99/4A DISK PERIPHERAL SOFTWARE OVERVIEW

Colin Hinson

This article was first printed in "International TI Lines" between July and December 1986.

The article draws upon a 1980 internal TI document called "GPL Interface Specification for the 99/4 Disk Peripheral".and other documents listed therein.

The article was reprinted in the magazine of the East Anglia Region 99ers in October 1988-April 1989.

Here is information about the TI99/4a Disk System- data storage on disk, the Device Service Routine, and the use of CPU and VDP Ram.

The Device Service Routine (DSR) ROM in the 99/4a Disk peripheral is designed to give the User access to the disk by means of a system using three different 'levels', which, with the addition of some utility routines gives the User complete access to a normally formatted disk without the need for any knowledge as to how the actual disk controller works.

Each level uses those features implemented at a lower level to add new features, (a sort of 'bui1ding block' system).

LEVEL 1 FEATURES

Communication with the FD1771 chip
Record read/write functions
Disk formatting Functions
Soft error corrections

This level is the only level which must know precisely what the disk hardware is. This allows higher levels to be independent of both the controller chip type, and the rest of the disk controller hardware. Each of the higher levels sees the disk simply as a linear storage device, addressed by disk unit-number, a physical record number, and a read or write operation.

If the disk controller chip is changed (such as the Myarc card) then it should only be necessary to replace this part of the software. A11 the higher levels are designed to be independent of the actual physical disk structure which this level deals with, except for sector size which is assumed to be 256 bytes. Smaller sector sizes could easily be supported by setting up the sectors in such a way that the total adds up to 256 bytes - for instance, if a sector size of 64 is used, each sector requested from a higher level would take up 4 sectors at level 1.

LEVEL 2 FEATURES

A11 level 1 features plus:
Creation and deletion of fi1es
File allocation dynamically extendable
Data accessed by filename and physical record displacement
'Mixed hybrid' File format (see below)

The actual 'file' concept is created at this level, with each File being known by its name and the displacement of the physical record within the file - a physical record being defined as one disk sector (256 bytes).

On each disk is maintained a directory and bit-map of the sectors. This allows for file and record management (i.e. deletion and creation). The file Format available is called the 'mixed hybrid' Format, and is a mixture of contiguous and non-contiguous (fragmented) file formats. A lot of overhead has to be carried by fragmented files in the form of pointers - these pointers are required in case relative access is required to the file and point to each data record of the File.

The files on this level are al1ocated in 'clusters' of contiguous records in order to combine the advantage of the flexible allocation of non-contiguous files with the low overhead, and the easy access of contiguous files. Whenever new records are requested, the clusters are expanded if possible, if a cluster cannot be expanded then a new one is started.

LEVEL 3 FEATURES

A11 level 2 features, plus:
Program and data files
Fixed and variable record formats
Relative and sequential access
Internal and ASCII data types

The disk management software is completed by the addition at this level, of the relative/sequential access and the fixed/variable record formats. This level takes care of the 'blocking' of one or more logical records into a physical record (as with DIS/VAR format). When relative access is required, it computes the physical record in which the logical record is to be found, updates that record and passes the physical record back to the level 2 file update routines.

UTILITY ROUTINES
As you may have noticed, there are some functions which have not been catered for, as they are not part of the normal file I/O system. These are catered for by means of some utility routines. These 'subprograms' are:

Direct level 2 file access
Direct sector (Allocatable unit) access
Modification of file protection
Disk formatting
File rename

Methods of accessing these routines will be described later

99/4a DISK PERIPHERAL - OPERATIONAL INFORMATION

There are three basic methods used to store data on the diskette these are:
'Program' (Memory Image) format
Variable format
Fixed format

Variable and Fixed Format are really ‘variations on a theme', in that each sector (or AU), contains as many records of either format as it is possible to write to that sector without overflowing it (i.e. without writing more than 256 bytes).

Program format is used to store an image of the data in memory on the diskette byte for byte, each byte in each sector being used (except for the last sector associated with the particular file, which may not be fully used due to the length of the file not being exactly divisible by 256).

Methods of access

There are three methods of access, each one being associated with one particular Format described above. The methods of access are (in order):-
1/. Physical I/O
2/. Sequential access
3/. Relative access

Sequential access

When the data records in a file are accessed strictly in the order of increasing address on the medium, each record is said to be ’sequential1y’ accessed. This is the access method used for accessing such things as magnetic and paper tapes etc, in which all the records up to and including the one required must be read in order to access the particular record required.

In this mode of access, the parameters for the data transfer do not specify a physical record number, as it is implied that the logical record currently indicated by some data transfer pointer is the one which is required. Restore/rewind operations are either implicitly done or explicitly done prior to the first data transfer. As each logical record is transferred, the access pointer moves to the first byte of the next logical record (which in case of this DSR is usually the length indicator).

Sequential access methods have the advantage that variable record lengths can be used (such as the well known "VAR 80") and so the number of records per sector can be increased by an amount dependent on the length of each particular record. For instance, ten 24 byte records could be written on a 256 byte sector, whereas if "FIXED 80" were to be used, then only 3 records (=240 bytes) could be written even though there are still only 24 bytes of usable data per record.

Variable format (sequential access) sectors are recorded on the disk with an extra byte added at the start of each record, and a final byte at the end of the last record of the sector.
The first byte of each record indicates the length of the record in bytes, a negative number (usually >FF) indicating that there are no more records on that sector.

