optimal results when using the parallel interface in most microcomputer environments.

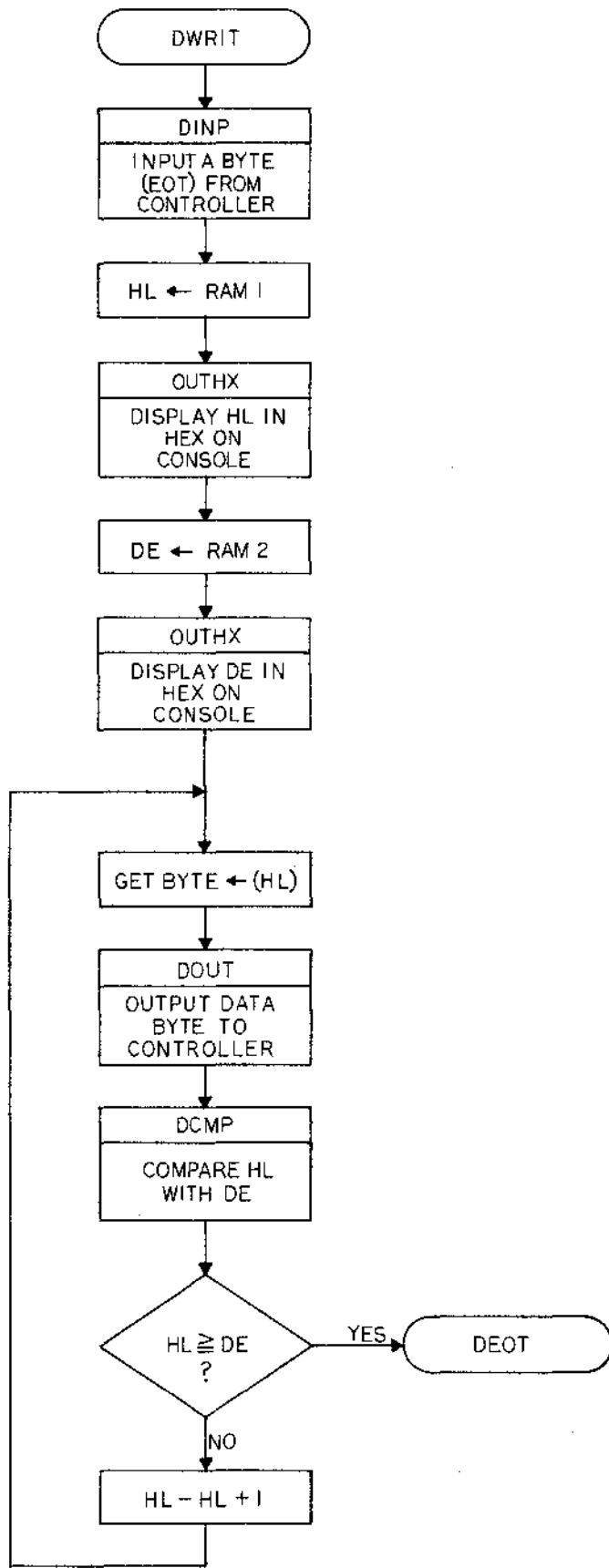
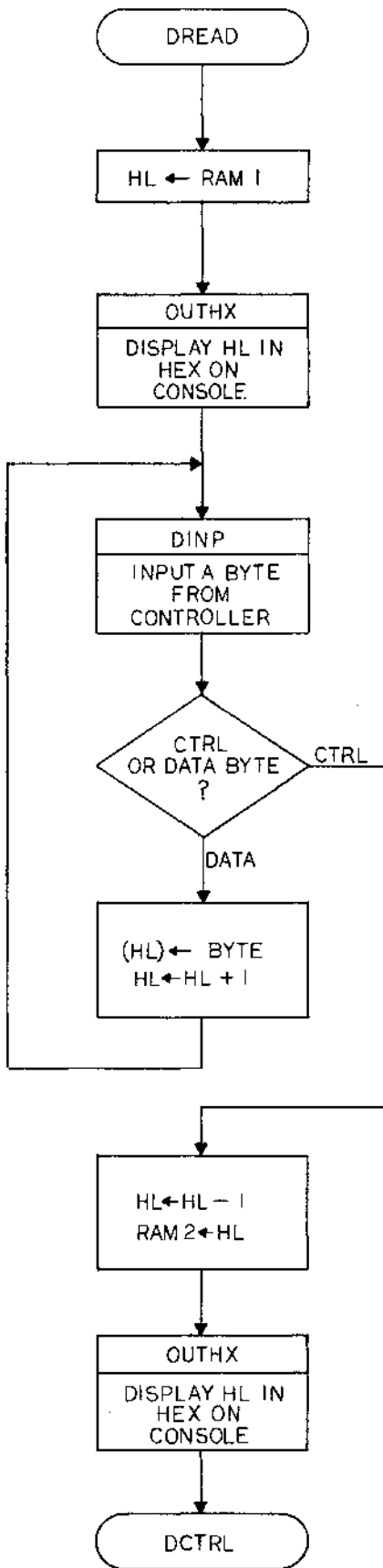
Additional information about these interleaved sector sequences and other diskette formatting considerations may be found in the following IBM document: "The IBM Diskette for Standard Data Interchange", GA 21-9182-0, File No. GENL-03/80.

APPENDIX A

Sample Driver Program Flowchart
Sample 8080 or Z80 Driver Program
Sample 6800 Driver Program



SAMPLE DRIVE PROGRAM FLOWCHART



SAMPLE DRIVE PROGRAM FLOWCHART

PerSci Model 1070 Intelligent Diskette Controller
Appendix B - Hardware Interfacing Information

APPENDIX B

Interface Schematic for S-100 Bus
Interface Schematic for 6800
Interface Timing Data

