A MICROPROCESSOR-BASED FLOPPY DISK OPERATING SYSTEM

BY

ANDREW NELSON FU

B.S., University of Illinois at Urbana-Champaign, 1977

THESIS

Submitted in partial fulfillment of the requirements
for the degree of Master of Science in Electrical Engineering
in the Graduate College of the
University of Illinois at Urbana-Champaign, 1979

Urbana, Illinois
ACKNOWLEDGEMENT

I would like to express my deepest appreciation to my thesis advisors, Professors Ricardo B. Uribe and Edward S. Davidson, for their support, patience and encouragement in this project.

My very special thanks to Professor Uribe, Bill Munson and Gary DeGregorio, for their many contributions in the construction and debugging of much of the hardware equipment involved in this project.

I am also grateful to many friends, involved in the Advanced Digital Systems Lab, who have given me excellent ideas and help for the completion of this project.
# TABLE OF CONTENTS

1. INTRODUCTION ................................................................................. 1
   1.1 Design Goals ........................................................................... 1
   1.2 Hardware Environment .......................................................... 2
      1.2.1 Processor .......................................................................... 2
      1.2.2 Floppy Disk System ......................................................... 2
      1.2.3 System Configuration ....................................................... 3
   1.3 Software Support ..................................................................... 4
      1.3.1 Resident DEBUG monitor .................................................. 4
      1.3.2 Cross software support .................................................... 4

2. THE SYSTEM CONSOLE COMMANDS ............................................... 6
   2.1 Editing and program control features ....................................... 7
   2.2 Console command syntax ........................................................ 7
   2.3 Command operands .................................................................. 8
   2.4 File reference ......................................................................... 9
   2.5 Description of DOS console commands ..................................... 11
      2.5.1 CHDR command ............................................................... 11
      2.5.2 DELETE command .......................................................... 12
      2.5.3 DIR command ................................................................ 13
      2.5.4 FORMAT command ........................................................ 14
      2.5.5 INIT command ................................................................ 14
      2.5.6 LOAD command ............................................................. 15
      2.5.7 MONITOR command ....................................................... 16
      2.5.8 MOUNT command .......................................................... 16
      2.5.9 RENAME command ......................................................... 17
      2.5.10 RESET command .......................................................... 17
      2.5.11 SAVE command ............................................................. 17
      2.5.12 SDATE command ........................................................... 18
      2.5.13 STIME command ........................................................... 19
      2.5.14 SYSGEN command ......................................................... 19
      2.5.15 USER command ............................................................. 20
   2.6 Error messages ....................................................................... 20
   2.7 Examples .............................................................................. 21

3. DISK I/O INTERFACE TO USER PROGRAM ..................................... 25
   3.1 Functional description of DOS ................................................. 25
   3.2 Disk I/O facilities .................................................................. 27
      3.2.1 File organization ............................................................. 27
      3.2.2 File Control Block .......................................................... 33
   3.3 Disk access primitives ............................................................ 33
      3.3.1 Select function ................................................................. 34
      3.3.2 OPEN function ............................................................... 35
      3.3.3 CLOSE function ............................................................. 36
      3.3.4 CREATE function .......................................................... 37
3.3.5 DELETE function ........................................ 38
3.3.6 GETR function ........................................ 38
3.3.7 PUTR function ........................................ 39
3.3.8 GETARG function ...................................... 40
3.3.9 SETDMA function ..................................... 41
3.3.10 INTDR function ...................................... 42
3.3.11 INTALOC function ................................... 42
3.3.12 WBOOT function ..................................... 43
3.3.13 ENDOS function ..................................... 43
3.4 Program Example ........................................ 43

4. IMPLEMENTATION OF DISK OPERATING SYSTEM .......... 47
4.1. System tables and directories ......................... 47
4.1.1 File Control Block .................................. 48
4.1.2 Disk Allocation Table ................................ 50
4.1.3 Directory Entry Table ................................ 51
4.1.4 Disk Record Buffer .................................. 52
4.1.5 Disk Buffer ........................................... 52
4.2 Implementation of Disk I/O routines .................... 53
4.2.1 OPEN routine ....................................... 53
4.2.2 CLOSE routine ....................................... 54
4.2.3 CREATE routine ..................................... 54
4.2.4 DELETE routine ...................................... 55
4.2.5 GETR routine ........................................ 56
4.2.6 PUTR routine ........................................ 56

5. FLOPPY DISK INTERFACE HARDWARE DESCRIPTION .......... 58
5.1 FD-402 Floppy Disk Controller ......................... 58
5.2 IMP-16 Interface Timing and Logic .................... 60
5.2.1 Command Input Timing and Logic .................... 60
5.2.2 Data Transfer Timing and Logic .................... 64
5.2.3 Status Timing and Logic ............................ 67
5.3 Floppy Disk Interface Specifications .................. 70
5.3.1 Cable conventions ................................... 72

6. CONCLUSION AND FURTHER WORK .......................... 74

REFERENCES .................................................. 76

APPENDIX A -- FDI hardware schematics ........................ 77
LIST OF TABLES

<table>
<thead>
<tr>
<th>TABLE</th>
<th>Description</th>
<th>Page</th>
</tr>
</thead>
<tbody>
<tr>
<td>TABLE 2.1</td>
<td>INIT format numbers</td>
<td>15</td>
</tr>
<tr>
<td>TABLE 2.2</td>
<td>Error codes and meaning</td>
<td>21</td>
</tr>
<tr>
<td>TABLE 2.3</td>
<td>Summary of built-in commands</td>
<td>24</td>
</tr>
<tr>
<td>TABLE 3.1</td>
<td>Disk Primitive Routines</td>
<td>45</td>
</tr>
<tr>
<td>TABLE 5.1</td>
<td>Floppy Disk controller functions</td>
<td>61</td>
</tr>
<tr>
<td>TABLE 5.2</td>
<td>Pin designation for FDI and FDC interface</td>
<td>73</td>
</tr>
</tbody>
</table>
LIST OF FIGURES

<table>
<thead>
<tr>
<th>FIGURE</th>
<th>Description</th>
<th>Page</th>
</tr>
</thead>
<tbody>
<tr>
<td>3.1a</td>
<td>Track and Sector Format</td>
<td>29</td>
</tr>
<tr>
<td>3.1b</td>
<td>Logical Organization of diskette</td>
<td>30</td>
</tr>
<tr>
<td>3.1c</td>
<td>Directory Track Format</td>
<td>31</td>
</tr>
<tr>
<td>3.1d</td>
<td>Data Track Format</td>
<td>32</td>
</tr>
<tr>
<td>4.1</td>
<td>File Control Block format</td>
<td>49</td>
</tr>
<tr>
<td>5.1</td>
<td>FD-402 I/O ports</td>
<td>59</td>
</tr>
<tr>
<td>5.2</td>
<td>Command Input Timing</td>
<td>63</td>
</tr>
<tr>
<td>5.3</td>
<td>Data Read/Write Timing</td>
<td>65</td>
</tr>
<tr>
<td>5.4</td>
<td>Status Timing</td>
<td>68</td>
</tr>
<tr>
<td>5.5</td>
<td>Status Words</td>
<td>69</td>
</tr>
<tr>
<td>5.6</td>
<td>IMP-16 Read Timing</td>
<td>71</td>
</tr>
<tr>
<td>5.7</td>
<td>IMP-16 Write Timing</td>
<td>71</td>
</tr>
</tbody>
</table>
This paper is a user's manual for the ADS-16 Disk Operating System (DOS). Chapter 2 describes the user commands for operating DOS. Chapter 3 presents information on how to interface to DOS. Chapter 4 describes the implementation of DOS system routines and is a guide to system expansion. Chapter 5 is the hardware reference manual for the Floppy Disk Interface (FDI). Chapter 6 gives a brief evaluation and possible future work relating to DOS.

1.1 Design goals

The design goal for the Disk Operating System is to provide easy access for a user to utilize the floppy disk as a data storage device. This is accomplished by implementing a named file structure on the disk. Also, the program is written in such a way that it can be called as a subroutine by the user.
Ease of expansion is the next goal. DOS has a modular structure and expansion of the system can be implemented by adding subroutines to the system program. User definable commands can easily be added by appending programs as files on the disk, thereby using only disk space for expansion. Such addition of nonresident routines allows maximum memory utilization for the user.

1.2 Hardware environment

The hardware environment of DOS consists of the Microprocessor System and the Floppy Disk System.

1.2.1 Processor

The National Semiconductor IMP-16 processor was selected for use as the host processor for all software development. The microprocessor system was developed by the Advanced Digital Systems Laboratory as the ADS-16 Microcomputer Development System. The ADS-16 contains the 16 bit IMP-16 processor with an extended instruction set which includes multiply, divide, block input/output and bit manipulation instructions.

1.2.2 Floppy Disk System
The Scientific Microsystems FD-402 Floppy Disk was selected because it was available for use on the ADS-16 system. The FD-402 system has two Shugart-802 double density 8-inch floppy disk drives and an intelligent disk controller which is implemented using a bit-slice processor. The controller can handle both single and dual density multiple formats (i.e., 128, 256, 512 or 1024 bytes per sector with either density). Nine macro commands provide all the required disk functions, minimizing IMP-16 system software overhead. Interfacing to the IMP-16 is done through three parallel data ports implemented as memory mapped I/O devices.

1.2.3 System configuration

The ADS-16 system is configured with 56K (57,344) 16 bit words of Random Access Memory (RAM) and 4K (4,096) words of Erasable Read Only Memory (EPROM). There are two serial I/O interfaces with programmable baud rates and interrupt capabilities. The floppy disk interface occupies 16 locations in the memory mapped I/O space and is switch selectable to occupy any address block in the memory space. Other serial I/O devices available include the console device (HP 2645 terminal); a COMDATA 203 modem for connection to other computers and a Microdata paper tape reader/punch.
1.3 Software support

1.3.1 Resident DEBUG Monitor

The DEBUG monitor is a debugging tool for use with the IMP-16 microprocessor. It allows the user to set break points in a program, alter and examine both memory and register contents, search for bit patterns in memory and transfer control to program residing in memory.

