Instruction Manual

for the

perSYMone FDC-1

Floppy-Disk Controller

15 June 1982

SYM-1 Users' Group

P. O. Box 319

Chico, CA 95927

WARRANTY

The FDC-1 is warranted to operate in the manner described in this manual when properly assembled and installed on a SYM-1 Microcomputer and correctly interfaced (including proper option settings) to one or a pair of Shugart SA4ØØ compatible 5 1/4" disk drive(s), or Shugart SA8ØØ compatible 8" disk drive(s).

Any parts or components found to be defective within ninety days from date of delivery will be replaced at no charge, provided only that the defective item is returned, if requested, at the user's expense, and that such defects were not induced by maltreatment by the user.

No other warranties, either express or implied, exist in connection with this product, including its hardware, firmware, software, and/or documentation.

SERVICES

The Users' Group is not prepared at this time to offer assembly, testing, or repair services for the FDC-1. If such help is required by the user, the names/addresses/phone numbers of several SYM users who have offered to provide these services will be supplied.

COPYRIGHT NOTICE

PREFACE

The FDC-1 represents the SYM Users' Group's first venture into producing a hardware product for the SYM-1. Because the FDC-1 was designed, originally, at least, for the SYM-1, and, in keeping with the fashion of using the names of fruits, as originated by the designers of the Apple II, the FDC-1 will henceforth be called, formally, the perSYMone FDC-1.

The original hardware design and the firmware in EPROM were developed by others, and distribution rights were assigned to the Users' Group, along with the tasks of completing the documentation and providing full product and customer support.

Completing the documentation proved to be a bigger task than we had anticipated (isn't it always?), for reasons to be given below, and this manual should, therefore, be considered as preliminary in nature. It is formatted to be edge punched and placed in a loose leaf binder, so that updates and corrections may easily be inserted, as these are made available as part of the continuing product support.

The original material for this manual was provided as rough draft hard copy, together with partial source code on an FDC-1 8" disk in System 65 format. Our evaluation system was configured only for 5 1/4" drives. but Joe Hobart was kind enough to transcribe the 8" disk files to SYM cassette for us and we were able to read the files with RAE, and convert them to RAE format. Joe also included a hex dump of the object code in his EPROM.

Joe's object code differed from that in our system, and neither of the object codes agreed with the source code, with major differences existing. The source code provided was apparently an early version, with "bugs" which would have kept it from working properly, while the object code supplied in our EPROM seemed to be free of problems when used extensively for over a month with a single-drive 5 1/4" system. [This of course does not guarantee freedom from problems with dual-drive systems (although we have begun to test a dual drive system), or with 8" systems (although Joe reports none, either, with his 8" system).]

The rough draft manual described several features implemented in the original source code which were dropped from the final object code to make room for the patches required to eliminate the bugs, thus necessitating far more editing and rewriting than expected. The firmware also differed from the source code in arrangement and usage of internal subroutines, as well as usage of System RAM, thus requiring additional hours of intensive, detailed study to ensure consistency between the manual and the firmware. Hopefully, all discrepancies between the text in this manual and the actually implemented firmware have been eliminated.

The "reverse engineering" involved in editing a multiple-file source code (with one very large gap!) to match the object code turned out to be too time consuming. It was decided, instead, to use Dessaintes' Disassembler on the "production" EPROM to reconstruct a new, single-file, "source" code, using the original only as a guide in assigning meaningful labels. The System 65 Assembler permits only six character labels, which can be quite cryptic; these were lengthened to up to ten characters for greater ease in understanding.

The reconstructed source code, with as many comments inserted as the limited time available permitted, is provided as an appendix to this manual. It is hoped that this source code, in conjunction with the 1791 spec sheet, also provided as an appendix, together with the descriptive material in the body of this manual, will provide the user with a sufficient understanding of the software requirements of the system to fully customize the system to his requirements. The user is encouraged to make such modifications, and to share them with others through SYM-PHYSIS.

Machine readable source code currently resides only on FODS 5 1/4" disks, and could be made available, on special order, on cassette. By early fall of 1982, however, we expect to have a 32K RAM SYM-1 system capable of operating with either an 8" or a 5 1/4" FDC-1 dual-drive system as well as with a dual-drive 5 1/4" FODS system. We can then provide FDC-1 readable source code, on disk (either size), much more fully commented. This system will have on-board RAM at $9ØØØ-$9FFF in place of the FDC-1's on-board EPROM to permit testing and evaluation of improved versions of SYMDOS. We will also relocate the FDC-1's I/O registers into the $ACØØ-$ACFF block to free the entire 2K block at $' ØØ-$F7FF for better uses.

We would like to extend our complements to the designers of the FDC-1, for a job well done, and to the two programmers who generated the original source code for the two separate portions of SYMDOS. Each section was well commented, and, with a copy of the 1791 spec sheet in one hand, the FDC-1 schematic in the other, and the rough draft manual handy for easy reference, the source code was elegantly easy to follow.

CONTENTS

CHAPTER 1

INTRODUCTION

The FDC-1 is a self-contained floppy disk controller board which will interface to the SYM-1, SYM-2, KIM-1, and AIM-65 microcomputer boards. One or two disk drives, either 5 1/4 or 8 inch, can be controlled by the FDC-1 (5 1/4 inch drives will be referred to as 5 inch drives in most sections of this manual). The physical layout of the board allows further expansion from the "E"connector.

The disk drive connections are configured such that a standard ribbon cable (with standard connectors) is all that is necessary to interface to any disk drive(s) using the industry standard "Shugart" bus. Changing between 5 inch and 8 inch drives is accomplished by changing jumpers J1-J3 (see Appendix C) and using the proper drive cable.

With the FDC-1 sitting in front of you with the name "FDC-1" reading from left to right, the following areas can be noted:

The card edge fingers along the left side connect to the "E" connectur on the SYM-1, etc. Along the right edge of the FDC-1 is a continuation of the "E" connector which brings through a subset of the signals available at SYM's "E" connector (see Appendix G).

Along the bottom of the FDC-1 are two disk drive connectors marked P3 for 5 inch drives, and P4 for 8 inch drives.

In the upper right hand corner are two turret pins marked T1 and T2. These pins are used to connect an external power supply to the FDC-1 and also to provide power to the extension of the "E" connector. T1 is the ground pin and T2 is the +5 volt pin.

The software provided in EPROM on the FDC-1 is a combination of commands and routines called SYMDOS. It is designed to provide the functional link between SYM and one or two floppy disk drives.

The software is linked to the SYM monitor (SUPERMON) in a manner which allows user extensions to the routines or user substitutions for varir s routines and default parameters. It permits execution of most built-.n functions either from the hex keypad or from an external terminal.

The software described in the following sections allows formatting disks and saving and loading files from SUPERMON. In addition, the disk directory can be listed to a display terminal from the monitor.

It provides for ENtering, LOading (with Append), and ASsembling (with .CT to disk) with the Resident Assembler/Editor (RAE-1) and loading and saving programs from BASIC (BAS-1).

This is a relatively primitive DOS, and the user may wish to extend its capabilities using the extension facilities provided. It is, however, this very basic structure which allows SYMDOS to operate in as small as 1K-byte system with no terminal, if so desired!

The disk is treated as one long file record split into linear segments end-to-end, and deleted files do not free their sectors for reuse. If a disk is full, but with a large residue of deleted files, the dead space may be recovered by loading active files from the full disk and saving them to an empty disk, one at a time, and then reformatting the old disk to clear its directory.

Alternatively, utility programs can be developed which will copy only active files from one drive to another, or if only a single drive is available, to prompt the user when the old or new disk is to be inserted, and wait for his/her signal before doing the appropriate loads and/or saves.

Another approach for the single drive user is the use of a packing, or compacting, utility, which physically moves the active files down to the space freed by the inactive files, but this is a relatively slow procedure, and should not be used unless back-up copies of all active files are available on other disks, in the event of a catastrophic failure during the packing process!

It is expected that a large number of such utility programs will very quickly be developed by FDC-1 users, and made available to the rest of the FDC-1 user community through SYM-PHYSIS. The availability of such routines is all that is lacking for SYMDOS to become a full-fledged DOS.