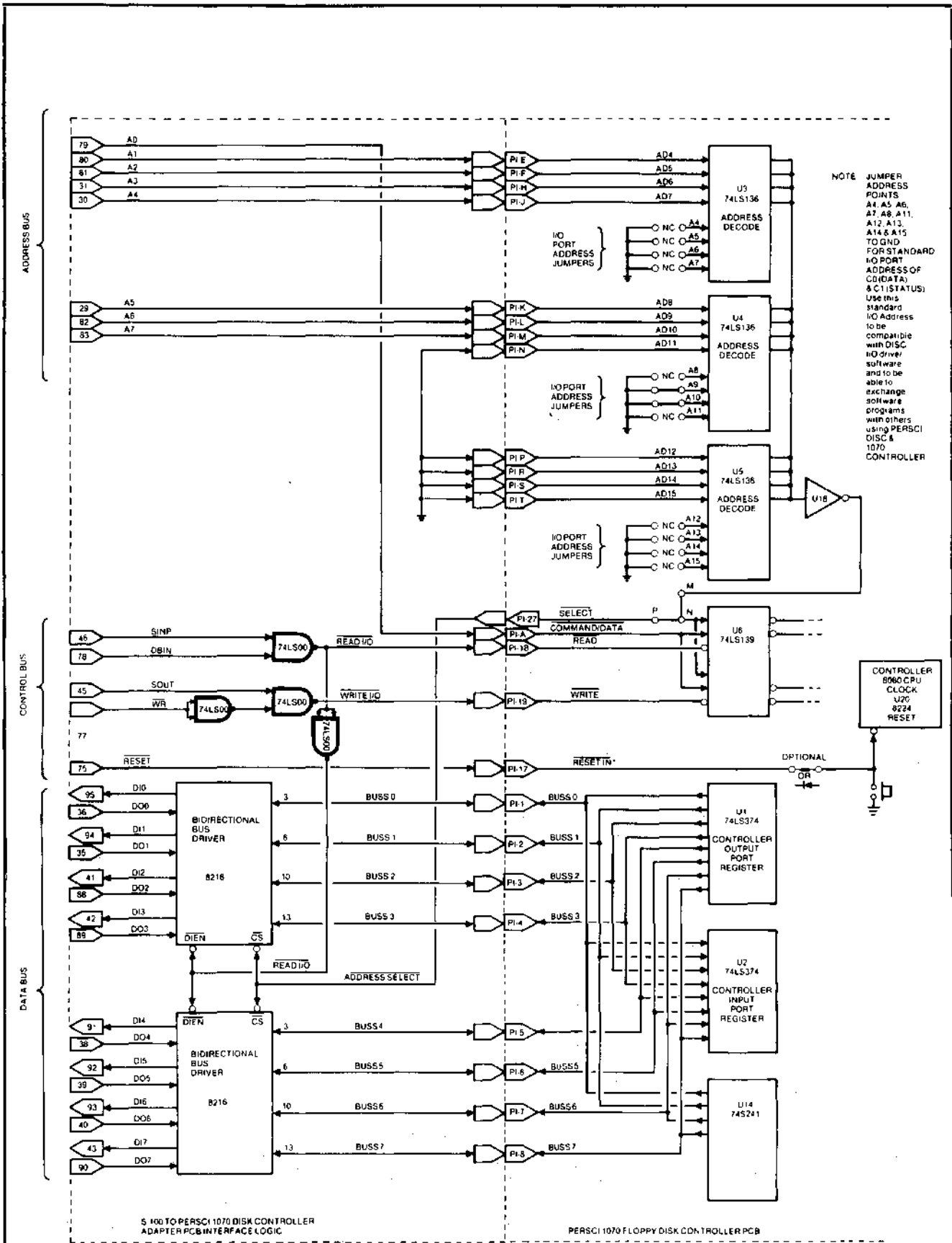
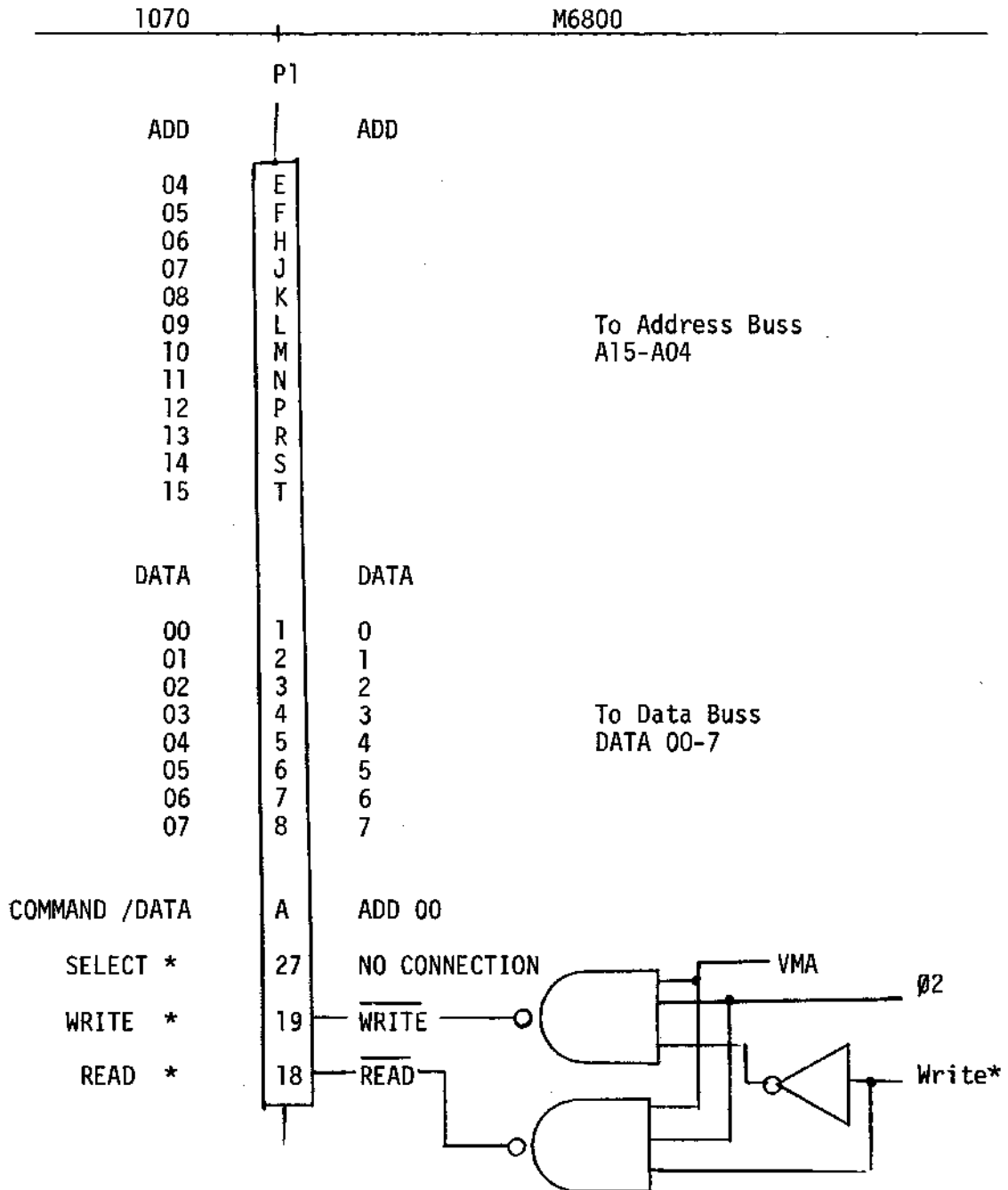


Figure 6. 8080S-100 Bus Controller Interface

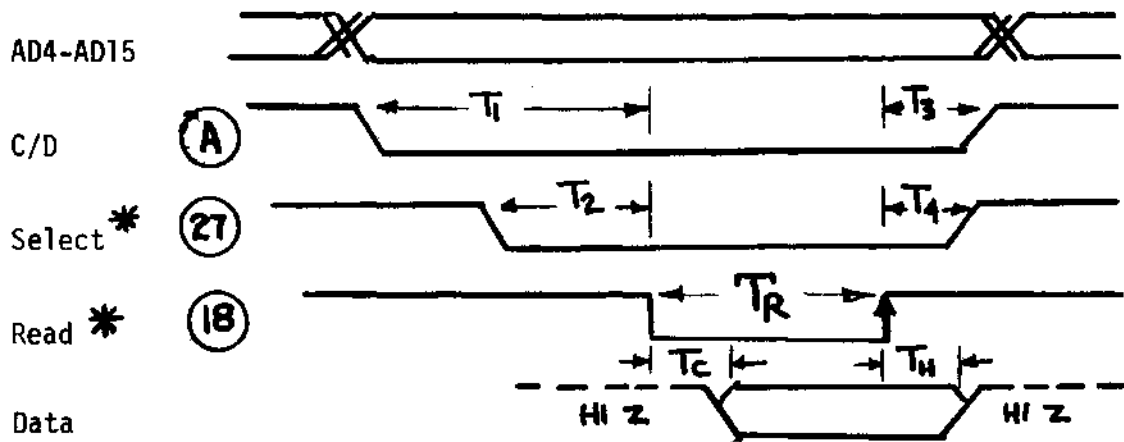
Appendix B-2

ASSUMPTIONS

1. Controller is operating on a parallel bus.
2. Controller is used as a memory ported device using two addresses with Address Bus 00 selecting command or data address.
3. 6800 MPU data bus enable (DBE) is held high for 50 nanoseconds after $\phi 2$ goes to zero volts.



READ TIMING

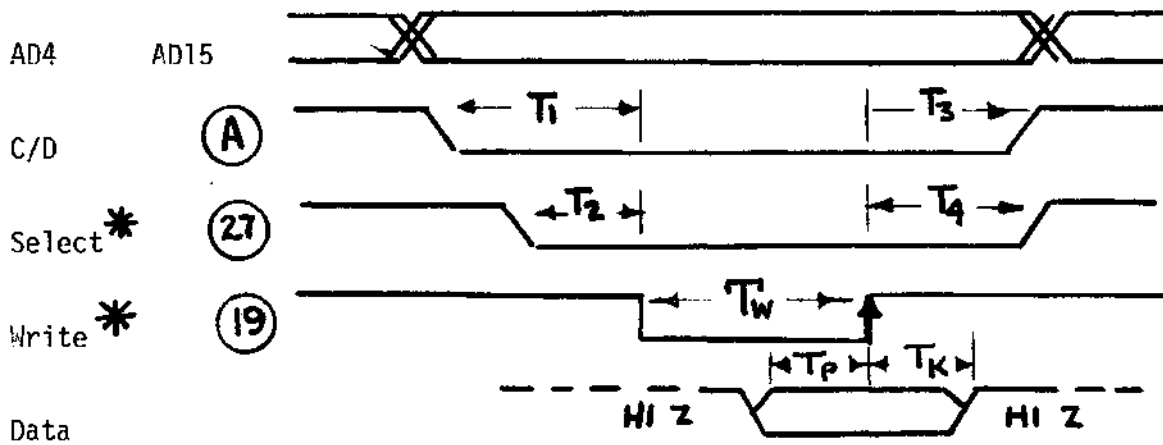


$$10\mu\text{sec} \geq T_R \geq 250\text{ns}$$

$$T_C < 100\text{ns}$$

$$T_H > 5''$$

WRITE TIMING



$$T_W \geq 250\text{ns}$$

$$T_P > 150\text{ns}$$

$$T_K > 100\text{ns}$$

$$T_1 \geq T_2 \geq 0$$

$$T_3 \geq T_4 \geq 0$$

* Active Low Signal

↑ Data Transfer Time

○ 1070 Controller Signal Pin Numbers

APPENDIX C

Brief History of the Model 1070 Controller
Option Jumper Data
Connector Data
Schematic for Controller

BRIEF HISTORY OF THE MODEL 1070 CONTROLLER

The PerSci Model 1070 Controller has evolved through several versions in reaching its present state. The stages (in terms of printed circuit board revisions) were:

PCB 200285-X1:

First production version. A number of cuts and jumpers were required on this PCB.

PCB 200285-X3 (Schematic 200287-X3):

Pull-up resistors (U34) were added to data and control lines from the diskette drives. Filter capacitors were added (C6 and C7). Jumper options were added (C,D,E,F,K,M,N,P,S,R). Two cuts and jumpers were required on this PCB.

PCB 200285-X3 "Kludge" (2114 RAMs on Adapter Boards):

The previously-used RAM chips (9130s and 9131s) used on the -X3 boards became unavailable in the Spring of 1977, and were temporarily replaced with 2114 RAMs mounted on miniature adapter PCBs to correct the incompatibilities in pinouts.

PCB 200249-A (Schematic 200351A):

This is the first production PCB based on the 2114 RAM. The etch is fully correct, with no cuts or jumpers. Space was added between jumper points C and D so that a diode could be used to tie the controller reset line to the host but leave the host reset line isolated from the controller reset pushbutton. A trace was added from U13 pin 15 to J1 pin 16 to enable the controller firmware to simultaneously load all heads when the controller is used with appropriately updated drives. A trace was added to tie U30 pins 8 and 12 to pin 2 (+5v) in order to permit a change from Western Digital 1941 to SMC COM9016 baud-rate generator chips in the optional RS232 serial interface.