A binary loader is also available, allowing the user to load programs from paper tape readers, HP magnetic tape cartridges and from the CDC Cyber timesharing system through a modem telephone link.

A special disk bootstrap loader is provided to allow cold starting the Disk Operating System from the first two tracks of any DOS floppy diskette. Bootstrapping is also possible from tracks other than track 0 and track 1, but care must be taken to ensure that the correct data format resides on those tracks.

1.3.2 Cross software support

Available for IMP-16 software development is a cross-assembler accessible through the CDC Cyber timesharing system.

The cross-assembler generates National's relocatable load modules which can be relocated to any address in memory during the process of downloading. Object codes can also be generated on paper tapes which can be read into the ADS-16 memory through the Microdata paper tape reader.
The listing of an assembled program can be printed on line printers available on the CDC Cyber system.
CHAPTER 2

The System Console Commands

The Disk Operating System can be loaded using the bootstrap loader available through the DEBUG monitor. After the insertion of a floppy disk into the system drive (drive 0), the bootstrap loader can be invoked by the "X" command in the DEBUG monitor. The bootstrap loader loads DOS into memory from the first two tracks of the disk and then transfers control to DOS now in memory.

Upon entering DOS, a signon message is printed on the console device indicating the version number of DOS loaded. Then DOS prompts with the character "%", signaling that system is up and ready for command input.

A command name is entered first, then its arguments and a carriage return for termination. DOS then processes the command with the given arguments and returns to the user. If an error occurs during processing, an error message will be printed on the console device. The user may enter a corrected command after the prompt appears.
2.1 Editing and program control features

When a user enters commands to DOS through the console device, several control characters are available for line editing and program control. Control characters are entered by depressing the indicating character key while holding down the keyboard CONTROL key.

The CONTROL-H character echoes a backspace and deletes the immediately preceding character from the input. Typing CONTROL-H at the beginning of a line has no effect.

The CONTROL-U character deletes the entire line from the input buffer. DOS will ignore the line and will prompt again for the next line.

The CONTROL-C character is activated when the operating system is in the execution state. DOS will abort the current disk operation immediately but termination is done only after a valid disk operation is totally completed. This ensures data integrity on the diskettes. Execution cannot be resumed after termination by a CONTROL-C.

The CONTROL-S character allows user to suspend output to the terminal device. DOS will wait for the user to enter a CONTROL-Q character to continue output again.

2.2 Console command syntax
All DOS console commands have the following general form:

\[<\text{keyword}> \ <\text{cr}>\]
\[\text{or}\]
\[<\text{keyword}> \ <\text{sp}> \ <\text{operand1}> \ <\text{cr}>\]
\[\text{or}\]
\[<\text{keyword}> \ <\text{sp}> \ <\text{operand1}> \ <\text{sp}> \ <\text{operand2}> \ldots \ <\text{cr}>\]

where \(<\text{keyword}>\) is the command name, \(<\text{operand1}>\) and \(<\text{operand2}>\) are command operands, \(<\text{sp}>\) is a "space" character used as a delimiter for all operands, and \(<\text{cr}>\) is a carriage return.

All characters in the keyword are used to identify the command. The command name consists of any alphanumeric characters except the first character which must be alphabetic.

2.3 Command operands

The command operands used may in general consist of any alphabetic or numeric characters. The operand can either be a filename or a numeric constant.

Numeric constants consist of a string of digits or alphabetic characters (letters A, B, C, D, E, F are used in hexadecimal constants). The first character of a hexadecimal constat must be a digit. For example, "FFFB" in base 16 must be written as "0FFFB" in order to differentiate it from a filename.

A filename consists of a string of up to 16 alphanumeric characters plus an optional file extension of 3 alphanumeric characters. Filenames longer than 16 characters are truncated to the first 16 characters. Similarly, file extensions longer than 3 characters are truncated to the first 3 characters. Filename and file extensions are separated by a
period "." as shown below.

ffff.xxx

where ffff is the primary filename and xxx is the file extension. Special characters may be used in representing ambiguous filenames. See Section 2.4 for details in file reference.

2.4 File reference

A file reference identifies a particular file or group of files on a particular disk logged-in on the system. These file references are classified as "unambiguous" and "ambiguous". An unambiguous file reference (ufn) uniquely identifies a single file, while an ambiguous file reference (afn) may be satisfied by a group of different files.

An ambiguous file reference is used for directory search and pattern searching. An afn is similar to an ufn except that the symbol "?" may be interspersed throughout the filenames and extensions. Any filename satisfies a match if it matches exactly in all character positions except where the symbol "?" appears. Therefore the ambiguous file reference

T????.?OM

is satisfied by the unambiguous filenames

TEST.COM
TEXT.TOM
T34T.XOM
Also a "*" can be used to represent an arbitrary length string of "?". The number of "?" represented by the "*" symbol equals the number of "?" needed to fill up the remaining filename or extension to their actual length. Thus, "ABCD.B*" is an abbreviation for any filename with a primary filename beginning with "ABCD" provided it has an extension beginning with B, regardless of the length of the primary filename and its extension. It also covers ambiguous filenames as "ABCD???????????????.B??". A useful example for the symbol "*" is the following:

DIR Test.*

which will list the file information of all files with primary filename "Test".

The ambiguous reference "*.8" can be used to match all the filenames on the disk whether the filenames contain any extensions or not.

Files that are not on the logged-in disk but reside on a different disk drive can be referenced by appending the drive number to the end of the filename. The symbol "/" is used to separate the filename and drive number. Thus if the currently logged-in disk is 0, then the file reference

XYZ.COM/1

will reference the file XYZ.COM on drive 1.

The console command "CHDR" allows the user to change the default logged-in drive. See section 2.5 on the console commands.
2.5 Description of DOS console Commands

The file references described in the previous section are used to fully specify the structure of the Disk Operating System commands. In the following sections, the commands and their use are discussed in alphabetical order.

All console commands are translated by the Console Command Processor (CCP) to uppercase, while filenames are upper and lowercase. The commands described are CHDR, DELETE, DIR, FORMAT, INIT, MONITOR, MOUNT, RENAME, RESET, SAVE, SDATE, STIME, SYSGEN and USER.

2.5.1 CHDR command

The CHDR (Change Drive) command allows the user to change the default working drive. DOS first bootstraps up from drive 0 and all subsequent default file references are made to that drive. File references which access files on drives other than the default drive will not affect the setting of the default drive.

The syntax of the CHDR command is

CHDR <drive number> <cr>

where <drive number> is a decimal constant specifying the new default drive to change to and <cr> is a carriage return.

Example of use of CHDR:

CHDR 1  
change default drive to 1

DELETE xyz.lst/0  
delete the file xyz.lst on drive 0

DELETE xyz.lst  
delete file xyz.lst on default drive
Current hardware implementation supports 2 drives, 0 and 1.

2.5.2 DELETE command

The DELETE command removes files from the currently logged-in disks. The files which are deleted are those which satisfy the ambiguous file reference.

The syntax of the DELETE command is

```
DELETE afn <cr>
```

where afn is an ambiguous filename and <cr> is a carriage return.

Following examples illustrate the use of DELETE:

```
DELETE xyz.obj
the file named xyz.obj on the default drive is removed from its directory.

DELETE xyz.*
all the files on the default drive with primary file name xyz are removed from their directories. Note: the file "xyz" with no extensions will not be deleted (This is an exception in the use of "*" so as to protect disk files).

DELETE *.obj
all the files on the default drive with file extension "obj" are removed from their directories.
```
DELETE xyz.obj/1 the file xyz.obj on drive 1 is removed from its directory regardless of what the current default drive is.

2.5.3 DIR command

The DIR (DIRectory) command causes file information of all files satisfying an ambiguous file reference to be listed at the console output device. The information displayed consists of filenames; file length in number of records; date and time of last modification and names of users who last modified the files.

The syntax of the DIR command is

DIR afn <cr>

where afn is an ambiguous file name.

Examples of some valid DIR commands are:

- DIR *.obj list all files with "obj" extensions on default drive.
- DIR x??.*/1 list all files x??.* on drive 1.
- DIR *.* list all files on default drive.
- DIR same as DIR *.*.

NOTE: if instead of a filename, a decimal constant is entered, CCP will use it as a drive number and all files on that drive will be listed. Therefore,

DIR <drive number> is the same as
2.5.4 FORMAT command

The FORMAT command is used to format a diskette with IBM 3740 disk format. The format to be used (i.e. length of sector and density) is specified by the INIT command (Section 2.2.5). The default format is 512 bytes/sector and double density.

The syntax of the FORMAT command is

```
FORMAT <drive number> <cr>
```

where <drive number> specifies which disk is to be formatted. The user is further required to type in "Y" or "N" (YES or NO) to confirm formatting the disk. Confirmation is desired since FORMAT will destroy all data on the disk. If anything other than "Y" is entered DOS will abort the FORMAT command before the disk is modified, else it will proceed to format the diskette.

2.5.5 INIT command

The INIT (INITialize) command allows the user to specify the format in which diskettes are to be read or written. The user should use this command only to read or write non-standard diskettes. The format used by DOS is 512 bytes/sector and double density.
The syntax of the INIT command is

\[ \text{INIT <format number> <cr>} \]

where \(<\text{format number}\) specifies what format to use in subsequent read and write operations. Table 2.1 shows the values and meanings of the format numbers.