CHAPTER 2

INSTALLATION

2.1 Interfacing to the SYM-1, KIM-1 and AIM-65

The FDC-1 connects to the SYM-1 through the "E" connector located at the top right edge of the SYM-1 board. This connector matches the connector marked "P1" on the FDC-1.

Several methods of installing the FDC-1 are available. A pair of dual-22 pin connectors is provided with the board. These may be soldered back-to-back to serve as a male-to-female adapter, or one of them may be soldered "backwards" to the FDC-1 edge fingers.

The FDC-1 now will connect directly to the SYM-1 "E" connector or to similar connectors on the AIM-65 or KIM-1 microcomputers. Devices which may have previously connected to the "E" connector may now be connected to the extension expansion connector, P2, on the FDC-1. Alternatively, you may elect to adopt some sort of motherboard approach.

2.2 Interfacing to the SYM-2

The SYM-2 has the same general connector format as the SYM-1 except that instead of card edge fingers, pads are provided for a right angle female dual 22-pin solder-tail connector. (This connector may be obtained as part of the PEX-2 Port Expansion Kit for the SYM-2.) The FDC-1 simply plugs into this connector, once it is installed in the SYM-2. Further expansion is again available by use of connector P2 on the FDC-1.

2.3 Connecting power

The only voltage which the FDC-1 requires is +5 volts which must be supplied separately to the turret pins on the FDC-1. This voltage is connected by soldering wires to the turret connections. T1 is ground (common to the SYM, FDC-1 and P2) and T2 is +5 volts (FDC-1 and P2 only). A 200 microfarad capacitor is provided across these terminals to provide filtering. Also, decoupling capacitors are spread around the FDC-1 to eliminate switching noise.

Power for the FDC-1 may be brought from the SYM-1 via external wiring but the SYM-2 power supply will not handle the extra load. Also, power required by any additional devices connected to FDC-1 expansion connector P2 must be allowed for in determining power requirements.

The user must supply all necessary voltages for the disk drives externally to the FDC-1.

2.4 Connecting a 5 1/4 inch drive

The FDC-1 connects to the control connector (drive select, step, direction, etc.) of the 5 inch disk drive through a 34 connector flat cable. This cable connects to the FDC-1 using a socket type connector (3M 3414 or equivalent) which plugs into the header strip provided at P3. Pin 2 of this connector is marked on the board. The appropriate jumpers on the FDC-1 must be configured for 5 inch drive systems.

2.5 Connecting an 8 inch drive

The FDC-1 connects to the control connector (drive select, step, direction, etc.) of the 8 inch disk drive through a 50 conductor flat cable. This cable connects to the FDC-1 using a socket type connector (3M 3425 or equivalent) which plugs into the header strip provided at P4. Pin 2 of this connector is also marked on the board. The appropriate jumpers on the FDC-1 must be configured for 8 inch drive systems.

CHAPTER 3

OPERATION

The following sections describe in detail how to use SYMDOS with either SYM-1 or SYM-2. It is assumed that the FDC-1 has been correctly installed according to the instructions given in Chapter 2, Installation.

In all cases, the use of SYMDOS involves two steps: First, a program must be called to link SYMDOS to either SUPERMON, BAS-1, or RAE-1. Second, the desired function is then called, using the new commands provided by SYMDOS.

3.1 Using SYMDOS with SUPERMON

SYMDOS may be used with SUPERMON either through a terminal or directly from the hex keypad. This first section will describe its use with a terminal.

After reset and logging on to the terminal, the user must link SYMDOS to SUPERMON (it is assumed that the user is familiar with the vario , features and commands of SUPERMON as described in the SYM-1 Reference Manual). This is done with the SYM G (GO) command, as follows:

```
            .G 9ØØ6 <cr>
```

Following the execution of this short program, you now have four new SUPERMON commands available to you. These are:

```
            .S9  for formatting a disk
            .S3  for saving to disk
            .L3  for loading from disk
            .L7  for listing a disk directory
```

3.1.1 Formatting a disk

Whenever a new disk is used with SYMDOS, it must be formatted. Formatting may also be performed, after a "temporary" exit from either BAS-1, or RAE-1, if it is discovered, after entering a long program, that no disk space is available on an already formatted disk and that other formatted disks are available!

To format a disk, enter the comand

```
            .S9 n,d,sl <cr>
```

where:

```
            n  is the drive number, Ø or 1
            d  is Ø for single density, 1 for single density
            sl is sector length
```

with:

```
            Ø for  128 bytes/sector
            1 for  256   "     "
            2 for  512   "     "
            3 for 1Ø24   "     "
```

As with other SUPERMON commands, the period and space are provided by the monitor. Either a comma (,) or a dash (-) may be used as the delimiter between parameters.

Some examples of this command are:

```
            .S9 Ø,Ø,Ø  <cr>   Format drive Ø, single density,
                              128 bytes per sector

            .S9 1,1,2  <cr>   Format drive 1, double density,
                              (5" drives only), 512 bytes per sector

            .S9 Ø      <cr>   Format drive Ø, single density,
                              128 bytes per sector.
```

Notice that, in the last command, only a single parameter was entered-- the drive number. In this case, default values are used to format in single density, 128 bytes per sector. In all cases, the drive number must be specified.

In the case of disks used in double sided drives, both sides of the disk will be formatted automatically.

As an aid in selecting sector length for formatting, refer to Table 3.1. Since data are stored on disks in whole sectors only, pick a sector l gth that will allow for minimum wasted sector space wherever possible. You may wish to format several disks, each with different sector lengths, just for this purpose.

For limitations on sector length, see Appendix D, FDC-1 Memory Map.

TABLE 3-1: SECTORS PER TRACK VS SECTOR LENGTH

FORMAT	SECTOR LENGTH			
	128	256	512	1Ø24
5" Single Density	18	1Ø	5	2
5" Double Density	3Ø	18	9	5
8" Single Density	26	15	8	4

3.1.2 Saving to disk

The format for saving data to disk is

```
            .S3 <cr>
            filename,u,sa,ea <cr>
```

where:

```
            filename is a name composed of one to ten characters
                (see Table 3-2 for list of acceptable characters)
            u is drive "unit" number (see Table 3-3)
            sa is starting address of data to save, in hex
            ea is ending address of data, in hex
```

Note that the save command takes two lines to enter the required information and that a <cr> must be entered after each line.

Since only whole sectors are stored and loaded on the disk, data beyond the ending address may be stored. This same "extra" data will be transfered to memory on subsequent loads.

Examples of the save command:

Save data in memory locations 200-FFF onto disk in drive 0. Give file the name "FILE1.HEX":

```
.S3 <cr>
FILE1.HEX,0,200,FFF <cr>
```

Save data in memory locations 1000-1225 onto disk in drive 1. Give file the name "NEWFILE" (note that additional locations will be saved to fill up a complete sector):

```
.S3 <cr>
NEWFILE-1-1000-1225 <cr>
```

Save data in memory locations E00-F0C onto side 2 of disk in drive 1. Give file the name "#1BUFFER":

```
.S3 <cr>
#1BUFFER,2,E000,F0C <cr>
```

Save data in memory locations C00-EFC onto side 1 of disk in drive 0. Do a read verify after writing to check integrity of data saved. Give file the name "%BACKUP.X":

```
.S3 <cr>
%BACKUP.X,4,C00,EFC <cr>
```

TABLE 3-2: ALLOWABLE CHARACTERS IN SYMDOS FILENAMES

A-Z	$	+	.
a-z	%	*	<
0-9	&	/	=
!	(	:	>
#	)	;	?

Do not use the following characters in a filename:

hyphen or dash (-)
comma (,)
quote (")
embedded spaces

Acceptable filenames:

NEW.FILE
$EXEC.MON
TOP>=BOT
1234@(%)

Unacceptable filenames:

A NEW ONE	(contains embedded spaces)
LAST-4	(contains a hyphen)
"THIS"TIME	(contains quotes)
FILES1,2	(contains comma)
LATEST.BASIC	(contains more than 10 characters)

TABLE 3-3: DISK DRIVE UNIT ASSIGNMENTS

UNIT NUMBER	ASSIGNMENT	
Ø	Drive Ø, Side 1	Read or Write
1	Drive 1, Side 1	Read or Write
2	Drive Ø, Side 2	Read or Write
3	Drive 1, Side 2	Read or Write
4	Drive Ø, Side 1	Read Verify after Write
5	Drive 1, Side 1	Read Verify after Write
6	Drive Ø, Side 2	Read Verify after Write
7	Drive 1, Side 2	Read Verify after Write

3.1.3 Loading from disk

There are two formats for loading from disk. The first is the most usual and loads a saved file into the same memory locations from which it was originally saved:

```
            .L3 <cr>
            filename,u <cr>
```

where filename and u are as defined for the save command.