PCB 200349-B:

This is now the definitive production printed circuit board for the Model 1070 controller. Primary change from the -A board is the use of a larger-capacity regulator IC for minus 5 volts, to eliminate the need for an add-on thermal radiator used on the previous regulator.

X- and F-Series Firmware:

There have been two different series of firmware used with the Model 1070. Earliest deliveries used various versions of the X-series firmware (X1 through X15), but PerSci no longer issues or supports this firmware. Since Spring of 1977, the controller has been delivered with the newer F-series File Management Firmware. This has been issued in two versions, F1.0 and F1.2, which are described in this document. (F1.1 was never issued.)

BRIEF HISTORY (continued)

PCB 20039-C

The "C" revision of the controller PCB was made the production standard in the spring of 1978. Primary change for this board was the addition of a 10 picofared capacitor in series with the 18.0 MHZ crystal used as the frequency reference for the controller.

FREQUENCY REFERENCE CHANGE

During production of the "B" PCB controllers, the controller frequency standard (Y1) was changed from an 18.432 MHZ crystal to an 18.0 MHZ crystal in series with a 10 picofared capacitor. (See schematic attached Drwg. NO. 200351C). This change was made to improve interchangeability of diskettes formatted by different controllers.

FIRMWARE FMF 1.3

Firmware used with the controller was updated to revision F 1.3 in April of 1978. The command set for this revision was changed such that the Kill Command requires a double KK. (KK volume/drive seq.) This change was in response to users request to reduce operators inadvertent deletion of diskettes files.

F 1.3 is issued in two versions. The first, F 1.3P, is coded for use only with controllers that do not have the serial (RS232) option.

The second version, F 1.3S, is coded for use with either the serial or parallel data ports.

FD1771 NEGATIVE VOLTAGE CHANGE

The negative voltage reference for the FD1771 was changed from minus 2.5 volts to minus 4.17 volts by changing R6 from 1K to 200 ohms. This change was made possible by improved chip performance and results in reduced noise sensitivity.

OPTION JUMPER DATA

A number of options are provided on the Model 1070 controller. They may be selected by connecting jumpers between points as described below:

A-to-B (Factory Installed):

This jumper enables the high-speed seek feature of the controller, which permits head positioning signals to operate at the speeds made possible by the PerSci voice-coil positioner.

C-to-D:

This jumper connects the controller reset line to P1 pin 17, where it may be tied to the host system reset line. On later production PCBs (200349), points C and D were separated to facilitate the use of an isolation diode (cathode at C) in place of a jumper.

E-to-F:

This jumper connects the controller reset complete signal to P1 pin U.

U-to-T:

This jumper is required only when the controller is used with three or four PerSci Model 70 (single) Diskette Drives.

R-to-S:

This jumper should be used if the optional RS232 serial interface is installed to ground the Clear-to-Send line when using RS232 devices which do not provide this control signal. The serial interface will not operate unless either a valid Clear-to-Send signal is present or this jumper is installed.

J-to-H:

This jumper connects the output of flip-flop U9B (receive data available) to P1 pin 22, so that it can be used as an interrupt or other signal to the host system that the controller has a data byte in its transmit buffer for the host.

M, N, P, K Combinations:

Jumpers between these points determine how the controller is selected, described below.

M-to-N-to P (Internal Address Decode):

This connection will allow the controller to be selected by a combination of 12 address signals (AD4 through AD15) determined by jumpers at points A4 through A15. The select signal (active low) is available at P1 pin 27 as an indication to the host system that the controller has recognized its address.

N-to-P (External Select):

This connection will allow the controller to be selected by an external signal (active low) at P1 pin 27.

K-to-P (Test Connection):

This connection makes the receive clock of the optional RS232 interface available at P1 pin 27. This connection is sometimes used by PerSci for test purposes.

PerSci Model 1070 Intelligent Diskette Controller
Appendix C - Supplementary Controller Data

A4-through-A15 (Address Selection):

Jumpers at these points determine which address will select the controller. Each jumper is associated with an address input line at P1 (e.g., jumper A7 with address line AD7). The jumper should be connected if the associated address bit should be high to select the controller.

For example, if the 12 most significant bits of a 16-bit host system address bus (A15 through A0) are connected to controller inputs AD15 through AD4, and if the least significant host system address line (A0) is connected to controller P1 pin A (Command/Data), and if jumpers are installed at points A15, A14, A9, and A5, then host address C200 hex will select the controller data port, and host address C201 hex will select the controller status port (addresses C202 through C20F hex are redundant and should not be used).

CAUTION: Controllers will usually be delivered with jumpers installed at M-to-N-to-P, A15, A14, and R-to-S (for the RS232 option). These jumpers are used by PerSci in checkout and final test. PerSci may change these jumpers to other combinations, without notice. Be certain to verify the proper jumper configuration for your application before placing the controller into service.

CONNECTION OF ADDITIONAL DRIVES

The 1070 Controller will accommodate two Model 277 drives or four Model 70's, without change to the controllers. However, the address logic of the drives added must be modified.

Address logic for the Model 70 and 277 drives is set by jumpers on a select module on the biggest PCB of the drives.

The following are the necessary jumpers:

Model 277

Drive 1 (Side 0 and 1)

Drive 2 (Side 2 and 3)

Select Module Jumpers (U11)

2 to 13, 4 to 11

1 to 14, 6 to 9

Model 70

Drive 1 (Side 0)

Drive 2 (Side 1)

Drive 3 (Side 2)

Drive 4 (Side 3)

Select Module Jumpers (U5)

7 to 8, 3 to 12

7 to 8, 4 to 11

7 to 8, 5 to 10

7 to 8, 6 to 9

1070 CONTROLLER/HOST INTERFACE

SIGNAL	CONTROLLER PINS				SIGNAL
	CONNECTOR PINS				
COMMAND(+)/DATA(-)	A	2	1	1	BUSS 00
	B	4	3	2	BUSS 01
	C	6	5	3	BUSS 02
	D	8	7	4	BUSS 03
AD 04	E	10	9	5	BUSS 04
AD 05	F	12	11	6	BUSS 05
AD 06	H	14	13	7	BUSS 06
AD 07	J	16	15	8	BUSS 07
AD 08	K	18	17	9	
AD 09	L	20	19	10	
AD 10	M	22	21	11	
AD 11	N	24	23	12	
AD 12	P	26	25	13	
AD 13	R	28	27	14	
AD 14	S	30	29	15	
AD 15	T	32	31	16	
RESET COMPLETE	U	34	33	17	RESET IN*
	V	36	35	18	READ*
	W	38	37	19	WRITE*
	X	40	39	20	
	Y	42	41	21	
	Z	44	43	22	
	AA	46	45	23	
	BB	48	47	24	
	CC	50	49	25	
	DD	52	51	26	
	EE	54	53	27	SELECT*
	FF	56	55	28	
RTS	HH	58	57	29	CTS
DTR	JJ	60	59	30	
TXD (XMT DATA)	KK	62	61	31	DSR
	LL	64	63	32	RXD (RCV DATA)
	MM	66	65	33	
-12V	NN	68	67	34	+12V
+5V	PP	70	69	35	+5V
GROUND	RR	72	71	36	GROUND

CONNECTOR, CONTROLLER INTERFACE

Optional Sector Interleave Sequence

DISK RECORD SEQUENCES

blank	01	02	03	04	05	06	07	08	09	10	11	12	13
1	1	1	1	1	1	1	1	1	1	1	1	1	1
2	2	3	4	5	6	7	8	9	10	11	12	13	14
3	3	5	7	9	11	13	15	17	19	21	23	25	2
4	4	7	10	13	16	19	22	25	2	2	2	2	15
5	5	9	13	17	21	25	2	2	11	12	13	14	3
6	6	11	16	21	26	2	9	10	20	22	24	26	16
7	7	13	19	25	2	8	16	18	3	3	3	3	4
8	8	15	22	2	7	14	23	26	12	13	14	15	17
9	9	17	25	6	12	20	3	3	21	23	25	4	5
10	10	19	2	10	17	26	10	11	4	4	4	16	18
11	11	21	5	14	22	3	17	19	13	14	15	5	8
12	12	23	8	18	3	9	24	4	22	24	26	17	19
13	13	25	11	22	8	15	4	12	5	5	5	6	7
14	14	2	14	26	13	21	11	20	14	15	16	18	20
15	15	4	17	3	18	4	18	5	23	25	6	7	8
16	16	6	20	7	23	10	25	13	6	6	17	19	21
17	17	8	23	11	4	16	5	21	15	16	7	8	9
18	18	10	26	15	9	22	12	6	24	26	18	20	22
19	19	12	3	19	14	5	19	14	7	7	8	9	10
20	20	14	6	23	19	11	26	22	16	17	19	21	23
21	21	16	9	4	24	17	6	7	26	8	9	10	11
22	22	18	12	8	5	23	13	15	8	18	20	22	24
23	23	20	15	12	10	6	20	23	17	9	10	11	12
24	24	22	18	16	15	12	7	8	26	19	21	23	25
25	25	24	21	20	20	18	14	16	9	10	11	12	13
26	26	26	24	24	26	24	21	24	18	20	22	24	26

Numbers at column top are interleave sequences specified by kill command.

Columns show sector sequence for specified interleave.

N.B. "Blank" sequence changes index track only.