The action of the computer when reading the sectors from the data buffer in VDP RAM is to read the first byte (length of record), then read the required number of bytes as the record from VDP RAM to a new location, read the next byte (length of record), etc. etc. until a negative number is found as the length. At this point another sector is read from the disk to VDP RAM and the process repeated again until all the data for the appropriate file has been read.

Relative (Random) access

Relative access allows data in a file recorded in fixed format to be accessed by logical record number. The records may be accessed in any order regardless of the order in which they were written or the order in which they appear in the file.

As the DSR software must be able to locate a record solely by its number, relative access can only be supported on either Indexed Files or Fixed Length files. In this DSR, only "Fixed length" files are supported. (Indexed files are files for which an "Index" is maintained on the diskette through which a particular record can be located by looking it up in the index.)

Physical I/O

Three possible texts here:
[TI Lines text: With the Physical I/O access, the data on the disk is considered to be organised in blocks of 256 bytes by the DSR software. Each byte can be of any value (ie -128 to +127) and no attempt is made to interpret these at data transfer. The existence of records or files is completely ignored by this access method. }
[EAR Text instead reads (ie -255 to +256]
[Original TI document reads: 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.]

You will notice that this method of access is a "Level 1" access. The rest of the disk software (i.e. Levels 2 & 3) reduces all access methods to physical I/O by converting logical record numbers into physical track and sector data. This information is used to specify the disk sector that is to be transferred by the Physical I/O. Physical I/O is not available directly to the User other than in the form of an assembly language sub- program within the DSR. (See later For Sub-Programs).

*

Directory Organisation

The directory organisation implemented within the DSR supports only a single level directory, that is to say that no FILE can be a directory containing pointers to other files. This means that each file on the disk can be readily identified by a single name, E.g.
DSK1.fi1ename which would specify a file called "Filename" on the diskette in drive 1

This approach to the diskette File organisation prevents access to a catalogue file as such on the disk, as no such File can physically exist. In order to overcome this problem, a semi-catalogue file can be created by the DSR software and accessed by the User. The file which is created (and remember it is not physically on the disk, so don't go looking for it with the Disk Manager!), is of the Fixed format, relative access type. The file contains 128 records, each containing information about the associated catalogue entry and is described in detail below. It can be accessed as:-

DSK1. or DSK.volname

as a standard data file but without a name

Please note that not all the file operations have been defined for this catalogue file, and only the standard OPEN, READ, and CLOSE are supported. Other operations such as DELETE, EOF etc are considered to be illegal, and an error will be returned if these operations are used.

Catalogue file access from Basic:

The Catalogue file can be accessed as a read-only file by the Basic User. The file has no name, and is of the INTERNAL, FIXED format type. The file can be opened by for examp1e:-

```
OPEN #1:"DSK.", INPUT, INTERNAL, RELATIVE
```

The record length will automafically be defaulted by Basic to the correct value, so this should not be entered. If however the User wants to specify the length, then it must be specified as 38 - all other lengths will result in an error message.

The Catalogue File acts as if it is Protected, and as mentioned above, it will only allow INPUT access.

The File is written in the normal Basic INTERNAL format, and each record contains four items: one string and three numerics. There are 128 records in the file, and they are numbered 0 through to 127.

Record 0

This record contains data about the volume for which the catalogue file was created. The string gives the name of the disk (up to 10 characters) and the numerical items are as fo11ows:

1/. A1ways 0 (for record 0)
2/. Total number of sectors on the disk
3/. Total number of Free sectors on the disk

Records 1 through to 127

These records contain information on the corresponding File in the Catalogue. Non-existent files will give a null string for the first item and Os (zeros) For the numeric items. Files which exist will give the file name for the string, and the following numeric items:

A/. = Filetype (if number is negative, File is protected) Value:

1 = DISPLAY/FIXED datafile
2 = DISPLAY/VARIABLE datafile
3 = INTERNAL/FIXED datafile
4 = INTERNAL/VARIABLE datafile
5 = Memory Image file (Program File)

B/. = Number of AUs allocated to the File

C/. = Number of bytes per record (0 for type 5 File)

Catalogue file access by application program or User

(Please read the preceding information first)

In order to enable access from assembly language programs, the following additional information is required:

The Catalogue file contains 128 records of 38 bytes and is output in INTERNAL format (i.e. a length byte followed by a data item.). Each of the records contains four of these data items:

i An ASCII string containing up to 10 characters, or a null string
ii Three numeric values in standard 8 byte floating point format

Record 0 contains information about the volume itself, while records 1 through to 127 contain information about the relevant file for each slot" in the catalogue.

The information in the records is as follows:

1. An ASCII string of up to 10 characters containing the name of the file in the specified slot. For record 0 this is the Volume name

2. A floating point value of between -5 and +5. These values represent the same information as given for Basic.

3. The number of AUs allocated for the file (record 0 = total AUs on the disk)

4. The number of bytes per logical record - 0 for Program file.
(record 0 = Free AUs remaining on the disk)

If a catalogue slot is empty, the filename will contain a null string and the numeric entries will contain floating point zeros.

====== ======= ======

INTERNAL DATA STRUCTURE

Physical device format
The physical device (diskette) is logically subdivided into "Allocatable Units" (AU's). An AU is defined as being an integral number of physical records on the device. The total number of AUs on any device is less than 4096 (ie each AU can be addressed by a 12 bit word). The first AU is numbered 0.

