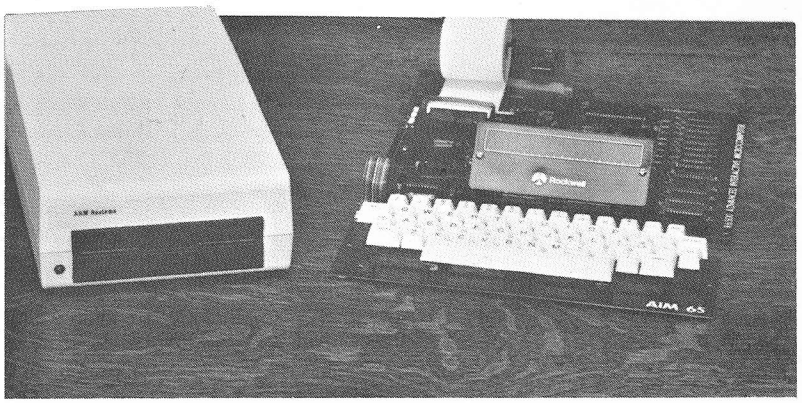
DATA SHEET

AIM-65 FLOPPY DISK SUBSYSTEM AH5050

The AH5050 offers a powerful yet inexpensive mass storage subsystem for the AIM 65 Microcomputer. It is fully compatible with the AIM 65's Editor, Assembler, Monitor, FORTH and Basic. Sequential, Random or Relative File types are supported. The AH5050 subsystem offers the capability of opening up to 10 files simultaneously, with features such as pattern matching and wild card file searches found only in larger systems. The file manager accessed through the "F1" key of the AIM 65 offers standard commands for Directory, Rename, Transfer, Delete, Format, Clear-plus special commands to run Command files.

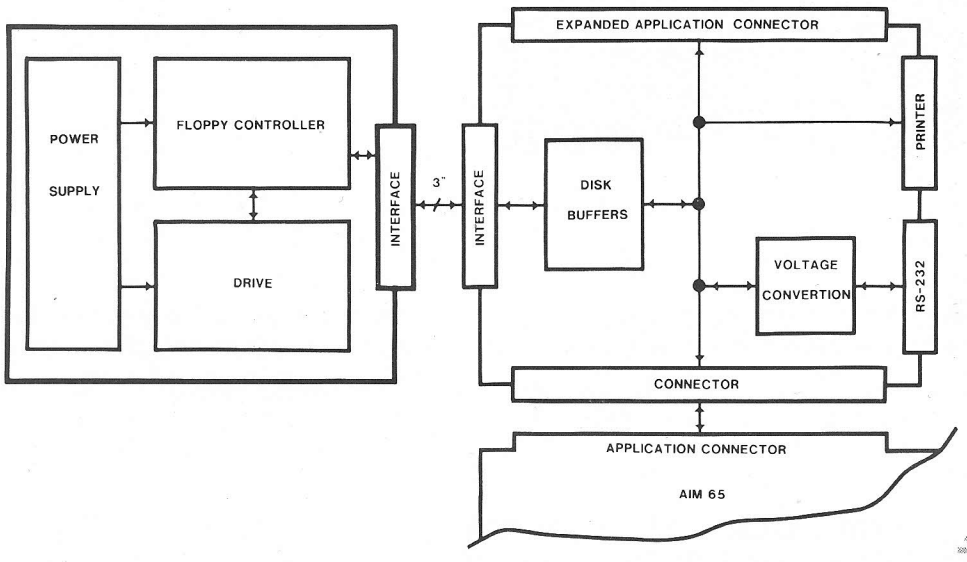
New BASIC and FORTH instructions such as OPEN, CLOSE, CHIN, CHOUT allow manipulation of disk data files during program execution. Other commands such as SAVEB/LOADB, store and retrieve BASIC or FORTH programs in binary form. This offers the user the capability to run BASIC programs like ordinary binary files even on power up. The AH5050 also includes Centronics and RS-232 compatible interfaces with supporting software.



FEATURES

- Fully compatible with the AIM 65 Microcomputer's BASIC, EDITOR, MONITOR, ASSEMBLER and FORTH
- Subsystem package includes intelligent controller, drive, power supply, case, cables, manual and software
- Added new commands to AIM BASIC and FORTH languages
- No motherboards or adaptor cards required. AIM 65 expansion connector remains free for other uses
- Compact size
- Centronics and RS-232 Interfaces
- Access to Sequential, Random and Relative type of files
- Wild card file searches
- Capability to open up to 10 files simultaneously
- File Manager commands:

Directory	Rename	Clear
Transfer	Merge	Initialize
Format	Scratch	Validate
Put	Run	



AH5050 FUNCTIONAL CAPABILITIES

The AH5050 Floppy Disk Subsystem is an inexpensive mass storage unit for the AIM 65. The AH5050 is designed for applications where low price is required, such as data acquisition, controllers, testers, etc. The AH5050 includes a floppy disk drive, intelligent floppy controller, power supply case, interface and software for the AIM-65 microcomputer.

The AH5050 is interfaced to the AIM-65 through the application connector using 5 lines of the User 6522 VIA.

The software for the AH5050 is initialized once upon power up. After the initialization access to the file management is done through the "F1" key and access to the files is done through the "user" vector in the AIM-65.

Access to files from the AIM-65 microcomputer's BASIC, Editor or Monitor is obtained through the existing commands of "Save / Load", "List / Read", "Dump / Load" respectively. Additional commands such as OPEN, CLOSE, CHIN, CHOUT, PRINT #, INPUT #, CMD, STATUS, SAVEB, LOADB; enhance the AIM BASIC and FORTH Languages so the user can manipulate sequential, Random or Relative Data files. BASIC and FORTH Programs can be saved and loaded in binary form. These binary files load very quickly since they are already in executable form. It is possible to automatically load and run a binary file upon power-up of the AIM. This virtually eliminates the need for storage of application programs in PROM.

The software opens and closes files automatically. The user has the capability to open up to ten files simultaneously.

SPECIFICATIONS

Storage

Total Capacity	174848 bytes per diskette
Sequential	168656 bytes per diskette
Relative	167132 bytes per diskette
Directory entries	144 per diskette
Sectors per track	17 to 21
Bytes per sector	256
Tracks	35
Sectors	683 (644 free)

Power Requirements

Voltage	100, 120, 220, or 240 VAC
Frequency	50 or 60 Hertz
Power	25 watts

Physical

Height	3.8 inches
Width	7.9 inches
Depth	14.7 inches

Miscellaneous Requirements

Free air operating Temperature: 0° to 55°C
Storage Temperature: +40° to 75°C
Operating humidity: 5% to 95% non-condensing
3" interface cable

Can be used with standard 5¼", single sided, single density soft sectored diskettes.

USERS' MANUAL

To obtain the User's Manual for the AH5050, ask for ABM document No. AL5051

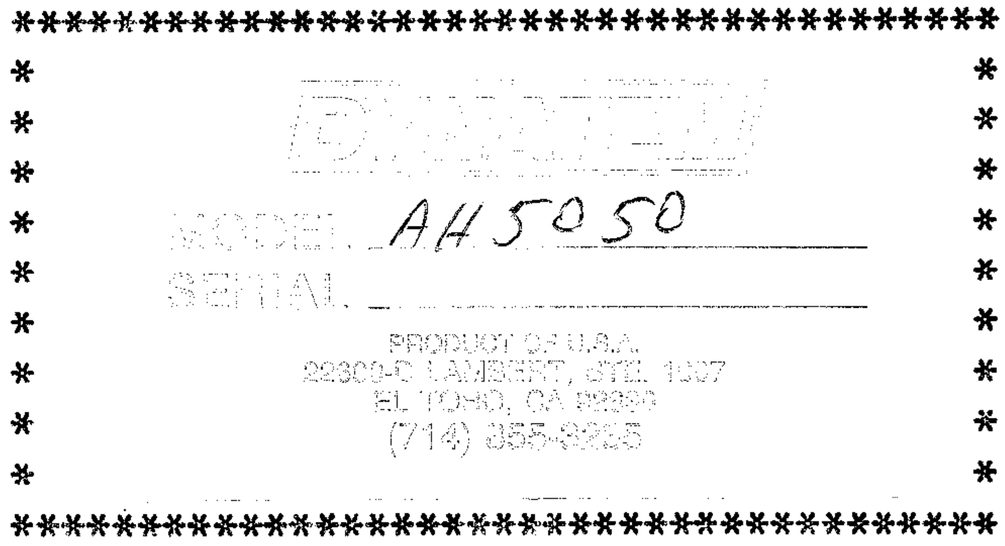
ORDERING INFORMATION

AH5050:	U.S. Floppy Disk Subsystem for AIM 65
AH5050E:	Euro Floppy Disk Subsystem for AIM 65
AH5052:	U.S. Floppy Disk Subsystem for CMOS CPU
AH5052E:	Euro Floppy Disk Subsystem for CMOS CPU

List Price: $495

AL5050
5/83

ABM Systems 14461 Silverbrook Tustin, CA 92680 (714) 832-7336



Specifications Subject to Change Without Notice

October 1 1983

AL5051

AH 5050 DISK SUBSYSTEM

Section 1: General Information

1.1 Introduction
1.2 Features
1.3 General Description
1.4 Specifications

Section 2: Installation

2.1 Introduction
2.2 Inspection
2.3 Installation
2.4 Memory Map
2.5 Disk Connector
2.6 Printer Connector
2.7 RS-232 Connector
2.8 Extended AIM Connector

Section 3: Operating System

3.1 Introduction
3.2 Initialization
3.3 File Name and Types (Seq,Prog,Rel)
3.4 Pattern matching, wild cards and replace
3.5 Log Files/and Log Devices
3.6 File Management (DIR,TYP,NAM,FRMT,CLR,MRG,PUT,GET,RUN...)
3.7 Existing Software Interface (MONITOR,EDITOR,ASSM,BASIC)
3.8 Machine Language Subroutines

Appendix A: Internal Structure

Appendix B: Error Summary

Appendix C: Command Summary

Appendix D: Interface Module Schematic

SECTION 1
GENERAL INFORMATION

1.1 INTRODUCTION

Welcome to the easiest, most efficient, most powerful and most inexpensive filing system for your AIM 65 microcomputer, the AH5050 floppy disk system.