Table 2.1

INIT format numbers

<table>
<thead>
<tr>
<th>format number</th>
<th>density</th>
<th>bytes/sector</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>single</td>
<td>128</td>
</tr>
<tr>
<td>1</td>
<td>single</td>
<td>256</td>
</tr>
<tr>
<td>2</td>
<td>single</td>
<td>512</td>
</tr>
<tr>
<td>3</td>
<td>single</td>
<td>1024</td>
</tr>
<tr>
<td>4</td>
<td>double</td>
<td>128</td>
</tr>
<tr>
<td>5</td>
<td>double</td>
<td>256</td>
</tr>
<tr>
<td>6</td>
<td>double</td>
<td>512</td>
</tr>
<tr>
<td>7</td>
<td>double</td>
<td>1024</td>
</tr>
</tbody>
</table>

2.5.6 LOAD command

The LOAD command is used to load portions of memory with data from a disk file. The data in the disk file must first be saved using the SAVE command (Section 2.5.11).

The syntax of the LOAD command is

\[ \text{LOAD ufn <start addr> <end addr> <cr>} \]

where ufn is an unambiguous filename (the file from which the data is to be loaded), <start addr> and <end addr> specify the starting address and
ending address in memory where data is to be loaded. If these addresses are not specified, the first two words of the file are used as the addresses. If they are specified the first two words of the file are ignored.

2.5.7 MONITOR command

The MONITOR command is used to exit from DOS and return to the ADS-16 DEBUG MONITOR.

The syntax of the MONITOR command is

```
MONITOR <cr>
```

2.5.8 MOUNT command

The MOUNT command is used the each time a diskette is mounted on a drive. DOS will read in the disk from the directory track and when a valid file entry is found, DOS will print file information on the console device. At the same time, the Disk Allocation Table (see Section 4.1.2) in memory is also updated using the record pointers found in the File Control Block of that file.

The complete directory is searched until there are no more valid files left.

The syntax of the MOUNT command is

```
MOUNT <drive number> <cr>
```

where <drive number> specifies the drive on which the mounting is to be done.
2.5.9 RENAME command

The RENAME command allows the user to change filenames of files on a logged-in disk. The file references given must be unambiguous filenames.

The syntax of the RENAME command is

RENAME ufn1 ufn2 <cr>

where the file named ufn1 is changed to ufn2. Example:

RENAME xyz.obj abc.obj

the file xyz.obj now becomes abc.obj.

2.5.10 RESET command

This command is used to reset the floppy disk controller. The command should be used only when controller errors occur. The RESET command should be issued if a diskette is accidentaly removed from a drive during a disk operation.

The syntax of the RESET command is

RESET <cr>

2.5.11 SAVE command

The SAVE command allows the user to place memory locations onto a diskette file and name it with a filename. The file SAVE-ed contains the exact core image of the locations specified.
The syntax of the SAVE command is

```
SAVE ufn <start addr> <end addr> <cr>
```

where ufn is an unambiguous filename, <start addr> specifies the starting address in memory and <end addr> specifies the ending address in memory. Both <start addr> and <end addr> are hexadecimal constants. These addresses are also SAVE-ed as the first two words in the file.

Example of the SAVE command

```
SAVE mem.com 0F800 0F900
```

will save memory locations F800 to F900 in a file called mem.com.

2.5.12 SDATE command

The SDATE (Set DATE) command is used to set the current date when DOS first comes up from bootstrap. Files modified subsequently will have this date as their "last modified" date.

The syntax of the SDATE command is

```
SDATE <mo> <dd> <yy> <cr>
```

where <mo> specifies the month, <dd> specifies the day and <yy> specifies the last 2 digits of the current year.

In the current implementation, it is the user's responsibility to update the date appropriately since there is not yet a routine for this purpose which uses the real time clock. Example,

```
SDATE 5 11 79
```

will set the current date to "MAY 11, 1979".
2.5.13 STIME command

The STIME (Set TIME) command allows the user to initialize the real time clock. The setting of the clock is done immediately after the user presses the carriage return key. Files modifies subsequently use the value of the real time clock to indicate the "last modified" time.

The syntax of the STIME command is

```
STIME <hh> <mm> <ss> <cr>
```

where <hh> specifies the hour of the day in 24-hour format, <mm> specifies the minute and <ss> specifies the seconds.

In the current implementation, a software real time clock is used. The current prototype temporarily requires all user routines to enable interrupt in order for the real time clock to function correctly. A hardware real time clock is under construction. Example of STIME:

```
STIME 20 34 44
```

sets the real time clock to "8:34:44 P.M.".

2.5.14 SYSGEN command

The SYSGEN (SYStem GENeration) command is used to place DOS on the first two tracks of a diskette for use in bootstrap. An empty directory is also created on the directory track. The disk should be formatted, using the FORMAT command (Section 2.5.4), to 512 bytes/sector double density before DOS is placed on the diskette.
The syntax of the SYSGEN command is

SYSGEN <drive number> <cr>

where <drive number> specifies the drive on which SYSGEN is to be done. If <drive number> is not entered, the default drive is assumed. The user is further required to type in "Y" or "N" (YES or NO) to confirm SYSGEN since SYSGEN will destroy data on the disk. If anything other than "Y" is entered DOS will abort the SYSGEN command else it will proceed to perform SYSGEN on the diskette.

2.5.15 USER command

The USER command allows the user to identify files, with the user's name. All files subsequently modified will use the user's name in the FUSER field (see Section 4.1.1) of the File Control Block (Section 4.1.1) of that file.

The syntax of the USER command is

USER <name> <cr>

where <name> is an ascii character string of length ten or less. If <name> is longer than ten characters, only the first ten will be used.

When DOS first comes up, the default user name is "SYSTEM". The USER command may be used any number of times during a logged-in session.

2.6 ERROR messages
The Console Command Processor (CCP) and the Basic Disk Operating System (BDOS) both detect errors in a command entered by the user. When an error is detected, the command is ignored and an error message is returned. The user may then enter a corrected command.

CCP detects input errors (i.e. illegal characters or syntax) while BDOS detects execution errors. If the command is entered through the User Function Processor (UFP) through a subroutine call, an error was found if the return code in register 0 has a negative value. Table 2.2 gives the values and meanings of possible errors.

Table 2.2
ERROR codes and meaning

<table>
<thead>
<tr>
<th>UFP return code</th>
<th>error message</th>
</tr>
</thead>
<tbody>
<tr>
<td>-1</td>
<td>Disk full</td>
</tr>
<tr>
<td>-2</td>
<td>File already exists</td>
</tr>
<tr>
<td>-3</td>
<td>End of disk buffer</td>
</tr>
<tr>
<td>-4</td>
<td>Invalid allocation vector</td>
</tr>
<tr>
<td>-5</td>
<td>Illegal filename</td>
</tr>
<tr>
<td>-6</td>
<td>File not found</td>
</tr>
<tr>
<td>-7</td>
<td>Illegal argument</td>
</tr>
<tr>
<td>-8</td>
<td>Input filename not found</td>
</tr>
<tr>
<td>-9</td>
<td>Illegal drive number</td>
</tr>
</tbody>
</table>

2.7 Examples
Following is an example of the use of the Console Commands. Comments are inside parentheses. DOS is relocatable and in the example below its entry point is 9000.

ADS-16 MONITOR V3.2
>I   (reset Floppy Disk Controller)
>X   (execute bootstrap loader)
>09000 (begin DOS)

ADS-16 DISK FILE SYSTEM    VERSION 2.0
% SDATE 5 11 79  (set date to MAY 11 1979)
% STIME 9 00 00  (set time to 9:00:00 am)
% USER DEMO  (user name is "DEMO")
% MOUNT 0

<table>
<thead>
<tr>
<th>DISKETTE - DEMO</th>
<th>ADS16 DOS VERSION 2.0</th>
</tr>
</thead>
<tbody>
<tr>
<td>FILENAME</td>
<td>LENGTH</td>
</tr>
<tr>
<td>-----------------</td>
<td>-----------</td>
</tr>
<tr>
<td>JUNK.COM</td>
<td>06</td>
</tr>
</tbody>
</table>

% DELETE JUNK.COM  (delete file on disk)
% DELETE JUNK.COM  (attempt to delete it again)
OPERATION ERROR # = 06  (ERROR! file already deleted)

% SAVE CORE.COM 9000 9FFF  (save memory image on disk)
% CHDR 1  (change default drive to 1)
% LOAD CORE.COM/0 1000 1FFF  (load file into 1000 to 1FFF)
% DIR 0  (display directory information on drive 0)

<table>
<thead>
<tr>
<th>DISKETTE - DEMO</th>
<th>ADS16 DOS VERSION 2.0</th>
</tr>
</thead>
<tbody>
<tr>
<td>FILENAME</td>
<td>LENGTH</td>
</tr>
<tr>
<td>-----------------</td>
<td>-----------</td>
</tr>
<tr>
<td>CORE.COM</td>
<td>06</td>
</tr>
</tbody>
</table>
A summary of the built in commands is given in Table 2.3.
Table 2.3
Summary of built in commands

<table>
<thead>
<tr>
<th>Command</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>CHDR</td>
<td>&lt;drive number&gt; &lt;cr&gt;</td>
</tr>
<tr>
<td>DELETE</td>
<td>&lt;ufn&gt; &lt;cr&gt;</td>
</tr>
<tr>
<td>DIR</td>
<td>&lt;afn&gt; &lt;cr&gt;</td>
</tr>
<tr>
<td>FORMAT</td>
<td>&lt;drive number&gt; &lt;cr&gt;</td>
</tr>
<tr>
<td>INIT</td>
<td>&lt;format number&gt; &lt;cr&gt;</td>
</tr>
<tr>
<td>LOAD</td>
<td>&lt;ufn&gt; &lt;start addr&gt; &lt;end addr&gt; &lt;cr&gt;</td>
</tr>
<tr>
<td>MONITOR</td>
<td>&lt;cr&gt;</td>
</tr>
<tr>
<td>MOUNT</td>
<td>&lt;drive number&gt; &lt;cr&gt;</td>
</tr>
<tr>
<td>RENAME</td>
<td>&lt;ufn1&gt; &lt;ufn2&gt; &lt;cr&gt;</td>
</tr>
<tr>
<td>RESET</td>
<td>&lt;cr&gt;</td>
</tr>
<tr>
<td>SAVE</td>
<td>&lt;ufn&gt; &lt;start addr&gt; &lt;end addr&gt; &lt;cr&gt;</td>
</tr>
<tr>
<td>SDATE</td>
<td>&lt;mo&gt; &lt;dd&gt; &lt;yy&gt; &lt;cr&gt;</td>
</tr>
<tr>
<td>STIME</td>
<td>&lt;hh&gt; &lt;mm&gt; &lt;ss&gt; &lt;cr&gt;</td>
</tr>
<tr>
<td>SYSGEN</td>
<td>&lt;drive number&gt; &lt;cr&gt;</td>
</tr>
<tr>
<td>USER</td>
<td>&lt;user name&gt; &lt;cr&gt;</td>
</tr>
</tbody>
</table>

where:

- <cr> carriage return
- <drive number> disk drive number, 0 or 1
- <ufn> unambiguous filename
- <afn> ambiguous filename
- <format number> format number, 0 to 7
- <start addr> starting address in base 16
- <end addr> ending address in base 16
- <mo> month, 1 to 12
- <dd> day, 1 to 31
- <yy> year, 0 to 99
- <hh> hour, 0 to 23
- <mm> minute, 0 to 59
- <ss> second, 0 to 59
- <user name> user name, 10 characters or less
CHAPTER 3

DISK I/O INTERFACE TO USER PROGRAM

The subroutines for disk primitives provide the user with all the necessary input/output functions associated with the disk. All assembly language interfaces are done through the calling of a single system entry point with the user providing a function identifier and the function arguments.

The following sections describe the function of DOS and calling conventions for user to DOS.

3.1 Functional description of Disk Operating System

DOS is logically divided into four distinct modules:

- BDOS - Basic Disk Operating System
- BIOS - Basic disk and device I/O System
- CCP - Console Command Processor
- UFP - User Function Processor
BDOS provides all the functions of disk file management. A directory of the independent files is kept on each disk. BDOS also implements disk allocation with random or sequential access and attempts to minimize head movement across the disk. BDOS allows the user to perform the following disk primitive operations:

- **OPEN** - open a file for further access.
- **CLOSE** - close a file after processing.
- **CREATE** - create a file.
- **DELETE** - delete an existing file.
- **GETR** - read a record from a file.
- **PUTR** - write a record into a file.

BIOS provides the primitive operations for interfacing to the floppy disk controller and standard peripherals. All system input/output is performed through BIOS. Any system I/O changes can be adapted by reflecting those changes in BIOS. Other modules of DOS need not be modified.

The CCP provides the symbolic interface between the user and the remainder of DOS. CCP receives commands from the console device which it interprets. Execution is then transferred to either BDOS or BIOS. The standard console commands are described in Chapter 2.

UFP allows the user to call DOS as an assembly language subroutine. The function number and all necessary arguments are passed to UFP through CPU registers. UFP decodes the function number and sets up the argument before transferring control to the corresponding routines in
BDOS or BIOS. Error codes are returned to the user through CPU registers.

3.2 Disk I/O facilities

The BDOS section provides user access to files stored on the logged-in diskettes. The following sections describe the overall file organization and file access mechanisms.

3.2.1 File organization

Files are implemented with a named file structure on each diskette, allowing any particular file to contain any number of records up to the full capacity of a diskette. Each diskette is logically distinct, with a complete operating system, disk directory and file data area.

The disk file names are in two parts, the primary filename which can be from one to sixteen alphanumeric characters and the file extension which consists of zero to three characters. The "file extension" names the generic category of a particular file, while the "primary file name" distinguishes a particular file within the category.

Some examples of "file extensions" are listed below. Although some extensions are established for specific use, they are in general arbitrary.

SRC ---- source file, mainly ascii characters
RLM ---- relocatable load module.
ABS ---- absolute binary file.
Details of these file types, except COM, are described in the INTERACT Operating System User's Manual [2].

Figure 3.1a shows the physical track and sector addressing of a diskette. The logical organization of a diskette is shown in Figure 3.1b. All 16 sectors on track 0 and 1 are used to store the bootstrap operating system. Sectors 1 to 15 in track 2 contain the disk directory. Each of these sectors in track 2 contains 8 File Control Blocks (see Section 3.2.2) for a total of 120 File Control Blocks. Sector 16 of track 2 contains the Directory Entry Table (see Section 4.1.3). Figure 3.1c shows the format of the directory track.

Tracks 3 to 77 are the data tracks containing the actual files on the disk. Each track is divided into three sector groups of 5 sectors each. Sector group 1 contains sectors 1, 4, 7, 10, 13; sector group 2 contains sectors 2, 5, 8, 11, 14 and sector group 3 contains sectors 3, 6, 9, 12, 15. Sectors of different sector groups are interleaved to provide time for software execution between reading or writing successive sectors. Sector 16 is not used on the data tracks. Figure 3.1d shows the organization of a data track.

Files may be made up of multiple records and each record is a sector group on a track. The records on a disk are numbered with record numbers from 0 to 221. The record on track T, sector group N has a record number \( T-3 + \frac{74}{N-1} \) where T ranges from 3 to 76 and N from 1 to 3. Note that T and N can be uniquely decoded from the 1 byte record number.
Figure 3.1a  Track and Sector Format
**Sectors**

<table>
<thead>
<tr>
<th>track</th>
<th>0</th>
<th>1</th>
<th>2</th>
<th>3</th>
<th>4</th>
<th>5</th>
<th>6</th>
<th>7</th>
<th>8</th>
<th>9</th>
<th>10</th>
<th>11</th>
<th>12</th>
<th>13</th>
<th>14</th>
<th>15</th>
<th>16</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td><strong>Bootstrap program</strong></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td><strong>Bootstrap program (continued)</strong></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td><strong>Disk directory</strong></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td><strong>File data area</strong></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

**Figure 3.1b Logical Organization of diskette**
<table>
<thead>
<tr>
<th>sector 1</th>
<th>FCB</th>
<th>FCB</th>
<th>FCB</th>
<th>FCB</th>
<th>FCB</th>
<th>FCB</th>
<th>FCB</th>
<th>FCB</th>
</tr>
</thead>
<tbody>
<tr>
<td>sector 2</td>
<td>FCB</td>
<td>FCB</td>
<td>FCB</td>
<td>FCB</td>
<td>FCB</td>
<td>FCB</td>
<td>FCB</td>
<td>FCB</td>
</tr>
<tr>
<td>sector 3</td>
<td>FCB</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>sector 15</td>
<td>FCB</td>
<td>FCB</td>
<td>FCB</td>
<td>FCB</td>
<td>FCB</td>
<td>FCB</td>
<td>FCB</td>
<td>FCB</td>
</tr>
<tr>
<td>sector 16</td>
<td>Directory Entry Table</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

Figure 3.1c Directory Track Format
<table>
<thead>
<tr>
<th>Sector</th>
<th>Record Type</th>
<th>Part</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>R</td>
<td>1</td>
</tr>
<tr>
<td>2</td>
<td>R+74</td>
<td>1</td>
</tr>
<tr>
<td>3</td>
<td>R+148</td>
<td>1</td>
</tr>
<tr>
<td>4</td>
<td>R</td>
<td>2</td>
</tr>
<tr>
<td>5</td>
<td>R+74</td>
<td>2</td>
</tr>
<tr>
<td>6</td>
<td>R+148</td>
<td>2</td>
</tr>
<tr>
<td>7</td>
<td>R</td>
<td>3</td>
</tr>
<tr>
<td>8</td>
<td>R+74</td>
<td>3</td>
</tr>
<tr>
<td>9</td>
<td>R+148</td>
<td>3</td>
</tr>
<tr>
<td>10</td>
<td>R</td>
<td>4</td>
</tr>
<tr>
<td>11</td>
<td>R+74</td>
<td>4</td>
</tr>
<tr>
<td>12</td>
<td>R+148</td>
<td>4</td>
</tr>
<tr>
<td>13</td>
<td>R</td>
<td>5</td>
</tr>
<tr>
<td>14</td>
<td>R+74</td>
<td>5</td>
</tr>
<tr>
<td>15</td>
<td>R+148</td>
<td>5</td>
</tr>
<tr>
<td>16</td>
<td>Not used</td>
<td></td>
</tr>
</tbody>
</table>

Figure 3.1d Data Track Format
3.2.2 File Control Block

Each file accessible by DOS has a corresponding File Control Block (FCB) which contains the name, allocation and other file information for that particular file. The FCB of each file is a 64-byte area on the directory track.

When a file is opened for read or write, its corresponding FCB is read from the directory track into memory. As file operations proceed, the FCB in memory is updated, and upon termination of the file operations, the FCB is written back onto the directory track when the file is CLOSE-d. This access method makes DOS file organization highly reliable, since its integrity can only be disrupted during the update of a single directory entry in the unlikely event of a hardware failure.

Each FCB describes up to 60K (61,440) bytes of a particular file. If a file is to contain more than 60K bytes, FCB extensions are automatically set up. Up to 16 FCB extensions are possible for a particular file. These can describe a file larger than the capacity of a single diskette.

3.3 Disk access primitives

The User Function Processor (UFP) allows the user to perform several kinds of disk I/O operations. Disk file operations are normally performed on the selected disk using file information in a particular FCB. These operations are described in the following sections.
In the following description, references are made to the assembler mnemonics used by the IMP-16 assembler. The LI mnemonic indicates a load accumulator with immediate data instruction. The JSR mnemonic indicates a jump to subroutine instruction. AC0, AC1, AC2 and AC3 are the four CPU registers. For more information about the IMP-16 assembler and instruction set see [1].

All calls are done through a JSR to the entry point of DOS (EDOS is the entry point). Register 3 contains the function number indicating what function DOS is to perform. Other registers contain arguments necessary for the call.

