SOFTWARE SPECIFICATION

FOR THE

HEXBUS FLOPPY DISK SYSTEM

Consumer Group
Mail Station 5890
2301 N. University
Lubbock, Texas 79414

TEXAS INSTRUMENTS
INCORPORATED

Date: August 2, 1983
Version 1.0

TABLE of CONTENTS

Paragraph Title

LIST of TABLES

LIST of FIGURES

SECTION 1

INTRODUCTION

1.1 General Description

This document is intended to give an operational specification of the ROM based software for the TI HEXBUS Floppy Disk System. This ROM based software is also called the disk Device Service Routine (DSR).

This Disk System will be compatible with the TI-99/4 Expansion System and will support the existing File Management System.

1.2 Hardware Description

The hardware design features a NEC 765 Floppy Disk Controller with 4K RAM, 16K ROM, and a 9995 microprocessor. The disk controller will be designed to support any 5.25" floppy disk drive, with a minimum step time of 20 milliseconds. More hardware details can be found in the HEXBUS Floppy Disk System Product Specification.

1.3 Software Description

The disk peripheral software is ROM-based. The ROM code is executed by the TMS 9995 located in the HEXBUS Disk System unit. The DSR software supports single sided single density, double sided double density, 40 or 96 Tracks Per Inch (TPI). (Note: 48 TPI actually has 40 tracks per side and 96 TPI actually has 77 tracks per side.) The HEXBUS Disk System DSR allows a maximum of 4 files to be open at the same time. Access to the disk peripheral software is facilitated through the file management system, as specified in the File Management Specification for the TI-99/4 Home Computer.

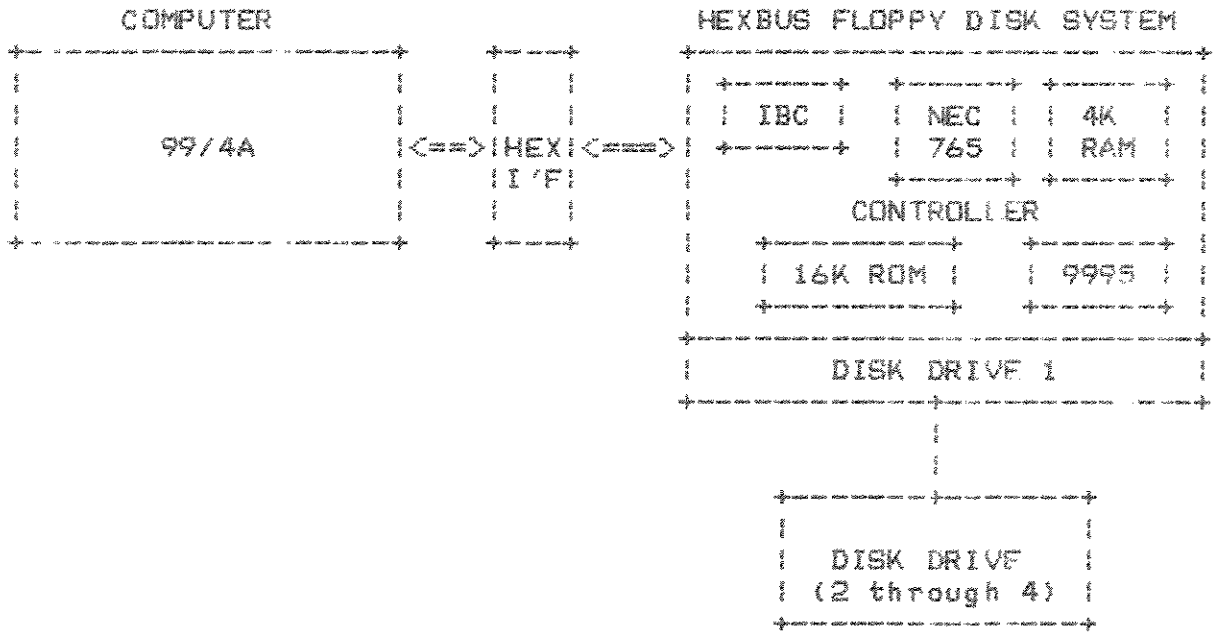


Figure 1-1 Block Diagram of the HEXBUS Disk System

SECTION 2

APPLICABLE DOCUMENTS

File Management Specification for the TI-99/4 Home Computer
(Version 2.5, Revised 25 February 1983)

Home Computer BASIC Language Specification
(Revision 4.1, 12 April 1979)

Home Computer Disk Peripheral Hardware Specification

Functional Specification for the 99/4 Disk Peripheral
(Version 3.0, Revised 28 March 1983)

GPL Interface Specification for the 99/4 Disk Peripheral
(Version 2.0, Revised 28 March 1983)

SECTION 3

SOFTWARE DESIGN CONSIDERATIONS

The disk peripheral software has been designed to support all options facilitated by the file management system. The supported options include:

* Sequential and Relative record (random access) files
* Fixed and Variable length records
* INTERNAL and DISPLAY file types
* OUTPUT, INPUT, UPDATE, and APPEND access modes
* Program LOAD and SAVE functions

Aside from these functions, a separate disk utility cartridge, the Disk Manager, supports the following utility programs:

* Single disk backup
* Disk-to-disk copy/backup
* Disk initialization/formatting
* Disk catalog
* Disk rename
* Disk tests
* File copy
* File rename
* File protection status
* Selective file deletion

The above mentioned utilities are accessable through menus, similar to the standard program selection menu featured by the TI-99/4 console. The utilities are currently available in three languages, ENGLISH, FRENCH, and GERMAN. Provisions are taken for

future expansion of these language capabilities.

More information about the Disk Manager is provided in the Texas Instruments Home Computer Disk Memory System manual.

SECTION 4

FUNCTIONAL OVERVIEW

This section will provide an overview of the functions of the disk peripheral software in each of the implementation levels. Each level uses the features implemented in a lower level, and builds new features with the building blocks provided by the lower levels.

4.1 Level 1 Features

* Disk formatting functions
* Record read/write functions
* Soft error correction functions
* Communication with the NEC 765 chip

Level 1 is the only level that must be familiar with the hardware, thus this implementation level allows for abstraction from the disk hardware. Every higher level will only know the disk as a linear storage device, addressed by physical record-number, disk unit-number, and read or write operation.

All the higher levels have been designed to be independent of the actual physical disk structure known at this level, except for the sector size, which is to be 256 bytes.

4.2 Level 2 Features

* All Level 1 features, plus:
* Data access by filename and physical record displacement
* File creation and deletion
* Mixed hybrid file format
* Dynamically extendable file allocation

This level creates the actual file concept. Each file is known by its name and the displacement of the physical record within the file. Each physical record is defined as one disk sector (256 bytes).

A directory and a Bit Map are maintained on every disk to allow for file and data-record management (creation and deletion). The file format available at this level is a mixture of contiguous and non-contiguous file formats, called the mixed hybrid format. Non-contiguous files (fragmented) carry a lot of overhead in the form of pointers to the location of each data-record of the file, in case relative access is required. In order to combine the advantage of the flexible allocation of non-contiguous files with the low overhead and easy access of contiguous files, the files on this level are allocated in clusters of contiguous records. These clusters are expanded if possible, whenever new data-records are requested. If a cluster cannot be expanded any more, a new cluster is started.

4.3 Level 3 Features

* All Level 2 features, plus:
* Fixed and Variable record formats
* Relative and Sequential access methods
* Program and data files
* Internal and ASCII data types
* Communication with the HEXBUS interface

The addition of relative/sequential access methods and fixed and variable record formats completes the disk management software. The software at this level takes care of the blocking of one or more logical records into a physical record. For relative access files it computes the physical record in which the logical record is located, updates that record, and passes the physical record back to the Level 2 file update routines.

A HEXBUS interface will be provided in the DSR to support conversion of HEXBUS formatted requests. This will be performed by Encode and Decode subroutines. When a request is made to the HEXBUS, the decode subroutine will translate this request and reformat to the 9995 processor file management format. Upon completion of the I/O request, the encode subroutine will encode the results back to the HEXBUS format. Proper error translation will be performed in the encode to correspond to the respective system error codes.