APPENDIX D

Applications Note for Simultaneous Head Load

APPENDIX D

APPLICATIONS NOTE FOR SIMULTANEOUS HEAD LOAD CIRCUIT MODIFICATION

CONTROLLER MODIFICATIONS

Applicable: PCB 200285 X1, X3

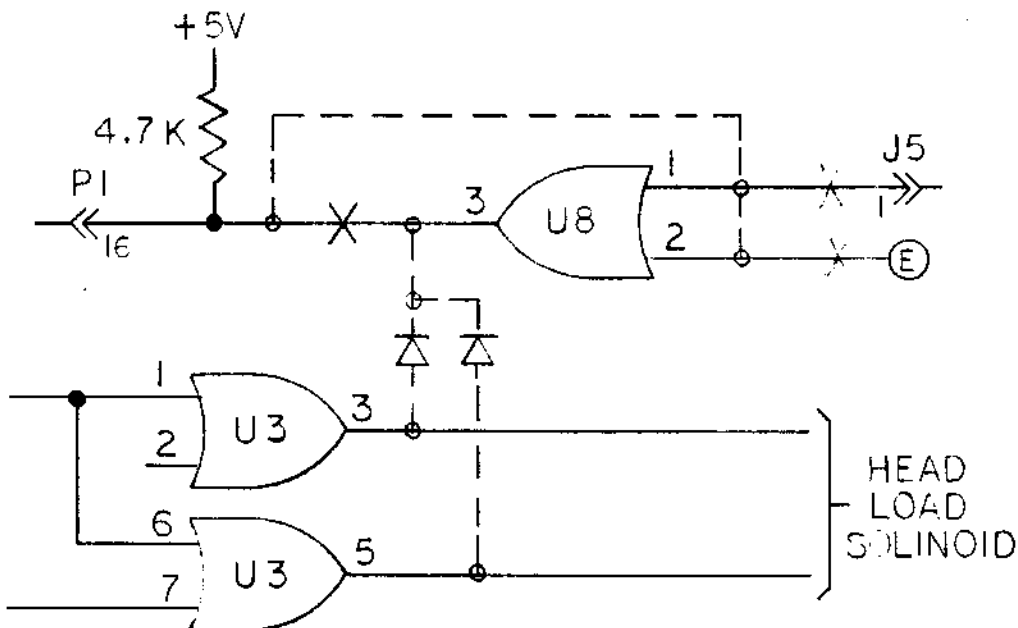
Add Jumper U13 Pin 15 → J1 Pin 16

DRIVE MODIFICATIONS

Applicable: Model 277 drive with Data and Interface Assembly
200263-003, Rev. A, B, C, D, E, or F (PCB 200262A).

1. Cut traces at U8 Pins 1, 2, and 3.
2. Add jumper from P1 Pin 16 to U8 Pins 1 and 2.
3. Add diode — anode at U3 Pin 3, cathode at U8 Pin 1.
4. Add diode — anode at U3 Pin 5, cathode at U8 Pin 1

The result should be as shown in the sketch below:



CONTROLLERS AND DRIVE WHEN MODIFIED AS ABOVE REQUIRE
FILE MANAGEMENT FIRMWARE VERSION F1.2

```

.SBTTL "Section 1 - Controller Interface Routines"
;
;
; This is the basic driver routine which sends console
; commands to the controller, controller messages to
; the console, and controls the transmission of files
; and records between the controller and microcomputer
; RAM.
;
0000 319B01 DRIVE: LXI SP,STACK ;INITIALIZE STACK
0003 DBC1 IN DSTAT ;GET DISK STATUS
0005 E6C0 ANI 0C0H ;IS DISK TALKING?
0007 C21500 JNZ DGET ;IF SO, LISTEN FIRST
000A CDAA00 CALL INPLN ;INPUT CONSOLE LINE
000D CD7200 CALL DLINE ;SEND COMMAND TO DISK
0010 3E04 DEOT: MVI A,EOT ;SEND "EOT" TO DISK
0012 CD8D00 CALL DOUTC ;AS CONTROL BYTE
0015 CD7C00 DGET: CALL DINP ;INPUT BYTE FROM DISK
0018 DA2100 JC DCTRL ;CONTROL OR DATA BYTE?
001B CD4C01 CALL OUTCH ;DATA, SEND TO CONSOLE
001E C31500 JMP DGET
0021 FE04 DCTRL: CPI EOT ;CONTROL, WHAT KIND?
0023 CA0000 JZ DRIVE ;"EOT", COMMAND IS DONE
0026 FE01 CPI SOH
0028 CA3300 JZ DREAD ;"SOH", DO DISK READ
002B FE05 CPI ENQ
002D CA5000 JZ DWRT ;"ENQ", DO DISK WRITE
0030 C31500 JMP DGET ;ELSE IGNORE (ERROR)
;
;
; This routine controls a disk read into RAM.
;
0033 2AA600 DREAD: LHLD RAM1 ;GET RAM STARTING ADDR
0036 CD0501 CALL OUTHX ;DISPLAY ON CONSOLE
0039 CD7C00 DREAL: CALL DINP ;INPUT BYTE FROM DISK
003C DA4400 JC DREAX ;CONTROL OR DATA BYTE?
003F 77 MOV M,A ;DATA, MOVE TO RAM
0040 23 INX H ;INCREMENT RAM ADDR
0041 C33900 JMP DREAL ;NEXT BYTE
0044 F5 DREAX: PUSH PSW ;CONTROL, SAVE BYTE
0045 2B DCX H ;DECREMENT RAM ADDR
0046 22A800 SHLD RAM2 ;SAVE RAM ENDING ADDR
0049 CD0501 CALL OUTHX ;DISPLAY ON CONSOLE
004C F1 POP PSW ;GET CONTROL BYTE
004D C32100 JMP DCTRL ;GO ANALYZE IT
;
;
; This routine controls a disk write from RAM.
;
0050 CD7C00 DWRT: CALL DINP ;INPUT BYTE FROM DISK
0053 D25000 JNC DWRT ;SHOULD BE AN "EOT"
0056 2AA600 LHLD RAM1 ;GET RAM STARTING ADDR
0059 CD0501 CALL OUTHX ;DISPLAY ON CONSOLE
005C EB XCHG
005D 2AA800 LHLD RAM2 ;GET RAM ENDING ADDR

```

```

0060 CD0501          CALL    OUTHX    ;DISPLAY ON CONSOLE
0063 EB            XCHG          ;START IN HL, END IN DE
0064 7E            DWRIL:  MOV     A,M      ;GET BYTE FROM RAM
0065 CD8700        CALL    DOUT     ;SEND DATA TO DISK
0068 CD3C01        CALL    DCMPL   ;COMPARE ADDR TO END
006B D21000        JNC     DEOT    ;AT END, SEND "EOT"
006E 23            INX     H         ;ELSE INCREMENT RAM ADDR
006F C36400        JMP     DWRIL   ;PROCESS NEXT BYTE
;
;
; This routine sends a command to the controller.
;
0072 CD2601        DLINL:  CALL    GETCH   ;GET CHAR FROM PUFFER
0075 B8            RC           ;EXHAUSTED, ALL DONE
0076 CD8700        CALL    DOUT     ;SEND CHARACTER TO DISK
0079 C37200        JMP     DLINL   ;PROCESS NEXT CHARACTER
;
;
; This routine inputs a byte from the controller
; and sets the carry flag if it is a control byte.
;
007C DBC1          DINP:   IN      DSTAT   ;GET DISK STATUS BYTE
007E E6C0          ANI     0C0H     ;RECEIVE DATA AVAILABLE?
0080 CA7C00        JZ      DINP     ;NO, WAIT UNTIL IT IS
0083 17            RAL          ;SET CARRY IF CONTROL
0084 DBC0          IN      DDATA    ;GET DISK DATA BYTE
0086 C9            RET          ;ALL DONE
;
;
; This routine outputs a data byte to the controller.
;
0087 CD9500        DOUT:   CALL    DOUTW   ;WAIT UNTIL READY
008A D3C0          OUT     DDATA    ;WRITE DISK DATA BYTE
008C C9            RET          ;ALL DONE
;
;
; This routine outputs a control byte to the controller.
;
008D CD9500        DOUTC:  CALL    DOUTW   ;WAIT UNTIL READY
0090 E3C1          OUT     DSTAT   ;WRITE DISK STATUS BYTE
0092 D3C0          OUT     DDATA    ;WRITE DISK DATA BYTE
0094 C9            RET          ;ALL DONE
;
;
; This routine waits for the disk transmit buffer to be
; empty and ready for another byte. It also arbitrates
; if disk and host try to transmit to one another at
; the same time.
;
0095 F5            DOUTW:  PUSH    PSW     ;SAVE BYTE TO SEND
0096 DBC1          IN      DSTAT   ;GET DISK STATUS BYTE
0098 E6C0          ANI     0C0H     ;IS DISK TRANSMITTING?
009A C2A400        JNZ     DOUTX   ;YES, BREAK THE TIE
009D DBC1          IN      DSTAT   ;GET DISK STATUS AGAIN
009F E603          ANI     03H     ;IS TRNSMT BUFFER EMPTY?