Note that the load command requires two lines, just as did the save command.

The filename must match identically the filename used when saving the file or a "file not found" error will be reported.

The second format for the load command allows loading the saved file into a different location that that from which it was saved (no address adjustments are made--the code is simply loaded intact into the new locations). The format is:

```
            .L3 <cr>
            filename,u,sa <cr>
```

where sa is the new starting address for the data.

Examples of the load command:

Load data from previously saved file on disk in drive Ø:

```
            .L3 <cr>
            FILE1.HEX,Ø <cr>
```

Load data from previously saved file on disk in drive 1, loading data into new memory locations 2ØØØ-2225:

```
            .L3 <cr>
            NEWFILE,1,2ØØØ <cr>
```

3.1.4 Listing a disk directory

As data is stored to the disk, SYMDOS automatically builds a directory with filenames, starting and ending memory locations from where the file was saved, and the track and sector where the file was placed on the disk. The format to list the information from the directory is:

```
            .L7 u <cr>
```

where u is the drive unit number as previously defined in section 3.1.3.

The response to this command will be a listing of all directory entries (all files saved), including those files which have been deleted by later saving another file with the identical filename as an earlier file. Only the latest copy of any file with a duplicate name is normally available to the user (see Appendix E for further information on disk format and directories).

A typical directory entry will look like:

```
            filename -bbbb-eeee-ttss
```

where filename is as previously defined, bbbb is the beginning address in memory, eeee is the ending address in memory, tt is the starting track number on disk, and ss is the starting sector number on disk.

3.1.5 Operation from the SYM hex keypad

The operations in the previous four sections (formatting, saving, loading and listing a directory) can all be performed directly from the SYM hex keypad without the need for an ASCII terminal connected to the SYM. Only the first three operations have real meaning since a terminal is needed to view the directory listing. To simplify the entry of parameters with the keypad, it is suggested that filenames be restricted to combinations of hexidecimal characters (Ø-F).

After RESET and

```
            .G 9ØØ6 <cr>
```

enter the following, using the hex keypad:

To format:

```
            .SHIFT USR1 u-d-sl <cr>
```

where u,d and sl are as previously defined in section 3.1.1.

To save:

```
            .SHIFT ASCII 1 F <cr>
            filename-u-sa-ea <cr>
```

where filename, u, sa, and ea are as defined in section 3.1.2.

To load:

```
            .SHIFT USRØ <cr>
            filename-u-sa <cr>
```

where filename, u, and sa are as defined in section 3.1.3.

If a display is connected, a directory may be listed.

To list the directory:

```
            .SHIFT USR4 u <cr>
```

3.2 Using SYMDOS with BASIC (BAS-1)

SYMDOS is used with BAS-1 in much the same manner as with SUPERMON. After logging on to BAS-1 (with .J Ø or .G CØØØ), you must run a small program that links SYMDOS in with BAS-1:

```
            X=USR(&"9ØØØ",Ø) <cr>
```

Following the execution of this short program, you now have three new SYMDOS functions available:

            Saving a BASIC program
            Loading a BASIC program
            Exiting BASIC to SUPERMON

3.2.1 Saving a BASIC program

After SYMDOS is linked to BASIC, programs you have entered may be saved on disk with the following command:

```
            #S filename,u
```

where filename is a name composed of one to ten characters taken from Table 3-2, and u is the drive unit number taken from Table 3-3. Note that BASIC enters the space after #S and does an automatic <cr> after the unit number.

Examples of the save command:

Save a program called PROG1 to side 1 of a disk in drive Ø:

```
            #S PROG1,Ø
```

Save a program called $ACCOUNTS to side 2 of a disk in drive 1:

```
            #S $ACCOUNTS,3
```

Save a program called BAS.PROG to side 1 of a disk in drive Ø, doing a read verify after write:

```
            #S BAS.PROG,4
```

3.2.2 Loading a BASIC program

The following command is used to load a previously saved BASIC program from disk:

```
            #L filename,u
```

where filename and u are as defined before in Section 3.2.1. As with the #S command, the space after #L and the <cr> after the unit number are entered automatically by BASIC.

Examples of the load command:

Load a program called PROG1 from side 1 of a disk in drive Ø:

```
#L PROG1,Ø
```

Load a program called $ACCOUNTS from side 2 of a disk in drive Ø:

```
#L $ACCOUNTS,2
```

3.2.3 Exiting BASIC to SUPERMON

A useful command, exit to SUPERMON, is provided in SYMDOS. The may be used, for example, to exit to the monitor to list the disk directory. You may return to BASIC by entering a .G <cr>. The format for this command is, simply, #M.

Notice it is not necessary to enter a <cr> after #M. You are now in the monitor and will get the monitor prompt (a dot or period). SYMDOS will remain linked to BASIC after entering .G <cr> so further #L, #S, and #M commands will be properly accepted. SYMDOS is linked to the monitor after a #M is used to enter the monitor.

3.3 Using SYMDOS with the Resident Assembler/Editor (RAE-1)

As with SUPERMON and BASIC, SYMDOS must first be linked with RAE-1 before use. After entering RAE-1 from the monitor with .G BØØØ <cr> enter the following:

```
>RU $9ØØ3 <cr>
```

SYMDOS will now be linked to RAE-1 and you will have the following new SYMDOS functions available to you:

ENtering source files to disk
LOading source files from disk
LOading and Appending source files from disk
ASsemble multiple source files from disk

3.3.1 ENtering source files to disk

The format for the command to ENter source files to disk is

```
>EN filename u <cr>
```

where filename is a name composed of one to ten characters taken from Table 3-2 and u is the drive unit number taken from Table 3-3. As with other RAE-1 commands, only the first two characters of the command need be used, if desired, and parameters are separated by spaces.

Examples of the ENter command:

ENter a file named RAEFILE to side 1 of a disk in drive Ø:

```
>EN RAEFILE Ø <cr>
```

ENter a file named FILE2.RAE to side 2 of a disk in drive Ø:

```
>EN FILE2.RAE 2 <cr>
```

ENter a file named SOURCE.TXT to side 1 of a disk in drive 1 with read verify after write:

```
            >EN SOURCE.TXT 5 <cr>
```

3.3.2 LOading source files from disk

The format to LOad source files previously entered to disk is

```
            >LO filename u <cr>
```

where filename and u are as defined in Section 3.4.1.

The following optional format will LOad a source file from disk and Append the file to a source file already in memory:

```
            >LO filename u A <cr>
```

As with other RAE-1 commands, the command may be shortened to two characters and the option may be shortened to one character, if desired. When LOading source files, especially when using the Append option, make sure sufficient source file space is available to contain the entire file (refer to the SEt command description in the RAE-1 Reference Manual).

Examples of the LOad command:

LOad a source file named RAEFILE from side 1 of a disk in drive Ø:

```
            >LO RAEFILE Ø <cr>
```

LOad a source file named FILE2.RAE from side 2 of a disk in drive Ø, and Append the file to the source file already in memory:

```
            >LO FILE2.RAE 2 A <cr>
```

3.3.3 ASsembling multiple source files from disk

As with cassette tape storage, SYMDOS allows RAE-1 source files to be segmented and placed on disk as a sequence of files. This allows source files to be assembled which would otherwise be too large to fit into the memory space available.

The procedure for assembling multiple source files from disk is very similar to that using cassette tape. The only difference is in the format of the continue to tape (.CT) pseudo-opcode. For continue to disk, the pseudo-opcode should be:

```
            .CT filename2 u
```

where filename2 is the filename of the next segment to be LOaded and u is the drive unit. Each source file except the last one must contain the .CT pseudo-opcode with the filename of the next segment and the drive unit where the file is located. The last file segment simply ends with the unual .EN pseudo-opcode.