The physical record length is the size of the block of data which can be read or written to the device at one time. For the Disk Peripheral, the AU and the Physical Record are equivalent to one disk sector (256 bytes).

Summary of system reserved sectors:

Sector 0 contains data concerning the volume, such as available (free) sectors, disk name etc.

Sector 1 contains pointers to other sectors which contain descriptions of the appropriate file. Normally there is a pointer in sector one for each file which exists on the disk.

Volume information block (VIB), sector 0

This block contains disk configuration data as required by the disk software. This includes available number of AUs, Volume name, format information etc. Included in this block is the "Allocation Bit Map":

The allocation bit map is used to indicate to the disk software the availability of individual sectors on the disk.
A "1" indicates that the sector associated with that "bit" has been allocated, and a "0" that the sector is available for use.
The first bit in the map is for sector 0, the second is for sector 1 and so forth.
When the disk is initialised (WITH VERIFY = YES if using DM1000 or similar), then the bits for bad AUs are set to "1" along with the bits for non-existent AUs and the 2 system reserved AUs. All the remaining bits are of course set to zero.

File Descriptor Index Record (FDI), sector 1.

This sector contains alphabetically sorted pointers to each File Descriptor Record (FDR), and enables the system to keep track of the location of each FDR on the disk.

NOTE: If either Sector 0 (VIB) or sector 1 (FDR) are bad or corrupted then the whole disk is considered bad by the system, as it can no longer keep track of information stored on the disk.

File Descriptor Record. (FDR) (any sector)

This record is used to map filenames into physical locations of the files on the disk. Each entry contains information about the file such as type, record type, data type, size of file etc.

File Control Block (FCB) in VDP RAM

This is a copy of the FDR which is maintained in VDP RAM while the file is open. It may additionally contain some more up to date information about the file. One FCB is required for each file which is currently opened. It is the memory taken by these FCBs which is affected when "CALL FILES" is used in BASIC.

==========

DETAILED DESCRIPTION OF DISK FORMAT

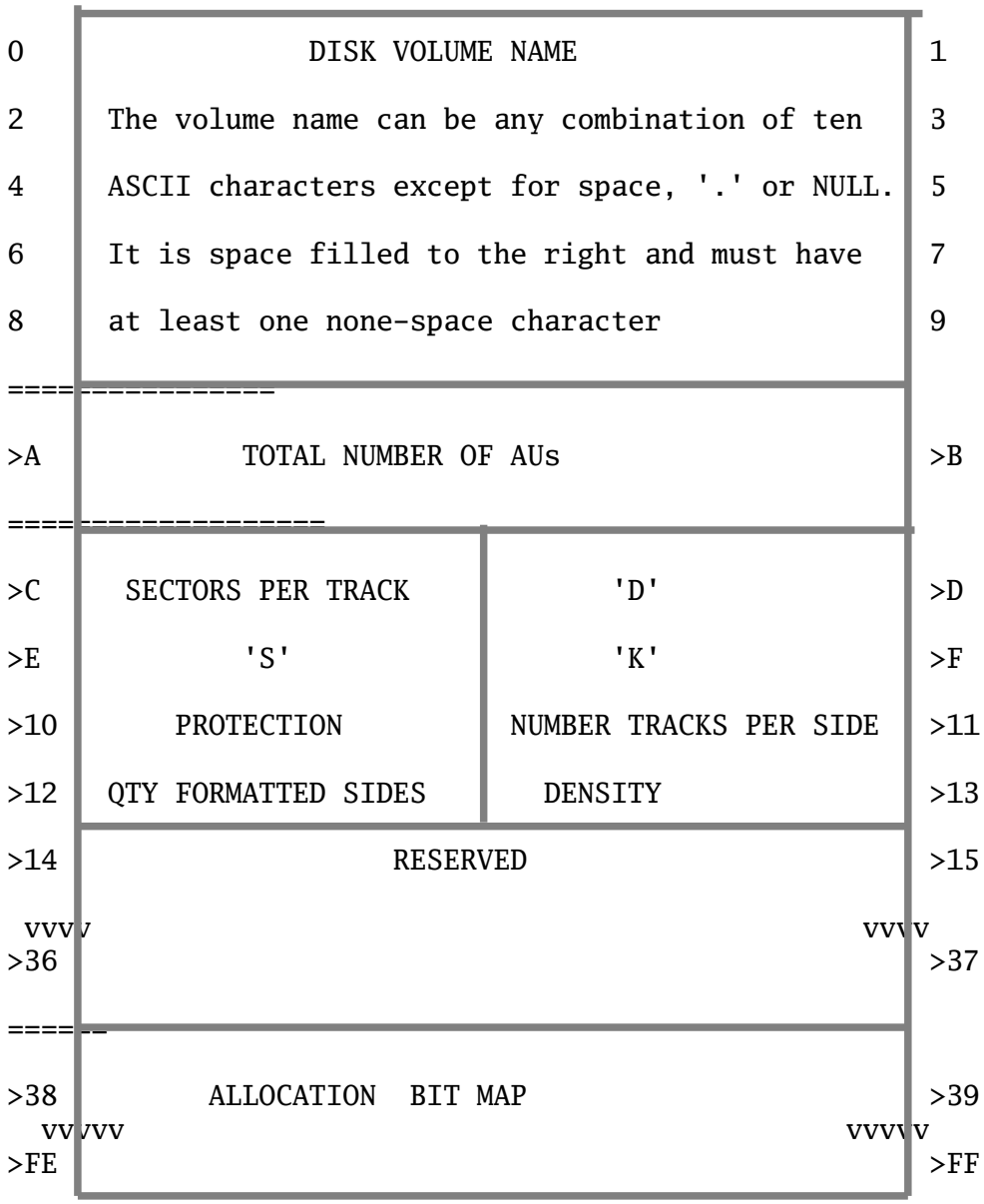
A single sided diskette used with the T.I. Disk Controller has the following specifications:-

```
Diskette type:   SA 104 (ANSI standard 5.25“)
Encoding method: FM single density
Capacity:        92160 Bytes per disk
                 2304 Bytes per track
                 256 Bytes per sector
                 40 Tracks per side
                 9 Sectors per track
```