```

TDL Z80 RELOCATING ASSEMBLER VERSION 1.2
Sample Driver Program for PerSci 1070 Controller
Section 1 - Controller Interface Routines

```
00A1 C29600      JNZ      DOUTW+1 ;NO, WAIT UNTIL IT IS
00A4 F1          DOUTX: POP      PSW      ;RESTORE BYTE TO SEND
00A5 C9          RET        ;ALL DONE
;
;
; Symbolic Equivalences
;
00C0            DDATA      = 0C0H   ;CONTROLLER DATA PORT
00C1            DSTAT      = 0C1H   ;CONTROLLER STATUS PORT
0004            EOT        = 04H    ;ASCII "EOT"
0001            SOH        = 01H    ;ASCII "SOH"
0005            ENQ        = 05H    ;ASCII "ENQ"
;
;
; RAM Working Storage
00A6 0000       RAM1:      .WORD    0      ;SAVE/LOAD START ADDR
00A8 0000       RAM2:      .WORD    0      ;SAVE/LOAD END ADDR
;
;
.PAGE
```

```

.SBTTL "Section 2 - Common Subroutines"
;
;
; This routine inputs a line from the console device
; into a RAM buffer, and processes backspace and
; line-delete functions.
;
00AA CDFA00      INPLN: CALL    CRLF      ;CR/LF TO CONSOLE
00AD 3E3E              MVI      A,'>'    ;GET COMMAND PROMPT
00AF CD4C01      CALL    OUTCH     ;SEND TO CONSOLE
00B2 215801      LXI      H,IBUFF    ;GET BUFFER ADDRESS
00B5 227801      SHLD    IBUFP     ;INITIALIZE POINTER
00B8 0E00              MVI      C,0      ;INITIALIZE COUNT
00BA CD4201      INPLI: CALL    INPCH     ;GET CHAR FROM CONSOLE
00BD E67F              ANI      7FH     ;STRIP PARITY BIT
00BF FE20              CPI      ' '      ;TEST IF CONTROL CHAR
00C1 DAD400      JC      INPLC     ;YES, GO PROCESS
00C4 77              MOV      M,A      ;NO, PUT IN BUFFER
00C5 3E20              MVI      A,32     ;GET BUFFER SIZE
00C7 B9              CMP      C        ;TEST IF FULL
00C8 CABA00      JZ      INPLI     ;YES, LOOP
00CB 7E              MOV      A,M      ;RECALL CHARACTER
00CC 23              INX      H        ;INCR POINTER
00CD 0C              INR      C        ;AND INCR COUNT
00CE CD4C01      INPLE: CALL    OUTCH     ;ECHO CHARACTER
00D1 C3BA00      JMP      INPLI     ;GET NEXT CHAR
00D4 FE08              INPLC: CPI      08H     ;TEST IF BACKSPACE
00D6 CAEB00      JZ      INPLB     ;YES, KILL CHAR
00D9 FE1B              CPI      1BH     ;TEST IF ESCAPE
00DB CAF500      JZ      INPLK     ;YES, KILL LINE
00DE FE0D              CPI      0DH     ;TEST IF RETURN
00E0 C2BA00      JNZ     INPLI     ;NO, IGNORE CHAR
00E3 79              MOV      A,C      ;GET COUNT
00E4 327A01      STA     IBUFC     ;SAVE IT
00E7 CDFA00      CALL    CRLF     ;SEND CR/LF TO CONSOLE
00EA C9              RET              ;DONE
00EB 2B              INPLB: DCX     H      ;DECREMENT POINTER
00EC 0D              DCR     C        ;DECREMENT COUNT
00ED F2CE00      JP      INPLE     ;IF NOT NEG, GO ECHO
00F0 23              INX     H        ;IF NEG, UNDO DECR
00F1 0C              INR     C        ;
00F2 C3BA00      JMP     INPLI     ;GET NEXT CHAR
00F5 AF              INPLK: XRA     A      ;KILL BY SETTING
00F6 327A01      STA     IBUFC     ;COUNT TO ZERO
00F9 C9              RET              ;DONE
;
;
; This routine sends a CR LF sequence to the console.
;
00FA 3E0D      CRLF: MVI      A,0DH     ;GET A CR
00FC CD4C01      CALL    OUTCH     ;DISPLAY IT
00FF 3E0A      MVI      A,0AH     ;GET A LF
0101 CD4C01      CALL    OUTCH     ;DISPLAY IT
0104 C9              RET              ;DONE
;

```



```

;
; This routine outputs the contents of registers H-L
; as a four-digit hexadecimal number on the console.
;
0105 3E20      OUTHX: MVI      A,' '    ;GET A SPACE
0107 CD4C01    CALL     OUTCH   ;SEND TO CONSOLE
010A 7C        MOV     A,H      ;GET TOP HALF OF WORD
010B CD0F01    CALL     OUTH1   ;DISPLAY IN HEX
010E 7D        MOV     A,L      ;SAME WITH BOTTOM HALF
010F F5        OUTH1: PUSH    PSW    ;SAVE LOW-ORDER DIG
0110 1F        RAR     ;GET HIGH-ORDER DIG
0111 1F        RAR
0112 1F        RAR
0113 1F        RAR
0114 CD1801    CALL     OUTH    ;DISPLAY HEX DIGIT
0117 F1        POP     PSW    ;GET OTHER DIGIT
0118 E60F      OUTH:  ANI     0FH    ;EXTRACT DIGIT
011A C630      ADI     '0'    ;ADD ASCII ZONE BITS
011C FE3A      CPI     '9'+1  ;TEST IF A-F
011E DA4C01    JC      OUTCH   ;NO, OUTPUT IT
0121 C607      ADI     'A'-'9'-1 ;YES, ADD BIAS FOR A-F
0123 C34C01    JMP     OUTCH   ;OUTPUT IT

```

```

;
; This routine obtains a character from the RAM buffer
; and sets the carry flag if exhausted.
;
0126 E5        GETCH: PUSH    H      ;SAVE REGS
0127 2A7801    LHLD   IBUFP   ;GET POINTER
012A 3A7A01    LDA    IBUFC   ;GET COUNT
012D D601      SUI    1       ;DECREMENT WITH CARRY
012F DA3A01    JC    GETCX   ;NO MORE CHARACTERS
0132 327A01    STA    IBUFC   ;REPLACE COUNT
0135 7E        MOV     A,M      ;GET CHARACTER
0136 23        INX    H      ;INCR POINTER
0137 227801    SHLD  IBUFP   ;REPLACE POINTER
013A E1        GETCX: POP     H      ;RESTORE REGS
013B C9        RET     ;DONE (CARRY IF NO CHAR)

```

```

;
; This routine compares D-E with H-L.
;
013C 7C        DCMP:  MOV     A,H      ;GET MOST SIGNIF
013D BA        CMP     D      ;COMPARE MOST SIGNIF
013E C0        RNZ     ;NONZERO, DONE
013F 7D        MOV     A,L      ;GET LEAST SIGNIF
0140 BB        CMP     E      ;COMPARE LEAST SIGNIF
0141 C9        RET     ;DONE

```

```

;
; These routines perform input and output from and to
; the console device, passing on character in the A-reg.
; They must be coded to work with the particular console
; I/O interface arrangement of each microcomputer. The
; two routines must not modify any registers other than

```

```

; the A-reg.
;
0142 DB00      INPCH:  IN      0      ;GET CONSOLE STATUS
0144 E601      ANI      01H     ;RECEIVE DATA AVAILABLE?
0146 C24201    JNZ      INPCH   ;NO, WAIT UNTIL IT IS
0149 DB01      IN      1      ;GET CONSOLE DATA
014B C9        RET          ;ALL DONE

;
014C F5        OUTCH:  PUSH    PSW   ;SAVE DATA TO BE SENT
014D DB00      IN      0      ;GET CONSOLE STATUS
014F E680      ANI      80H     ;TRANSMIT BUFFER EMPTY?
0151 C24D01    JNZ      OUTCH+1 ;NO, WAIT UNTIL IT IS
0154 F1        POP     PSW   ;GET SAVED DATA
0155 D301      OUT     1      ;SEND TO CONSOLE
0157 C9        RET          ;ALL DONE

;
;
; RAM Working Storage
;
0158          IBUFF:  .BLKB  32    ;INPUT TEXT BUFFER
0178          IBUFP:  .BLKB  2     ;INPUT POINTER
017A          IBUFC:  .BLKB  1     ;INPUT COUNTER
017B          .BLKB  32    ;STACK AREA
019B          STACK = .      ;TOP OF STACK
;
;
0000          .END    DRIVE   ;END OF ASSEMBLY