To do an assembly, the first file segment is loaded into memory with the LOad command and the ASsemble command is entered. Alternatively, you may enter

```
            >1 .CT filename1 u <cr>
            >AS <cr>
```

where filename1 is the name of the first source segment, and SYMDOS will then LOad the first file segment and continue until pass one is complete.

At the end of pass one, the message "READY FOR PASS 2" will appear. You must then reload the first file segment by either using the LOad command or by entering the .CT filename1 u pseudo-opcode as above. In either case, you next enter

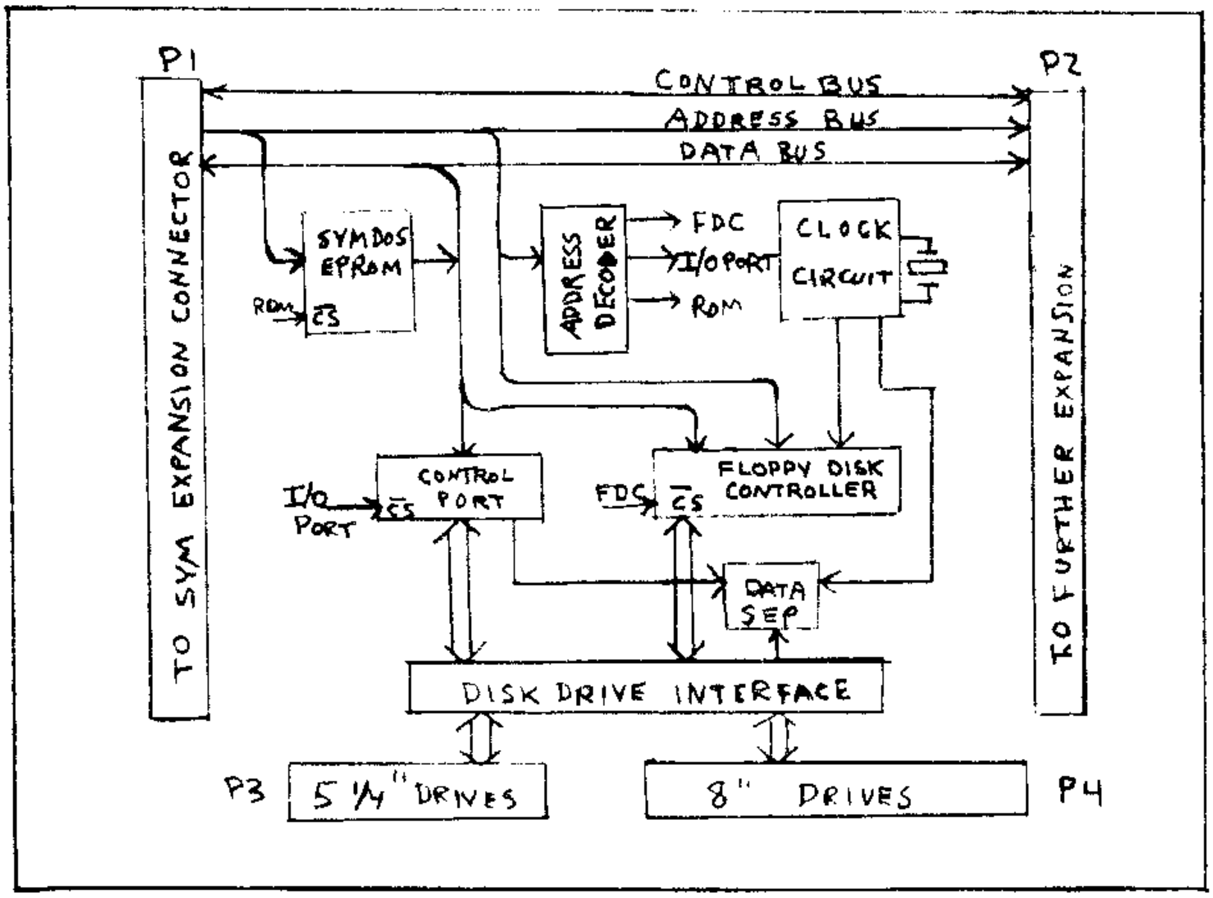
```
            >PA <cr>
```

to complete the assembly.

CHAPTER 4

HARDWARE

Note: This brief discussion is intended only to cover the basic information flow of the FDC-1, and not to explain its operation in any great detail. For more detailed information refer to the FDC-1 Schematic, the SY1791-Ø2 Specifications, and the SYMDOS Source Code Listing provided as supplements to this manual.



The sketch above shows the major functional elements of the FDC-1. The function of each of the elements is briefly described below (the interface connectors P1-P4 are not covered since their functions should be clear).

Address Decoder

This PROM generates the required chip selects for the three subsystems on the FDC-1; the floppy disk controller chip, the output control port, and the SYMDOS EPROM. Since these addresses are determined by PROM, custom memory maps are possible with the FDC-1. A 256 x 4 PROM is connected to A8-A15, and its outputs are used to generate the required chip selects, resulting in a very versatile memory map. Note that the minimum address space for any chip select signal is one whole memory page.

SYMDOS EPROM

This EPROM contains the firmware necessary to save and load between disk and the SYM's SUPERMON, BASIC, and RAE. Both disk I/O and primitive file handling routines are included in this 4K x 8 EPROM. Both address and data lines from the SYM are connected to this EPROM, which is located at hex addresses $9ØØØ to $9FFF by the address map PROM.

Clock Circuit

This circuit generates the necessary clock signals used by the FDC-1. The data separator requires three clocks, while the floppy disk controller chip requires two clocks. These signals are all generated and controlled by this circuit. A 16 MHz crystal is used as the source from which these clocks are derived, and two lines from the control port control these various clocks.

Control Port

Various lines needed to control the subsystems on the FDC-1 are provided by this circuit, which interfaces the SYM's data bus to the necessary latched control lines on the FDC-1. The actual location of this port on the SYM's memory map is determined by the address decode PROM. As shipped from the factory, this is located at hex address $F1XX, using the entire page. This port is write only.

Floppy Disk Controller

A single board floppy disk controller with the simplicity of the FDC-1 is made possible only through the use of LSI technology to incorporate the majority of the disk controller functions into a single chip. This one chip controls most of the functions of the FDC-1 as well as the disk drive(s). Commands such as Read Sector, Format and others are performed by this chip with the firmware/CPU assisting in moving data around. This chip has four registers, and therefore requires two address lines, AØ and A1, and also the SYM's data bus to function correctly.

The controller location is selected by the address decode PROM; this has been assigned at the factory as hex address $FØXX (again, its four registers are repeated every four bytes, to fill the entire page).

Data Separator

This circuit is better called a "Data Synchronizer", and its function is to generate a clock which perfectly "frames" the incoming data stream so that the FDC chip can reliably separate data pulses from clock pulses in the incoming data stream. This circuit must also be able to accomodate various data speeds resulting from the various disk size/density combinations supported. The data separator acts on the incoming read data from the disk, and generates the required "Read Clock" for the FDC.

Disk Drive Interface

This circuit is a hardware interface between the signals from the two disk drives, which are typically "open collector", and the various "TTL" level input, output, or control signals which the FDC-1's control circuit primarily uses. Open collector lines are used for their high noise immunity, and long distance drive capability. This circuit connects to the drives through connectors P3 and P4.

CHAPTER 5

SOFTWARE

The purpose of this chapter is to provide background and supporting information to supplement the highly condensed, but definitive information contained in the source code listing of SYMDOS in Appendix J.