The capacities given are for a single sided, single density system.
Using double sided will of course double the bytes per disk, using double density (Myarc type controller) will double the capacity again.

For ease of description, the following information assumes that the diskette is addressed as a 'linear' medium, that is to say, sector 0 is the first sector of track zero, sector 1 is the second and so on -- sector 359 being the last sector of track 39. It should be noted that this is not strictly correct as the sectors are in fact 'interleaved' on each track to obtain faster access when reading. If a double sided set up is being used then the physical layout of the second side is the reverse of the first side, that is to say, sector 360 is physically on the opposite side of the disk to sector 359, and sector 719 is opposite sector 0.

VOLUME INFORMATION BLOCK LAYOUT



Bytes >A - >B show the total number of allocation units on the volume. This information should match the Allocation Bit Map data.

Bytes >D - >F contain the ASCII letters ’DSK’. These letters are checked by the T.I. disk managers to see if the disk hasbeen initialised.

Byte >10 Contains the ASCII ’P’ if the disk is Proprietary Protected. This byte will normally otherwise be an ASCII space.

Bytes >12 - >37 are reserved for what were intended to be future expansion. This could be date and time of creation etc. The T.I. controller will set all these to zero.

Bytes >3B - >FF contain the allocation bit map. The map can control up to 1600 256-byte sectors (=400K bytes) - this will allow double sided, double density diskettes without modification to the map layout. Each bit in the map represents one sector on the disk. A logical one in the bit map means that the corresponding sector has been allocated, a logical zero means that the sector is available for use.

The Volume name can be used as an alternative to the drive name - that is to say the User can specify a disk drive in either of the following ways:

DSK.vo1name.fi1ename or DSK1.fi1ename

If the volume name is specified, then the system will look at each drive in sequence until it finds the specified volume. If more that one drive contains a volume with that name, then the lowest drive number will be assigned.

FILE DESCRIPTOR INDEX RECORD. (Sector 1)

This sector contains up to 127 two byte entries. Each of these points to a File Descriptor Record, and are alphabetically sorted according to the file name in the File Descriptor Record. The list starts at the beginning of the block, and ends with a zero entry.

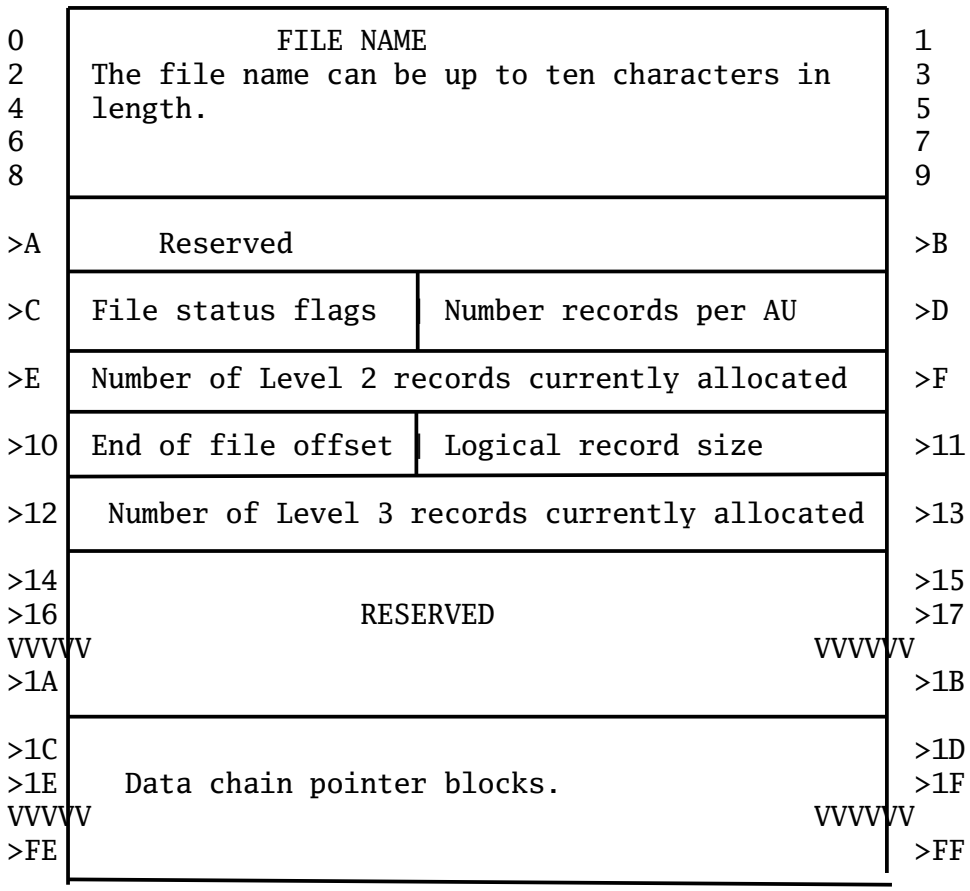
As the file descriptors are alphabetically sorted, a binary search can be used to find any given filename. This limits the maximum number of searches to 7 if more than 63 files are defined. Generally if between 2^(N-1) and 2^N files are defined, a file search will take at the most N disk searches. To obtain faster directory response times, data blocks are normally allocated in the area above sector >22, the area below this being used for File descriptors and only used for file data when there are no more sectors available above >22.