A summary of the disk primitives is presented at the end of this chapter.

3.3.1 SELECT function

The SELECT function is used to select the default disk drive. All subsequent disk operations will assume the selected drive as the default drive. The only time when the selected default drive is not used is when the file reference contains a drive number.

The calling sequence for SELECT is

```
LD ACO,<drive number>
LI AC3, 1
JSR EDOS
```

where ACO is loaded with the drive number (0 or 1) and AC3 is set to 1, the function number for SELECT. EDOS is the entry point of DOS.
SELECT saves the contents of registers 1, 2 and 3, and puts a return code in register 0. The return codes and their meanings are listed below.

<table>
<thead>
<tr>
<th>Code</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>SELECT successful, selected drive 0.</td>
</tr>
<tr>
<td>1</td>
<td>SELECT successful, selected drive 1.</td>
</tr>
<tr>
<td>-5</td>
<td>SELECT failed, illegal drive number.</td>
</tr>
</tbody>
</table>

3.3.2 OPEN function

OPEN establishes a connection between the user program and a particular file on the disk. OPEN should be the first system call preceding any read or write operations.

Before calling the OPEN function, the user must set up an FCB area for the file. The user's responsibility is to fill in the FN (filename) and FNX (file-extension) field of the FCB. The address of the FCB in memory is then passed to the OPEN function. DOS will then perform the necessary disk operations so as to read in the matching FCB from the disk directory to the FCB area in memory.

The calling sequence for OPEN is

```
LD    AC2,<FCB address>
LI    AC3, 2
JSR   EDOS
```

where AC2 is loaded with the FCB starting address and AC3 is set to 2, the function number for OPEN. EDOS is the entry point to DOS.
OPEN saves the contents of registers 1, 2 and 3, and puts a return code in ACO. The values and meanings of the return code are listed below.

<table>
<thead>
<tr>
<th>Value</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>OPEN successful.</td>
</tr>
<tr>
<td>-1</td>
<td>OPEN failed -- file already open.</td>
</tr>
<tr>
<td>-5</td>
<td>OPEN failed -- illegal filename.</td>
</tr>
<tr>
<td>-6</td>
<td>OPEN failed -- file not found.</td>
</tr>
</tbody>
</table>

3.3.3 CLOSE function

The CLOSE function performs the reverse action of the OPEN function by disconnecting the user from a file and updates the FCB on the disk. CLOSE should be the last operation on a file after all the input/output operations are completed. DOS will update the FCB in the disk directory using the current updated FCB in memory.

The calling sequence for CLOSE is:

```
LD AC2, <FCB address>
LI AC3, 3
JSR EDOS
```

where register 2 is loaded with the starting address of the FCB and register 3 is set to 3, the function number for CLOSE. EDOS is the entry point to DOS.

CLOSE saves the contents of registers 1, 2 and 3, and puts a return code in register 0. The values and meanings of the return code are listed below.

<table>
<thead>
<tr>
<th>Value</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>CLOSE successful.</td>
</tr>
<tr>
<td>-1</td>
<td>CLOSE failed -- file not open.</td>
</tr>
</tbody>
</table>
### 3.3.4 CREATE function

The CREATE function allows the user to create a file on the disk with a specific number of records assigned to it. A new directory entry is also created for that file containing all the allocation information.

The calling sequence for CREATE is

```
LD ACO,<number of records to allocate>
LD AC2,<FCB address>
LI AC3, 4
JSR EDOS
```

where register 0 is loaded with the number of records to be allocated, register 2 is the FCB starting address and register 3 is set to 4, the function number for CREATE. EDOS is the entry point for DOS.

The user is responsible for setting up the filename and extension field in the FCB. DOS will fill in the Allocation Map (see Section 4.1.1) of the FCB corresponding to the records allocated on the actual disk. The FCB is then written back onto the directory track with the updated allocation information.

CREATE saves the contents of register 1, 2 and 3, and puts a return code in register 0. The values and meanings of the return code are listed below.

- **0** CREATE successful
- **-1** CREATE failed -- disk or directory full.
- **-2** CREATE failed -- file already on disk.
- **-5** CREATE failed -- illegal filename.
- **-7** CREATE failed -- argument error.
3.3.5 DELETE function

DELETE performs the reverse action of CREATE. DELETE removes a file from the disk directory and frees all the records from the file. The allocation table in memory is updated to include all the now available free records.

The calling sequence for DELETE is

LD AC2,<FCB address>
LI AC3, 5
JSR EDOS

where register 2 is loaded with the starting address of the FCB, register 3 is set to 5, the function number for DELETE. EDOS is the entry point to DOS.

DELETE saves the contents of registers 1, 2 and 3, and puts a return code in register 0. The values and meanings of the return code are listed below.

0 DELETE successful.
-5 DELETE failed -- missing or filename error.
-6 DELETE failed -- file not found.

3.3.6 GETR function

The GETR (GET Record) function is used to read a single record from a disk file into memory. The disk file must be previously opened with its FCB residing in memory. The record to be read is pointed to by the NXR (NeXt Record) field in the FCB. The record is stored in a buffer of which the starting address is set by the SETDMA function (See Section 3.3.9).
The calling sequence for the GETR function is

LD AC2,<FCB address>
LI AC3, 6
JSR EDOS

where register 2 is loaded with the starting address of the FCB and register 3 is set to 6, the function number for GETR. EDOS is the entry point to DOS.

GETR saves the contents of registers 1, 2 and 3, and puts a return code in register 0. The values and meanings of the return code are listed below.

0 GETR successful
-1 GETR failed -- end of file.
-4 GETR failed -- invalid AM field found (Section 4.1.1).

3.3.7 PUTR function

The PUTR (PUT Record) function is similar to the GETR function except data is written from memory into a particular record of a disk file. The disk file must be previously opened with its FCB residing in memory. The record to be written is pointed to by the NXR field in the FCB. The record buffer in memory is pointed to by the DMA address, presettable by the SETDMA function. (See Section 3.3.9)

The calling sequence for the PUTR function is

LD AC2,<FCB address>
LI AC3, 7
JSR EDOS

where register 2 is loaded with the starting address of FCB and register 3 is set to 7, the function number for PUTR. EDOS is the entry point for DOS.
PUTR saves the contents of registers 1, 2 and 3, and puts a return code in register 0. The values and meanings of the return code are listed below.

<table>
<thead>
<tr>
<th>Code</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>PUTR successful</td>
</tr>
<tr>
<td>-1</td>
<td>PUTR failed -- disk full.</td>
</tr>
<tr>
<td>-4</td>
<td>PUTR failed -- invalid AM field found.</td>
</tr>
</tbody>
</table>

3.3.8 GETARG function

The GETARG (GET ARGument) function is used to get an input argument from the console input device. The arguments are entered at the same time a command file is called.

Upon entering the command file (simply a user program), GETARG allows user to access arguments entered on the same line with the command file call.

The calling sequence for GETARG is

```
LD ACO,<parameter number>
LI AC3, 8
JSR EDOS
```

where <parameter number> is a numeric constant indicating which parameter is to be gotten from the input string. Register 3 is set to 8, the function number for GETARG and EDOS is the entry point for DOS.

If the parameter is a filename then a 64 byte FCB is allocated for that file in memory. The filename and file extension fields are filled using the input filename. Register 2, upon return, contains the starting address of this FCB.
If the parameter is a numeric constant (first character being numeric) both the hexadecimal (AC2) and decimal (AC1) values of the constant are returned.

GETARG saves the contents of register 3 and puts a return code in register 0. The values and meanings of the return code are listed below.

- 0  GETARG successful -- filename is found.
- 1  GETARG successful -- numeric constant is found.
- -1 GETARG failed -- argument missing.

3.3.9 SETDMA function

The SETDMA (SET Direct Memory Address) function is used to define the starting address of the record buffer used by GETR and PUTR. The length of the buffer is 5 x 256 words long.

The calling sequence for SETDMA is

LD AC0,<buffer address>
LI AC3, 9
JSR EDOS

where register 0 is loaded with the starting address of the record buffer to be used. Register 3 is set to 9, the function number for SETDMA and EDOS is the entry point for DOS.

SETDMA saves the contents of all registers.
3.3.10 INTDR function

The INTDR (INterrogate DRive) function allows the user to find out what the current default drive is set to.

The calling sequence for INTDR is

\[
\begin{align*}
\text{LI} & \quad \text{AC3}, 10 \\
\text{JSR} & \quad \text{EDOS}
\end{align*}
\]

where register 3 is set to 10, the function number for INTDR and EDOS is the entry point for DOS.

INTDR saves the contents of registers 1, 2 and 3, and returns with the drive number in register 0.

3.3.11 INTALOC function

INTALOC (INterrogate ALLOCation) function returns to the user the number of records allocated for the currently logged-in disk.

The calling sequence for INTALOC is

\[
\begin{align*}
\text{LI} & \quad \text{AC3}, 11 \\
\text{JSR} & \quad \text{EDOS}
\end{align*}
\]

where register 3 is set to 11, the function number for INTALOC and EDOS is the entry point for DOS.

INTALOC saves the contents of registers 1, 2 and 3, and returns the allocation value in register 0.
3.3.12 WBOOT function

The WBOOT (Warm BOOT) function is used to reload part of DOS that has been overlayed by the user program. CCP, BDOS and BIOS are the parts to reload from the disk. If UFP is also overwritten by the user program, only the cold boot routine in ROM can reload DOS.

The calling sequence for WBOOT is

```
LI AC3, 12
JMP EDOS
```

where register 3 is set to 12, the function number for WBOOT.

3.3.13 ENDOS function

The ENDOS function allows the user to re-enter DOS. ENDOS is used only when DOS is not destroyed by the user program, else WBOOT or the cold start loader must be used.