2.1 User Controllable Parameters

URCNEW	New unrecognized command vector. SYMDOS links into URCVEC and allows user to establish his own command link here instead of URCVEC. Defaults to \$8171 (ERMSG) at \$A62E-\$A62F
DOSEXT	Address for user extensions to SYMDOS commands Defaults to \$9559 (RSTXY) at \$A62C-\$A62D
UCMDVC	Unrecognized command vector for the DISKIO primitives May be used to add more "primitives" Defaults to \$9DFD (BADCMD) at \$A6ØF-\$A61Ø BADCMD returns \$36 error code (invalid command)
WRKBUF	SYMDOS needs lots of RAM working space whose location may be defined by the user. It consists of 32 bytes for working variables and 96 bytes needed for RAE-1 continue on disk name function, followed by a RAM space used by the disk as a single sector buffer. WARNING - this buffer will be fully utilized by the disk routine! If the user formats disks for 1Ø24 bytes per sector, the disk routine will fill 1Ø24 locations during a read, beginning at the location calculated as the start of the disk buffer. Defaults to \$ØE8Ø, where workspace goes from \$ØE8Ø-\$ØE9F, RAE buffer from \$ØEAØ-\$ØEFF and the default sector buffer from \$ØFØØ-\$ØFFF. At \$A62A-\$A62B
NOTRACKS	Defaults to 35 (5") or 77 (8") at \$A614
STEPRT	Defaults to \$Ø3 (5") or \$Ø2 (8") at \$A611 See SY1791 Specifications on step rate settings
RETRIES	Defaults to 3 at \$A617
FXBFLG	Fixed block format flag - see section 5.5

2.2 Using the DISKIO primitives to examine or alter a directory entry

The DISKIO parameter control block is located at $A6ØØØ-$A6Ø5. The byte contents are explained also in a separate section, however they are repeated here for use in directory work. To examine a typical directory entry, from the monitor, deposit, with .D A6ØØ <cr>, the following bytes:

```
    A6ØØ      ØØ        ØØ        Ø2        ØØ        ØF        2Ø
              !         !         !         !         !         !

                                                                IFLAGS, turns off
                                                                disk after the read

                                                      Address $ØFØØ, where the data is to
                                                      be read into (disk sector buffer
                                                      default)

                                            Sector 2, first sector of directory

                                  Track Ø, the first track (Ø and 1) of the directory
                                  Most significant bit must be set if to read second
                                  side of a two sided disk

              Unit number (Ø or 1)
```

To read the selected sector from the disk to memory put an Ø4 in the accumulator from the monitor with:

```
.R <cr>
P XXXX, <spacebar>
S   XX, <spacebar>
F   XX, <spacebar>
A   XX,Ø4 <cr>
```

Then .B 98ØØ <cr>. This will read the selected sector. If these are 1[illegible] byte sectors, the data will be read into $ØFØØ-$ØF7F; if 256 byte sectors, into $ØFØØ-$ØFFF, etc. Only complete sectors are read.

To write a selected sector, follow the same instructions, but use an Ø5 instead of Ø4 in the accumulator.

After reading the data, examine the memory area into which the data was read. Use the Verify function or equivalent. Two typical entries would appear as follows (these would be the only entries in a freshly formatted disk):

```
ØFØØ  C1  42  43  44  2Ø  2Ø  2Ø  2Ø
ØFØ8  2Ø  2Ø  Ø2  ØØ  Ø3  FF  Ø2  Ø1
ØF1Ø  41  42  43  44  2Ø  2Ø  2Ø  2Ø
ØF18  2Ø  2Ø  Ø2  ØØ  Ø3  FF  Ø2  Ø5
ØF2Ø  ØØ  ??  ??  ??  ??  ??  ??  ??
ØF28  ??  ??  ??  ??  ??  ??  Ø2  Ø9
```

First look at $ØF1Ø-$ØF19. This represents the ASCII for file name ABCD with 6 trailing blanks to fill our 1Ø characters (the user does not need to enter the blanks). Locations $ØF1A, $ØF1B contain Ø2ØØ, the hex start address for the file. If no relocation is requested, this will be the true start address. Locations $ØF1C, $ØF1D contain Ø3FF, the ending address from which the file was originally stored. The last two bytes of the 16 byte directory entry show the track and sector where the file starts. This file starts at track Ø2 sector Ø5 and uses the following full sectors:

```
            Ø2 Ø5 (128 bytes)
            Ø2 Ø6 (128 bytes)
            Ø2 Ø7 (128 bytes)
            Ø2 Ø8 (128 bytes)
```

for a total of 512 bytes ($Ø2ØØ-$Ø3FF).

Now look at locations $ØFØØ-$ØF1F. These contain a directory entry for file ABCD also. But this one is a down-level "old" copy of ABCD which got "deleted" when you stored the second one. Now, if you say:

```
     .L3 <cr>
     ABCD Ø <cr>
```

SYMDOS will load only the second one because the ASCII characters all match exactly.

This directory resulted from the following sequence:

```
     .S9 Ø <cr>

     .S3 <cr>
     ABCD Ø-2ØØ-3FF <cr>

     .S3 <cr>
     ABCD Ø-2ØØ-3FF <cr>
```

As SYMDOS tried to locate a place to store the second file in the directory, it encountered an old copy of the file name before it found an open entry. Therefore it "smudged" the old copy by setting the MSB of the first letter of the old name. Then it found an open entry (first spot where first character of file name = ØØ) and stored the file starting with the track and sector data it found in that open entry ([illegible] $ØF2Ø-$ØF2F for a typical directory ending entry).

To reaccess a file accidently deleted (preferably to rename the deleted entry) you need only:

1. Read the directory tracks
2. Change the entries to suit your purpose
3. Rewrite that directory entry to the disk

2.3 The Read Routine

a) The name for directory search is loaded into NAMBUF+Ø through +9.

b) The directory search is started. If the name is not found, this is reported. If found, directory entry for start and end of file are moved from directory into NAMBUF+1Ø through +13.

c) If the file is to be relocated, the relocation address replaces the original address and a new file extent is calculated to match the original file extent.

NOTE: BASIC files are not relocatable. They are always loaded into the same space they originally occupied. Only SUPERMON and RAE files can be relocated.

d) Directory data indicating sector and track are moved into NAMBUF+14 and +15 and used there to control the seek to the first sector of the data.

e) Subsequent seeks and reads are done on sequential sectors to the end of a track, continuing on to the first sector of the next sequential track until the entire file has been loaded.

2.4 The Write Routine

a) The directory is searched to see if this file name is already present on the disk. If yes, the old entry is deleted by setting the most significant bit of the first letter of the name on the old entry. This part of the directory is rewritten and the directory search is continued.

b) When search hits the first directory entry having a ØØ as the first byte of the name, it examines the last two bytes of that first ØØ entry to obtain the starting track and sector for the new file storage. This saves calculation time involved in search for last used entry, file start and file extent.

c) File storage is done on an incremental sector by sector basis until the entire file is stored. Maximum file size is 32K if single density is used, 64K if double density is used. The memory area where the data is stored is taken as the source buffer for the drive data transfer. If read/verify is enabled (a readback after every write to assure data integrity on the disk), readback data enters the disk buffer one sector at a time as it is written. No validity check is made, the DISKIO routine will report only whether successful readback occurs.

d) If a readback error occurs part way through the file transfer to the disk, the directory entry for that partial file will not be the name requested originally. Instead it will be entered as "BAD?SEGMNT" with the file start address in that entry the same as the original file start and the file end address reflecting the total number of sectors occupied by this "bad file", up to and including the sector on which the error occurred. As the error is sensed, in addition to marking this disk segment bad, the following directory entry (a ØØ byte in first letter) is marked in its last two bytes with the track number and sector number of the next usable sector. The error is reported and the file store attempt is aborted, returning to the calling routine.

e) If a file store attempt is successful, the directory entry for that attempt is made normally and the directory entry immediately following it is rewritten to show the next available track and sector as explained in d) above. The DOS returns control to the calling program.

2.5 Fixed Block Format Data Storage

The normal method of storing new copies of old files is to delete the old entry in the directory but not to reuse the data sectors (again, a reminder that this is a very basic DOS). Thus, if a user stores multiple updates of a single file name on the disk, the data area will fill up with numerous useless files. The user would then have to run a copy file or duplicate disk utility to "compact" only current files onto a new disk.

A typical example might look like this:

```
      FIRST       SECOND      THIRD      CURRENT
     /FILE1      /FILE1      /FILE1       FILE1
   !..........!..........!..........!..........!..........!
 TRK2        TRK5        TRK8        TRK11       TRK14
 SEC1        SEC2        SEC3        SEC4        SEC5
```

This example shows four copies of the same file (each slightly modified as compared to the previous). Each earlier entry with the same name has been "smudged" or deleted from current access, but the data areas are not reused. Notice that all four use 3 tracks + 1 sector.