File Descriptor Records

-

The File Descriptor Record (FDR) contains the general information for its associated file. In order for the system to function, all the information to access and update the file must be contained within this record.

Layout of an FDR is as follows



Bytes >A and >B were reserved for an extension of the number of data chain pointers. As this was never implemented, these bytes are always 0.

Byte >C. The file status flags are as follows, where bit 0 is the LSB:

Bit No	Description
0	File type indicator 0 = Data file 1 = Program file
1	Data type indicator 0 = ASCII data (DISPLAY file) 1 = Binary data (INTERNAL or PROGRAM file)
2	This bit was reserved for expansion of the data type indicator
3	PROTECT flag 0 = NOT protected 1 = Protected
4,5,6	These bits were reserved for expansion of ????
7	Fixed/variable flag 0 = Fixed record lengths 1 = Variable record lengths

Bytes >E & >F The number of 256 byte records allocated on level 2.

Byte >10 Contains the EOF offset within the highest physical AU for variable length records and program files.

Byte >11 Contains the logical record size in bytes.
For variable length records this byte contains the maximum permissible record size.

Bytes >12 & >13 Contain the number of records allocated on level 3.
For variable length records these bytes will contain the number of level 2 records actually used.
NOTE! these bytes are in the reverse order.

Bytes >14 to >1B These bytes were reserved For future expansion
and will always be 0
.

Bytes >1C to >FF Contain blocks of three bytes which indicate the clusters which have been allocated for the file

12 bits of each 3 byte block indicate the address of the first AU in the cluster, and the remaining 12 bits indicate the highest logical record offset in the cluster of contiguous records.
(This method of indication reduces the computation required for relative record file access).

The layout of the data within the 3 byte blocks is shown below:

BYTE 1		BYTE 2		BYTE 3	
A2	A1	L1	A3	L3	L2

Where
A3 = AU's times >100
A2 = AU's times >10
A1 = AU's times 1

L3 = offset times >100
L2 = offset times >10
L1 = offset times 1

ALLOCATION OF DATA FILES

A Data file is built of clusters of contiguous data records, each data file can contain up to 76 of these data record clusters, with each cluster containing at least one data record.
The DSR 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 cluster, then a new cluster of contiguous records is started. If 76 clusters have been allocated and a new cluster is requested then the write operation will be aborted.
This will only occur when the data records on the disk have become too scattered (i.e. the file is badly segmented) - the problem can be corrected by copying the disk with the Disk Manager (or with DM1000 in file mode), which will cause the records for the files to be allocated in 1 (or at the most 2) clusters on the new disk.
Note that at worst case this scheme still allows for 19k bytes per file (76 * 256 bytes).

Because of the system used, each physical record within the file can be accessed at random, without any need for large 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 at random or sequentially and therefore the disk software does not have to distinguish between relative record and sequential Files.

This has some implication for sequential fixed length record access, as 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 is stored at the start of the record itself. The result of this is that since a record cannot cross an AU (sector) boundary, the maximum record length for variable length records is limited to 254 bytes, as the 'end of records on this sector' (>FF) has to be written too.

PROGRAM FILE ALLOCATION

Program file allocation is identical to data file allocation. The Program file (segment) is split into 256 byte records which are stored as a standard data file. As the disk software marks the file as a program file it can prevent data access to program Files (and vice versa).

In order to prevent problems with VDP ram ‘wrap round' (i.e. continuing to write to VDP ram after address >3FFF will write to >OOOO) the DSR software notes the actual number of bytes used in the last data record and will return exactly as many bytes as were originally written to the disk, even though this may not be a multiple of 256.

When accessing the disk from a high level (i.e. using file access as opposed to sector access), current information about the disk and fi1e(s) etc are held in VDP RAM so as to avoid having to re-access the relevant sectors on the disk each time a sector is read from a file.
In addition to this data space is also required to buffer the data being read from the disk. The memory used for this is at the high end of VDP RAM and is permanently allocated by the power up routine within the DSR ROM of the disk controller, and as a consequence cannot be used by application programs, although its size can be changed by a (GPL) utility routine.

The allocated VDP memory is roughly sub-divided into three categories:

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

Each of these is discussed below

DRIVE CONTROL INFORMATION

In order to control the Disk hardware, the software needs to know the current status of each drive before it can access it. A11 this information is readily available (some through checking the actual current status of the drive directly).