4.4 Utility Routines

The ROM-code also provides the subprograms for special utility routines which do not use the standard file I/O system. These subprograms are:

* Direct Level 2 file access
* Logical sector/Allocatable Unit (AU) access
* File rename
* File protection modification
* Disk formatting

The subprograms will be provided in the form of GPL subprograms, i.e. assembly language routines located in ROM, for the Home Computer family. These GPL subprograms are specified in the GPL Interface Specification for the 99/4 Disk Peripheral. These subprograms can also be used by the Compact Computer family.

SECTION 5

DETAILED OPERATIONAL SPECIFICATIONS

This section will go into the operational specifications in more detail. Record formats and access methods, as well as file type will be discussed.

5.1 Record Formats

A file attribute specified at time of creation is the record format. This attribute describes the logical organization of the file. Two such formats are currently supported by the disk software:

1. Variable length records
2. Fixed length records

5.1.1 Variable Length Records.

In applications which must store data structures of unpredictable length, the variable length record format provides an economical way to use the disk space. Since the length of the records is variable, the length of each individual record must be recorded together with the data. This can be done by providing each record with a pointer to the next record, or equivalently, by providing each record with a header byte indicating the number of bytes required to represent the data structure.

Variable length records can also be used to record fixed length data structures in which repeated character strings are expected (i.e. trailing blanks). Such a fixed length structure can become variable by virtue of data compression. However, the current implementation of the disk peripheral software does not perform any data compression.

5.1.2 Fixed Length Records.

Records of fixed (constant) length can be used if relative access to particular records is desired. When the information

structures to be recorded are almost or exactly equal in length to the record size, fixed length records are appropriate, since there is no overhead associated with record headers or compression indicators. The records consist of an unmodified copy of the data as presented in the user's data buffer. This is obviously a more efficient use of bulk storage devices where relative access is supported by the medium.

As we shall see in the next section, fixed length records are also very convenient for relative access, since their length is a known quantity.

5.2 Access Methods

Several methods of accessing data in files are supported. These methods are:

* Physical I/O
* Sequential access
* Relative access

5.2.1 Physical I/O.

In the physical I/O access method, the data on the disk is considered by the disk software to be organized in blocks of 256 bytes each. Each byte contains any of the 256 possible 8-bit combinations, with no attempt to interpret at data transfer time. Any existance of records or files is completely ignored when this access method is being used.

The rest of the disk software reduces all access methods to physical I/O, by converting logical record numbers to physical track/sector data, which can be used to specify the disk sector that is to be transferred by the physical I/O software. Physical I/O has been made available to GPL software only in the form of an assembly language subprogram.

5.2.2 Sequential Access.

When data records in a file are accessed strictly in the order of increasing addresses on the medium, the records are said to be sequentially accessed. This is typically the access method associated with magnetic tape and other linear storage media. The data transfer parameters do not specify a physical record

number. It is implied that the logical record currently indicated in some data transfer pointer, is the one desired. Rewind/Restore operations are implicitly or explicitly done, to set such pointers to the beginning of the file, prior to the first data transfer. As each logical record is transferred, the pointer moves to the first byte of the following one (possibly the length indicator).

5.2.3 Relative Access.

This access method, also called random access, allows data reference by logical record number. Logical data records may be accessed in any sequence, without regard to the order in which they were written, or their relative position in the file.

Since the disk software must be able to locate a record based solely on its number, relative access can be supported on indexed files or on fixed length record files only. Indexed files are not supported in this implementation, so the relative access method is supported for fixed length record files only.

5.3 Library Organization

The library organization implemented in the disk software only supports a single level library. This implies that no file can be of the catalog type (a file pointing to other files). Each file can be identified by a single name, for example:

DSK1.filename

which specifies a file called "filename" on the diskette in disk drive 1.

Since this approach prohibits access of a catalog file as such, a semi-catalog file has been created. This file is of the fixed length, relative access type. It contains 128 records, each containing information about the associated catalog entry. The semi-catalog file, which will be described in more detail in section 8, can be accessed as:

DSK1. or DSK.volname.

as a general file, with an empty filename.

Notice that not all general file operations have been defined for the catalog file. Only the standard OPEN, READ, and CLOSE calls are supported. All the other operations, such as

DELETE, RESTORE, WRITE, and EOF, are illegal, and will cause an error to be returned.

5.4 Internal Data Structure Overview

This section describes the internal data structure implemented on the disk peripheral. A description of the external data structure can be found in the File Management Specification for the TI-99/4 Home Computer.

5.4.1 Phusical Device Format.

The physical device is logically subdivided into Allocatable Units (AUs). An AU is defined to be an integral number of physical records on the device. The total number of AUs on any device should be less than 4096 (i.e., each AU can be addressed in a 12-bit word). AUs are numbered with zero origin, (i.e. the first AU is number 0).

The physical record length is the block of data read from or written to the device at one time. For the disk peripheral, both the AU and the physical record are currently equivalent to one diskette sector (256 bytes).

5.4.2 Volume Information Block.

The Volume Information Block (VIB) is located at AU number 0. If this AU is bad, the entire device will be considered bad. This block contains configuration data as required by the disk software, such as available number of AUs, volume identification field, and format information.

A major part of the VIB has been allocated for the Allocation Bit Map.

5.4.3 Allocation Bit Map.

The Allocation Bit Map is used to indicate the availability of individual allocation units. A binary 1 in a bit position indicates that the allocation unit associated with that bit has been allocated. The first bit (bit 0) is associated with allocation unit 0, the second bit (bit 1) with allocation unit 1, etc. During disk initialization, bits corresponding to system-reserved AUs, non-existing AUs, and bad AUs, are set to 1. All other bits are set to zero.

5.4.4 File Descriptor Record.

The File Descriptor Record (FDR) is used to map filenames into physical locations of the files on the disk. Each entry contains information such as filename, file type, record type, data type, location, and size information for the file.

5.4.5 File Control Block.

The File Control Block (FCB) is a copy of the File Descriptor Record that is maintained in memory while the file is open. In addition to the FDR information, the FCB contains some up-to-date file information.

5.4.6 File Descriptor Index Record.

The File Descriptor Index Record enables the system to keep track of the location of each file descriptor record on the disk. It contains alphabetically sorted pointers to each File Descriptor Record.

The File Descriptor Index Record is located at AU number 1. If this AU is bad, the entire disk is considered to be bad.

SECTION 6

DETAILED DISKETTE FORMAT SPECIFICATION

The HEXBUS disk controller hardware and DSR software will format, read, and write diskettes of the following configurations:

Table 6-1 Valid Diskette Configurations

No. Trks.	No. Sides	Data Density	Sectors/ Track	Sectors/ Diskette	Data Bytes/ Diskette
35	1	S	9	315	80640
35	1	D	16	560	143360
40	1	S	9	360	92160
40	1	D	16	640	163840
40	2	S	9	720	184320
40	2	D	16	1280	327680
77	1	S	9	693	177408
77	1	D	16	1232	315392
77	2	S	9	1386	354816
77	2	D	16	2464	630784

All sectors are 256 data bytes long. Single data density follows IBM 3740 FM encoding format and double density is IBM 34 MFM format.

Data density is automatically determined by the low level DSR in the read or write mode.

The following section contains a description of the logical structure on each diskette in terms of records.

6.1 Physical Diskette Format

The general diskette format used in the TI-99/4 Disk Peripheral is the following:

Sector 0 contains the Volume Information Block (VIB). This block contains general information about the diskette like:

* Volume Name
* Number of available AUs
* Number of sectors/track
* Allocation Bit Map

Sector 1 contains pointers to file descriptor records.

Sector 2 thru 359 contain File Descriptor Records and data blocks.

The File Descriptor Records contain general information about the file, such as:

* File name
* File status data
* File data access blocks

6.1.1 Volume Information Block.

As mentioned previously, this block contains general information about the diskette. A more detailed description of each entry and its contents will be given in this section.