If the user were certain that the new copy with the same file name would occupy the same number of (or fewer) sectors than the original, it would fill up the disk far less if the new copy could reuse (over-write) the old copy.

The user can cause this overwrite by storing any nonzero number in location $A624 (FXBFLG) while DOS is active. All files of the same name will then reuse the old file's storage area.

The user accepts the responsibility to see that the new file size occupies the same number or fewer sectors than the original.

>>>NOTE: NO ERROR CHECKING IS PERFORMED FOR THIS OPERATION!<<<

This facility is typically most useful for a language such as FORTH, where 1Ø24 byte blocks are always loaded. Thus a single disk may be formatted as a set of files titled SCREEN1, SCREEN2, SCREEN3, etc. Then if FXBFLG ($A624) is nonzero, the same "screen" is always stored in the same place on the disk each time it is updated.

2.6 Subroutine Description

Following is a partial listing of the major routines and subroutines in SYMDOS, each with a brief summary of its function(s). To make this listing even more useful, the column headed "REGS" should contain an indication of the registers modified by the routine, and of the registers used to pass parameters to and from each routine.

It is hoped that a revised version of this table will become available, including not only the register information, but a more complete listing, in the very near future.

SYMDOS SUBROUTINES

NAME	ADDS	REGS	FUNCTION
BASENTRY	9ØØØ		BASIC entry to SYMDOS
RAEENTRY	9ØØ3		RAE entry to SYMDOS
MONENTRY	9ØØ6		Cold entry to SYMDOS. Sets up monitor link Destroys jump vector table from $A625-$A62F Leaves system RAM unprotected. Initializes new input and output vectors to allow hex keypad to be used with FDC-1. SYMDOS uses entire SCOPE BUFFER ($A6ØØ-$A61F)
LOADIT	9Ø9E		Uses parameters entered with L3 to search the directory and load the file, with or without relocation
USET	9ØBF		Splits a value in A called "UNIT" into drive (Ø,1), side (Ø,1) and read verify after write (Ø,1) Bit 2 - Read verify if 1 Bit 1 - Side select Bit Ø - Drive select
DRINIT	9ØD8		Presets directory parameters to start search at track Ø sector 2
DIRSRCH	9ØE5		Searches all tracks of directory on selected drive and side for exact match to the name in the first ten locations of NAMBUF (WORKSP to WRKSP+9) ØØ in A if it found entry, carry = Ø 8Ø in A if it found end of directory without finding selected file Variables DIRTRK and DIRSEC in WORKSP will contain track and sector in which the selected entry is located Variables DRNTRY and DRNTRY+1 in WORKSP will contain LO, HI address in disk buffer of the first character of the name (saved for later update)
MORTRKS	9ØE8		Subentry of DRSCH. Allows DOS to continue search where it left off if it stopped to delete a file during a new store of the same filename
SECSRCH	9ØF6		Searches directory whether 128, 256, 512, or 1Ø24 bytes per sector -- divides sector length by 16 and searches that number of entries for a name match
DECREMENT	9133		Decrements directory counter for number of

		entries in a sector examined
CMPSUB	914E	Compares two ASCII strings, one pointed to by address in FC, FD; the other in FE, FF. X contains number of characters compared. Flags show results. Z = 1 if compare is OK
ASCLP	9163	Inputs an ASCII string from the currently active input device and places it character by character into the CRT buffer. Carriage return also enters the buffer and terminates the string. Delete ($7F) causes display of a backslash on the terminal and a backspace in the buffer. Backspacing to the left of the first character causes exit with nothing in the ASCII buffer except a $ØD in the first position
NMBLANK	9199	Places blanks into the first ten positions of NAMBUF
DSKPTR	91A5	Sets up FE, FF as LO, HI addresses of the single sector buffer. Set up as WORKSP+128
FIXPTR	91B6	Calls DSKPTR, sets DISKIO IADDR to sector buffer address
ASCPTR	91C4	Sets FA, FB as LO, HI address of CRT buffer
PTNTRY	91CD	Restores last directory entry as the pointer into sector buffer for update
ADJCNT	91D9	Divides directory count by 16 to get how many times 128 must be added to determine next load or save address
BMPNTR	91F5	Uses ADJCNT in adding correct number to P3 which is the address counter during load and save operations
DIFFP2TOP3	925Ø	Compares P3 to P2 to see if load or save complete
SECCNT	91FD	Calculates how many directory entries in a sector based on disk parameters in DISKIO section
INTPRM	92ØE	Moves DATA, TRACK, SECTOR variables from workspaceto DISKIO parameters
GETDTA	921F	Communicates with the user. Accepts an ASCII line, then interprets it. First parameter is name, up to 1Ø characters max. Then at least one blank space. Then up to 3 parameters in normal monitor input format (all hex) separated by hyphens or commas
DOLOAD	9235	Loads the file using previously defined parameters and directory entries. Start

		address in P1. End address in P2. Moving counter from P1 to P2 in P3 bumped as each sector is loaded or saved
STOPTR	925E	Saves directory pointer for update later if file is successfully stored else points to entry where "BAD.SEGMNT" will be entered
MOVEADDRS	926A	Moves address data into P1, P2, P3 for load data. If relocation requested, recalculates and replaces directory address info with relocation data. No change made to directory on disk if relocation is requested
DOSTORE	932F	Directs the file store funtion. Uses parameters entered with S3 command to search directory for open entry, delete old copies of the same name, store the file, then update the directory
MOVPARMS	9360	Moves address P1, P2 into directory
WRTNAM	936D	Moves 10 characters from name buffer into correct directory entry in disk buffer after a successful store
SMUDGE	9381	Sets most significant bit of current directory entry. Has the effect of deleting that name
NXNTRY	938D	Adds 16 to Buffer Pointer (each directory entry is 16 bytes)
READVERIFY	9399	Does the read verify. Write comes out of actual memory area being stored. Read verify data goes only into sector buffer. Original data in memory from file being stored is not hurt if read verify is used
DIRPARMS	93DB	Moves directory track and sector data to DISKIO parameters (saved from earlier DIRSRCH)
BMPDIR	95E7	Points to next entry to compare during search. Bumps sector then track until whole directory examined until first empty entry
DTAPNT	93EC	Ending routine of S3. Updates directory, stores available track and sector in first open directory entry
DISKPARMS	9033	Sets up system defaults if disk present in drive, else returns
FORMAT	944E	Translates from user entered 128, 256, etc., to DISKIO required 0,2 etc. for sector size. Disk formatter: User parameters entered are unit, density, bytes/sector.

		Unit = 0 or 1 (required entry) Density = 0 Single 1 Double (default) Bytes/sector = 128 256 (default) 512 1024
DOWRITE	94E9	Writes to disk using present parameters
DOREAD	9500	Reads from disk using present parameters
BMPDD	9507	Increments sector pointer for data or directory work. If sector exceeds max resets to 1 and increments track pointer. If track exceeds max, sets carry return
SAVEXY	9529	Saves X and Y for later return to BASIC
DELIMITERS	954A	Tests for delimiters, blank, double quote, comma, or hyphen. Z = 1 if delimiter in A
RSTXY	9559	Restores X and Y for BASIC return
NEWINPUT	9563	New input character routine for BASIC only. Tests for #L, #S for DOS load or save. If no #, then returns normally. Else stays in DOS until command is done, gives BASIC a <cr>
BMPDIR	95E7	Increments directory entry pointer to search next entry
SETUPRAE	96BA	Presets parameters for a RAE load or save
SETPARMS	96F2	Move parameters of disk into current parameter block. User may have changed disks betweeen jobs. Reads the current density, sector length, etc., and sets parameters to match

APPENDIX A - SYMDOS COMMANDS

In all the following commands it is assumed that a carriage return is entered after each line in the command except for the "#" type commands in BASIC:

COMMAND TABLE

FUNCTION	SUPERMON	BASIC	RAE
Link to SYMDOS	.G 9ØØ6	X=USR(&"9ØØØ",Ø)	>RUn $9ØØ3
Format a disk	.S9 u,d,sl		
Save to disk	.S3 filename,u,sa,ea	#S filename,u	>ENter filename u
Load from disk	.L3 filename,u	#L filename,u	>LOad filename u
Load and relocate	.L3 filename,u,sa		
Load and append			>LOad filename u
Continue to disk			>n .CT filename u
Exit to monitor		#M	>BReak or C
List directory	.L7 u		