The power up routines take care of the FD1771 register intialisation

FILE ALLOCATION INFORMATION

File allocation information is held in the File Control Blocks (FCBs), each open file having an FCB associated with it.

The information contained in the FCB is identical to the File Descriptor Record (FDR) data held on the disk for each file, with the addition of 6 bytes of dynamic information about each file.
These six bytes are stored in front of the FDR informat1on (i.e. the FDR starts at FDB byte 6).
As the length of the FDR is 256 bytes and there is a data buffer for each file of 256 bytes, the total length of an FDB is therefore 256 + 256 + 6 = 518 bytes.

The format of the 6 byte extension of the FDR is:
Bytes -6 and -5 = Current logical Record offset on Level 2
Bytes -4 and -3 = Physical Record location of the FDR
Byte -2 = Logical Record offset
Byte -1 = Drive ID

The meanings of these entries are as follows

Drive ID

Contains the Drive number on which the associated file resides. If the highest bit of this byte is set, then the current data block has been modified and it will have to be written back to the drive before the file is closed or a new data block is accessed.

Logical Record offset

Contains the offset of the next logical record in the current physical record. If during a READ operation this points to a byte entry of >FF then this indicates an end of record for the current physical record.

Note that this entry is used only for variable length records. For fixed length records, the actual AU and the position within that AU is computed before each I/O operation, and therefore the logical record offset byte is irrelevant.

During WRITE operations, this offset 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.
The first logical record in a physical record can never cause the physical record to overflow as the maximum logical record length is 254 and the physical record length is 256

Physical location of the FDR:
Points to the physical sector on the disk where the FDR resides for the associated file, and is used when it is necessary to re-write the FDR to the disk. It is maintained on read only accesses even though it is not required.

Current Logical Record Offset on Level 2:
Contains the physical record offset of the most recently processed physical record and is independant of READ or WRITE operations.
Always contains the logical offset for Level 2 of the datablock which is currently in memory.

It should be noted that this system causes fixed length sequential files to be accessed as relative access files on Level 2.

DATA BUFFERING

In order to buffer the data to and from the disk, a 256 byte buffer is maintained for each OPEN File. The buffer is located immediately above the fCB memory area.

One of the VDP RAM buffers is permanently assigned for processing VIBs (Volume Information Blocks - see previous information. If more than one drive is used in WRITE mode, then the bit maps are moved in and out of this area as required. This buffer is accessed for each access to the disk VIB.

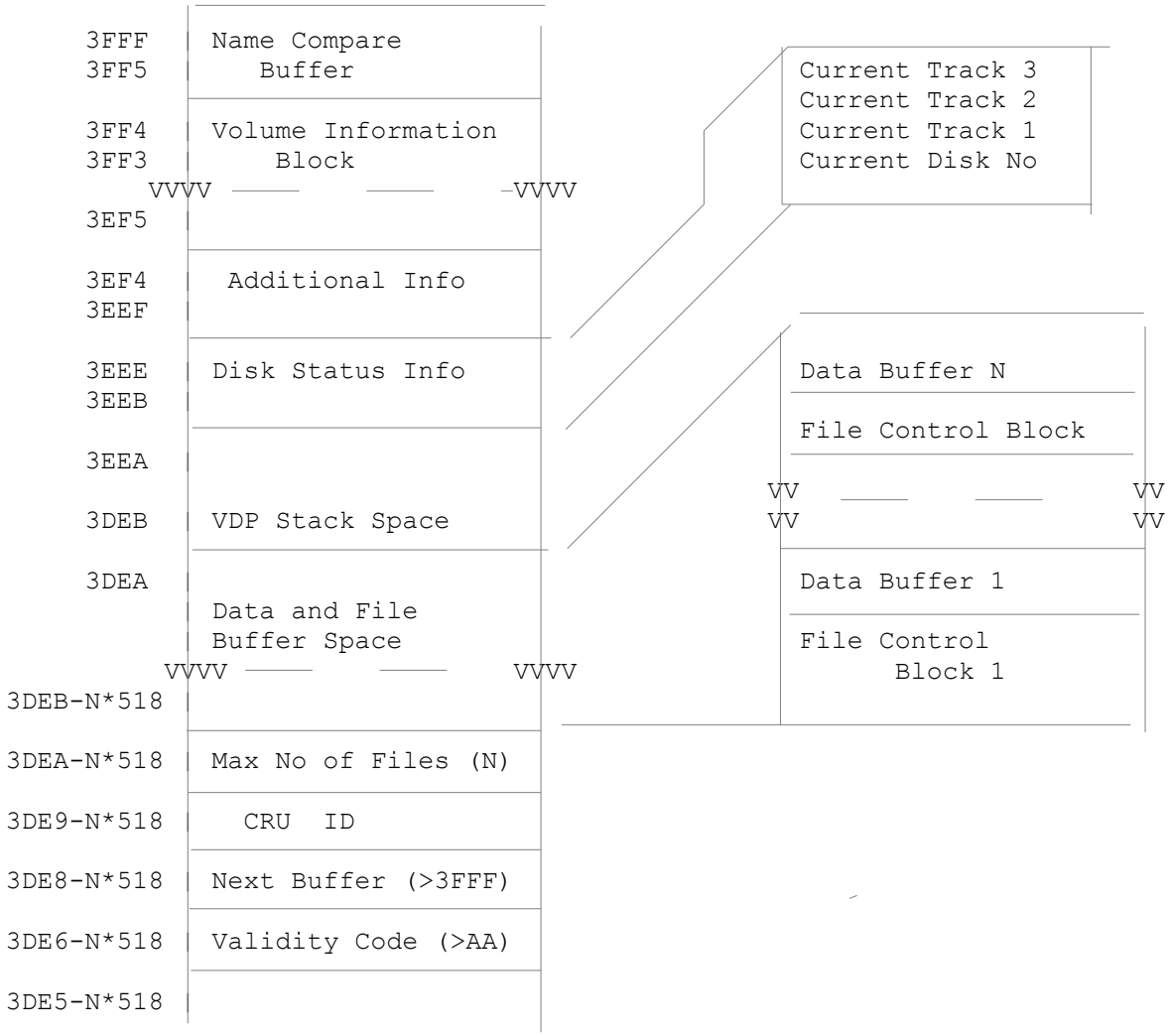
Every Level 3 WRITE operation to a file will eventually be passed on to Level 2 as a physical sector WRITE. To keep the number of disk accesses to a minimum, a flag (the MS bit of the Drive ID) to indicate that the current data buffer has been modified. The data buffer is only written to the disk if the next physical record access involves another physical record. If the file is closed then the last data buffer is written onto the disk if required (this is the reason why the ends of files go missing if you forget to close them!).