The intelligent AH5050 Floppy system consists of a 5"1/4 disk drive, a 6502 CPU based controller with 16K bytes of software in ROM, power supply and case. In addition there is 4K bytes of software in PROM installed in the AIM 65 to handle all interfaces so they are transparent to the user. The interface module that plugs on the application connector of the AIM 65 also expands the capabilities of the microcomputer by providing Centronic's type parallel and RS-232 serial I/O ports.

1.2 FEATURES

* Fully integrated with the AIM 65 Microcomputer's Basic, Editor, Monitor, Assembler and Forth.

* Subsystem package includes intelligent controller, drive, power supply, case, cables, manual and software.

* Added new commands to AIM BASIC and FORTH languages.

* No expensive motherboards required. Free AIM Expansion connector.

* Centronics and RS-232 type Interfaces.

* Sequential, Random and Relative files supported.

* Wild card & pattern matching features.

* Capability to open up to 10 files simultaneously.

* File Manager built-in: Directory, Type, Format, etc.

1.3 GENERAL DESCRIPTION

The AH5050 allows you to store up to 144 files on a 5"1/4 disk.

Included in the drive is circuitry for both the disk controller and a complete disk operating system, a total of 16K of ROM and 2K of RAM memory. This circuitry makes the AH5050 an "intelligent" device. This means that the AH 5050 does it's own processing without requiring any memory from your AIM 65 computer.

The AH5050 Floppy system contains a dual "serial bus" interface. The signals of this bus resemble the parallel IEEE-488 interface used in

other computers, except that only one wire is used to communicate data instead of eight. The two ports at the rear of the drive allows more than one device to share the serial bus at the same time. This is accomplished by "daisy-chaining" the devices together. Up to 4 drives and one printer can share the bus simultaneously.

1.4 SPECIFICATIONS

Storage:

Total capacity	174848 bytes per diskette
Sequential files	168656 bytes per diskette
Relative files	167132 bytes per diskette
	65535 records per file
Directory entries	144 per diskette
Sectors per track	17 to 21
Bytes per sector	256
Tracks	35
Blocks	683 (664 blocks free)

Physical:

Height	97 mm
Width	200 mm
Depth	374 mm

Electrical:

Power Requirements

Voltage	100, 120, 220 or 240 VAC
Frequency	50 or 60 Hertz
Power	25 Watts

Media:

Diskettes	Standard mini 5 1/4", single sided, single density

SECTION 2
INSTALLATION

2.1 INTRODUCTION

This section contains instructions for inspection and installation of the AH5050 system. This section also contains memory map of the system to avoid any conflict with other modules.

2.2 INSPECTION

Upon receipt of the AH5050 system, inspect it for any broken, damaged, or missing parts. Included with the AH5050 drive unit you should find a power cable,a serial bus cable,an interface board,a PROM and a manual.

2.3 INSTALLATION

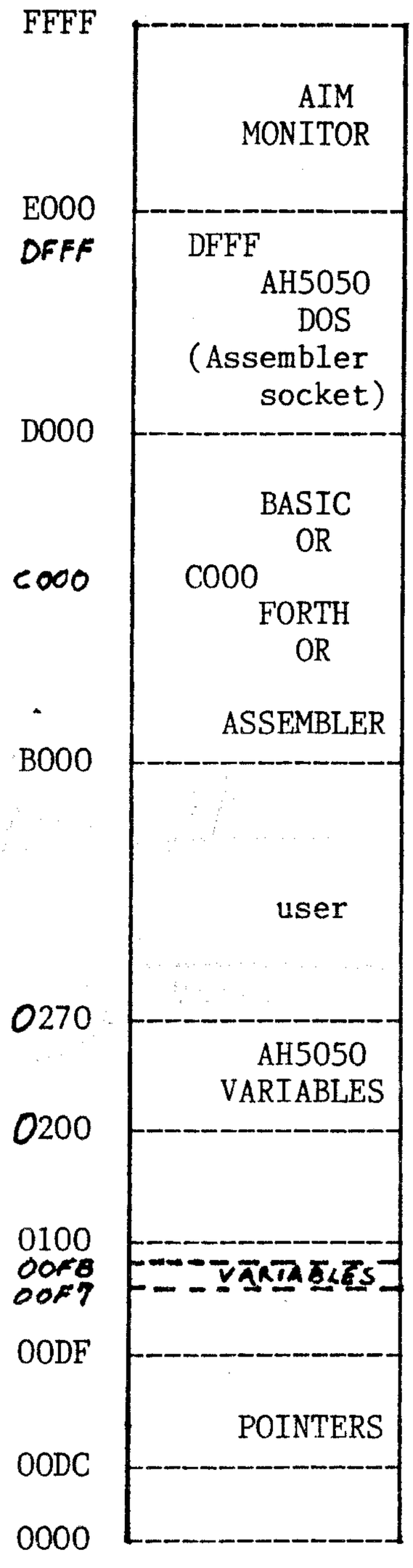
Be sure to disconnect all power before making any attempt to install the interface board. The interface to the disk and printer requires only +5Vdc. The current loop interface in the AIM 65 is converted into a voltage interface so it can be used to drive RS-232 devices. Although the voltages used to drive the RS-232 port are not the full specified +/- 12 Vdc, they are sufficient since the threshold for RS-232 devices is set at +/- 3Vdc. In this way there is no need for an additional power supplies of +/- 12 Vdc.

2.4 MEMORY MAP

The AH5050 interfaces to the AIM 65 through the on-board user 6522 VIA device using just 5 signals. Additional signals from this device are used if an interface to a Centronics printer is desired. A powerful operating system resides in the controller itself. But a PROM of 4K bytes installed at $D000 (the assembler socket) is required for the file management handling, expanded instructions for the BASIC, FORTH and general interface to the Editor, Assembler and Monitor on the AIM 65.

A minimal amount of RAM memory is used for variables and pointers as shown in the diagram.

FIGURE 2.1
MEMORY MAP

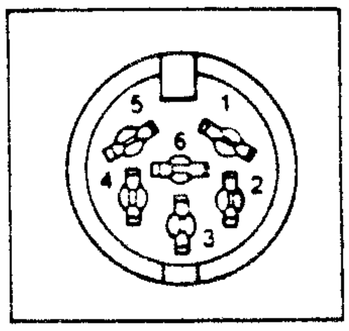


2.5 DISK CONNECTOR

The AH5050 Floppy drive interfaces to the AIM 65 through a serial bus cable which has the same connector on both ends,a 6-pin DIN plug which attaches to the interface board or another drive. The signal pin-out of this cable (J2) is shown below.

FIGURE 2.2
DISK INTERFACE (J2)

PIN	DESCRIPTION	
1	SERIAL SRQ	IN
2	GND	
3	SERIAL ATN	I/O
4	SERIAL CLK	I/O
5	SERIAL DATA	I/O
6	GND	



2.6 PRINTER CONNECTOR

Another connector (J3) in the interface board is provided for printer interface. Figure 2.3 shows the pin assignment of this connector. The interface is Centronics type parallel with 8 data lines, one Strobing and one Acknowledgeling for proper handshaking. The handshaking of data is controlled by the software that operates the I/O device.

FIGURE 2.3
PRINTER PIN ASSIGNMENT (J3)

STROBE (CA2)	---	1	2	---GND
DATA 0	---	3	4	---GND
DATA 1	---	5	6	---GND
DATA 2	---	7	8	---GND
DATA 3	---	9	10	---GND
DATA 4	---	11	12	---GND
DATA 5	---	13	14	---GND
DATA 6	---	15	16	---GND
DATA 7	---	17	18	---GND
ACKNOWLEDGE (CA1)	---	19	20	---GND
NOT USED	---	21	22	---NOT USED
NOT USED	---	23	24	---NOT USED
NOT USED	---	25	26	---NOT USED

2.7 RS-232 CONNECTOR

There are 4 signals used to interface to RS-232 compatible devices. The signals are obtained from the current loop offered in the AIM 65. Figure 2.4 shows the connector used (J4).

FIGURE 2.4
RS-232 INTERFACE (J4)

PIN	AH5050 MODULE	CONNECTION TO RS232 TERMINAL PIN
1	SERIAL-OUT	RD* 3
2	SERIAL-IN	TD* 2
3	-5V	-- -
4	GND	GND 1,7

NOTE: In order to receive the RS-232 signal on pin Y of the AIM 65 Applications Connector the user must change the resistor value of R24 from 1K to 3.3K . Presently in the AIM-65, R24 and R23 provide a divider circuit which makes it impossible to be able to drive the following input gate.