```

CRLF	00FA	DCMP	013C	DCTRL	0021	DDATA	00C0
DEOT	0010	DGET	0015	DINP	007C	DLINE	0072
DOUT	0087	DOUTC	008D	DOUTW	0095	DOUTX	00A4
DREAD	0033	DREAL	0039	DREAX	0044	DRIVE	0000
DSTAT	00C1	DWRIL	0064	DWRIT	0050	ENQ	0005
EOT	0004	GETCH	0126	GETCX	013A	IBUFC	017A
IBUFF	0158	IBUFP	0178	INPCH	0142	INPLB	00EB
INPLC	00D4	INPLE	00CE	INPLI	00BA	INPLK	00F5
INPLN	00AA	OUTCH	014C	OUTH	0118	OUTH1	010F
OUTHX	0105	RAM1	00A6	RAM2	00A8	SOH	0001
STACK	019B						

```

00001          NAM          DRIVER
00002          *
00003          * ADAPTED TO 6800 FROM PERSCI 8080
00004          * PROGRAM BY MIKE SMITH
00005          * D.I.Y. INDUSTRIES
00006          * 17315 S.E. RIVER ROAD
00007          * MILWAUKIE, OREGON 97222
00008          *
00009          *
00010          *****
00011          *SAMPLE DRIVER PROGRAM TO INTERFACE WITH
00012          *PERSCI MODEL 1070 DISKETTE CONTROLLER
00013          *****
00014          *
00015          *THIS PROGRAM OPERATES ON A 6800 BASED MICRO-
00016          *COMPUTER. IT ASSUMES THAT THE PERSCI MODEL
00017          *1070 DISKETTE CONTROLLER IS INTERFACED VIA
00018          *ITS PARALLEL PORT IN SUCH A MANNER THAT ITS
00019          *DATA AND STATUS BYTES APPEAR TO THE 6800 AS
00020          *MEMORY LOCATIONS E000 AND E001 HEX. RESPECT-
00021          *IVELY. IT ALSO ASSUMES THAT AN ASCII CONSOLE
00022          *DEVICE IS CONNECTED TO THE MICROCOMPUTER.
00023          *
00024          *
00025          *THIS PROGRAM HAS BEEN MODIFIED TO BE
00026          *CALLED AS A SUBROUTINE.
00027          *
00028          *
00029          *THIS PROGRAM LISTING IS DIVIDED IN TWO SEC-
00030          *TIONS. SECTION ONE CONTAINS THOSE ROUTINES
00031          *WHICH ARE UNIQUE TO THE DISKETTE CONTROLLER
00032          *INTERFACE. IT REQUIRES ONLY 151 BYTES OF
00033          *PROGRAM STORAGE AND 5 BYTES OF RAM.
00034          *
00035          *SECTION TWO CONTAINS GENERAL I/O SUBROUTINES
00036          *WHICH ARE ROUTINELY A PART OF MOST MICRO-
00037          *COMPUTER OPERATING SYSTEM OR MONITORS, AND
00038          *THUS WHICH WILL NOT NEED TO BE DUPLICATED IN
00039          *MOST INSTALLATIONS.
00040          *
00041          *
00042          *****SECTION ONE*****
00043          *
00044          *
00045          *THIS IS THE BASIC DRIVER ROUTINE WHICH SENDS
00046          *CONSOLE COMMANDS TO THE CONTROLLER, CONTROLLER
00047          *MESSAGES TO THE CONSOLE, AND CONTROLS THE
00048          *TRANSMISSION OF FILES AND RECORDS BETWEEN THE
00049          *CONTROLLER AND MICROCOMPUTER RAM.
00050          *
00051          *
00052          *
00053          *
00054          *

```

```

00055 *
00056 *
00057 D000 ORG $D000
00058 OPT P
00059 OPT S
00060 OPT M
00061 *
00062 *
00063 D000 B6 E001 DRIVE LDA A DSTAT GET DISC STATUS
00064 D003 84 C0 AND A #S00 SEE IF READY YET
00065 D005 26 0A BNE DGET IF NOT THEN CLEAN UP
00066 D007 BD D098 START JSR INPLN INPUT CONSOLE LINE
00067 D00A 8D 56 BSR DLINE SEND COMMAND TO DISK
00068 D00C 86 04 DEOT LDA A #S04 SEND "EOT" TO DISK
00069 D00E BD D07E JSR DOUTC AS CONTROL BYTE
00070 D011 8D 59 DGET BSR DINP INPUT BYTE FROM DISK
00071 D013 25 05 BCS DCTRL CONTROL OR DATA BYTE?
00072 D015 BD FEAA JSR OUTCH DATA, SEND TO CONSOLE
00073 D018 20 F7 BRA DGET GET NEXT BYTE
00074 D01A 81 04 DCTRL CMP A #S04 CONTROL, WHAT KIND?
00075 D01C 26 01 BNE GO EOT, COMMAND IS DONE
00076 D01E 39 RTS RETURN TO CALLER
00077 D01F 81 01 GO CMP A #S01
00078 D021 27 06 BEQ DREAD SOH, DO DISK READ
00079 D023 81 05 CMP A #S05
00080 D025 27 1C BEQ DWRT ENQ, DO DISK WRITE
00081 D027 20 E8 BRA DGET ELSE IGNORE
00082 *
00083 *
00084 *THIS ROUTINE CONTROLS A DISK READ INTO RAM
00085 *
00086 *
00087 D029 FE E100 DREAD LDX RAM1 GET RAM STARTING ADDR
00088 D02C BD D11F JSR OUTHX DISPLAY ON CONSOLE
00089 D02F 8D 3B DREAL BSR DINP INPUT BYTE FROM DISK
00090 D031 25 05 BCS DREAX CONTROL OR DATA BYTE?
00091 D033 A7 00 STA A 0,X DATA, MOVE TO RAM
00092 D035 08 INX INCREMENT RAM ADDR
00093 D036 20 F7 BRA DREAL NEXT BYTE
00094 D038 36 DREAX PSH A CONTROL, SAVE BYTE
00095 D039 09 DEX DECREMENT RAM ADDR
00096 D03A FF E102 STX RAM2 SAVE RAM ENDING ADDR
00097 D03D BD D11F JSR OUTHX DISPLAY ON CONSOLE
00098 D040 32 PUL A GET CONTROL BYTE
00099 D041 20 D7 BRA DCTRL GO ANALYZE IT
00100 *
00101 *
00102 *
00103 *
00104 *
00105 *
00106 *
00107 *
00108 *

```

```

00109          *THIS ROUTINE CONTROLS A DISK WRITE FROM RAM
00110          *
00111          *
00112 D043 8D 27  DWRIT  BSR          DINP          INPUT BYTE FROM DISK
00113 D045 24 FC          BCC          DWRIT          SHOULD BE AN EOT
00114 D047 FE E100      LDX          RAM1          GET RAM STARTING ADDR
00115 D04A BD D11F      JSR          OUTHX         DISPLAY ON CONSOLE
00116 D04D FE E102      LDX          RAM2          GET RAM ENDING ADDR
00117 D050 BD D11F      JSR          OUTHX         DISPLAY ON CONSOLE
00118 D053 FE E100      LDX          RAM1          GET STARTING ADRS
00119 D056 A6 00  DWRIL  LDA  A          0,X          GET BYTE FROM RAM
00120 D058 8D 1E          BSR          DOUT          SEND DATA TO DISK
00121 D05A BC E102      CPX          RAM2          COMPARE ADRS TO END
00122 D05D 27 AD          BEQ          DEOT          AT END, SEND EOT
00123 D05F 08          INX          ELSE INCREMENT RAM ADDR
00124 D060 20 F4          BRA          DWRIL         PROCESS NEXT BYTE
00125          *
00126          *
00127          *THIS ROUTINE SENDS A LINE TO THE CONTROLLER
00128          *
00129          *
00130 D062 BD D14A  DLINE  JSR          GETCH          GET CHAR FROM BUFFER
00131 D065 24 01          BCC          CONT          CHECK IF DONE
00132 D067 39          RTS          DONE? THEN RETURN
00133 D068 8D 0E  CONT    BSR          DOUT          SEND CHARACTER TO DISK
00134 D06A 20 F6          BRA          DLINE         PROCESS NEXT CHARACTER
00135          *
00136          *
00137          *
00138          *THIS ROUTINE INPUTS A BYTE FROM THE CONTROLLER
00139          *AND SETS CARRY=1 IF A CONTROL BYTE
00140          *
00141          *
00142 D06C B6 E001  DINP    LDA  A          DSTAT          GET DISK STATUS BYTE
00143 D06F 84 C0          AND  A          #5C0          RECEIVE DATA AVAILIABLE?
00144 D071 27 F9          BEQ          DINP          NO, WAIT UNTL IT IS
00145 D073 49          ROL  A          SET CARRY IF CONTROL
00146 D074 B6 E000      LDA  A          DDATA          GET DISK DATA BYTE
00147 D077 39          RTS          AND RETURN
00148          *
00149          *
00150          *THIS ROUTINE SENDS A DATA BYTE TO THE CONTROLLER
00151          *
00152          *
00153 D078 8D 0D  DOUT    BSR          DOUTW         WAIT UNTIL READY
00154 D07A B7 E000      STA  A          DDATA          WRITE DISK DATA BYTE
00155 D07D 39          RTS          ALL DONE RETURN
00156          *
00157          *
00158          *
00159          *
00160          *THIS ROUTINE SENDS A CTRL BYTE TO THE CONTROLLER
00161          *
00162          *
00163 D07E 8D 07  DOUTC  BSR          DOUTW         WAIT UNTIL READY

```

```

00164 D080 B7 E001      STA A      DSTAT      WRITE DISK STATUS BYTE
00165 D083 B7 E000      STA A      DDATA      WRITE DISK DATA BYTE
00166 D086 39           RTS          ALL DONE, RETURN
00167                  *
00168                  *
00169                  *THIS ROUTINE WAITS FOR THE DISK TRANSMIT BUFFER
00170                  *TO BE EMPTY AND READY FOR ANOTHER BYTE. IT ALSO
00171                  *ARBITRATES IF DISK AND HOST TRY TO TRANSMIT
00172                  *TO ONE ANOTHER AT THE SAME TIME.
00173                  *
00174                  *
00175 D087 36           DOUTW PSH A      SAVE BYTE TO SEND
00176 D088 B6 E001      LDA A      DSTAT      GET DISK STATUS BYTE
00177 D08B 84 C0        AND A      #5C0      IS DISK TRANSMITTING?
00178 D08D 26 07        BNE          DOUTX      YES, BREAK THE TIE
00179 D08F B6 E001      LDA A      DSTAT      GET DISK STATUS AGAIN
00180 D092 84 03        AND A      #503      IS TRNSMT BUFFER EMPTY?
00181 D094 26 F2        BNE          DOUTW+1    NO, WAIT UNTIL IT IS
00182 D096 32           DOUTX PUL A      RESTORE BYTE TO SEND
00183 D097 39           RTS          ALL DONE RETURN
00184                  *
00185                  *
00186                  *SYMBOLIC EQUIVALENCES
00187                  *
00188                  *
00189          E000      DDATA EQU          $E000      CONTROLLER DATA BYTE
00190          E001      DSTAT EQU          $E001      CONTROLLER STATUS BYTE
00191                  *
00192                  *
00193                  *RAM WORKING STORAGE
00194                  *
00195                  *
00196          E100      RAM1 EQU          $E100      RAM START ADDR
00197          E102      RAM2 EQU          $E102      RAM END ADDR
00198          E104      XTEMP EQU          $E104      TEMP INDEX STORE
00199                  *
00200                  *