VDP MEMORY LAYOUT

The memory layout is outlined in the diagram (such as it is!) below. This block of memory is reserved by the power up routine in the Disk DSR ROM. The size of the area from then on depends upon the number of files which are allowed to be open at the same time, and is initially set to 3. This number can be varied between one and 16 by calling a subprogram (through CALL FILES from BASIC). Each extra File will of course take up 518 bytes.

As for each peripheral, the disk peripheral identifies its reserved area through its CRU address (unique for each peripheral). The area is 'validated' by an >AA byte, followed by the address of the previous top of memory. As the disk peripheral has the highest priority on power up, this entry will always point to the actual top of memory. The disk system does not use this however, and so will work equally well on other CRU locations.

The First entry after the CRU ID contains the number of Files for which the area is reserved, and directly determines the length of the reserved area. After this entry come the areas reserved for the FCBs and the associated buffers for each file. In order to simp1ify(?) the buffer allocation buffers are not allocated on demand, but as soon as a file is opened. The FCB and buffer are associated with the file for its entire 'open' life.



The VDP Stack area is used to simulate a stack based machine with the TMS 9900, giving the programmer the advantage of being able to use the multilevel stack oriented CALL/RETURN system, rather than the single level BL system used by the 9900 series processors. The stack can of course also be used to PUSH and POP registers and data to and From it

The disk status information area is used to save the current track numbers of the (3) drives, and the most recently accessed drive number.

The additional information area is no longer used (by the 99/A)

The Volume Information Block buffer is described above.

At the top of memory, an 11-byte buffer is reserved which is used for name comparison. Every high level entry point automatically saves the drive number and the 10 character file name in this entry. If less than 10 characters are available, the buffer is padded with spaces

DISK PERIPHERAL D. S. R. SUBPROGRAMS

LEVEL 1 SUBRDUTINES

The lowest routines in the disk DSR are called level 1 subroutines. These routines make the higher levels independent of the physical disk medium, e.g. changing the disk software for a double density disk would only involve changing the routines on this level provided the physical sector size remains at 256 bytes.

There are two Sub-programs available on this level:

1 Sector Read/Write
2 Format disk

The following paragraphs contain a description of the subprograms and their call requirements. A11 parameters are transferred through the FAC block in CPU RAM. This block is located at a relative position of >4A (which for the 99/4A is >834A).

A11 the subprogvams are called by a 'BLWP @DSRLNK' followed by a data statement: 'DATA >A'. (Note that the Editor Assembler manual is WRONG - it gives >1O for the data on the third line of the first paragraph on page 262).
Before calling any of the subprograms, location >8356 (name length pointer) must be set up to point at the location in VRAM where the name length and the subprogram number have been written.

e.g. If subprogram >14 is called then a location in VRAM (say >1000) must contain 2 bytes, the first of which is >O1 (the name length), and the second of which is the subprogram number >14. Location >8356 in CPU RAM points to the first of these bytes - i.e. contains >1000.

Error codes are returned in >8350

SECTOR READ/WRITE - SUBPROGRAM >10

The transfer block for this subprogram:

```
>834A       (Sector Number)
>834C   Unit #       |   READ/WRITE
>834E   VDP Buffer start address
>8350         Sector number
```

The meaning of each entry is:

```
Unit Number: - Indicates the disk drive on which the operation is to
               be performed. For a T.I. controller, this has to be
               either 1, 2, or 3
READ/WRITE   - Indicates the direction of data flow:
               0 =  WRITE
           NOT 0 = READ

VDP buffer start address. - Indicates the start of VDP buffer for
               data transfer. The number of bytes transferred will
               always be 256
```

Sector number - Number of the sector to be written or read. Sectors are addressed as logical sectors (0-359 for a single sided single density disk), rather than as a track and sector number, which would require a knowledge of the physical layout of the floppy disk. The sector number has to be given in CPU RAM locations >8350 and >8351, and will be returned in CPU RAM locations >834A and >834B

.