The user can connect a terminal to the AIM 65 through the AH5050 interface board. The user must change the KB/TTY switch on the AIM to the TTY position, press reset and type one key on the terminal so the AIM calculates the Baud Rate.

The AIM does not properly calculate Baud Rates higher than about 2000. Nevertheless the user can set the proper baud rates in software by setting the variables CNTH30 and CNTL30 with values given in the AIM 65 User's Manual. This could be done in the "STARTUP" program (see section 3.2)

2.8 EXTENDED AIM APPLICATION CONNECTOR

This connector is an edge connector. All signals from the AIM 65 are passed across the interface board to this connector. This connector (J5) looks exactly like the AIM 65's application connector (J1).

SECTION 3
OPERATING SYSTEM

3.1 INTRODUCTION

This section explain the procedures and capabilities of the Disk Operating System supplied with the AH5050 Floppy System.

The software for the AH5050 Operating System is contained in 16K bytes of ROM inside the controller and 4K bytes of PROM on the AIM 65.

The disk drive controller is 6502 CPU based, simplifing the handling of files. However the controller still requires the opening, channel switching, data transfer and closing of files.

Included in the 4K bytes of software ($D000 - $DFFF) on the AIM 65 are the interface subroutines to both the drive and to the AIM 65 existing software, as well as AIM 65 enhancements for BASIC and FORTH languages .

3.2 INITIALIZATION

To initialize the AH5050 all the user must do is to start progran executing at location $D000. This can be accomplished in two ways : first by pressing the "N" key for the AIM keyboard. The second way will provide execution upon power-up of AIM 65. This requires the changing of 3 bytes in the monitor at location $E141, to $4C $00 and $D0 (JMP $D000). Substitution of this code can be easily done by reading the ROM at Z2 ($E000-$EFFF) into RAM memory, changing the three bytes and burning 4K bytes back into a 2532 EPROM with a PROM programmer.

Pointers and variables for the AH5050 and the AIM 65 user vectors are initialized upon execution starting at location $D000. In addition a file named "START UP" is fetched from the drive and executed. This is a very powerful feature since the file "START UP" can be created from BASIC, FORTH or an Assembly program.

The file "START UP" could be used to initialize other devices such as CRT controllers and then run a BASIC program. Different initializations can be accomplished by simply changing the file.

The AH5050 System offers the capability to create programs in executable form from BASIC or FORTH . There is no need for the lengthly process of loading the source in order to run the programs. Since files in executable form do not require compiling, the loading into memory is very fast. The "START UP" program is of this kind. From a user BASIC or FORTH program other programs in executable form can also be executed. This allows the running of programs in a very small amount of memory. Observe that in order to run the "START UP" file, if created in BASIC or FORTH, the BASIC or FORTH languages must be installed at $B000-$CFFF.

The starting address of the "START UP" file is recorded in the file, so execution could be at any place in RAM memory.

If the file "START UP" is not found the AIM-65 will print "File not found" and return to the Monitor.

More about this in the BASIC and FORTH interface sections.

3.3 FILENAMES AND TYPES

Files on disk are identified by names of up to 15 character each. File names can be made up of any ASCII characters except "@" or ":".

There are essentially three types of files supported. They are Sequential, Random and Relative. When the directory is listed the user will see file types of the ones from the chart below

SEQ	Sequential
PRG	Program
REL	Relative
USR	User

3.3.1 SEQUENTIAL FILES

Sequential files are the most commonly used. They are limited by their sequential nature, which means they must be read from beginning to end. The Editor, Assembler, Monitor and BASIC use this type of files to store programs or text.

For example:

```
=<T>        (TOP OF EDITOR)
=<L>        (LIST TEXT TO
/.           A SEQUENTIAL FILE)
```

It is a good practice if the user specifies something such as "TEX" in the file name to determine later the origin of the sequential file. For example 'TEST.TEX' and 'TEST.OBJ' identify one file as created by the Editor and the other by the Monitor. Sequential files can be divided into three catogories which the disk drive treats equally. There are, sequential files, program files and user files.

The essential difference between sequential files and program files is that program files normally are made to execute the contents of the file. Program files contain a start address in the first 2 bytes of data. Program files are created by new commands in the AH5050 software such as 'PUT' and 'SAVEB' to put binary data in a file which is intended to run as a program later on.

When a sequential or program file is opened the user must specify the direction for Read or Write.

Example:

```
10  OPEN 3,D1, "0: FILE1,S,W"
```

Where "S" means sequential and "W" means open for write.

More of this later on.

3.3.2 RANDOM FILES

Sequential files are fine when working with a continuous stream of data. The random access feature allows the user to read/write data from/to a sector on the disk. Although the random access is not really a file, it has uses of its own, espepecially when working with machine language. The relative discussed in the next section are more convenient for data handling operations.

Each diskette is divided into 683 sectors of 256 bytes each. There are 35 tracks starting with track 1 at the outside to track 35 at the center. Track 18 is used for the directory, and the DOS fills up the diskette from the center outward. Since there is more room on the outer tracks, there are more sectors there as shown below.

Example:

Track Number	Sector Range	Total Sectors
1 to 17	0 to 20	21
18 to 24	0 to 18	19
25 to 30	0 to 17	18
31 to 35	0 to 16	17

Commands are provided for reading and writing directly to any track and sector on the diskette.

Random files can be created and accessed from machine language or high level language programs. (See section 3.8 for programing instruction).

3.3 RELATIVE

Relative files are designed so the user can access any part of a file at random, relative to the beginning of a specific file. The user does not have to read sequentially through all the data in a file until the desired piece of information is found. Realtive files are structured into records and into fields within those records.

The DOS in the drive keeps track of the tracks and sectors used and allows records to overlap from one block to the next. It is able to do this because it establishes side sectors, a series of pointers for the beginning of each record. Each side sector can point to up to 120 records,and there may be 6 side sectors in a file. There can be up to 720 records in a file, and each record can be up to 254 characters long. The file could be as large as the entire diskette.

The Replace option (next section) does not erase a relative file. The file can be expanded, read, and written into. To create a relative file

the record length is given when opening the file. To access and existing relative file the record length is omitted. See the sections for Machine Language and High Level Language to interface to Relative files.

3.4 PATTERN MATCHING AND WILD CARDS

When using the AH5050 you can load, delete, type and manipulate files using what is called pattern matching. The asterisk (*) is a special character to designate this. For example if you want to load the first file on the disk starting with the letters "TE" you can type "TE*" for the file name.

Example:

```
<L> IN=U
DEV=D1    FILE=TE*
```

If only the "*" is used for the name, the last program accessed on the disk is the one loaded. If no program has yet been loaded, the first one listed in the directory is the one used. To delete all files that start with the letters "FIL" the user can enter:

```
<F1>D(IR,T(YP,N(AM,S(CRTH,F(RMT,C(LR,M(RG,I(NI,V(AL,P(UT,G(ET,R(UN S

DEV=D1      FILE=FIL*
```

The question mark "?" can be used as a wild card on the disk. The file names on the disk are compared to the file name given by the user. However all characters where there is a question mark in the given name are not checked. For instance, when a file is specified to be read from disk such as "C?T", files that match include: CAT, CUT, COT, etc.

If a program already exists on the disk it is often necessary to make some changes and store it back under the same file name. It would be inconvenient to have to erase the old version and then save the corrected version.

If the first character of a file name is preceded by character "@" the AH5050 will erase the data of the file name specified and replace it with the new data provided. For example to replace a file called "TEST" from the Editor that has been corrected the user will enter:

```
=<T>      (Top of Editor)
=<L>
 /.       (List all lines)
OUT=U     (To floppy)
DEV=D1    FILE=@TEST
```

3.5 LOGICAL FILES AND LOGICAL DEVICES

The AH5050 can support up to 10 logical files at a time. The logical file numbers can be any number from 2-14. Logical files 0,1 are reserved for loading, saving and file management. The logical File number is issued throughout the program to identify which file is being accessed. The user can open several files simultaneously to different devices, then

data is directed from/to these logical files by switching the input and output logical file.

Before explaning this in more detail, we will discuss the logical device names. The devices supported by the AH5050 operating systems are assigned a number from 0-11 (decimal). From BASIC, FORTH, and File Management the user refers to devices by name. For instance, in BASIC in order to specify "Disk 1" the user will enter "D1".

Example:

Log File	Device	File Name
10 OPEN	3, D1,	"0:TEST,S,R"

The following table shows the relationship between device names and device numbers.

TABLE

DEV#=	0	1	2	3	4	5	6	7	8	9	10	11
DEVNAME=	KB	PRI	SER	U1	SPRI	-	-	-	D1	D2	D3	D4

KB refers to the keyboard/display devices; PRI is the parallel printer output J3 on the interface board; SER is the RS232 serial port J4; U1 is a user vector at address $212 - $215; SPRI is a printer serial port off J2; the drives D1-D4 are disks supported off the disk DIN connector (J2).

Location $212 - $213 represent device "U1" user vector for input and locations $214 - $215 is device "U1" user vector for output. In both cases the carry flag will be clear when opening and set when data transfer is required. The user must preserve the registers specially in data transfers.

When a file is opened either in a High Level Language or machine code the logical file and the device are entered in a table. The table maintains up to 10 entries. The user can direct inputs or outputs of data to come from any logical file in the table and therefore from the specified device. If the device is a disk drive the logical file will determine which file from the directory the user is working with. See BASIC interface for more details.