The calling sequence for ENDOS is

```
LI AC3, 13
JSR EDOS
```

3.4 Program Example

The following program example creates a file called "F.X" with 1 record in it. The record is then written with data starting from memory location "1234" for the length of the record. Finally, the file is updated back on the disk (CLOSE).
OPEN = 2
CLOSE = 3
CREATE = 4
PUTR = 7
SETDMA = 9
FN = 2
FNX = 18
AC0 = 0
AC1 = 1
AC2 = 2
AC3 = 3

; SET UP FUNCTION NUMBERS

START:
LD AC2,FCBP ; GET FCB STARTING ADDRESS
LI AC0,'F'/256 ; FILENAME "F" IN ASCII
ST AC0,FN(AC2) ; PUT "F" IN FCB FN FIELD
LI AC0,'X'/256 ; FILE EXTENSION "X" IN ASCII
ST AC0,FNX(AC2) ; PUT "X" IN FCB FNX FIELD
LI AC3,CREATE ; SET UP FUNCTION NUMBER IN AC3
LI AC0,1 ; LENGTH OF FILE = 1 RECORD
JMP EDOS ; CALL OPERATING SYSTEM
JMP ERROR ; FOUND ERROR IN CREATE
LD AC0,OPEN ; SET UP FUNCTION NUMBER IN AC3
JMP EDOS ; CALL DOS TO OPEN FILE "F.X"
JMP ERROR ; OPEN FAILED, GOTO ERROR ROUTINE
LD AC0,ADDRESS ; GET STARTING ADDRESS IN MEMORY
JMP ERROR ; NEGATIVE RETURN CODE IMPLIES ERROR
JMP ERROR ; CLOSE FAILED, GOTO ERROR ROUTINE
RTS ; SUCCESSFUL RETURN

ERROR: ...(Process error here)....

NEG: .WORD -1
FCBP: .WORD FCB
EDOS: .WORD 09000 ; ENTRY POINT FOR DOS
ADDRESS: .WORD 1234
FCB: . = .+64 ; FILE CONTROL BLOCK AREA

A summary of the Disk Primitive Routines is given in Table 3.1.
<table>
<thead>
<tr>
<th>Number</th>
<th>Function</th>
<th>Entry parameters</th>
<th>Return value in ACO</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>SELECT</td>
<td>ACO = &lt;drive number&gt;</td>
<td>0 Successful, drive 0</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td>1 Successful, drive 1</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td>-5 Illegal drive number</td>
</tr>
<tr>
<td>2</td>
<td>OPEN</td>
<td>ACO = &lt;FCB address&gt;</td>
<td>0 Successful</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td>-1 File already open</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td>-5 Illegal filename</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td>-6 File not found</td>
</tr>
<tr>
<td>3</td>
<td>CLOSE</td>
<td>ACO = &lt;FCB address&gt;</td>
<td>0 Successful</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td>-1 File not open</td>
</tr>
<tr>
<td>4</td>
<td>CREATE</td>
<td>ACO= &lt;number of records&gt;</td>
<td>0 Successful</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td>-1 Disk full</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td>-2 File already exist</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td>-5 Illegal filename</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td>-7 Argument error</td>
</tr>
<tr>
<td>5</td>
<td>DELETE</td>
<td>ACO = &lt;FCB address&gt;</td>
<td>0 Successful</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td>-5 Illegal filename</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td>-6 File not found</td>
</tr>
<tr>
<td>6</td>
<td>GETR</td>
<td>ACO = &lt;FCB address&gt;</td>
<td>0 Successful</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td>-1 End of file</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td>-4 Invalid AM field</td>
</tr>
<tr>
<td>7</td>
<td>PUTR</td>
<td>ACO = &lt;FCB address&gt;</td>
<td>0 Successful</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td>-1 Disk full</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td>-4 Invalid AM field</td>
</tr>
<tr>
<td>8</td>
<td>GETARG</td>
<td>ACO = &lt;parameter number&gt;</td>
<td>0 AC2 = FCB address</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td>if parameter is filename</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td>1 AC1 = decimal ,</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td>AC2 = hexadecimal</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td>if parameter is numeric</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td>-1 Missing argument</td>
</tr>
<tr>
<td>Number</td>
<td>Function</td>
<td>Entry parameters</td>
<td>Return value</td>
</tr>
<tr>
<td>--------</td>
<td>------------</td>
<td>--------------------------</td>
<td>--------------------</td>
</tr>
<tr>
<td>9</td>
<td>SETDMA</td>
<td>[ACO = &lt;DMA address&gt;]</td>
<td>None</td>
</tr>
<tr>
<td>10</td>
<td>INTDR</td>
<td>None</td>
<td>[ACO = \text{drive number}]</td>
</tr>
<tr>
<td>11</td>
<td>INTALOC</td>
<td>None</td>
<td>[ACO = \text{allocation}]</td>
</tr>
<tr>
<td>12</td>
<td>WBOOT</td>
<td>None</td>
<td>None</td>
</tr>
<tr>
<td>13</td>
<td>ENDOS</td>
<td>None</td>
<td>None</td>
</tr>
</tbody>
</table>

\(<\text{drive number}>\) 0 or 1
\(<\text{FCB address}>\) starting address of FCB (4 digit hex)
\(<\text{number of records}>\) number of records in the file (0 to 24)
\(<\text{parameter number}>\) number of the parameter to be fetched (positive integer)
\(<\text{DMA address}>\) disk record buffer address (4 digit hex)
CHAPTER 4

IMPLEMENTATION OF DISK OPERATING SYSTEM

This chapter describes the implementation of DOS. Section 4.1 contains the description of all system tables and buffer areas. Their meanings and functions are discussed. Section 4.2 describes the actual implementation of the disk primitives. These primitives are OPEN, CLOSE, CREATE, DELETE, GETR and PUTR.

4.1 System tables and directories

DOS keeps track of all the file and disk information with three tables. These tables together with the disk buffers comprise the data area of DOS. These tables are the File Control Block (FCB), Disk Allocation Table (DAT), and the Directory Entry Table (DET).
4.1.1 File Control Block

The File Control Block stores information about a particular disk file. The FCB is a 64 byte long data block and can be placed anywhere in memory. DOS's data area supports up to 16 FCB's, while extra FCB's can be placed in the user data area. Fig 4.1 shows the format of the File Control Block.

The DE (DElete marker) field is used to indicate an invalid FCB. If DE contains the hexadecimal value "0E5" then that particular FCB is considered to be empty or void. Values other then "0E5" indicate a valid FCB.

The FNB (FCB Number) field contains the FCB number in the directory track (track 2). Each sector on the directory track holds up to 8 FCB's. Sector 1 contains FCB 0 to FCB 7, sector 2 contains FCB 8 to FCB 15 and so on for the other sectors. Fifteen sectors are used to store all the FCB's, for a total of up to 120 FCB's. Thus allowing a maximum of 120 distinct files on a single diskette.

The FN (FileName) and FNX (FileName eXtension) fields contain the filename and filename extension of the file. FN is 16 bytes long containing the ascii character filename and FNX is 3 bytes long containing the ascii character filename extension. Each field is padded with blanks if its name is shorter than the length of the field.

The NXR (NeXt Record) field contains a pointer to the next record to be read or written for that FCB. Each GETR and PUTR actually accesses one record of the file using the current NXR byte, and then increments that byte. Nonsequential record accessing from a file can be accomplished by altering the NXR byte in the FCB image in memory.
<table>
<thead>
<tr>
<th>byte</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>DE     Delete marker</td>
</tr>
<tr>
<td>1</td>
<td>FNB    FCB number</td>
</tr>
<tr>
<td>2</td>
<td>FN     Filename is ascii</td>
</tr>
<tr>
<td>17</td>
<td>FNX    File extension</td>
</tr>
<tr>
<td>21</td>
<td>NXR    Next record pointer</td>
</tr>
<tr>
<td>22</td>
<td>NR     Number of records in the file</td>
</tr>
<tr>
<td>23</td>
<td>FCBX   Next FCB extension</td>
</tr>
<tr>
<td>24</td>
<td>AM     Allocation Map</td>
</tr>
<tr>
<td>47</td>
<td>F柄OUR Hours of modification</td>
</tr>
<tr>
<td>49</td>
<td>FMIN   Minutes of modification</td>
</tr>
<tr>
<td>50</td>
<td>FSEC   Seconds of modification</td>
</tr>
<tr>
<td>51</td>
<td>FMONTH Month of modification</td>
</tr>
<tr>
<td>52</td>
<td>FDAY   Day of modification</td>
</tr>
<tr>
<td>53</td>
<td>FYEAR  Year of modification</td>
</tr>
<tr>
<td>54</td>
<td>FUSER  User name of modification</td>
</tr>
</tbody>
</table>

Figure 4.1 File Control Block Format
The NR (Number of Records) field contains a value equal to the total number of records allocated on disk for the file.

The FCBX (FCB eXtension) field is a link to the next FCB extension used for files containing more than 24 records. The value in this field is the FCB number of the next FCB to be linked to.

The AM (Allocation Map) field is 24 bytes long and each byte contains a record number. Byte i of the AM field is the record number of the i-th record in that file. Any sequence of record numbers in AM is legal, that is, the records of the file can be distributed arbitrarily to physical record locations on the disk.

The FHOUR, FMIN and FSEC fields store the time of the last modification to the associated file in hour, minute and seconds format. FHOUR is an integer between 0 and 23, FMIN and FSEC are integers between 0 and 59. The values of these fields are updated every time a file is CLOSE-ed.

The FMONTH, FDAY and FYEAR fields are similar to the time fields above except they represent the month, day and year of the last modification.

The FUSER field contains a 10 byte ascii string representing the name of the user who last modified the file.

4.1.2 Disk Allocation Table
The Disk Allocation Table (DAT) stores information about record allocation on the logged-in diskette. DAT is 74 words long, each representing a single track on the disk (starting with track 3). The $N$-th bit of a word in the DAT is set to "1", if the $N$-th sector group in the corresponding track has been allocated.