PARAMETER DEFINITIONS

filename is a name composed of one to ten characters taken from Table 3-1

u is the drive unit number taken from Table 3-2

d is Ø for single density, 1 for double density

sl is sector length

Ø for	128	bytes/sector	
1 for	256	"	"
2 for	512	"	"
3 for	1Ø24	"	"

sa is starting address of file in memory

ea is ending address of file in memory

n is a line number

APPENDIX B - SYMDOS ERROR CODES

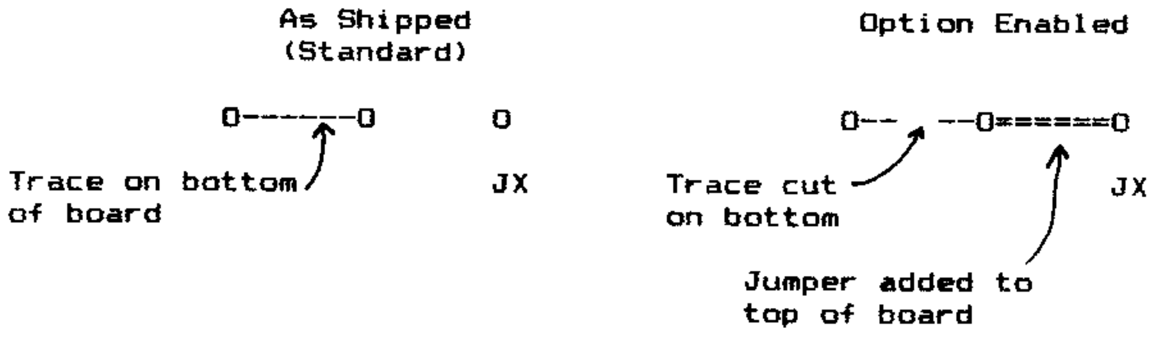
Error Code	Probable Meaning
31	Not available (drive or formatted disk)
32	Invalid drive number
33	Invalid side number
34	Invalid track number
35	Invalid sector number
36	Invalid command code
37	Read/write timeout
38	Seek timeout
39	Record not found
3A	Seek error
3B	Write protect
3C	FDC chip busy
3D	Lost data
3E	CRC error
3F	Not ready
4Ø	Density select error
5Ø	Syntax error
51	Directory full
52	Disk full
53	Name not found
54	Buffer input error

APPENDIX C - JUMPER OPTIONS

The FDC-1 has five jumper options which control three different functions. Jumpers may consist of sockets and jumper pins or of circuit etch and jumper pads. For the socket/jumper pin, simply remove the pin and position it in the desired socket contacts. For the etch/jumper pads, circuit trace must be cut from the center pad to the outer pad, and a jumper wire installed and soldered into the desired pad holes. Any trace which must be cut will always be on the bottom side of the board. Jumper wires may be soldered on the top or bottom of the board as desired.

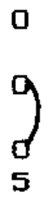
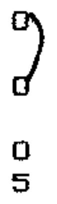
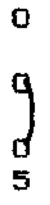
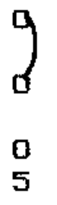
The general philosophy followed with jumpers is that the option is enabled by cutting the trace between the center pad of a jumper and one of the outside pads, and soldering in a jumper from the center pad to the other pad.

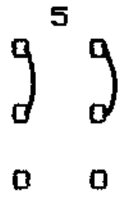
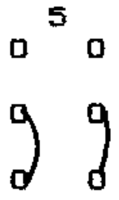
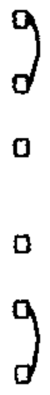
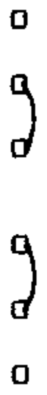
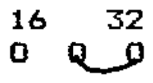
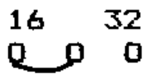
Example:



Editor's note: The above material is intended for a "factory" assembled version of the FDC-1. In the kit version all options are selected by removable jumpers.

The following figure explains the significance of each of the jumpers:

JUMPER	DESCRIPTION	STANDARD	OPTION	COMMENTS
J1	Configures "Ready" input to function correctly			Std is for 5" Opt is for 8"
J2	Connects the proper clock (1 or 2 MHz) for FDC chip			Same as above

J3	Sets up data separator control inputs correctly			Same as above Note two jumpers
J4	Configures pins 2 and 4 of FDC for 1791 or 6591			Std is for 1791 Opt is for 6591 Note two jumpers Also, 6591 requires special firmware
J5	Configures U1 for 2716 or 2732			Std is for 2732 Opt is for 2716

APPENDIX D - FDC-1 MEMORY MAP

SYMDOS is contained in an EPROM located at addresses $9ØØØ-$9FFF. In addition, locations are used in pages ØØ, Ø1, FØ, F1, and in system RAM.

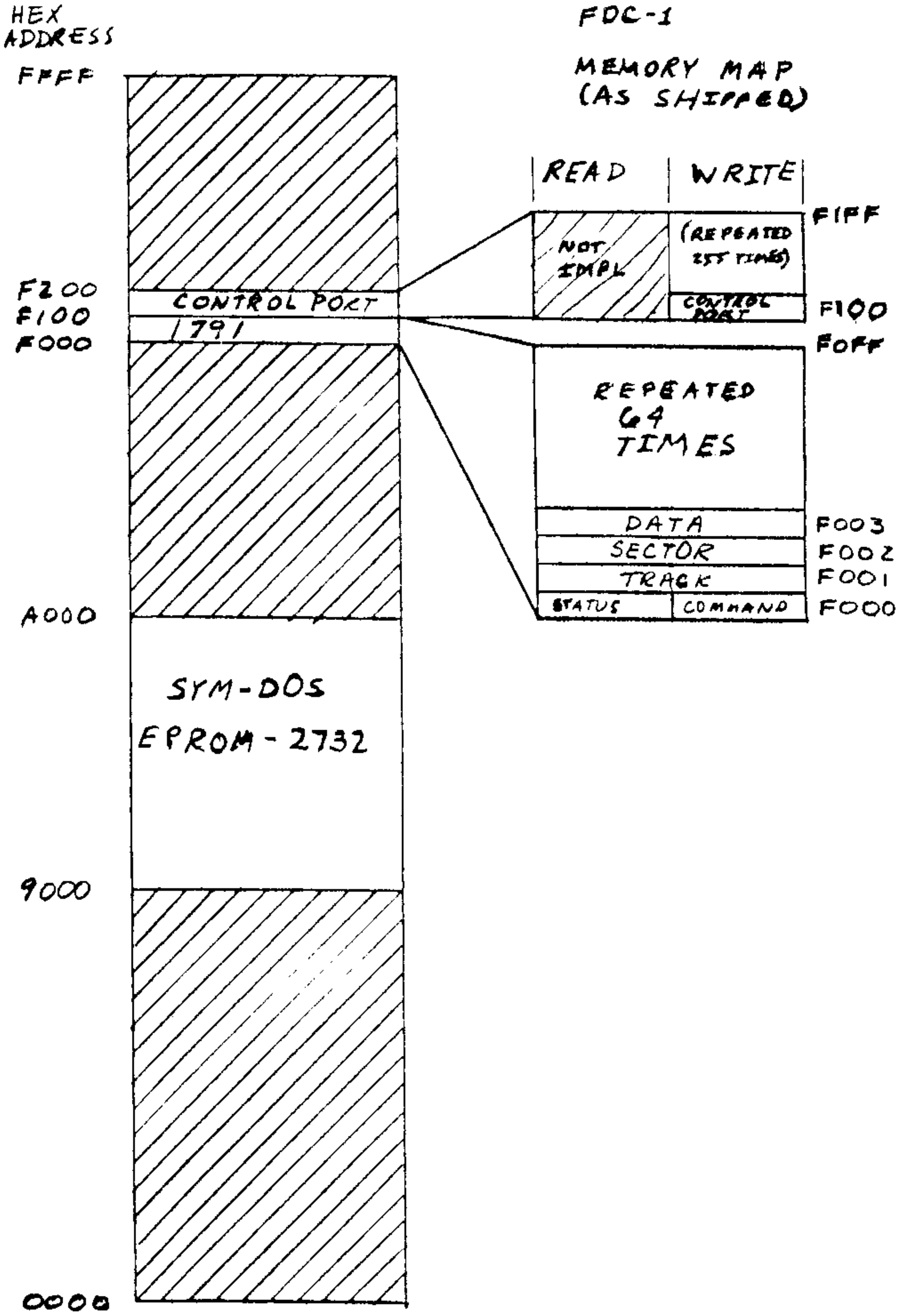
Also, default values are set to use $ØE8Ø-$ØEFF and one or more blocks from $ØFØØ to $12FF for sector buffers. For a 4K SYM-1, sector length must be limited to 256 bytes if the standard defaults are used. For larger sector length, or for different RAM configurations, the sector buffers may be moved by changing $A62A-$A62B in system RAM to hold a 128 byte workspace plus the maximum desired sector length (in multiples of 128 bytes).