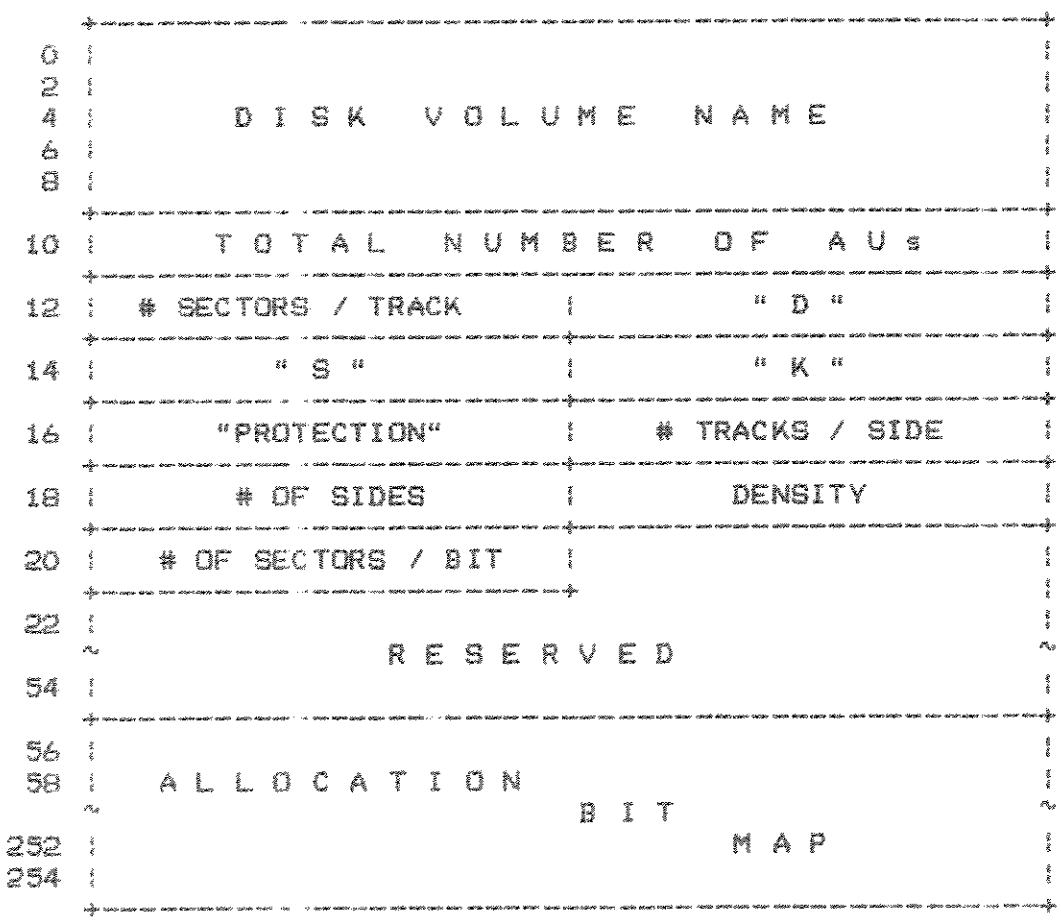


Figure 6-1 Volume Information Block

Bytes 0-9 contain the volume name of the diskette. The volume name can be any combination of ten ASCII characters, except for the space or period (".") characters and the null character (ASCII code 0). The name is space filled to the right in case of less than 10 characters. The volume name must contain at least one non-space character.

Bytes 10-11 give the total number of allocation units (AUs) on the volume. This datum should match the Allocation Bit Map.

Byte 12 indicates the number of sectors per track.

Bytes 13-15 contain the ASCII code for "DSK", which is used by the disk manager software to check if the diskette has been initialized.

Byte 16 contains the ASCII code for "P" if the diskette is protected (a protected disk is also called a proprietary disk), otherwise this byte contains a >20.

Byte 17 indicates the number of tracks per side.

Byte 18 indicates the number of formatted sides on the diskette.

Byte 19 indicates the density of the diskette.

Byte 20 indicates the number of sectors per AU bit. With 48 TPI each AU bit represents one sector. With 96 TPI each AU bit represents two sectors.

Bytes 21-55 are reserved for future expansions like date and time of creation. In the current version of the disk software these bytes are set to zero.

Bytes 56-255 contain the Allocation Bit Map. This 200 byte map can control up to 1600 256-byte records (total controllable storage capacity = 400K bytes), which make it useable for a double density, double sided diskette. The disk allocation system uses a conventional method of allocating disk space called Bit Map. Each bit in the Bit Map represents one sector on the disk. A logical one in the Bit Map means that the corresponding sector has been allocated. A zero means that the sector is still available.

The volume name can be used as an alternative to the actual disk drive name, i.e. the user can specify a disk drive in either of the following ways:

DSK.volname.filename or DSK1.filename

If the voluma is specified, rather than the physical drive number, the system will look in sequence on every drive in the system, until it finds the specified volume. If more than one volume of the same name exists, the drive with the lowest drive identification number will be assigned.

6.1.2 File Descriptor Index Record.

The File Descriptor Index Record contains up to 127 two byte entries, each pointing to a file descriptor record. These pointers are alphabetically sorted according to the filename in the associated file descriptor record. The pointer list starts at the beginning of this block, and ends with a zero entry.

Since the file descriptors are alphabetically sorted in this block, a binary search method can be used to find any given filename, limiting the maximum number of disk searches to 7 if more than 63 files are defined. In general if between 2**(N-1) and 2**N files are defined, a file search will take at most N disk searches. To obtain faster directory search response times, the system will prefer to allocate data blocks in the area above AU number 34. Only if no AU can be allocated in that area will the disk data block allocator start allocating blocks in the AU area 2-33.

6.1.3 File Descriptor Record.

The File Descriptor Record (FDR) contains general information about the associated file. All the information the system needs to know to access and update the file has to be contained within the file descriptor record.

The physical layout of an FDR is:

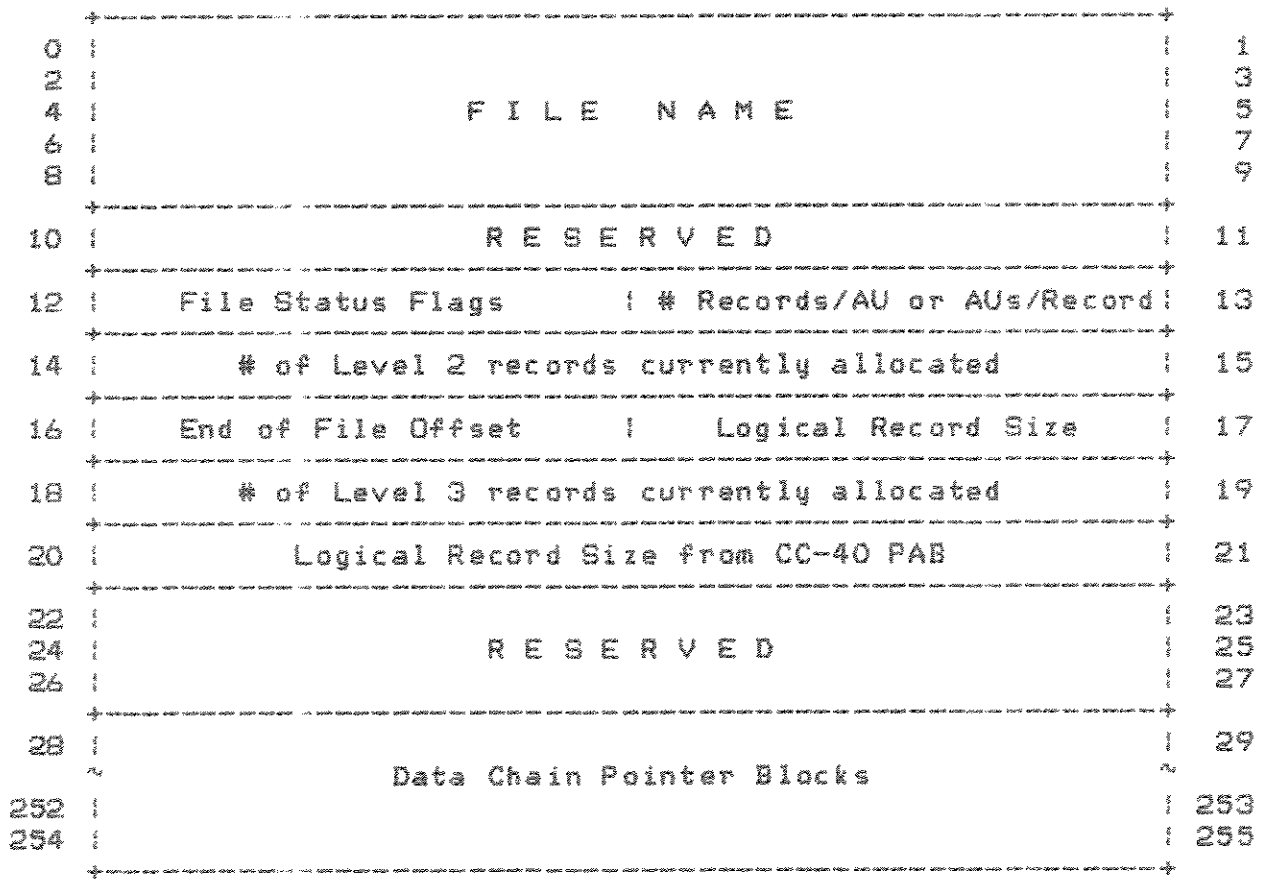


Figure 6-2 File Descriptor Record

Bytes 0-9 contain a filename up to ten characters in length.