3.6 FILE MANAGEMENT

The file management is provided so the user can manipulate files on the disk. The user can look up a directory of files on the disk, send files to the screen or printer, erase files, format a new disk, etc.

The file management is entered through the "F1" key on the AIM 65.

Example:

```
<F1> D(IR,T(YP,N(AM,S(CRTH,F(RMT,C(LR,M(RG,I(NI,V(AL,P(UT,G(ET,R(UN
```

The computer responds by displaying the menu of interactive commands in the file manager. Each command is accessed by typing the first letter of the command name. Following is a description of each of these commands.

3.6.1 DIRECTORY (D)

This command will display the directory of files on the disk. It lists number of occupied sectors, file names and types of files. The file names can be up to 15 characters long. The list can be directed to an active device.

```
<F1>D(IR,T(YP,N(AM,S(CRTH,F(RMT,C(LR,M(RG,I(NI,V(AL,P(UT,G(ET,R(UN D

DEV=D1
OUT=(CR)

0000 "SAMPLE" 01 2A       (Disk Name)
0001 "START UP"     PRG   (File Name)
0001 "IN1BASIC"     PRG   (File Name)
0640  FREE BLOCKS
```

3.6.2 TYPE (T)

This command will transfer Data on a sequential file on the diskette to a specified output device. Typing of other file types such as program (PRG) can also be done if the file name is followed by ",P,R". P stands for program and R for read. But the typing of program files is not recommended since these are binary files which contain data that the terminal will decode as control commands resulting in erratic output.

Example:

```
<F1>D(IR,T(YP,N(AM,S(CRTH,F(RMT,C(LR,M(RG,I(NI,V(AL,P(UT,G(ET,R(UN T

DEV=D1          FILE=TEST
OUT=(CR)        (TO TERMINAL)
                (TEXT DISPLAYED)
<               (BACK TO MONITOR)
```

3.6.3 RENAME (N)

This command allows the user to change the name of a file once it is in the disk directory.

Example:

```
<F1>D(IR,T(YP,N(AM,S(CRTH,F(RMT,C(LR,M(RG,I(NI,V(AL,P(UT,G(ET,R(UN N

DEV=D1          FILE=NEWNAME=OLDNAME
```

3.6.4 SCRATCH (S)

This command erases unwanted files from the disk. The blocks are made available for new information. The user can erase files one at a time or in groups by using pattern matching and/or wild cards.

Example:

```
<F1>D(IR,T(YP,N(AM,S(CRTH,F(RMT,C(LR,M(RG,I(NI,V(AL,P(UT,G(ET,R(UN S

DEV=D1        FILE=TEST
```

If the user had used "T*" for the file name all files starting with the letter T would have been scratched.

3.6.5 FORMAT (F)

This command is necessary when using a diskette for the first time. Note that when a brand new diskette is not formatted forthe first time all other commands will produce errors. The command formats the entire diskette with timing and block markers and creates the directory. The name goes in the directory as the name of the entire disk. Formating should take about one minute.

Example:

```
<F1>D(IR,T(YP,N(AM,S(CRTH,F(RMT,C(LR,M(RG,I(NI,V(AL,P(UT,G(ET,R(UN F

DEV=D1        FILE=SAMPLE
```

3.6.6 CLEAR (C)

The clear command clears out the directory of an already-formatted diskette. This is faster than re-formatting the whole disk. The name goes into the directory as the name of the entire disk.

Example:

```
<F1>D(IR,T(YP,N(AM,S(CRTH,F(RMT,C(LR,M(RG,I(NI,V(AL,P(UT,G(ET,R(UN C

DEV=D1        FILE=SAMPLE
```

3.6.7 MERGE (M)

This command allows the user to merge up to 4 files into 1 file on the same disk.

Example:

```
<F1>D(IR,T(YP,N(AM,S(CRTH,F(RMT,C(LR,M(RG,I(NI,V(AL,P(UT,G(ET,R(UN M
```

DEV=D1 FILE=NEWNAME=0:FILE1,0:FILE2,0:FILE3,0:FILE4

The "0:" must be specified between file names.

3.6.8 INITIALIZE (I)

If an error condition on the disk prevents the user from performing some operations, this command will return the disk drive to the same state as when powered up.

Example:

```
<F1>D(IR,T(YP,N(AM,S(CRTH,F(RMT,C(LR,M(RG,I(NI,V(AL,P(UT,G(ET,R(UN I

DEV=D1
<
```

3.6.9 VALIDATE (V)

When files have been repeatedly stored and scratched they may leave small gaps on the disk, a block here and a few blocks there. These blocks never get used because they are too small to be useful. The Validate command will re-organize the diskette so the user can get the most from the available space.

The command also will collect the blocks from opened files which were never properly closed. This command should never be used with a diskette that uses random files since blocks allocated in random files will be de-allocated.

Example:

```
<F1>D(IR,T(YP,N(AM,S(CRTH,F(RMT,C(LR,M(RG,I(NI,V(AL,P(UT,G(ET,R(UN V

DEV=D1

<
```

3.6.10 PUT (P)

This command is used to transfer data from the specified address in memory to a file as a program type. The first 2 bytes record the starting address in memory. The rest of the data is an image of the data in memory. This command is used to put a machine code program into a program file.

Example:

```
<F1>D(IR,T(YP,N(AM,S(CRTH,F(RMT,C(LR,M(RG,I(NI,V(AL,P(UT,G(ET,R(UN P

FROM=0300  TO=0FFF  (HEX ADDRESSES)      DEV=01  FILE=TESTER
<
```

3.6.11 GET (G)

This command is the opposite of PUT. This command will load into memory the specified file. The starting address in memory is the one specified in the first 2 bytes of the file.

Example:

<F1>D(IR,T(YP,N(AM,S(CRTH,F(RMT,C(LR,M(RG,I(NI,V(AL,P(UT,G(ET,R(UN G

DEV=D1 FILE=TESTER

3.6.11 RUN (R)

The RUN command will load and run any file of the type Program (PRG). These type ot files are stored on disk by the PUT command from the monitor and the SAVEB commands in the BASIC or FORTH languages.
RUN is a powerful command since it can be used to run Editor, Assemblers, or user application programs. Note that programs that will run can be created in BASIC or FORTH without the need for recompiling. This is an interactive command in which the user is prompted with the device and file name. The AH5050 also provides the capability for the user to specify in software the file name to be run. See BASIC, FORTH and machine language interfaces.

Example:

<F1>D(IR,T(YP,N(AM,S(CRTH,F(RMT,C(LR,M(RG,I(NI,V(AL,P(UT,G(ET,R(UN R

DEV=D1 FILE=TESTER

3.7 EXISTING SOFTWARE INTERFACE

Reading or writing files from the EDITOR, ASSEMBLER, MONITOR and BASIC on the AIM 65 is done through the user vector. The user must type "U" after the "OUT" or "IN=" prompts. This directs input/outputs through the user vectors, which the software in the AH5050 uses to read/write to a file in the floppy disk.

Since the AH5050 uses locations $0212-$025F for variables, the user must not use locations lower than $0260 when the AH5050 is installed.

The following sections explain each of these interfaces.

3.7.1 MONITOR INTERFACE

Besides the file manager access from the Monitor, the user can use the regular LOAD (U) and DUMP (D) commands to read or write object files using the format given by the AIM 65. The object files have more overhead than the binary files since they contain data for the number of bytes in the record, address and check sum. Object files are used by PROM programmers or other devices which may need to load several sections of code into different portions of memory.

a) Monitor write example:

```
<D>
FROM = 500 TO=5FF
OUT=U                          (OUTPUT TO AH5050)
DEV=D1    FILE=DEMO
MORE?Y                         (DUMP ADDITIOANL MEMORY ...)
FROM=700 TO=750          (UNDER SAME FILE)
MORE?N
```

b) Monitor read example:

```
<L>IN=U                        (LOAD TO MEMORY)
DEV=D1    FILE=DEMO
```

This will load object data into memory. The address to be loaded into memory is included on each of the records.

It is a good practice to add to the file name an extension such as ".OBJ" that mentions where the file was created from. The drawback is that the user must specify it also when reading the file unless something like "DEMO*" is used.

3.7.2 EDITOR INTERFACE

The user can Read or Write text files through the Editor by using the 'Read' and 'List' commands. The Editor buffer start address should not be set under $260, in order to avoid conflict with the AH5050 variables.

a) Editor write example:

```
=<T>           (GO TO TOP OF EDITOR)
=<L>           (LIST ALL LINES)
/.
OUT=U          (OUTPUT TO FLOPPY)
DEV=D1  FILE=DEMO2
END                (PROMPT FROM EDITOR)
```

b) Editor read example:

```
<E>
EDITOR
FROM=300   TO=1000   (EDITOR BUFFER)
IN=U
DEV=D1  FILE=DEMO2
```

3.7.3 ASSEMBLER INTERFACE

In order to use the ASSEMBLER with the AH5050, the user must have the Assembler installed at $B000. The Assembler at $B000 is the same as the assembler offered by Rockell International at $D000 but relocated to $B000 so the AH5050 Operating System can be installed at $D000. This

relocated assembler is offered by ABM. If RAM is available at $B000 the assembler could be loaded from disk.