DAT is first initialized when a MOUNT is performed. All the valid FCB's on the disk directory are scanned and the DAT is initialized using the information stored in the AM field of the FCB's. Each valid AM byte in the FCB correspond to a single bit in the Disk Allocation Table. Each byte of the AM field is an integer from 0 to 221 representing the record number $(T-3) + 74(N-1)$. This integer is uniquely decoded into $T$ and $N$ and bit $N$ of DAT entry $T$ is set to a "1".

4.1.3 Directory Entry Table

The Directory Entry Table (DET) stores information about which entries in the disk directory are allocated. This table is 120 words long, each of the words corresponds to one directory entry (FCB) on the directory track. The contents of DET represent the length of every valid FCB entry. A zero value indicates a null FCB. The DET is thus equivalent to a collection of the DE's of the FCB's in a single table.

DET is initialized when a MOUNT is performed. DET is filled by scanning all the valid entries on the directory track and reading the length of each such entry. A copy of DET is also stored as the last sector of the disk directory. This copy of the DET is used by the DIR command.
The i-th entry in the DET corresponds to the i-th FCB on the directory track.

4.1.4 Disk Record Buffer

The Disk Record Buffer is used by the GETR and PUTR routines to input and output a record between disk and memory. This buffer is 1,280 words long and can be allocated anywhere in memory. The SETDMA function allows the user to set the starting address of this buffer area. This function provides a "software DMA" type of access, thus allowing the user to specify an arbitrary memory location for direct reading and writing of file records to and from memory.

A single word in the Disk Record Buffer represents 2 bytes on the disk. The disk I/O routines automatically convert between pairs of data bytes in the disk record and words in the Disk Record Buffer. They use the Disk Buffer, described below, as intermediate storage for this purpose.

4.1.5 Disk Buffer

The Disk Buffer is a 512 word area of memory set aside for use by the disk I/O routines. Only the lower 8 bits of each word is used making a direct correspondance to the single sector, 512 byte FD-402 floppy disk controller buffer.
During a disk read, the disk controller puts one sector of data, 512 bytes, into its buffer. Then BIOS reads the controller buffer into the Disk Buffer in memory. The write operation is similar except data flow is reversed.

This buffer is used by system routines and should not be used by the user normally.

4.2 Implementation of Disk I/O routines

The disk I/O routines are accessible to the user through the User Function Processor. These routines allow user access to all disk operations and data in files stored on the disk.

The operation of each of the disk primitives, OPEN, CLOSE, CREATE, DELETE, GETR and PUTR is described in the following sections.

4.2.1 OPEN routine

The OPEN routine activates a particular file and initiates the processing necessary for input and output. The routine first checks to make sure that the file is not currently opened and returns an error if it is. OPEN then searches the directory to locate the file's FCB. If the FCB is found, it is read into the FCB area in memory and a table entry is created in that OPEN table indicating that the current file is opened.
In case the file is not found, an error code is returned to the user. The user is responsible for further action after an error occurs. OPEN returns to the user after it is finished.

See Section 3.3.2 for a description of the OPEN calling sequence and return codes.

4.2.2 CLOSE routine

The CLOSE routine reverses the OPEN function. CLOSE first checks whether the file was previously opened and returns an error code if it was not. CLOSE next updates the File Control Block of that file in the directory track. Changes in length of file, allocation information, and historical information, are updated at the same time. The Directory Entry Table is also updated on the disk to reflect any changes that occur in the directory. CLOSE then returns to the user.

See Section 3.3.3 for a description of the CLOSE calling sequence and return codes.

4.2.3 CREATE routine

The CREATE routine creates a named file entry on the disk and allocates disk space for that file. CREATE's first action is to make an attempt to open the file. If opening succeeds, a file with the same name already exists on the disk and an error code is returned to the user. Only if open fails will CREATE continue. CREATE then sets up an empty FCB for the file with only the filename field filled. The
Directory Entry Table is then scanned for the allocation of a new entry. Once an entry is found the FNB field is then set to the entry number found. The Disk Allocation Table is next scanned for allocation of disk space requested by the user. The AM field of the FCB is then filled with the record pointers allocated and DAT is updated to include the new allocation. In the event the directory is full or not enough disk space is available, an error code is returned to the user.

After the allocation, CREATE updates the FCB on the directory track and returns to the user. See Section 3.3.4 for the calling sequence of CREATE and the meanings of return codes.

4.2.4 DELETE routine

The DELETE routine removes a file from the disk directory entry table. DELETE first checks to see if the file exist on the disk by an attempt to open that file. Once the file is opened, DELETE scans the AM field in the FCB and one by one deletes the record pointers from the Disk Allocation Table. The DE field is then set to "OE5" to indicate it is now deleted. The FCB is then written back on the disk directory and DELETE returns to the user.

See Section 3.3.5 for a description of the DELETE calling sequence and return codes.
4.2.5 GETR routine

The GETR routine starts an input operation on a file and reads one record from that file into memory. The record to be read is pointed to by a record pointer in the FCB AM field. The NXR field points to a vector in the AM field which is used as the record pointer. GETR checks to see if the pointer is valid or not, so as to ensure valid data from the disk. An error code is returned to the user if an invalid pointer is found.

GETR reads in each of the sectors corresponding to the record in turn and places them in the Disk Buffer. After each sector is read, GETR moves the Disk Buffer into the appropriate part of the Disk Record Buffer until all the sectors are transferred.

See Section 3.3.6 for a description of the GETR calling sequence and the meanings of the return codes.

4.2.6 PUTR routine

The PUTR routine starts an output operation and writes the contents of the Disk Record Buffer one sector at a time through the Disk Buffer into a file on the disk.

The function of PUTR is similar to GETR except that the flow of data which is reversed. Both PUTR and GETR return to the user only after the complete record is transferred.
See Section 3.3.7 for a description of the calling sequence for PUTR and the meanings of the return codes.
CHAPTER 5

FLOPPY DISK INTERFACE HARDWARE DESCRIPTION

This chapter describes the FD-402 Floppy Disk System interface to the ADS-16 system. The logic diagram of the interface is located in Appendix A.

5.1 FD-402 Floppy Disk Controller

The FD-402 Floppy Disk Controller (FDC) is a complete preprogrammed controller for floppy disk drives. It performs control functions to transfer data between two Shugart double density 8-inch floppy disk drives, and a host system. The FD-402 interfaces to IMP-16 through the use of three I/O Ports shown in FIG 5.1. These three I/O ports provide control, status and data paths between the IMP-16 and controller via 19 TTL interface signal lines.
Figure 5.1 FD-402 I/O ports
The DATA port is used for read data, write data, command data and FDC status. The STATUS port is used to transmit handshake signals to the IMP-16 and to identify status type information available on the DATA port. The CONTROL port is used to initiate FDC operations and to acknowledge handshake signals from FDC when data transfers are occurring between the FDC and the IMP-16.

The FD-402 system provides nine macro functions which implement all the basic operations required to address, read and write user data on the disk. These nine functions are described in Table 5.1. For detailed descriptions of the functions, see [3].

5.2 IMP-16 interface timing and logic

The following sections describe the interface timing and logic necessary for correct command, data and status transfers between the IMP-16 and the FDC. References are made to the Floppy Disk Interface (FDI) schematics in Appendix A. The notation "IC-XX" represents the Integrated Circuit labelled XX on the schematics. The FDI is a card in the IMP-16 system.

5.2.1 Command Input Timing and Logic
<table>
<thead>
<tr>
<th>COMMAND BYTES</th>
<th>FUNCTION</th>
</tr>
</thead>
<tbody>
<tr>
<td>byte 1</td>
<td>byte 2</td>
</tr>
<tr>
<td>0X0XX000</td>
<td>XXXXXXXX</td>
</tr>
<tr>
<td>1X0XX000</td>
<td>XXXX1ff</td>
</tr>
<tr>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
</tr>
<tr>
<td>XX0dd011</td>
<td>X&lt;--T--&gt;</td>
</tr>
<tr>
<td>hb0dd010</td>
<td>X&lt;--S--&gt;</td>
</tr>
<tr>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
</tr>
<tr>
<td>XXXdd011</td>
<td>XXXXXXXX</td>
</tr>
<tr>
<td>hb0dd100</td>
<td>X&lt;--S--&gt;</td>
</tr>
<tr>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
</tr>
<tr>
<td>hb0dd101</td>
<td>X&lt;--S--&gt;</td>
</tr>
<tr>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
</tr>
<tr>
<td>XX0dd110</td>
<td>X&lt;--T--&gt;</td>
</tr>
<tr>
<td>XX0XX111</td>
<td>XXXXXXXX</td>
</tr>
</tbody>
</table>

Note: "X" represents a don't care.
All floppy disk controller activity is initiated by the host system via commands, which may be issued whenever the floppy disk controller is not busy. The state of the FDC is indicated by the BUSY and DONE bits in the status port.

When status bits BUSY=0 and DONE=1, the host system may request a new disk operation by initiating a command. The associated command input timing is shown in Figure 5.2.

The IMP-16 initiates a command sequence by placing the first command byte on the DATA port and raising the CMD line. The CMD line is held high and the DATA port stays valid until the FDC raises the XFR line, signaling acceptance of the first command byte. The IMP-16 then drops the CMD line, places the second command byte on the data port and raises the CMD line. Again the CMD line is held high and the DATA port stays valid until the FDC raises the XFR line, signaling acceptance of the second command byte.

The control line is completely asynchronous and the only timing considerations for the IMP-16 are the FDC's response delays.