Bytes 10-11 are reserved for future extension of the number of data chain pointers through linkage to a data chain pointer block chain. In the current version these bytes are always 0.

Byte 12 contains the file status flags. These flags are to be interpreted as follows (bit 0 is the least significant bit):

Bit #	Description
0	Program/data file indicator. 0 = Data file 1 = Program file
1	Binary/ASCII data 0 = ASCII data (DISPLAY file) 1 = Binary data (INTERNAL or program file)
2	Reserved for future data type expansion
3	PROTECT flag 0 = Not protected 1 = Protected
4-6	Reserved for future expansion
7	FIXED/VARIABLE flag 0 = Fixed length records 1 = Variable length records

Byte 13 contains the number of logical records per AU.

Bytes 14-15 contain the number of logical records allocated on Level 2 (256 byte records).

Byte 16 contains the EOF offset within the highest physical AU for variable length record files and program files.

Byte 17 contains the logical record size in bytes. In case of variable length records, this entry will indicate the maximum allowable record size.

Bytes 18-19 contain the number of records allocated on Level 3. For variable length records, this entry is replaced with the number of Level 2 records actually used. (NOTE: The bytes in this entry are in reverse order.)

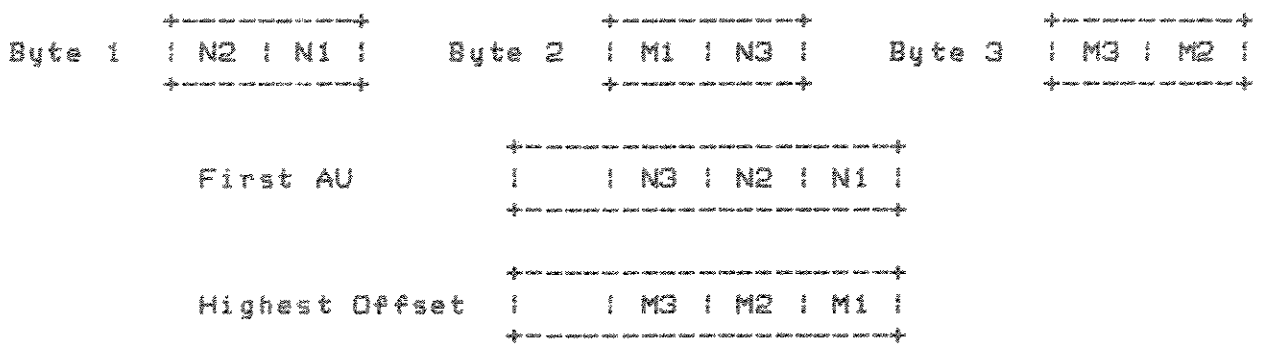
Bytes 20-21 contain a zero if the logical record size is less than 254. Otherwise these bytes contain the actual logical record size received from the CC-40 PAB. These two bytes are only used for variable length records.

Bytes 22-27 have been reserved for future expansion. They will be fixed to 0 in this implementation of disk peripheral

software.

Bytes 28-255 contain three byte blocks indicating the clusters that have been allocated for the file. The first 12 bits in each entry indicate the address of the first AU in the cluster. The second 12 bits indicate the highest logical record offset in the cluster of contiguous records. This indication has been chosen, rather than the number of data-records in the chain, since it reduces the amount of computation required for relative record file access.

The following diagram shows how each three byte entry relates to the address of the first AU and the highest logical record offset in the cluster.



6.2 Data File Allocation

A data-file is built out of clusters of contiguous data records. Each data-file can contain up to 76 of those data record clusters. Each data cluster can contain at least one record. The disk software will allocate as many contiguous records as possible upon request. If a new record is requested, and no more records can be added to the current contiguous cluster, a new cluster of contiguous records is started. If 76 of those clusters have been allocated, and a new cluster is requested, the data-records on the disk have become too scattered, and the write-operation is aborted. Worst case this scheme still allows for a minimum of 19K bytes per file (76 * 256 bytes).

An additional advantage of this scheme is that each physical record within the file can be accessed at random, without any need for big areas of contiguous disk space. This means that as long as the logical records within a file have a fixed length, the file can be accessed either sequentially or at random. Therefore the disk software does not make any distinction between

relative record or sequential files. Note that this has some implications for sequential fixed length record access, since now the record number is being used, rather than the current record number and offset.

For variable length records, the length of the logical record will be stored together with the record itself. This means that, since we do not cross physical record boundaries for any file- or record-type, the maximum record length for a variable length record file is limited to 254 bytes. The end of an AU with variable length records will be marked with an "all ones" byte.

6.3 Program File Allocation

The allocation of a program file is identical to the allocation of a data file. The program segment is blocked into 256-byte records which are stored as a standard data-file. However, the disk software will mark a program file as such, and will not allow data access to program files and vice versa.

To avoid any problems with memory wrap-around, the disk software will also note the actual number of bytes used in the last data record and it will return exactly as many bytes as have been stored originally, even if this number is not a multiple of 256.

SECTION 7

MEMORY USAGE

Since the disk peripheral software will have to use buffer areas to buffer control information, the disk software will allocate part of controller RAM memory for its internal usage.

The allocated RAM memory can roughly be subdivided into three usage categories:

1. Drive control information
2. File allocation information
3. Data buffering

Each of these categories will be discussed in more detail in the next sections.

7.1 Drive Control Information

In order to be able to control the disk drive hardware, the software has to know what the current status of each disk drive is before it can access it. This information is available in section 6 of the HEXBUS Floppy Disk System Product Specification.

7.2 File Allocation Information

The file allocation information is maintained in the File Control Blocks (FCBs). Each "open" file has an FCB associated with it.

The information maintained in the FCB is identical to the FDR information described in section 6.1.3. In addition, the disk software also maintains some dynamic information about each file. This information is stored in front of the standard FDR information (i.e. the FDR starts at FCB location 6). The total length of an FCB is therefore 512 + 6 = 518 bytes, including its 256 byte data buffer (see next section).

The format of the FDR extension is outlined below.

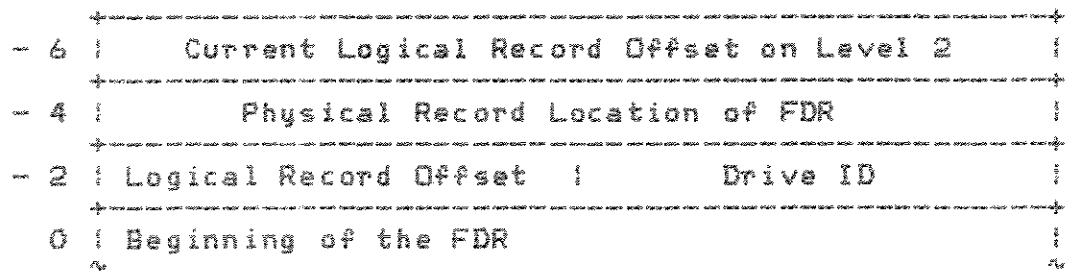


Figure 7-1 FDR Extension

The meaning of each entry in this additional information block is:

Drive ID - Contains the drive number (1-3) of the drive on which the associated file resides. If the highest bit of this entry is set, the current data block has been modified and will have to be written back to the disk before closing the file, or accessing a new data block.