```

```

00202 *
00203 *
00204 *****SECTION TWO*****
00205 *
00206 *
00207 *THIS ROUTINE INPUTS A LINE FROM THE CONSOLE
00208 *INTO A RAM BUFFER, AND PROCESSES BACKSPACE
00209 *AND LINE DELETE FUNCTIONS.
00210 *
00211 *
00212 D098 8D 7A INPLN BSR CRLF CR/LF TO CONSOLE
00213 D09A 86 3E LDA A #53E GET COMMAND PROMPT >
00214 D09C BD FEAA JSR OUTCH SEND TO CONSOLE
00215 D09F CE E106 LDX #IBUFF GET BUFFER ADDRESS
00216 D0A2 FF E126 STX IBUFF INITIALIZE POINTER
00217 D0A5 7F E129 CLR CTEMP INITIALIZE COUNT
00218 D0A8 BD FD61 INPLI JSR INPCH GET CHAR FROM CONSOLE
00219 D0AB 84 7F AND A #57F MASK OUT PARITY
00220 D0AD 81 40 CMP A #540 CHECK FOR NO PRINT 0
00221 D0AF 26 08 BNE EXCL NO THEN CONT
00222 D0B1 BD FEAA JSR OUTCH ECHO
00223 D0B4 7F E12A CLR PRINT SET NO PRINT
00224 D0B7 20 DF BRA INPLN GO BACK FOR MORE
00225 D0B9 81 21 EXCL CMP A #521 TEST IF SET PRINT 1
00226 D0BB 26 0A BNE EQUAL NO THEN CONT
00227 D0BD BD FEAA JSR OUTCH ECHO
00228 D0C0 86 FF LDA A #5FF GET PRINT CH
00229 D0C2 B7 E12A STA A PRINT SET TO PRINT
00230 D0C5 20 D1 BRA INPLN GO BACK FOR MORE
00231 D0C7 81 3D EQUAL CMP A #53D TEST IF EQUAL SGN
00232 D0C9 26 05 BNE UTST NO THEN CONT
00233 D0CB BD D164 JSR SETUP GO SET ADDRESSES
00234 D0CE 20 C8 BRA INPLN GO BACK FOR MORE
00235 D0D0 81 3C UTST CMP A #53C TEST IF UTILITY <
00236 D0D2 26 03 BNE GO1 NO THEN CONTINUE
00237 D0D4 7E FE32 JMP UTIL GO TO UTILITY
00238 D0D7 81 20 GO1 CMP A #520 TEST IF CONTROL CHAR
00239 D0D9 25 14 BCS INPLC YES, GO PROCESS
00240 D0DB A7 00 STA A 0,X NO, PUT IN BUFFER
00241 D0DD 86 20 LDA A #32 GET BUFFER SIZE
00242 D0DF B1 E129 CMP A CTEMP TEST IF FULL
00243 D0E2 27 C4 BEQ INPLI YES, LOOP
00244 D0E4 A6 00 LDA A 0,X RECALL CHARACTER
00245 D0E6 08 INX INCREMENT POINTER
00246 D0E7 7C E129 INC CTEMP AND INCR COUNT
00248 D0EA BD FEAA INPLE JSR OUTCH ECHO CHARACTER
00249 D0ED 20 B9 BRA INPLI GET NEXT CHAR
00250 D0EF 81 0F INPLC CMP A #50F TEST IF BACKSPACE 10
00251 D0F1 27 0F BEQ INPLB YES, KILL CHAR
00252 D0F3 81 18 CMP A #518 TEST IF 1X
00253 D0F5 27 19 BEQ INPLK YES, KILL LINE
00254 D0F7 81 0D CMP A #50D TEST IF RETURN
00255 D0F9 26 AD BNE INPLI NO, IGNORE CHAR
00256 D0FB B6 E129 LDA A CTEMP GET COUNT

```



```

00257 D0FE B7 E128 STA A IBUFC SAVE IT
00258 D101 39 RTS DONE, RETURN
00259 D102 09 INPLB DEX DECREMENT POINTER
00260 D103 A6 00 LDA A 0,X GET DELETED CHARACTER
00261 D105 7A E129 DEC CTEMP DECREMENT COUNT
00262 D106 2C E0 BGE INPLE IF NOT NEG, GO ECHO
00263 D10A 08 INX IF NEG, UNDO DECR
00264 D10B 7C E129 INC CTEMP IF NEG, INC COUNT
00265 D10E 20 98 BRA INPLI GET NEXT CHAR
00266 D110 7F E128 INPLK CLR IBUFC KILL COUNT TO 0
00267 D113 39 RTS DONE, RETURN
00268 *
00269 *
00270 *
00271 *THIS ROUTINE SENDS A CR/LF TO CONSOLE
00272 *
00273 D114 86 0D CRLF LDA A #S0D GET A CR
00274 D116 BD FEAA JSR OUTCH DISPLAY IT
00275 D119 86 0A LDA A #S0A GET A LF
00276 D11B BD FEAA JSR OUTCH DISPLAY IT
00277 D11E 39 RTS DONE, RETURN
00278 *
00279 *THIS ROUTINE OUPUTS THE CONTENTS OF THE INDEX
00280 *REGISTER AS A FOUR DIGIT HEXADECIMAL NUMBER.
00281 *
00282 D11F 7D E12A OUTHX TST PRINT TEST FOR PRINT
00283 D122 26 01 ENL OUTHX1 YES THEN PRINT
00284 D124 39 RTS NO THEN RETURN
00285 D125 86 20 OUTHX1 LDA A #S20 GET A SPACE
00286 D127 BD FEAA JSR OUTCH SEND TO CONSOLE
00287 D12A FF E104 STX XTEMP SAVE INDEX REG
00288 D12D B6 E104 LDA A XTEMP GET HI BYTE
00289 D130 8D 03 BSP OUTH1 DISPLAY IN HEX
00290 D132 B6 E105 LDA A XTEMP+1 GET OTHER HALF
00291 D135 36 OUTH1 PSH A SAVE LOW ORDER DIG
00292 D136 44 LSR A GET HIGH ORDER DIG
00293 D137 44 LSR A
00294 D138 44 LSR A
00295 D139 44 LSR A
00296 D13A 8D 01 BSP OUTH DISPLAY HEX DIGIT
00297 D13C 32 PUL A GET OTHER DIGIT
00298 D13D 84 0F OUTH AND A #S0F EXTRACT DIGIT
00299 D13F 8B 30 ADD A #S30 ADD ASCII ZONE BITS
00300 D141 81 39 CMP A #S39 TEST IF A-F
00301 D143 23 02 BLS CON IF CLEAR THEN CONT
00302 D145 8B 07 ADD A #S07 YES, ADD BIAS FOR A-F
00303 D147 7E FEAA CON JMP OUTH AND PRINT IT
00304 *
00305 *THIS ROUTINE OBTAINS A CHARACTER FROM THE RAM
00306 *BUFFER AND SETS CARRY=1 IF EXHAUSTED.
00307 *
00308 D14A FF E104 GETCH STX XTEMP SAVE INDEX REG
00309 D14D FE E126 LDX IBUFP GET POINTER
00310 D150 B6 E128 LDA A IBUFC GET COUNT