The source input (text file) can be entered from disk or from text in memory. Only one of the two outputs of the Assembler (Object or Listing) can be directed to disk at a time. The symbol table for the Assembler should not be set lower than $260.

The AH5050 software permits the user to assemble more than one text file by linking them with the '.FILE' directive.

When assembling a single text file, the user must terminate the source program wit the '.END' assembler directive with no file-name appended. This enables the second pass and closes files.

a) Single-file assembly example:

```
PORT=$4000          (FILE-NAME IN DIRECTORY=
*=$600              'DEMOA')
LDA #$45
STA PORT
.END                (END ASSEMBLER DIRECTIVE)
```

On the other hand, when a text file is too long to fit in the text Editor buffer, the file can be divided into smaller text files and stored in disk. In this manner, larger programs can be created. If all these text files are to be assembled together, the '.FILE' and '.END' directives with the next file-name appended are used. These directives tell the software which file to open next and when to close the files. The following example shows hows to assemble three text files linked by the '.FILE' directive.

b) Multi-file assembly example:

```
PORT=$4000          (FILE-NAME IN DIRECTORY=
*=$600              'DEMOA')
LDA #$45
STA PORT
.FILE DEMOB         (CONTINUE ON DEMOB)


LDA PORT            (FILE-NAME IN DIRECTORY=
EOR #$FF            'DEMOB')

STA PORT
.FILE DEMOC         (CONTINUE ON DEMOC)
```

```
LDX #05             (FILE;NAME IN DIRECTORY=
LOOP JSR OUTPUT     'DEMOC')
DEX
BNE LOOP
BRK
.END DEMOA          (FILE-NAME OF FIRST FILE
```

i.e. 'DEMOA' MUST BE GIVEN)

When assembling either a single text file or multiple test files from disk, the user has the option to direct the object code to memory or disk.

c) The following example will assemble a multi-text file and put the object back to the disk.

<5>

ASSEMBLER
FROM=800 TO=900 (SYMBOL TABLE)

IN=U (INPUT FIRST FILE
OF LINKED FILES)
DEV=D1 FILE=DEMOA
LIST?Y
LIST-OUT=(CR) (LIST TO TERMINAL)
OBJECT?Y
OBJECT-OUT=U
DEV=D1 FILE=DEMO-OBJ (TO FLOPPY)

3.7.4 BASIC LANGUAGE INTERFACE

In order to use BASIC with the AH5050, the user must have the BASIC language installed at $B000-$CFFF. If RAM is available at those locations the BASIC Language could be loaded from disk.

If a BASIC program is going to be written for the first time, file 'INIBASIC' must be run first. This asures proper set up of the vectors in the BASIC so the user program starts above $260 and there is no conflict with the variables in the AH5050.

Example:

<F1> D(IR,T(YP,N(AM,S(CRTH,F(RMT,C(CLR,M(RG,I(NI,V(AL,P(PUT,G(ET,R(UN
R

DEV=D1 FILE=INIBAS

After this initialization the user is prompted for the memory size in case some memory at the top should be reserved for other purposes. If the user responds with <CR>, the available memory is calculated.

If the user escapes from the BASIC program to the Monitor, reentering BASIC can be accommplished by typing Key 6.

Following is a Brief description of added commands to the AIM 65 BASIC language. Some examples are also given to explain the potential of using sequential, relative and random files. The loading of these files into BASIC takes time because the BASIC has to tokenize the incoming data.

3.7.4.1 EXISTING LOAD/SAVE

The existing 'Load' and 'Save' commands in the BASIC are used to read and write source BASIC files.

a) Saving source BASIC program example:

```
SAVE (CR)          (SAVE COMMAND)
OUT=U              (OUTPUT TO FLOPPY DISK)
DEV=D1  FILE=DEMO
```

b) Loading source BASIC program example:

```
LOAD (CR)          (LOAD COMMAND)
IN=U
DEV=D1  FILE=DEMO (AIM will load File from disk and display it on the terminal)
```

3.7.4.2 LOADB/SAVEB

These new commands are used to read and write the already "tokenized" program into the disk. Files are loaded/saved faster since there is no compilation.

a) Saving binary BASIC program example:

```
SAVEB (CR)         (SAVE BINARY)
OUT=U              (TO DISK)
DEV=D1  FILE=DEMO
```

b) Loading binary BASIC program example:

```
LOADB
IN=U
DEV=D1  FILE=DEMO
```

The LOADB command will replace any existing BASIC file in memory. These binary file programs are already in executable form. So the user can run these type of files from the monitor using the run command.

Example:

```
    <F1>  D(IR,T(YP,N(AM,S(CRTH,F(RMT,C(CLR,M(RG,I(NI,V(AL,P(PUT,G(ET,R(UN
R
    DEV=D1  FILE=DEMO
```

If the file name for the user BASIC program saved with the command 'SAVEB' is 'STARTUP', the file will run automatically when the microprocessor runs locations $D000. The file will run automatically from power up or reset if the changes to the monitor are made as explained in Section 3.2. Note that the BASIC language must be installed in $B000-$CFFF.

The new LOADB/SAVEB commands can be normally used instead of the existing

LOAD/SAVE to develop the user programs. But since the files are already in executable form, they will not allow relocation of BASIC programs when loaded by LOADB. If relocation is needed the user can use SAVE followed by a LOAD. Relocating may be necessary when the memory is used for other purposes also. This is very uncommon.

3.7.4.3 NEW OPEN

In order to Read or Write data from devices the user must open a logical file specifying the device and the file name. Up to 10 logical files can be open simultaneously.

Example:

```
10  OPEN 2,D1,"0:FILE1,S,W"
20  OPEN 3,D1,"0:FILE2,S,W"
30  PRINT#2, "DATA TO FILE 1"
40  PRINT#3, "DATA TO FILE 2"
50  CLOSE 2: CLOSE 3
```

Lines 10 and 20 open two logical files (2,3) to device D1 (8) as Sequential files and to be Written into. The logical files number can be from 2 to 14. Logical file 0,1 are used for the Load and Save commands.

When opening a file a replace option could be used if the file name starts with "@" sign.

Example:

```
10  OPEN 2,D1,"0:@FILE1,S,W"
```

This will delete the old FILE1 and replace it with data from the new FILE1.

3.7.4.4 NEW PRINT#

The PRINT# is used exactly as the existing PRINT command except the logical number assigned in the OPEN command must be supplied. The software will open the logical channel momentarily to output the data specified with the PRINT# statement and close the logical channel so any following outputs will go to the terminal.

Example:

```
05  A=2
10  OPEN 1,D1,"DATA TO FILE1,S,W,"
20  PRINT#A, "DATA TO FILE1"
30  PRINT "DATA TO TERMINAL"
40  PRINT#A, "MORE DATA TO FILE1"
50  CLOSE A
```

The PRINT# command in this example works exactly like the PRINT statement, except that output is re-directed to the disk drive. All the

formatting capabilities and rules of the PRINT statement for punctuation and data types, apply here too. Therefore be careful when putting data into your files.

FORMAT FOR WRITING TO FILE WITH PRINT#:

```
PRINT#file#,data list
```

The file# is the one used in the OPEN statement when the file was created.

The data list is the same as the regular PRINT statement-a list of variables and/or text inside quote marks. However, you must be especially careful when writing data so it would be as easy as possible to read back later.

When using the PRINT# statement, if you use commas (,) to separate items on the line, the items will be separated by blank spaces, as if they were being formated for the screen. Semicolons (;) do not result in any extra spaces.

In order to more fully understand what is happening, here is a diagram of a sequential file created by the statement OPEN 5,D1,"0:TEST,S,W":

String data entering the file goes in byte by byte, including spaces.

For instance, let's set up some variables with the statement A$= "HELLO"; B$= "THERE". Here is a picture of a file after the statement PRINT#5,A$;B$ is executed.

```
    H  E  L  L  O  T  H  E  R  E  CR eof
char 1  2  3  4  5  6  7  8  9  10 11 12
```

CR stands for the CHR$ code of 13 (decimal), the carriage return, which is PRINTed at the end of every PRINT or PRINT# statement unless there is a comma or semicolon at the end of the line.

NOTE: Do not leave a space in PRINT#, and do not try to abbreviate the command to ?#.

3.7.4.5 NEW INPUT#

The INPUT# is used exactly as the existing INPUT command except inputs come from the logical file specified in the OPEN statement.

The software will open the specified logical file momentarely, read data into the variables and close the inputs so following data will come from the keyboard.

Example:

```
05  A=2
10  OPEN A,D1,"0:FILE1,S,R"
20  INPUT# A, D$
30  CLOSE A
```

FORMAT FOR INPUT# STATEMENT:

INPUT#file#,variable list

When using the INPUT# to read data, there should be a way to tell whether the input is supposed to be one long string or made up of multiple strings. The file requires string separators. Characters used as separators include the CR, a comma or a semicolon. The CR can be added easily by just using one variable per line in the PRINT# statement, since the system puts one there automatically. The statement PRINT# 5,A$: PRINT# 5,B$ puts a CR after every variable being written, providing the proper separation for a statement like INPUT#5, A$,B$. Also a line like Z$= ",": PRINT#5,A$;Z$;B$ can be used, which will also conserve space .The example file after such a line looks like this:

	H	E	L	L	O	,	T	H	E	R	E	CR	eof
char	1	2	3	4	5	6	7	8	9	10	11	12	13

Putting commas between variables results in extra space on the disk being used. A statement like PRINT#5, A$,B$ makes a file that looks like:

	H	E	L	L	O		T	H	E	R	E	CR	eof	
char	1	2	3	4	5	6	7	8	9	10	11	12	13	14

You can see that much of the space in the file is wasted. Also looks like one single variable when you try to read it.