Logical Record Offset - This entry contains the offset of the next logical record in the current physical record. If, during READ operations, this entry points to a byte count of >FF, this will indicate an end of record for the current physical record.

This entry is only used for variable length records. For fixed length record access, the actual position AU and the position within that AU is recomputed before every I/O operation. The logical record offset byte is therefore superfluous in this case.

During WRITE operations, this offset always points to the first free byte in the physical record. If the next logical record would leave less than one byte in the current record, a byte count of >FF will be written, and the logical record will be located in the next physical record. Note that the first logical record in a physical record can never cause that physical record to overflow, since the maximum logical record length is 254, and the physical record length is 256.

Physical Record Location of FDR - Points to the physical sector location of the FDR on the disk. This is important if we ever want to rewrite the FDR on the disk. Even though it is not required, it is still maintained during read-only access.

Current Logical Record Offset on Level 2 - Contains the physical record offset of the most recently processed physical record. Independent of READ or WRITE operations, this entry always contains the logical offset for Level 2 operation of the datablock that is currently in memory.

Notice that this approach causes fixed length sequential files to be accessed as relative access files on Level 2.

7.3 Data Buffering

For the purpose of data buffering, the disk software will maintain one 256-byte buffer for each "open" file, located directly above the FCB buffer.

One of the controller RAM buffers is continuously assigned to VIB processing. In case more than one drive is used for WRITE mode, the Bit Map will be moved in and out of this buffer as demanded by the disk software. For every Bit Map operation, this buffer will be used to access the Volume Information Block.

Every Level 3 WRITE operation to a file will ultimately be passed onto Level 2 as a physical sector WRITE. To minimize the number of disk accesses, a flag will be set to indicate that the current data buffer has been modified. The data buffer will only be physically written to the disk if the next physical record access involves another physical record than the one currently residing in the data buffer. If the file is closed for further access, the last data buffer will be written onto the disk if required.

7.4 Controller RAM Memory Layout

The 4K controller RAM memory layout used for the disk peripheral is outlined in Figure 7-3. The memory block in this figure is allocated upon power up by special power up code. Four files is the maximum number of files allowed to be open at the same time. Each file takes up 518 bytes of memory.

Each file has its own FCB and data buffer associated with it. To simplify the buffer allocation mechanism, the buffers are not allocated on demand, but rather as soon as a file is opened, an FCB and databuffer are associated with it for the entire "open" life of the file.