```

```

00311 D153 82 01          SBC A          #S01      DECREMENT WITH CARRY
00312 D155 25 09          BCS          GETCX      NO MORE CHARACTERS
00313 D157 B7 E128        STA A        IBUFC     REPLACE COUNT
00314 D15A A6 00          LDA A        0,X      GET CHARACTER
00315 D15C 08             INX          INCR POINTER
00316 D15D FF E126        STX          IBUFP     REPLACE POINTER
00317 D160 FE E104 GETCX LDX          XTEMP     RESTORE INDEX REG
00318 D163 39             RTS          DONE, CARRY IF NO CHAR
00319                      *
00320                      *
00321                      *THIS ROUTINE ALLOWS THE SETTING OF THE BEGINNING
00322                      *AND ENDING ADDRESSES IN RAM1 AND RAM2 WITHOUT
00323                      *HAVING TO RETURN TO YOUR MONITOR PROGRAM.
00324                      *
00325                      *
00326 D164 BD FEAA SETUP JSR          OUTCH      GO PRINT =
00327 D167 8D 18          BSR          BYTEA     MSB OF ADDR5
00328 D169 B7 E100        STA A        RAM1      AND STORE
00329 D16C 8D 13          BSR          BYTEA     GET LSB
00330 D16E B7 E101        STA A        RAM1+1    AND STORE
00331 D171 86 20          LDA A        #S20     GET A SPACE
00332 D173 BD FEAA        JSR          OUTCH     AND PRINT IT
00333 D176 8D 09          BSR          BYTEA     INPUT MSB
00334 D178 B7 E102        STA A        RAM2      AND STORE
00335 D17B 8D 04          BSR          BYTEA     GET LSB
00336 D17D B7 E103        STA A        RAM2+1    AND STORE
00337 D180 39             RTS          AND RETURN
00338 D181 8D 0A          BSR          INHEX     PUT IN HEX CH
00339 D183 48             ASL A        PUT IN HIGH HALF
00340 D184 48             ASL A
00341 D185 48             ASL A
00342 D186 48             ASL A
00343 D187 16             TAB
00344 D188 8D 03          BSR          INHEX     SAVE A
00345 D18A 1B             ABA          INPUT OTHER HEX CH
00346 D18B 16             TAB          ADD
00347 D18C 39             RTS          AND RETURN
00348 D18D 8D 18          BSR          INCHA     INPUT HEX CH
00349 D18F 84 7F          AND A        #S7F     MASK OUT PARITY
00350 D191 BD FEAA        JSR          OUTCH     AND PRINT IT
00351 D194 80 30          SUB A        #S30
00352 D196 2B 17          BMI          C1        NOT HEX
00353 D198 81 09          CMP A        #S09
00354 D19A 2F 0A          BLE          INIHG
00355 D19C 81 11          CMP A        #S11     NOT HEX
00356 D19E 2B 0F          BMI          C1        NOT HEX
00357 D1A0 81 16          CMP A        #S16
00358 D1A2 2E 0B          BGT          C1        NOT HEX
00359 D1A4 80 07          SUB A        #S07
00360 D1A6 39             RTS          AND RETURN
00361 D1A7 7E FD61 INCHA JMP          INPCH
00362 D1AA 86 3F          LDA A        #S3F     SEND A ?
00363 D1AC BD FEAA        JSR          OUTCH
00364 D1AF 7E D000 C1     JMP          DRIVE

```

```

00365      *
00366      *
00367      *THESE ROUTINES PERFORM INPUT AND OUTPUT FROM
00368      *AND TO THE CONSOLE, PASSING ONE CHARACTER IN
00369      *THE A ACCUMULATOR. THEY MUST BE CODED TO WORK
00370      *WITH THE PARTICULAR CONSOLE I/O INTERFACE
00371      *ARRANGEMENT OF EACH MICROCOMPUTER.
00372      *
00373      *
00374      FE32  UTIL   EQU          $FE32  START OF UTILITY
00375      FD61  INPCH EQU          $FD61  CONSOLE INPUT ROUTINE
00376      FEAA  OUTCH EQU          $FEAA  CONSOLE OUTPUT ROUTINE
00377      *
00378      *
00379      *RAM WORKING STORAGE
00380      *
00381      *
00382  E106      ORG          XTEMP+2
00383      *
00384  E106 0020  IBUFF  RMB          32      INPUT TEXT BUFFER
00385  E126 0002  IBUFP  RMB          02      INPUT POINTER
00386  E128 0001  IBUFC  RMB          01      INPUT COUNTER
00387  E129 0001  CTEMP  RMB          01      PHONY C REGISTER
00388  E12A FF    PRINT  FCB          $FF    PRINT INDICATOR
00389      *
00390      *
00391      *
00392      *
00393      END

```

```

DRIVE  D000
START  D007
DEOT   D00C
IGET   D011
DCTRL  D01A
GO      D01F
DREAD  D029
DREAL  D02F
DREAX  D038
DWRIT  D043
DWRIL  D056
DLINE  D062
CONT   D068
DINP   D06C
DOUT   D078
DOUTC  D07E
DOUTW  D087
DOUTX  D096
DDATA  E000
DSTAT  E001
RAM1   E100
RAM2   E102
XTEMP  E104
INPLN  D098
INPLI  D0A8

```

EXCL	D0B9
EQUAL	D0C7
UTST	D0D0
GOI	D0D7
INPLE	D0EA
INPLC	D0EF
INPLB	D102
INPLK	D110
CRLF	D114
OUTHX	D11F
OUTHX1	D125
OUTH1	D135
OUTH	D13D
CON	D147
GETCH	D14A
GETCX	D160
SETUP	D164
BYTEA	D181
INHEX	D18D
IN IHG	D1A6
INCHA	D1A7
CI	D1AF
UTIL	FE32
INPCH	FD61
OUTCH	FEAA
IBUFF	E106
IBUFP	E126
IBUFC	E128
CTEMP	E129
PRINT	E12A

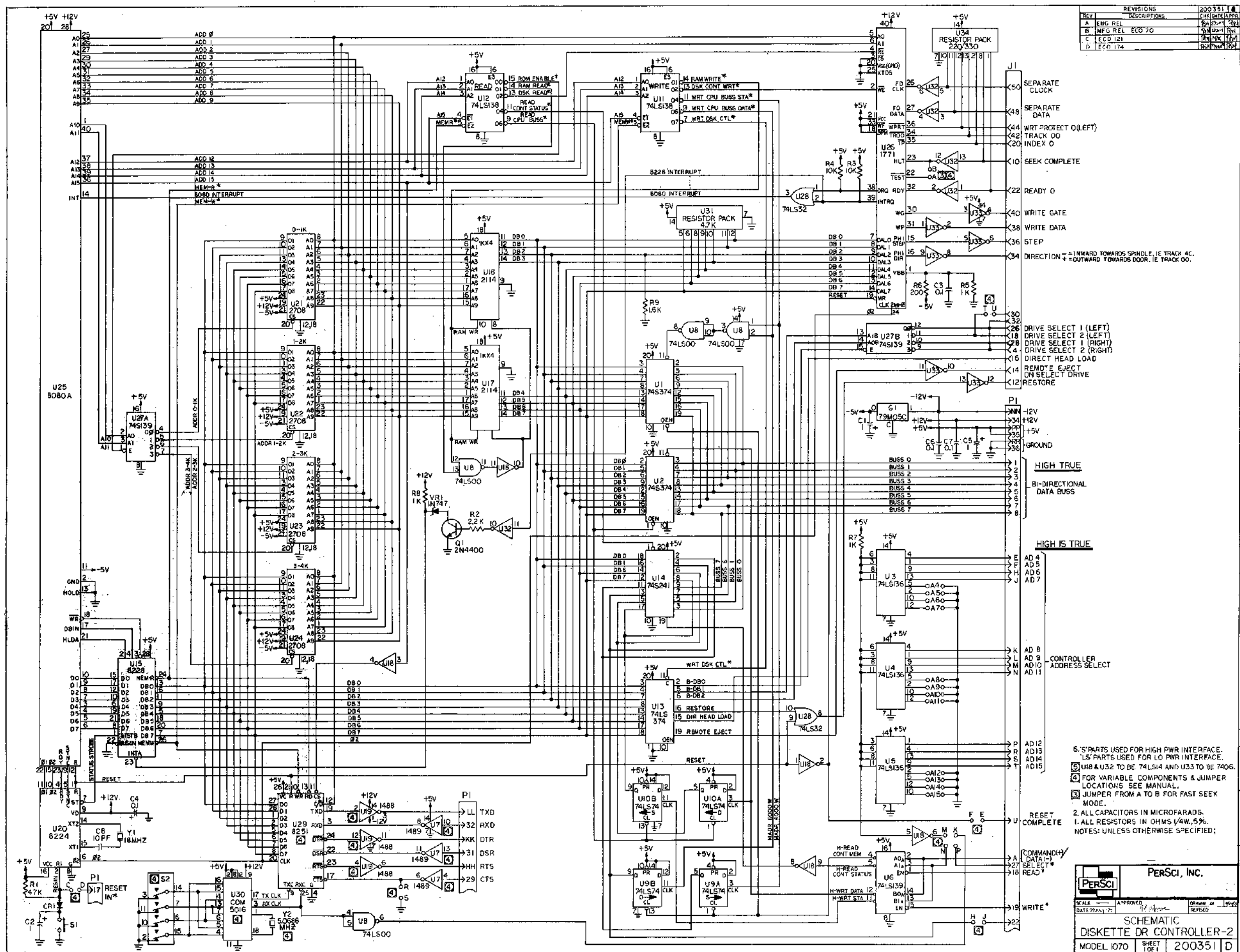
TOTAL ERRORS 00000

PERSCI, INC.

**12210 Nebraska Ave
W Los Angeles
CA 90025**

APRIL, 1978

REVISIONS		
REV	DESCRIPTIONS	CHK DATE APPR
A	EDG REL	8/17/70
B	MFG REL ECO 70	8/24/70
C	ECD 121	10/16/70
D	ECO 174	10/16/70



- 5. S' PARTS USED FOR HIGH PWR INTERFACE.
- LS' PARTS USED FOR LO PWR INTERFACE.
- 6. U18 & U32 TO BE 74LS14 AND U33 TO BE 7406.
- 4. FOR VARIABLE COMPONENTS & JUMPER LOCATIONS SEE MANUAL.
- 3. JUMPER FROM A TO B FOR FAST SEEK MODE.
- 2. ALL CAPACITORS IN MICROFARADS.
- 1. ALL RESISTORS IN OHMS 1/4W, 5%.
- NOTES: UNLESS OTHERWISE SPECIFIED;

PERSCI, INC.

SCALE: _____ APPROVED: _____ DRAWN BY: _____
DATE: 2/24/77 REVISION: _____

SCHEMATIC
DISKETTE DR. CONTROLLER-2

MODEL 1070 SHEET 1 OF 1 200351 D

200351 D