The moral of all this is: take care when using PRINT# to format data and separate variables so it will be in order for reading back by an INPUT statement.

Numeric data written in the file take the form of strings, as if the STR$ function had been performed on them. The first character will be a blank space if the number is positive, and a minus sign (-) if the number is negative. Then comes the number, and the last character is a SPACE character. This format provides enough information for the INPUT# statement to read them in as separate numbers if several are written with no special separators.

Here is a picture of the file after the statement PRINT#5, 1;3;5;7 is performed:

		1			3			5			7		CR	eof	
char	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15

3.7.4.6 NEW CHOUT

The PRINT# can be used to output data to any logical file. But after the output the software resets the logical channel so any following PRINTs without the "#" sign will be directed to the terminal. Since the PRINT# statement must open and close the logical channel for every print statement, this imposes an overhead when a lot of data is sent to that

logical file. The CHOUT command sets the logical channel so PRINT statements will send data to the logical file. It is the users responsibility to close the logical channel when done, so following outputs will be sent to the terminal. There are two ways to reset the logical channel, one by typing "CHOUT 255" and the other by the PRINT# command.

Example:

```
10  OPEN 5,D1,"0:FILE1,S,W"
20  CHOUT 5
30  FOR I=1 TO 5
40  PRINT "DATA TO FILE #":I: ","
50  NEXT
60  PRINT#5, "END" : REM CLOSE CHANNEL
70  PRINT "THIS GOES TO TERMINAL"
80  CLOSE 5    : REM CLOSE FILE
```

3.7.4.7 NEW CHIN

This is the opposite of CHOUT and is used to open the logical channel only once so normal INPUT statement can be used. This lowers the overhead when a lot of data is to be read.

Example:

```
10  OPEN 3,D1,"0:FILE1,S,R"
20  CHIN 3
30  FOR I=1 TO 5
40  INPUT A$(I)
50  NEXT
60  CHIN 255   : REM CLOSE CHANNEL
70  CLOSE 3    : REM CLOSE FILE
```

To close the input channel the user can do 'CHIN 255' or do a last INPUT# of data.

3.7.4.8 NEW CMD

This command is used to send instructions to the intelligent drive. The command is normally used for Random and Relative files to set the position for the Track, Sector and Pointer where the BASIC will read or write data. For now we will give the general format. For more detail see Random and Relative files.

Example:

```
10 OPEN 5,D1,"#" :REM RANDOM FILE.
20 CMD D1, "B-R:" 5;0;18;2

      BLOCK-READ LOG-FILE TRACK SECTOR
```

3.7.4.9 NEW STATUS

The status command is intended to read the information from a disk drive. The user can determine if a record exist, if it is a non-existing file, or any error in general. Look at Appendix B for all possible errors. If an error occurs and the user does not read the status with this command the drive LED will blink until the user executes the STATUS command. The STATUS command will read an error number, descri

Example:

```
10  STATUS D1,ER,D$,TK,SE
20  PRINT ER,D$,TK,SE
```

Where ER is an integer error number, D$ is a string description, TK is the track integer and SE is the sector integer.

The user can print the variable contents after STATUS if desired. The STATUS command does not print them automatically because under software control this is not always desired, for example when a record does not exist.

3.7.4.10 NEW CLOSE

It is very important that the user CLOSEs files once he is finish using them. Closing the file causes the DOS to properly allocate blocks and to finish the entry in the directory. If the user does not CLOSE the file, all the data will be lost!

Example:

```
50 CLOSE 3  (Close logical file 3)
```

3.8 MACHINE LANGUAGE SUBROUTINES

This section illustrates how to read and write data through the AH5050 subsystem using assembly language programs. See Appendix C for a list of all the subroutines and their addresses.

There are different levels of entry. For example the user may want just to run a file without concerning himself with the opening, loading, and closing before running the file. On the other hand the user may want to open many files and work with all files at the same time.

Also some subroutines will ask questions on the terminal about the device and file name. These are called interactive subroutines. Other subroutines assume pointers and filenames are already set under software control so no questions are asked.

3.8.1 FUNCTION NAME: RUN

Arguments: none
Registers affected: .A,.X,.Y

This routine opens, loads, closes and runs the program file given by the user. It prompts the user for DEVICE and FILE name before attempting to open any file. The only devices allowed are 'D1, D2, D3, D4' since the subroutine has been optimized for speed to read from disk. This routine is the one executed by the "RUN" command in the File Manager.

Example:

```
JSR RUN    ;Run file given by user
JMP MONI   ;Back to Monitor if does
           ;not exitst
```

3.8.2 FUNCTION NAME: GET

Arguments: None.
Registers affected: .A,.X,.Y

This routine does the same as the 'RUN' function except that it is a subroutine and it does not run the program, instead it always returns to the user. This routine is used by the 'GET' command in the file manager.

Example:

```
JSR GET       ;Load file program only.
JMP MONITOR   ;Back to AIM Monitor.
```

3.8.3 FUNCTION NAME: RUNSET

Arguments: A=Log Fil, X=DEV, FNLEN=FIL LEN, FNADR=FILENAME PTR.

Registers affected: A,X,Y

This routine is similar to 'RUN' routine except it does not ask questions on the terminal. Instead parameters are passed to the routine and the file name length and pointer are expected to be set at variables FNLEN and FNADR previous to calling the subroutine. The pointer and length can be set by the SETNAME routine. To call SETNAME the Accumulator contains the file length and X,Y the address where the file name is.

Example:

This is an implementation of the power up routine.

```
LDA #NAME2-NAME     ;Length of file name
LDX #<NAME          ;Address of file name
LDY #>NAME
JSR SETNAM          ;Set FNLEN & FNADR Variables.

LDA #00             ;Use logical file=0
```

```
      LDX #08              ;DEVICE #8 (D1)
      JMP RUNSET

NAME .BYT 'STARTUP,P,R'   ;File name
NAME2
```

3.8.4 FUNCTION NAME: GETSET

Arguments: A=Log Fil, X=D3V, FNLEN=FILE LEN, FNADR=FILE NAME PTR
Registers affected: A,X,Y

This is basically the same routine as RUNSET except the program is loaded into memory but the code is not run.

3.8.5 FUNCTION NAME: WHEREI

Arguments: None
Registers: A,X,Y

The previous functions are intended to run one program from the disk and not to manipulate file data. The WHEREI function is used to open files for input so data can be input from the device specified through the 'INALL' routine. The WHEREI function will ask questions on the terminal regarding the device and file names requested.

Example:

```
      JSR WHEREI      ;prompt for input device & file desired.
        .
        .
        .
        .
        .
```

3.8.6 FUNCTION NAME: WHEREO

Arguments: None
Registers: A,X,Y,

This function is the counterpart of 'WHEREI'. WHEREO opens a file for output so data can be output to the specified device through the 'OUTALL' routine.

3.8.7 FUNCTION NAME: OPEN

Arguments: A=log FILE, X=DEV, FNLEN=FILE LEN, FNADR=NAME ADR PTR
Registers Affected: A,X,Y

WHEREI and WHEREO are fine if the user wants the operator to enter the device and file name from the keyboard. Sometimes the user wants to specify the device and file name under software control, with no operator intervention. In this case the user must use the OPEN function.

The Open routine opens a file and logs-on the logical file and device into a table. Parameters for the logical file and device are passed to the routine. Also the file name pointer and file name length must be set previous to calling this routine. After a logical file is opened routines such as 'CHIN', 'CHOUT' set the I/O channel flags so the routines 'INALL' and 'OUTALL' can direct the data to the proper device.

Example:

```
LDA #NAME2-NAME    ;Length of file name.
LDX #<NAME         ;Address of file name.
LDY #>NAME
JSR SETNAM         ;Set FNLEN, FNADR

LDA #05            ;Logical File (2-14)
LDX #08            ;Dev #
JSR OPEN           ;Logon logical file,dev and open device.
  .
  .
  .
```

The subroutine SETNAME will set the length of the file given in the Accumulator into FNLEN and the address given by X & Y into FNADR. Then the user opens the file where the Accumulator specifies the logical file and X the device number from the allowed devices in section 3.5.

3.8.8 FUNCTION NAME: INALL

Arguments: None
Registers affected: Acc

This function reads a byte into the ACCumulator register from the open logical channel.

Example:

```
     LDY #$00          ;Prepare counter
LOOP JSR INALL         ;Input from Keyboard defaulted log channel.
     STA DATA,Y        ;Store
     INY
     CMP #CR           ;terminate with CR
     BNE LOOP
```

3.8.9 FUNCTION NAME: OUTALL

Arguments: Acc=data
Registers Affected: None

This function outputs a byte from the ACCumulator register to the open logical channel which in this case is a centronics printer. Notice that no file name is required since the output to the printer does not check for it.

Example:

```
     LDA #03        ;Logical file 03
     LDX #01        ;device=01 (Centronics PRInter)
     JSR OPEN       ;OPEN DEV=01

     LDA #03        ;Set output channel=log file 03
     JSR CHOUT

     LDY #00
LOOP LDA DATA,Y     ;Data from memory
     JSR OUTALL     ;To printer
     CPY #05

     :
     :
```

3.8.10 FUNCTION NAME: CLOSE

Arguments: Acc=Logical file
Registers Affected: A,X,Y

This function closes the logical file specified in the Accumulator. This number is the same used when the file was opened using the OPEN command.