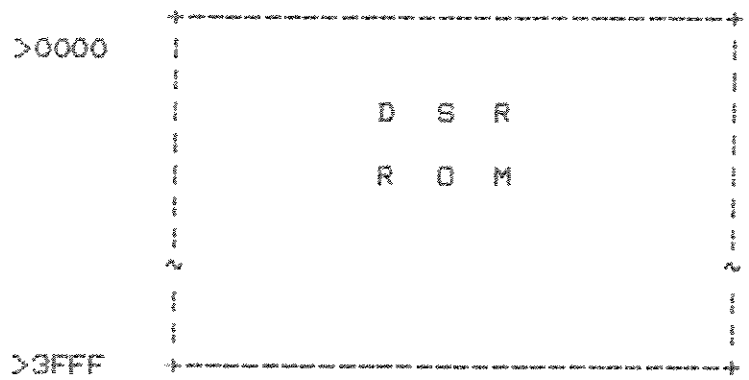


Figure 7-2 16K ROM

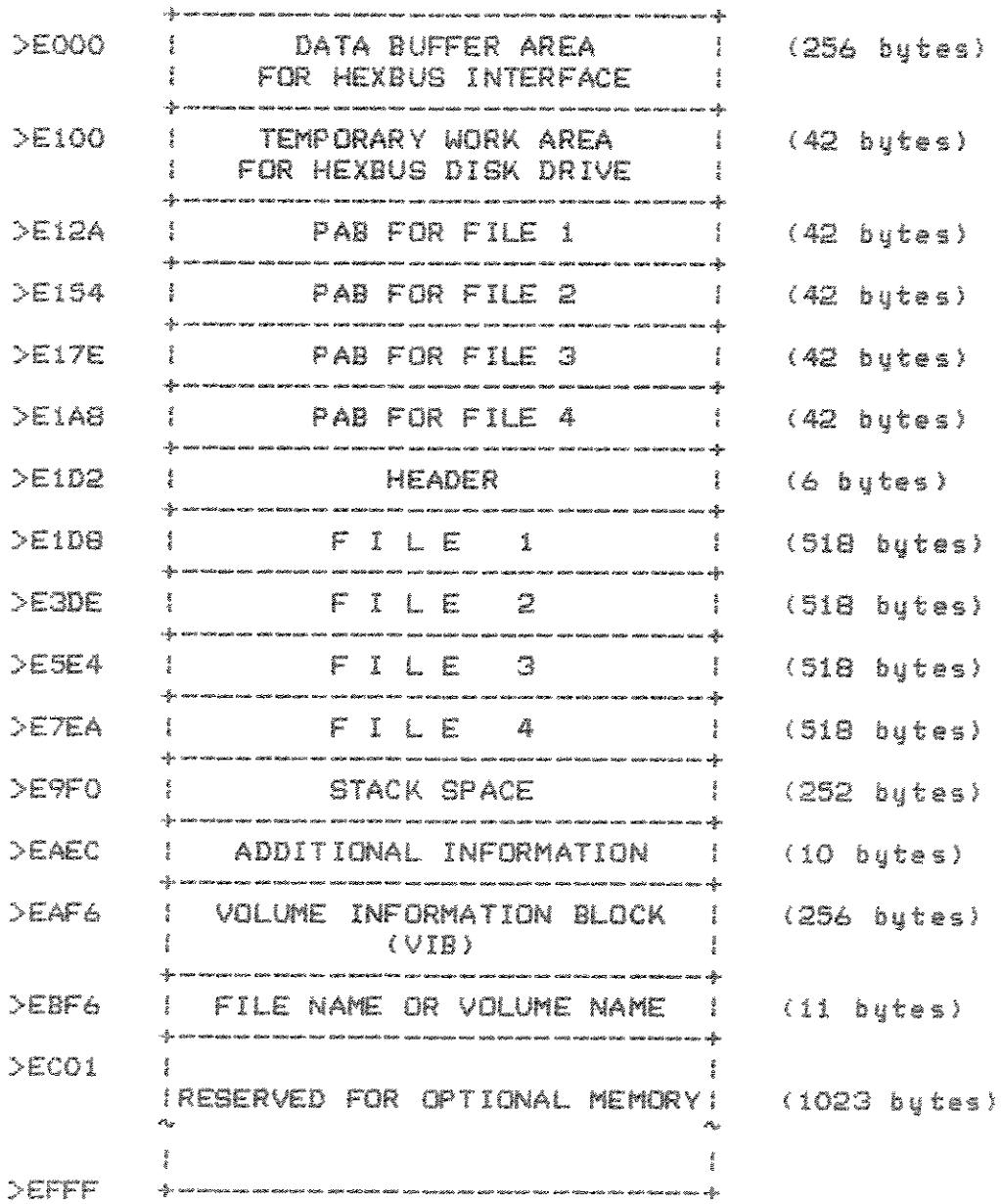


Figure 7-3 4K Controller RAM

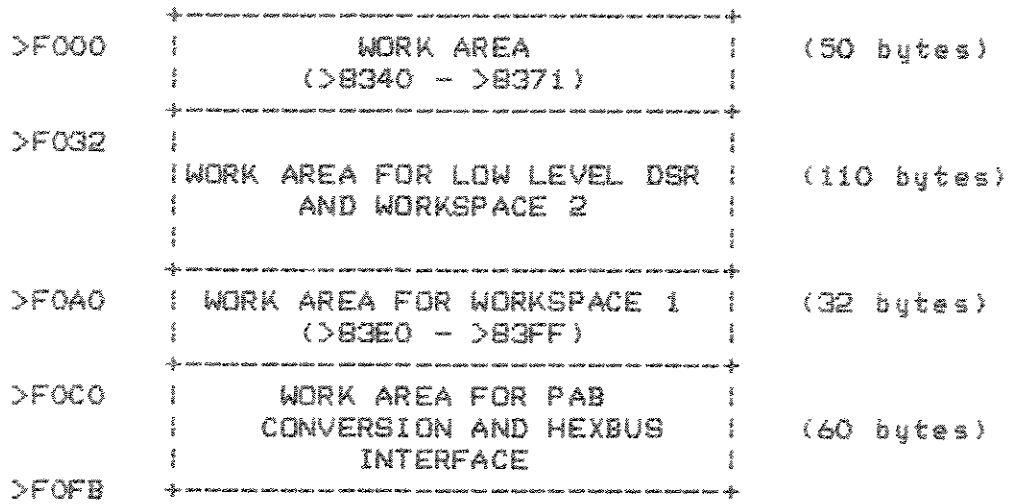


Figure 7-4 9995 Fast Scratch Pad RAM

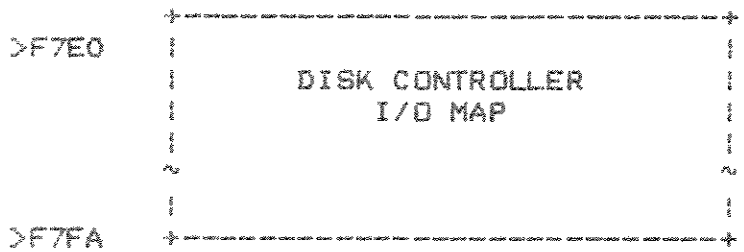


Figure 7-5 Disk Controller I/O Map

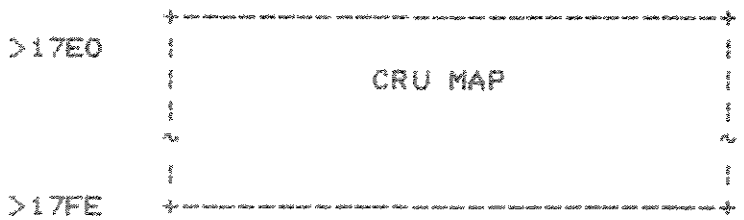


Figure 7-6 CRU Map

SECTION 8

IMPLEMENTATION

The DSRs should present a uniform interface between the File Management System and the peripherals. This section will give implementational details on this interface. Some remarks are being made on a possible implementation of the file system for random access devices. However, no details are given for such an implementation.

8.1 Peripheral Access Block Definition

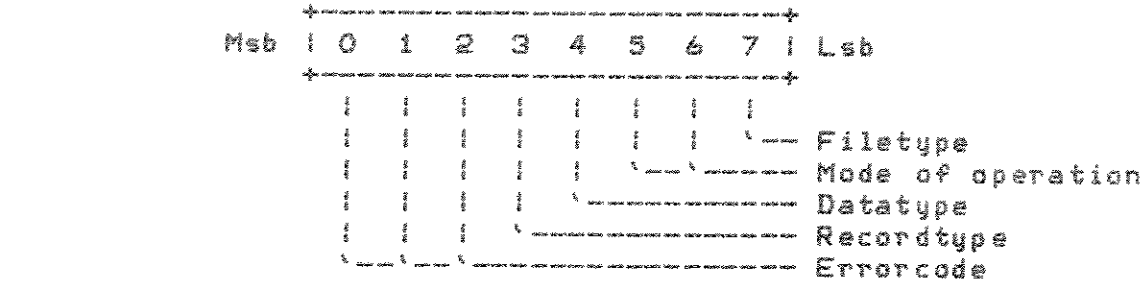
All DSRs are accessed through a Peripheral Access Block (PAB). The definition for these PABs is the same for every peripheral. The only difference between peripherals, as seen by any application program, is that some peripherals will not support every option provided for in the PAB.

All PABs are physically located in controller RAM. They are created before the OPEN call, and are not to be released until the I/O has been closed for that device or file.

Figure 8-1 shows the layout of the new PAB. The PAB has a variable length, depending upon the length of the file descriptor. The meaning of the data within the new PAB is explained below.

Byte	Bit	Meaning
0	All	I/O opcode - Contains opcode for the current I/O-call. A description of the valid opcodes will be given in section 3.2.
1	All	Flags - File-type, mode of operation and data-type is stored in this byte. The meaning of the bits within this flagbyte is:

Byte	Bit	Meaning



1	0-2	Reserved (Set to zero).
	3	Recordtype - Indicates type of record used. 0 = Fixed length records 1 = Variable length records
	4	Datatype - Indicates type of data stored in the file. 0 = DISPLAY type data 1 = INTERNAL type data
	5,6	Mode of operation - Indicates operation mode file has been opened for. 00 = UPDATE 01 = OUTPUT 10 = INPUT 11 = APPEND
	7	Filetype - Indicates file-type. 0 = Sequential file 1 = Relative record file
2,3	All	Data buffer address - Address of the data buffer in controller RAM the data has to be written to or read from.
4,5	All	Character count - Number of characters to be transferred for a WRITE opcode, or the number of bytes actually read for a READ opcode.
6,7	All	Record number - Only required if the file opened is of the relative record type. Indicates the record number the current I/O operation is to be performed upon (this limits the range of record-numbers to 0 - 32767). The highest bit will be ignored by the DSR.

Byte	bit	Meaning
8,9	All	Logical record length - Indicates the logical record length for fixed length records, or the maximum length for a variable length record (see flagbyte).
10	All	Screen offset - Offset of the screen characters in respect to their normal ASCII value. (Normally >60 while a BASIC is running, >00 otherwise.) This byte is used by cassettes, disks and the RS232. For disks using the STATUS opcode, this byte will be the returned status.
11	All	Logical unit number - Contains the number assigned to each file by the HEXBUS Interface DSR. Each file is associated with a unique number between 1 and 255. This logical unit number is only used by the HEXBUS Disk System.
12	All	Error status - Indicates the CC-40 error code. Section 8.4 describes the HEXBUS Peripheral error codes.
13	All	Name length - Length of the file descriptor following the PAB.
14+	All	File descriptor - The device name and, if required, the filename and options. The length of this descriptor is given in byte 13.

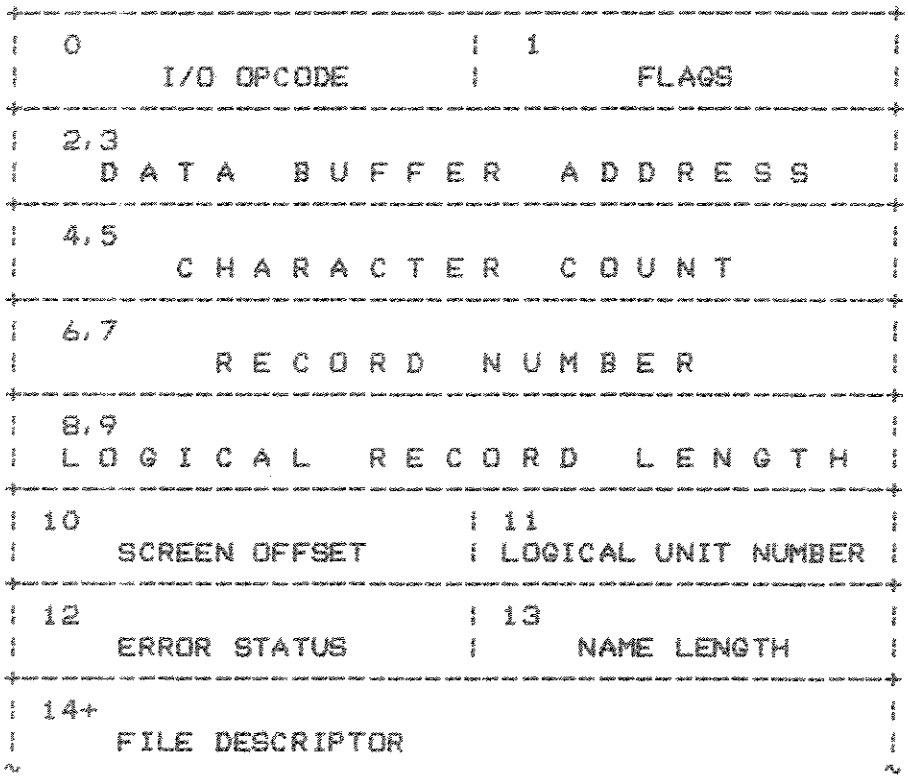


Figure 8-1 New PAB Format

The following figure shows the layout of the old PAB. This figure is included only for reference purposes.