Details on pages ØØ and Ø1 usage, and on the use of system RAM, are given in the detailed source code listing provided as a supplement to this manual.

The figure on the following page shows how the four FDC-1 registers and the one control port are echoed throughout the two pages from $FØØØ-$F1FF, due to the partial decoding technique used. This is similar in intent to the I/O decoding technique employed on the SYM itself, in that in a 4K system there is memory space to waste.

For those who resent this waste of valuable memory space and who wish to add even more than 32K RAM to SYM, say an added 4K at $9ØØØ, the following suggestion is offered:

Since full source code is provided, SYMDOS may be reassembled at any 4K location desired, e. g., $FØØØ-$FFFF (suppressing the SYSRAM echo, and putting indirect jumps to SYSRAM at the top of SYMDOS, since there is lots of room there), and the FDC registers relocated to, say, page $A8, or $AC.


HEX ADDRESS
FDC-1
MEMORY MAP (AS SHIPPED)
FFFF
F200
F100
F000
CONTROL PORT
1791
A000
SYM-DOS
EPROM - 2732
9000
0000
READ
WRITE
NOT IMPL
(REPEATED 255 TIMES)
CONTROL PORT
F1FF
F100
F0FF
REPEATED 64 TIMES
DATA
SECTOR
TRACK
STATUS
COMMAND
F003
F002
F001
F000

APPENDIX E - FILE AND DIRECTORY STRUCTURES

Disk Format

```
        Track Ø, sector 1: System parameters set by user (if
                           not user set, defaults are used)

        Track Ø, sectors 2 to max value,
        Track 1, sectors 1 to max value:    Directory Data

        Track 2 to max value, all sectors:  Data Files
```

Directory Format

There are 16 bytes per directory entry, with:

```
                 8 entries/sector if single density
                16 entries/sector if double density
```

Each entry consists of:

1Ø bytes:	Filename
2 bytes:	Hex start address of file
2 bytes:	Hex end address of file
1 byte :	Track number where file starts (Max = $7F; MSB = Ø if side 1, 1 if side 2)
1 byte :	Sector number where file starts (Max value depends on drive type, $12 if 5 inch, $1A if 8 inch)

File Structure

All files are stored in sequential sectors on the disk. When the end of a track is reached, the disk advances to sector 1 of the next higher track number.

If read-verify is enabled and an error occurs during the readback, the disk directory will reflect an entry titled "BAD?SEGMNT" occupying an area from the beginning of the file up to and including the sector on which the eroor occurred.

APPENDIX F - CHIP FUNCTIONS

CHIP	TYPE	FUNCTION
1	2732	EPROM-Firmware
2	7438	Clock Switching Data Sep Clock Misc Gating
3	74LS197	Master Clock Divider
4	SY1791-Ø2	Floppy Disk Controller
5	74LS368	SYM Signal Buffering
6	74LSØØ	Misc Gating
7	74SØ4	Master Oscillator Misc Inverting
8	7416	Open Collector Interface
9	82S129	Address Map Decoder ROM
1Ø	74LS74	Read Data Pulse Shaping
11	74LS163	Data Separator
12	83Ø3	Data Bus Buffer
13	74LS174	Control Port Latch
14	7438	Disk Control Outputs

APPENDIX G - CONNECTOR PINOUTS

P1 - "E" CONNECTOR

1	NC	A	ABØ
2	NC	B	AB1
3	ϕ1	C	AB2
4	IRQ	D	AB3
5	RO	E	AB4
6	NC	F	AB5
7	RES	H	AB6
8	DB7	J	AB7
9	DB6	K	AB8
1Ø	DB5	L	AB9
11	DB4	M	AB1Ø
12	DB3	N	AB11
13	DB2	P	AB12
14	DB1	R	AB13
15	DBØ	S	AB14
16	NC	T	AB15
17	NC	U	ϕ2
18	NC	V	R/W
19	NC	W	R/W*
2Ø	NC	X	NC
21	NC	Y	ϕ2*
22	GND	Z	RAM R/W

Pins 1-22 are on the component side, from right to left when facing the edge, opposite to SYM.

Pins A-Z are on the solder side.

Connector type: Dual 22 card edge on Ø.156" centers.

P2 - EXTENSION "E" CONNECTOR

The Extension Expansion Connector pinout is identical with that of the Expansion Connector except for pin 21, which is connected to the +5 V supplied by turret pin T2, and the numbers read from left to right when facing the edge, same as on SYM.

DISK DRIVE CABLE WIRING LIST

P4 DUAL 25 (Ø.1") 8 INCH DRIVES		P3 DUAL 17 (Ø.1") 5 1/4 INCH DRIVES	
2	No Connection		
4	No Connection		
6	No Connection		
8	No Connection		
1Ø	No Connection		
12	No Connection		
14	Side Select (Note 1)		
16	No Connection		
18	No Connection	2	No Connection
2Ø	Index	4	No Connection
22	Ready	6	No Connection
24	No Connection (Note 2)	8	Index/Sector
26	Drive Select 1	1Ø	Drive Select 1
28	Drive Select 2	12	Drive Select 2
3Ø	No Connection	14	No Connection
32	No Connection	16	Motor On
34	Direction	18	Direction
36	Step	2Ø	Step
38	Write Data	22	Write Data
4Ø	Write Gate	24	Write Gate
42	Track ØØ	26	Track ØØ
44	Write Protect	28	Write Protect
46	Read Data	3Ø	Read Data
48	No Connection	32	Side Select (Note 1)
5Ø	No Connection	34	No Connection

All odd numbered pins are at logic zero (i. e., ground).

Note 1: Not provided by single sided drives, but the FDC-1 defaults to side 1 if this signal is not available.

Note 2: Hard-sectored disk drives, e.g., the SA8Ø1, use this line for Sector. These drives may be used (with soft sectored disks only) with the FDC-1.

APPENDIX H - PHYSICAL SPECIFICATIONS

Power Requirements:

Volts:	+5.0V DC +/- 0.1V
Amps:	0.4A typical, 0.5A max
Watts:	2.5W max

Weight:

With edge connector:	5 ounces (141.75 grams) max

Temperature Range:

Operating:	0 to +70 degrees Celsius
Storage:	-30 to +80 degrees Celsius

Physical Size - P.C. Board:

Width:	4.2" +/- 0.04"
Length:	5.7" +/- 0.05" (6.2" with connector)
Height:	0.5" max

Electrical Characteristics:

Host System Clock Speed:	1 MHz max
Disk Transfer Rates:	31250, 15625 bytes/second
Disk Sizes Supported:	8", 5 1/4"; 1 or 2 sided
Recording Densities/Encoding:	Single/FM Double/MFM (5" only)

APPENDIX I - CONTROL PORT DEFINITIONS

D7 D6 D5 D4 D3 D2 D1 DØ

DØ	When low enables "DS1"
D1	When low enables "DS2"
D2	When low enables "Motor On"
D3	The side select line to the disk drive follows this line
D4	When low enables the 1791 HLT input, and also the 1791 IRQ to the SYM
D7	When low enables the single density mode. High enables double density, and should be done in 5 inch mode only

Note 1: Control port is write only

Note 2: On reset, all control port outputs are set high