Example:

```
     LDA #03
     JSR CLOSE
       .
       .
       .
```

3.8.11 FUNCTION NAME: CLOALL

Arguments: None
Registers Affected: A,X,Y

This routine calles CLRCHN and closes all logical files open. Therefore there is no need to load into the Accumulator the logical file and call CLOSE for each file open.

Example:

```
     JSR CLOALL
     :
     :
```

3.8.12 FUNCTION NAME: CLRCHN

Arguements: None
Registers Affected: A,X

This routine closes the Input/Output logical channels so they default to the keybord and display terminal. This routine does not affect the logical files and devices in the logical table. Therefore CHIN, CHOUT can be used to switch the input or output channels to any of the logical files that are still open.

Example:

```
JSR CLRCHN     ;Close I/O Channels
```

3.8.13 FUNCTION NAME: CHIN

Arguments: ACC=Log file.
Registers Affected: A,X

This routine opens the input channel to the device which matches the logical file specified in the Accumulator. This routine is used after logical files are opened and before the 'INALL' routine is called.

Example:

```
LDA #05        ;Prepare to input from
JSR CHIN       ;Logical file #05
:
```

3.8.14 FUNCTION NAME: CHOUT

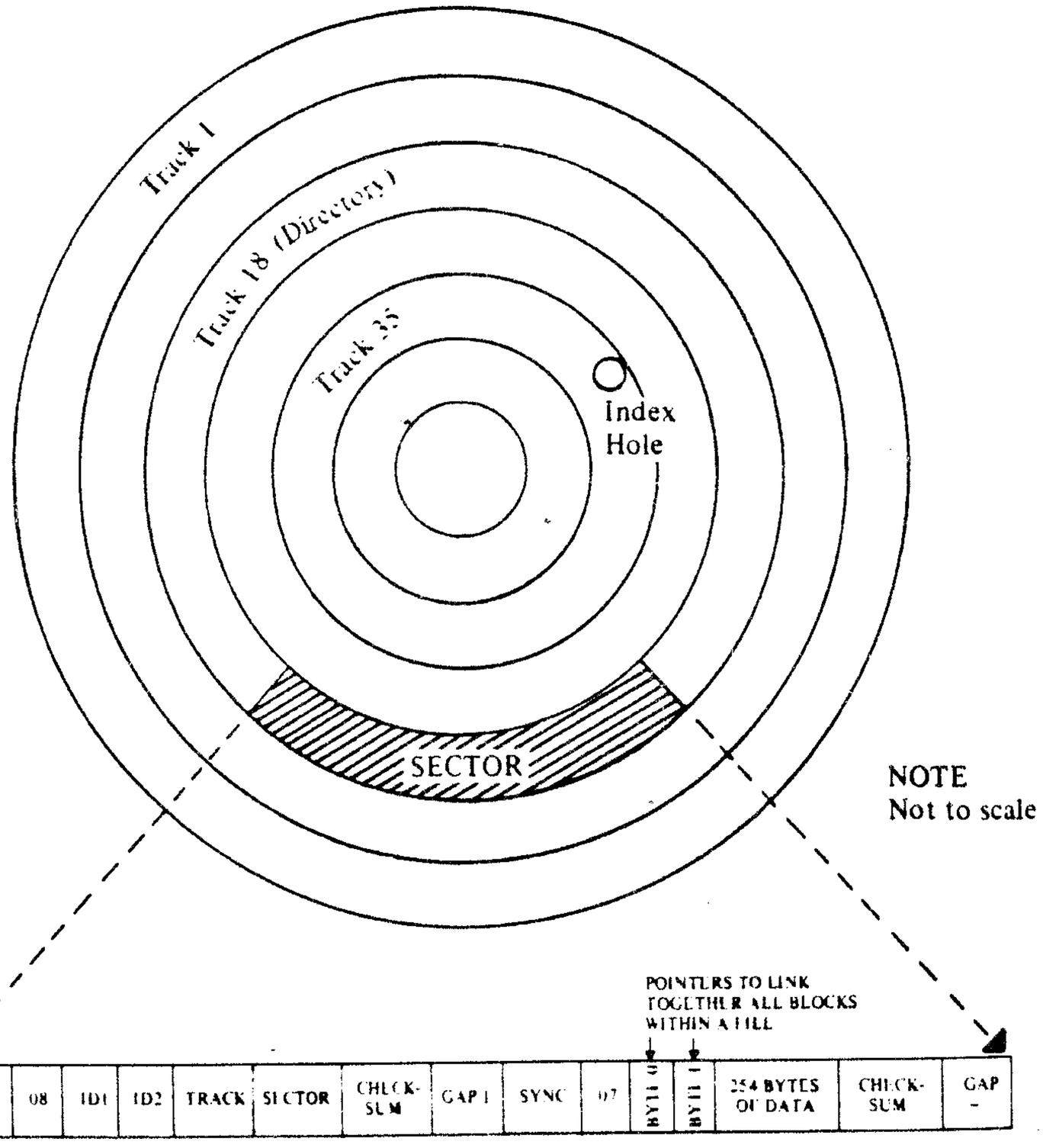
Arguments: ACC=Log File
Registers Affected: A,X

This is the counter part of CHIN for the output channel,

Example:

```
LDA #06     ;Prepare for output from
JSR CHOUT   ;Logical file #06
:
```

A.1 DISK FORMAT



Format: Expanded View of a Single Sector

A.4 DIRECTORY HEADER

Track 18. Sector 0.		
BYTE	CONTENTS	DEFINITION
144-161		Disk name padded with shifted spaces.
162-163		Disk ID.
164	160	Shifted space.
165-166	50.65	ASCII representation for 2A which is DOS version and format type.
166-167	160	Shifted spaces.
177-255	0	Nulls, not used.
Note: ASCII characters may appear in locations 180 thru 191 on some diskettes.		

A.5 DIRECTORY FORMAT

Track 18. Sector 1	
BYTE	DEFINITION
0-1	Track and sector of next directory block
2-31	*File entry 1
34-63	*File entry 2
66-95	*File entry 3
98-127	*File entry 4
130-159	*File entry 5
162-191	*File entry 6
194-123	*File entry 7
226-255	*File entry 8

A.6 DIRECTORY ENTRY

BYTE	CONTENTS	DEFINITION
0	128+type	File type OR'ed with $80 to indicate properly closed file. TYPES: 0 = DELeted 1 = SEQential 2 = PROGram 3 = USER 4 = RELative
1-2		Track and sector of 1st data block.
3-18		File name padded with shifted spaces.
19-20		Relative file only: track and sector for first side sector block.
21		Relative file only: Record size.
22-25		Unused.
26-27		Track and sector of replacement file when OPEN@ is in effect.
28-29		Number of blocks in file: low byte, high byte.

A.7 SEQUENTIAL FILE FORMAT

BYTE	DEFINITION
0-1	Track and sector of next sequential data block.
2-256	254 bytes of data with carriage return as record terminators.

A.8 PROGRAM FILE FORMAT

BYTE	DEFINITION
0,1	Track and sector of next block in program file.
2-256	254 bytes of program info stored in CBM memory format (with key words tokenized). End of file is marked by three zero bytes.

A.9 RELATIVE FILE FORMAT

DATA BLOCK	
BYTE	DEFINITION
0,1	Track and sector of next data block.
2-256	254 bytes of data. Empty records contain FF (all binary ones) in the first byte followed by 00 (binary all zeros) to the end of the record. Partially filled records are padded with nulls (00).

SIDE SECTOR BLOCK	
BYTE	DEFINITION
0-1	Track and sector of next side sector block.
2	Side sector number (0-5).
3	Record length
4-5	Track and sector of first side sector (number 0)
6-7	Track and sector of second side sector (number 1)
8-9	Track and sector of third side sector (number 2)
10-11	Track and sector of fourth side sector (number 3)
12-13	Track and sector of fifth side sector (number 4)
14-15	Track and sector of sixth side sector (number 5)
16-256	Track and sector pointers to 120 data blocks

APPENDIX B
ERROR MESSAGES

0 OK, no error exists
1 Files scratched response. Not an error condition.
2-5 No error condition given by drive.
6 Over 10 files
7 Log file opened
8 No file opened
9 1st byte eof
10 DEV not present
11-19 No error condition
20 Block header not found on disk
21 Sync character not found.
22 Data block not present.
23 Checksum error in data.
24 Byte decoding error
25 Write-verify error.
26 Attempt to write with write protect on.
27 Checksum error in header
28 Data extends into next block.
29 Disk ID mismatch.
30 General syntax error.
31 Invalid command
32 Long line
33 Invalid filename
34 no file given
39 Command file not found.
50 Record not Present.
51 Overflow in record.
52 File too large.
60 File open for write.
61 File not open.
62 File not found.
63 File exists.
64 File type mismatch
65 No block.
66 Illegal track or sector.
67 Illegal system track or sector
70 No channels available
71 Directory error.
72 Disk full or directory full.
73 Power up message, or write attempt with DOS Mismatch
74 Drive not ready. (8050 only)