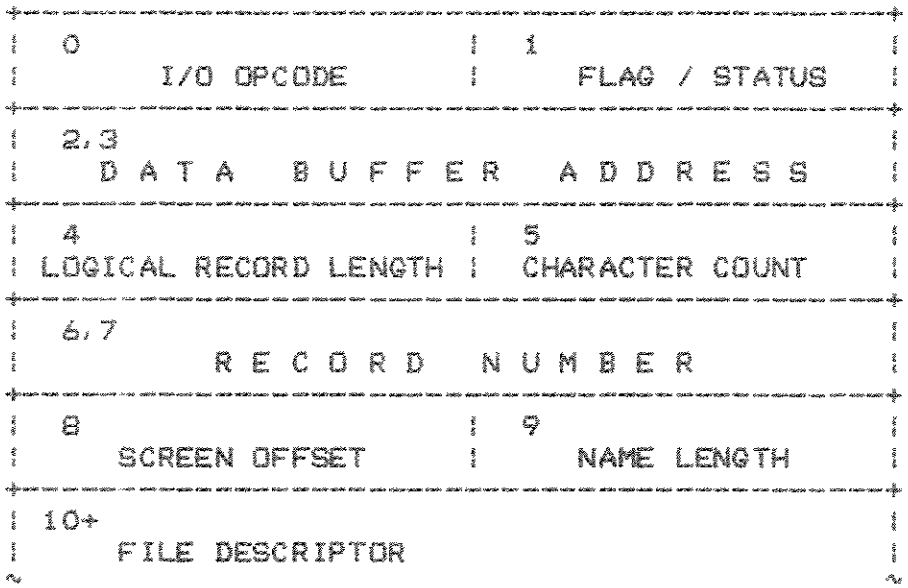


Figure 8-2 Old PAB Format

8.2 I/O Opcodes

This paragraph describes the valid opcodes that can be used in a PAB. Valid opcodes are shown in Figure 8-3.

Opcode	Meaning
00	OPEN
01	CLOSE
02	READ
03	WRITE
04	RESTORE/REWIND
05	LOAD
06	SAVE
07	DELETE
08	SCRATCH RECORD
09	STATUS

Figure 8-3 I/O Opcodes

The following describes the general actions invoked by an I/O-call with each of the I/O-opcodes. Each I/O-call returns any

error-codes in byte 12 (Error Status) of the PAB.

8.2.1 OPEN.

The OPEN operation should be performed before any data-transfer operation except those performed with LOAD or SAVE. The file remains open until a CLOSE operation is performed. The mode of operation for which the file is to be opened must be indicated in byte 1 (Flags) of the PAB. In case this mode is UPDATE, APPEND or INPUT, and a record length of zero is given in bytes 8 and 9 (Logical Record Length), the assigned record length (which depends on the peripheral) is returned in bytes 8 and 9. If a non-zero record length is given, it is used after being checked for correctness with the given peripheral. For OUTPUT, the record length can be specified, or a default can be used by specifying record length zero.

For any device, an OPEN operation must be performed before any other I/O operation. The DSR need only check the record length and I/O mode on an OPEN. Changing I/O modes after an OPEN may cause unpredictable results.

8.2.2 CLOSE.

The CLOSE operation closes the file. It informs the DSR that the current I/O sequence to that DSR has been completed. If the file or device was opened in OUTPUT or APPEND mode, an End Of File (EOF) record is written to the device or file before deallocating the PAB. After the CLOSE operation, the space allocated for the PAB may be used for other purposes. As long as a PAB is connected to an active device, the contents of that PAB must be preserved.

8.2.3 READ.

The READ operation reads a record from the selected device and copies the bytes into the buffer specified in bytes 2 and 3 (Data Buffer Address) of the PAB. The size of the buffer is specified in bytes 8 and 9 (Logical Record Length) of the PAB. The actual number of bytes stored is specified in bytes 4 and 5 (Character Count) of the PAB. If the length of the input record exceeds the buffer size, the remaining characters are discarded.

8.2.4 WRITE.

The WRITE operation writes a record from the buffer specified in bytes 2 and 3 (Data Buffer Address) of the PAB to the specified device. The number of bytes to be written is specified in bytes 4 and 5 (Character Count) of the PAB.

8.2.5 RESTORE/REWIND.

The RESTORE/REWIND operation repositions the file READ/WRITE pointer either to the beginning of the file, or, in the case of a relative record file, to the record specified in bytes 6 and 7 (Record Number) of the PAB. This operation can only be used if the file was opened in INPUT or UPDATE mode. For relative record files, a RESTORE can be simulated in any I/O mode by specifying the record at which the file is to be positioned in bytes 6 and 7 (Record Number) of the PAB. The next I/O operation then automatically uses the indicated record.

8.2.6 LOAD.

The LOAD operation loads an entire memory image of a file from an external device or file into controller RAM. All the control information the application program needs should be concatenated to the program image. Since no intermediary buffers are used, the LOAD operation requires as much buffer in controller RAM as the file occupies on the diskette or other device. The entire memory image is dumped starting at the specified location.

The LOAD operation is a stand alone operation, i.e. the LOAD operation is used without a previous OPEN operation.

For this operation, the PAB needs to contain only the following information:

Byte 0 : I/O opcode.

Bytes 2,3 : Start address of the controller RAM memory dump area.

Bytes 6,7 : Maximum number of bytes to be loaded.

Byte 13 : Name length.

Bytes 14+ : File descriptor information.

8.2.7 SAVE.

SAVE is the complementary operation for LOAD. It writes memory images from controller RAM to a peripheral. The SAVE operation is used without a previous OPEN operation. It copies the entire memory image from the buffer in controller RAM to the diskette or other device. All necessary control information should be linked to the memory image, so that the information plus program image use one contiguous memory area. Again, only a small part of the PAB is used. The PAB contains:

Byte 0 : I/O opcode.

Bytes 2,3 : Start address of the controller RAM memory area.

Bytes 6,7 : Number of bytes to be saved.

Byte 13 : Name length.

Bytes 14+ : File descriptor information.

8.2.8 DELETE.

The DELETE operation deletes the specified file from the specified peripheral. This operation also performs a CLOSE.

8.2.9 SCRATCH RECORD.

The SCRATCH RECORD operation removes the record specified in bytes 6 and 7 (Record Number) of the PAB from the specified relative record file. This operation causes an error for peripherals opened as sequential files. No device currently supports this operation.

8.2.10 STATUS.

The STATUS operation is used for obtaining information about a file. This information can be examined at any time, although bits 6 and 7 only have meaning if a file has been opened.

To indicate the current status of the file, byte 10 (Screen Offset) is used. Upon the DSR-call, byte 10 should contain the usual screen characters base address. The DSR can only use this byte, and is guaranteed not to destroy any other entry in the PAB.

The meaning of the bits within byte 10 after return from the DSR is shown in the following table.

Table 8-1 Meaning of byte 10 after return from DSR

Bit	Information
0	If this bit is set, the requested file doesn't exist. If reset, the file does exist. On some devices, such as a printer, this bit is never set since any file could exist.
1	PROTECT flag. If set, the file is protected against modifications. If reset, the file is not protected.
2	Reserved for future use. Fixed to zero in the current peripherals.
3	Data type. If set, the data type is binary (INTERNAL). If reset, the data type is ASCII (DISPLAY) or file is program file.
4	Filetype. If set, the file is a program file. If reset, the file is a data file.
5	Record type. If set, the record type is VARIABLE length. If reset, the record type is FIXED length.
6	Physical end of file. If set, no more data can be written, since the physical limits of the device have been reached. Generally this means an end of medium has been detected on the device.
7	Logical end of file. If set, the file is at the end of its previously created contents. This is usually the case if the file has been opened for APPEND mode. Depending upon the mode of operation for which the file has been opened, data can still be written to the file (APPEND, OUTPUT or UPDATE mode), however, a "read" operation will cause an ATTEMPT TO READ PAST EOF error to occur.