Below is an IMP-16 assembly language routine for sending commands to the FDC with complete handshake. This routine assumes that checking has previously been done to insure that DONE is 1 and BUSY is 0.

```
LD ACO, BYTE1 ; GET FIRST COMMAND BYTE
ST ACO, @DATA ; PUT COMMAND INTO FDC DATA PORT
LI AC1, 080 ; SET UP CMD BIT IN REGISTER 1
ST AC1, @CONTROL ; RAISE CMD LINE

WAIT: LD ACO, @STATUS ; READ FDC STATUS PORT
      AND ACO, XFRMASK ; MASK OUT XFR BIT
      BOC NZRO, WAIT ; IF XFR (BIT 7)=0 THEN PROCEED
                      ; ELSE LOOP BACK
```

(repeat the above code for second command byte)
Figure 5.2 Command Input Timing
After the program places the first command byte into the FDI Data port, the CMD line is set by writing a "1" in the CMD bit of the Control port. This sets the output of flip-flop IC-D4 to a "1" which is then driven to the FDC CMD line. The FDC acknowledges by raising the XFR line immediately. The output of the flip-flop IC-D4 is reset at the falling edge of XFR. This completes the handshake for the first command byte. The second byte is sent to the FDI with similar handshake. Some commands (not used in this system) require four bytes and use the extended handshake.

5.2.2 Data Transfer Timing and Logic

When Data is transferred between the FDC and the IMP-16, the XFR line and ACK line are used to complete a handshake between the two.

In the FD-402 controller, a data byte transfer (i.e., data available from the floppy disk controller or data requested from the IMP-16) is indicated when the FDC sets the XFR line. The IMP-16, in turn, acknowledges the data transfer by setting the ACK line. The IMP-16 holds the DATA valid during a WRITE until the XFR line goes low.

Figure 5.3 shows the timing for both buffered reading and buffered writing by the IMP-16. In this mode, the floppy disk controller raises XFR after placing a data byte on the DATA port (buffered read) or when it is waiting for a new byte from the IMP-16 (buffered write). When the IMP-16 acknowledges the data transfer by raising ACK, the FDC lowers the
Figure 5.3  Data Read/Write Timing
XFR line and waits for the ACK line to go low. There are no timing constraints placed on the IMP-16's response.

If the user requests a data transfer between the floppy disk controller and its buffer, no user interaction is required during the data transfer.

Below is an IMP-16 assembly language routine for data transfer handshake between the IMP-16 and the FDC.

```
WAIT:  LD ACO,@STATUS ; READ FDC STATUS PORT
       AND ACO,XFRMASK ; MASKOUT XFR BIT
       BOC ZRO,WAIT ; IF XFR=0 THEN PROCEED,
                      ; ELSE LOOP BACK
READ:  LD ACO,@DATA ; READ DATA FROM FDC DATA PORT
       INTO REGISTER 0. SUBSTITUTE
             ; "LD" WITH "ST" IF THE OPERATION
             ; IS A WRITE
```

When the FDC is ready for data transfer, the XFR bit (bit 7) is set in the Status port. The program waits for that bit to be a "1" then proceeds with the data transfer operation by reading or writing the Data port of the FDC. This operation sets the output of flip-flop IC-C5 to a "1" which is then gated to the ACK line of the FDC. Data are latched in IC-E4 and IC-E5 for a read operation and IC-E2 and IC-E3 for a write operation. The ACK line (flip-flop IC-C5 output) is reset at the falling edge of the XFR line. XFR will fall before WAIT can be reentered following READ or WRITE. Experience in fact has shown that the BOC instruction always fails for successive bytes in a block transfer after the first byte signifying that XFR always returns to "1" before the program loop is completed. The WAIT loop is thus unnecessary for this system except for the first byte of the sector. Multiple byte
5.2.3 Status Timing and Logic

Status is available during two occasions -- in response to a STATUS command and following an attempt to execute any command (except READID).

If an error occurs during an operation, the ERROR flag is set and the kind of error is identified in STATID. The actual cause of the error is reported in the secondary status byte placed on the DATA port. If no error occurs during an operation, the status returned will indicate normal completion. In this case, the ERROR flag is set to zero.

At the end of an operation (BUSY=1, DONE=1), the secondary status is placed on the DATA port, the status bits, ERROR and STATID, are set on the STATUS port, and the XFR line is asserted. There are no timing constraints placed on the user's response to the status reporting, but the user must acknowledge the status (ACK=1) before the FDC will clear its BUSY line and accept a new command. Figure 5.4 shows the Status Timing and Figure 5.5 shows the meanings of the status words.

Below is an IMP-16 assembly language routine that waits for completion of a disk operation and then acknowledges the FDC for status.

```assembly
WAIT:  LD  ACO,@STATUS   ; READ FDC STATUS
       AND ACO,DONEMASK ; MASK OUT DONE BIT
       BOC ZRO,WAIT    ; IF NOT DONE, LOOP AND WAIT
       LD  ACO,@STATUS ; ELSE CHECK STATUS FROM FDC
       SKAZ ACO,ERRORMASK ; CHECK IF ERROR BIT SET
       JMP ERROR        ; ERROR FOUND
       LD  ACO,@DATA    ; NO ERROR, READ SECONDARY
                        ; STATUS FROM DATA PORT
```
Figure 5.4 Status Timing
STATID bits | Interpretation
---|---
0 0 | No secondary status
0 1 | Invalid format type
  | Invalid track address
  | Invalid sector address
  | Invalid buffer specification
1 0 | Disk drive not ready
  | Disk drive unsafe
  | Disk write protect
  | Bad format track
  | Seek in progress
1 1 | Read/write aborted
  | Head positioning error
  | Data address mark missing
  | Data overrun
  | Sector unrecoverable
  | Data CRC error
  | Read write completed
  | Delete data read

Figure 5.5 Status Words
ERROR: (process disk error here)

After the FDC places the status on the Status port and raises the XFR line, the program proceeds to read the Status port and check for errors. Then the program acknowledges the Status by reading the Data port, which sets the output of flip-flop IC-C5 (the ACK line). Then the FDC clears the BUSY line after it lowers the XFR line which resets the output of flip-flop IC-C5.

5.3 Floppy Disk Interface Specifications

The IMP-16 Floppy Disk Interface (FDI) is designed as memory mapped I/O in the ADS-16 system. The address block for the FDI is dip-switch selectable and occupies 16 words of memory locations in the IMP-16 memory space. Only four of the sixteen locations are currently used to implement the interface. The I/O ports, Data, Control and Status shown in Figure 5.1 correspond, respectively, to the first three of these locations, the fourth location, when read or written resets the FDC, regardless of the data.

Timing specifications on the read and write operations are shown in Figure 5.6 and Figure 5.7 respectively. Detailed timing information is available in [1].
Figure 5.6 IMP-16 Read Timing

Figure 5.7 IMP-16 Write Timing
5.3.1 Cable Conventions

A 50 conductor Scotch-flex flat cable is used for electrical connection between the FDC and the IMP-16 FDI. Table 5.2 shows the pin designations for the 50 conductor cable.
Table 5.2

Pin designation for FDI and the FDC interface

<table>
<thead>
<tr>
<th>pin number</th>
<th>names</th>
</tr>
</thead>
<tbody>
<tr>
<td>2</td>
<td>XFR</td>
</tr>
<tr>
<td>4</td>
<td>ACK</td>
</tr>
<tr>
<td>6</td>
<td>CMD</td>
</tr>
<tr>
<td>8</td>
<td>DCTRL</td>
</tr>
<tr>
<td>10</td>
<td>STATID 0</td>
</tr>
<tr>
<td>12</td>
<td>STATID 1</td>
</tr>
<tr>
<td>14</td>
<td>DONE</td>
</tr>
<tr>
<td>16</td>
<td>BUSY</td>
</tr>
<tr>
<td>18</td>
<td>D0</td>
</tr>
<tr>
<td>20</td>
<td>D1</td>
</tr>
<tr>
<td>22</td>
<td>D2</td>
</tr>
<tr>
<td>24</td>
<td>D3</td>
</tr>
<tr>
<td>26</td>
<td>D4</td>
</tr>
<tr>
<td>28</td>
<td>D5</td>
</tr>
<tr>
<td>30</td>
<td>D6</td>
</tr>
<tr>
<td>32</td>
<td>D7</td>
</tr>
<tr>
<td>34</td>
<td>D8</td>
</tr>
<tr>
<td>36</td>
<td>NC</td>
</tr>
<tr>
<td>38</td>
<td>NC</td>
</tr>
<tr>
<td>40</td>
<td>NC</td>
</tr>
<tr>
<td>42</td>
<td>NC</td>
</tr>
<tr>
<td>44</td>
<td>NC</td>
</tr>
<tr>
<td>46</td>
<td>NC</td>
</tr>
<tr>
<td>48</td>
<td>RESET</td>
</tr>
<tr>
<td>50</td>
<td>NC</td>
</tr>
</tbody>
</table>

where NC is "No Connection" and all ODD number pins are GROUND.
CHAPTER 6

CONCLUSION AND FURTHER WORK

The IMP-16 Floppy Disk Interface has been constructed and tested for functionality. All the disk functions described in this paper functioned correctly. A number of utility programs have been written using DOS functions and all have worked.

The next step to be taken is the building of more utility routines to facilitate data transfers. One very useful utility program is a loader between the IMP-16 and the CDC Cyber so that files on Cyber can be backed-up on floppy diskettes and vice-versa. The User Function Processor allows very easy access to the disk functions used in the writing of these utility routines.

Further work is now possible in the area of resident assemblers, text editors and high level language compilers where the use of the Floppy Disk will greatly ease the implementation and use of these programs.
Other system modifications which could be implemented are: the construction of a high speed Direct Memory Access Channel for fast disk I/O, software memory management, dynamic memory allocation and memory paging capabilities.
REFERENCES


2. INTERACT Operating System User's Manual Master's Thesis by Larry Lewis Hanes, University of Illinois at Urbana-Champaign, 1978

Appendix A
FDI hardware schematics
Figure A1 Floppy Disk Interface schematic
Figure A2 Floppy Disk Interface schematic
Figure A3 Floppy Disk Interface schematic