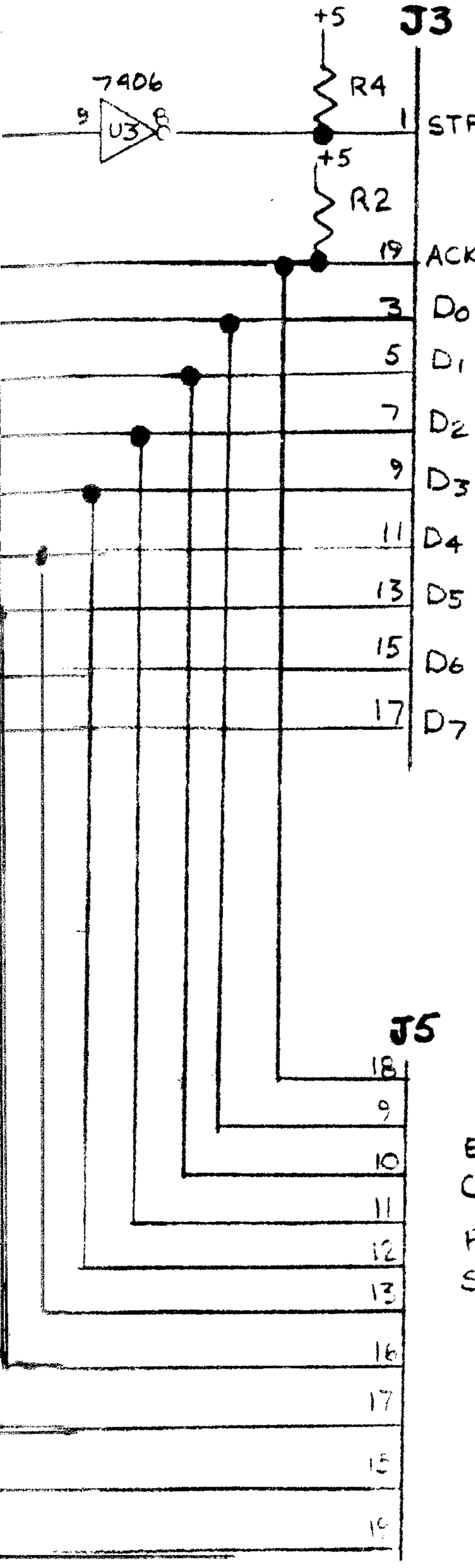
```
E848 WHEREI --- open input file: ask dev& name
E871 WHEREO --- open outside file; ask dev & name
DFEB SETNAM --- set variables FNLEN & FNADR used by OPEN
DFEE OPEN   --- open file; A=log, X=dev, FNLEN=lenname,
                FNADR=nameprt (name,type,r/w)

E993 INALL  --- input one byte  from logical file
E9BC OUTALL --- output one byte from logical file

DFF1 CLOSE  --- close logical file; Acc=log file
DFF4 CLOALL --- close all log files

DFF7 CHIN   --- open logical channel for input; Acc=log file
DFFA CHOUT  --- open logical channel for output; Acc=log file
DFFD CLRCHN --- close both I/O channels & default to terminal
```

Additional commands which are transparent to the user are included into the AH5050 floppy subsystem. They provide functions for read/write data from the Editor, Assembler, Monitor, BASIC and FORTH.



AIM 65
EXTENDED APPLICATION
CONNECTOR

PLUS ALL OTHER
SIGNALS FROM J1

J1 - AIM 65 APPLICATION CONNECTOR

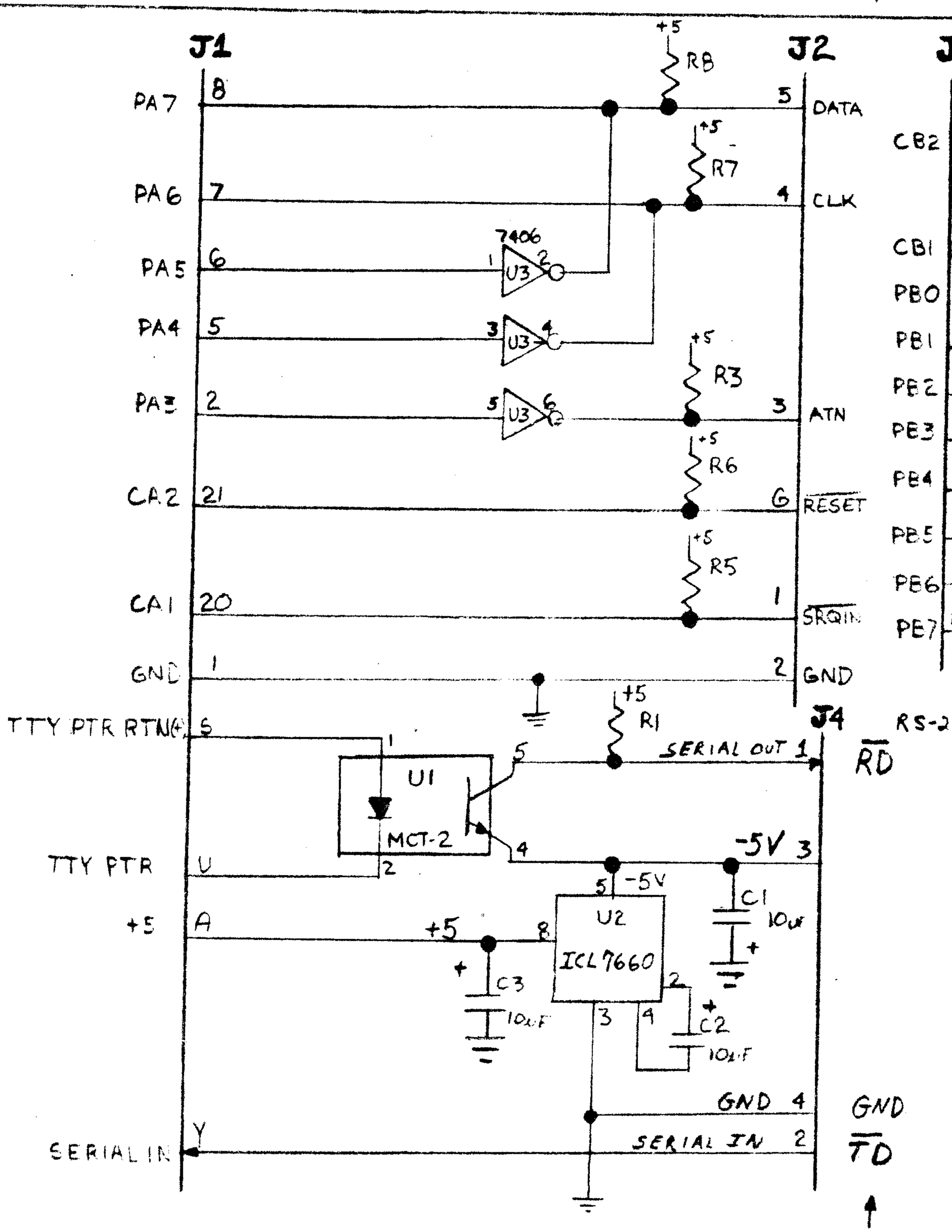
J2 - DISK INTERFACE

J3 - PRINTER INTERFACE

J4 - RS-232C INTERFACE

J5 - AIM 65 EXTENDED APPLICATION CONNECTOR

DR M SMITH	ABM Systems			
CHK	DISK INTERFACE AH5051 MODULE			
ENG A R	DATE 7-8-83	NO		A
APP	REL	B		SHT 1 OF 1



3) ALL RESISTORS ARE 3.3K 1/4 WATT
2) ALL CAPACITORS ARE 10UF
1) ALL SIGNALS FROM J1 CONNECT TO J5

NOTES: UNLESS OTHERWISE SPECIFIED

APPENDIX C

COMMAND SUMMARY

C.1 FILE MANAGER COMMANDS

D(IR	--- list directory
T(YP	--- type contents of file
N(AM	--- rename a file
S(CRTH	--- scratch a file from directory
F(RMT	--- format entire new diskette
C(LR	--- clear directory from all files
M(RG	--- merge up to 4 files
I(NI	--- initialize drive as power up
V(AL	--- validate diskette, collect unused space
P(UT	--- put a binary file into disk
G(ET	--- get binary file from disk into memory
R(UN	--- run binary disk file

C.2 NEW BASIC COMMANDS

SAVEB	--- save binary basic program
LOADB	--- load binary basic program
SYS	--- execute machine language program
OPEN	--- open logical file. sequential, prog, relative, user
CLOSE	--- close logical file
PRINT#	--- print data to logical file
INPUT#	--- input data from logical file
CHOUT	--- open logical channel for output
CHIN	--- open logical channel for input
EXEC	--- execute binary basic file
CMD	--- send commands to drive, for random relative files
STATUS	--- read error status from drive

C.3 NEW FORTH COMMANDS

SAVEB	--- save binary forth program
OPEN"	--- open logical file
CLOSE	--- close logical file
CHOUT	--- set logical file for output channel
CHIN	--- set logical file for input channel
EXE"	--- execute binary forth file.

C.4 ASSEMBLY SUBROUTINE COMMANDS

ADDR	COMMAND	DESCRIPTION
DFDC	RUN	--- open, load, close & run program file; ask dev, name.
DFDF	GET	--- open, load, close program file; ask dev, name
DFE2	RUNSET	--- open, load, close & run program file; set ACC=log, X=dev ,FNLEN & FNADR
DFE5	GETSET	--- open, load, close program file; set like RUNSET