Bits 0 - 5 have meaning even if the file is not open. Bits 6 and 7 only have meaning for files that are currently open, otherwise a zero should be indicated in these two bits.

8.3 Directory Handling

The GROM cartridge containing the DSR for a device that supports files, shall also contain a CATALOG program, which can be used to list the current contents of the medium.

8.4 Error Codes

The File Management System supports a number of error codes. Errors are indicated in byte 12 (Error Status) of the PAB. These error codes are completely compatible with computers in the Compact Computer family, but have to be interpreted for computers in the Home Computer family.

The following table divides error codes into two categories: Home Computer and Compact Computer. Home Computer is then divided into Disk Manager III and BASIC error codes. The Disk Manager III error codes are in the left-most column in the following table. They correspond to the error codes described in the Disk Memory System Manual. The error codes generated by BASIC are defined by two digits. The left-most digit corresponds to the value of one of the I/O Opcodes described in Figure 8-3. The right-most digit corresponds to the middle column of the next table. The third column in the table is the Compact Computer's error codes. The following table describes the HEXBUS Peripheral error codes and their meanings.

Table 8-2 HEXBUS Peripheral Error Codes and Meanings

ERROR CODES			MEANING
HOME COMPUTER DM III	COMPACT COMPUTER BASIC		
00	0		BAD DEVICE NAME
01	1	09	DEVICE IS WRITE PROTECTED
	1	09	DATA FILE OR PROGRAM FILE IS WRITE PROTECTED
	2	02	BAD OPEN ATTRIBUTE
	2	02	FIXED/APPEND ATTRIBUTE ERROR
	2	02	RECORD NUMBER IS TOO LARGE
	2	02	OPEN ATTRIBUTE HAS TO BE FIXED/INTERNAL; TO OPEN A CATALOG FILE, THE OPEN CODE HAS TO BE FIXED/INTERNAL
	2	0C	RECORD LENGTH HAS TO BE 38 FOR CATALOG FILE
	2	0C	INPUT BUFFER SIZE (LOGICAL RECORD LENGTH) IS TOO BIG
	2	0E	FILE NOT OPENED FOR WRITE
	2	0F	FILE NOT OPENED FOR READ
	2	11	FILE TYPE (RELATIVE/SEQUENTIAL) IS INCORRECT OR IS NOT SUPPORTED
	2	17	FILE TYPE (INTERNAL/DISPLAY) IS INCORRECT OR IS NOT SUPPORTED
	3	0D	ILLEGAL OPERATION
	3	0D	I/O OP-CODE IS LARGER THAN 9
	3	1E	OUTPUT/APPEND MODE, WHICH IS NOT ALLOWED FOR 'RESTORE' OPERATION
	4	0B	DIRECTORY FULL
04	4	20	OUT OF SPACE; I.E. THE DISKETTE IS FULL
	4	21	ATTEMPTED TO EXCEED ALLOWED NUMBER OF ASSIGNED LUNOS
	5	07	ATTEMPTED TO READ BEYOND END OF FILE

HEXBUS Peripheral Error Codes and Meanings (continued)

ERROR CODES			MEANING
HOME COMPUTER DM III	BASIC	COMPACT COMPUTER	
06	6	06	DEVICE ERROR, I.E. DISK DRIVE/DISKETTE ERROR (DISKETTE MAY NOT BE INITIALIZED)
16	6	61	NO DISKETTE OR NO DRIVE
17	6	62	INVALID INPUT PARAMETERS, I.E. LOGICAL SECTOR NUMBER IS TOO LARGE
11	6	63	SEEK ERROR
41	6	64	MISSING ADDRESS MARK ERROR
51	6	65	NO DATA (SECTOR NOT FOUND) ERROR
21	6	66	RECORD NOT FOUND ON A READ
22	6	67	CYCLIC REDUNDANCE CODE ERROR ON A READ
23	6	68	LOST DATA ON A READ
28	6	69	BAD COMPARE ON A WRITE
31	6	6A	RECORD NOT FOUND ON A WRITE
33	6	6B	LOST DATA ON A WRITE
34	1		WRITE PROTECTED DISK
	7	03	FILE ERROR
	7	03	SPECIFIED DISK DRIVE DOES NOT EXIST, I.E. DISK DRIVE IS NOT 1, 2, 3, OR 4
	7	03	ATTEMPT TO CLOSE, READ, OR WRITE A NONEXISTENT FILE
	7	04	FILE/DEVICE NOT OPEN
	7	05	ATTEMPTED TO OPEN AN ACTIVE FILE, I.E. TRY TO OPEN A FILE WHICH IS ALREADY OPEN
	7	1F	FILE NAME IS TOO LONG, I.E. FILE NAME IS MORE THAN 10 CHARACTERS IN LENGTH
	7	1F	INCORRECT FILE NAME, I.E. USED ' ' AS A CHARACTER IN THE FILE NAME
	7	1F	EMPTY FILE NAME
	7	1F	END OF OLD FILE NAME NOT FOUND
	7	1F	OLD FILE NAME TOO LONG
	7	1F	NEW FILE NAME TOO LONG
		08	DATA/FILE TOO LONG
		0A	NOT REQUESTING SERVICE (RESPONSE TO POLL INQUIRY)
		18	VERIFY ERROR
99			COMPREHENSIVE TEST DATA WRITTEN ON TEST 5 COULD NOT BE VERIFIED DURING TEST 6

Compact Computer error codes >50 - >EF are reserved for device dependent errors.

The following table describes in detail the meaning of the right-most digit of the error code generated by BASIC in the Home Computer.

Table 8-3 Home Computer BASIC Error Code Definitions

Error Code	Meaning
0	BAD DEVICE NAME The device indicated is not in the system.
1	DEVICE WRITE PROTECTED
2	BAD OPEN ATTRIBUTE One or more of the given OPEN attributes are illegal or do not match the file's actual characteristics. This could be: * File type * Record length * I/O mode * File organization
3	ILLEGAL OPERATION Either an issued I/O command was not supported, or a conflict with the OPEN mode has occurred.
4	OUT OF TABLE/BUFFER SPACE The amount of space left on the device is insufficient for the requested operation.
5	ATTEMPT TO READ PAST EOF This error may also be given for non-existing records in a relative record file.
6	DEVICE ERROR Covers all hard device errors, such as parity and bad medium errors.
7	FILE ERROR Covers all file-related errors like: program/data file mismatch, non-existing file opened for INPUT mode, etc.

SECTION 9

CATALOG FILE ACCESS

In order to enable access to a disk catalog from a user or application program, the CATALOG file has been added to the disk software.

The CATALOG file is a datafile of the INTERNAL/FIXED type. The record length for this file is 38 bytes. The file can be accessed under the name:

DSKx. or DSK.volname.

as a standard datafile, but without a name.

The CATALOG file contains 128 records of 38 bytes. The data in this file is stored in an INTERNAL format (i.e. a length byte followed by a data-item). Each record contains four of those data-items:

* An ASCII string of up to 10 characters or a null-string
* Three numerics in standard 8-byte floating point notation

Record 0 contains information about the volume itself, whereas records 1 through 127 contain information about specific slots in the catalog. Record 1 contains information about file 1, record 2 about file 2, etc.

The information contained in the records is as follows:

* An ASCII string up to 10 characters in length containing the name of the file in the specified directory slot. For record 0 this is the name of the volume.
* A floating point type code between -5 and +5. A negative value means that the file is write protected. The individual codes are given in Figure 9-1.
* The number of AUs allocated for the file. Record 0 contains the total number of AUs on the disk.
* The number of bytes per logical record. For a program file this entry is 0. Record 0 contains the remaining

number of AUs in this entry.

If a specified catalog slot is empty, the filename will be the null-string, and all numeric entries will contain floating zeroes. The following figure shows the floating point type codes found in the CATALOG file.

Code	Meaning
0	Volume info record or empty catalog entry
1	DISPLAY/FIXED data file
2	DISPLAY/VARIABLE data file
3	INTERNAL/FIXED data file
4	INTERNAL/VARIABLE data file
5	Memory image file (program)

Figure 9-1 CATALOG Type Codes