DISK FORMATTING - SUBPROGRAM >11

The transfer block for this subprogram:

```
      >834A        (# of sectors/disk)
      >834C  DSR ver  |  Unit #1   |  # of tracks
      >834E       VDP Buffer start address
      >8350     Density     |     # of sides
```

The meaning of each entry is:

# of sectors/disk - Is returned by the routine to provide compatibility between the normal controller and double density or SA200 systems.

DSR Version (This is the Most Significant nibble)

0 indicates the format reAuires nothing special and can be done on any version of the DSR

1 indicates the format requires the 2nd version of the DSR for one of two reasons. It may be because a double sided format is requested, or it may be because a # of sectors other than 35 or 40 is requested (but see below!)

Unit Number: - Indicates the disk drive on which the operation is to be performed. For a T.I. Controller, this has to be either 1, 2, or 3. This is the Least Significant nibble.

# of tracks: - Indicates the number of tracks to be formatted. In the only versions released, this entry has to be either 35 or 40!!! Upon return, this entry contains the number of sectors per track.

VDP buffer start address - Indicates the start address of the VDP buffer that can be used by the disk controller to write tracks The amount of memory used depends on the disk format. For a single density format, the buffer memory used is a nominal 3125 bytes. This can vary with disk motor speed to a maximum of 3300 bytes. To be compatible with double density versions of the controller (such as MYARC), the minimum buffer size must be 8K bytes.

Density - 0 = single

% of sides - Indicates the number of sides to format.

The above subprogram will format the entire disk on the given unit unless the disk in the unit has been hardware write protected. It can use any VDP memory starting at the location given in the transfer block.

LEVEL. 2 SUBROUTINES

The Level 2 subroutines use the "file" concept, rather than the "logical sector number". Note that the file concept on this level is limited to an abstract type of file which has NO properties such as "program file" or "data file". A file on this level is merely a collection of data, stored in logical blocks of 256 bytes each.

The logical blocks on this level are accessed by filename and logical block offset. This offset starts with block 0 and ends with block N-1 for a file with a length of N blocks.

MODIFY FILE PROTECTION - SUBPROGRAM >12

The transfer block for this subp*ogr‘avn is

```
>834C   UNIT #    |    Protect Code
>834E     Pointer to file name
```

The protect bit For the indicated file will be set or reset according to the information given in CPU RAM location >4D:

0 -Reset the file protect bit. The file is no longer protected against modification or deletion

1 -Set the file file protect bit. Disallow SAVE and OPEN for OUTPUT, APPEND, or UPDATE mode.

The pointer to the file name must point to the VDP RAM location of the first character of the file name. The name must be left adjusted in a 10 character field, right filled with spaces. No checks are made to ensure the legality of the file name.

FILE RENAME ROUTINE - SUBPROGRAM >13

The transfer block for this subprogram is

```
   >834C     UNIT #   |   Unused
   >834E      Pointer to new name
   >8350      Pointer to old name
```

Both pointers to the File names must point to the VDP RAM location of the first character of a file name. Each name must be left adjusted in a 10 character field, right filled with spaces. No checks are made to ensure the legality of the file names.

Since the rename has to be done on the same disk, only one unit number entry is required

Error codes are returned, as usual, at location >8350. The error codes returned are identical to the standard file management error codes, i.e. only the upper three bits of the error byte are significant

* Subprograms 14 and 15 deal with Direct File Access and can be found after Subprogram 16 data. This reflects the order the article was printed in TI Lines.

BUFFER ALLOCATION ROUTINE - SUBPROGRAM >16

The transfer block for this subprogram is:

```
   >834C            ZERO
   >834E  # of files |  ZERO
   >8350           ZERO
```

The "argument" for this subprogram is the number of file buffers to be allocated. whilst it is not actually necessary to clear >834C, & >8350, experience has shown that strange results sometimes occur if you don't.

The effect of this routine is that an attempt is made to allocate enough VDP RAM space for disk usage to facilitate the simultaneous opening of the given nuumber of files. This number has to be between 1 and 16.

The disk software automatically relocates all buffer areas that have deem linked in the following manner (see also previous text VDP RAM a11ocation):-

Byte 1 - Validation code
Bytes 2 & 3 - Top of memory before allocation of this buffer
Byte 4 - High byte of CRU address for given buffer area. For programs this byte is 0

The linkage to the first buffer area is made through the current top of memory, given in CPU RAM location >8370

The top of memory is also automatically updated after successful completion of this subprogram

A check is made that the current request leaves at least >800 bytes of VDP RAM space for screen and data storage. If this is not the case or if the total number of buffers is zero or greater than 16, the request is ignored and an error code will be indicated in CPU RAM location >8350.

Successful completion is indicated by a 0 byte in CPU RAM location >8350. A non-zero byte indicates unsuccessful completion

DIRECT FILE ACCESS ROUTINES

The direct File access routines can be used for accessing disk files without paying attention to the type of disk File (PROGRAM or DATA). The level of access is equivalent to the level 2 disk software, which means that access is performed on the basis of straight AUs. However, Level 3 information can be passed at file open time.

Since the input and output direct access subprograms can be used together to copy files, the user has to be very careful with the information returned by the input File subprogram, since some of this information may be used by the output file subprogram.

Direct File Input - Subprogram >14

The transfer block for the subprogram is:

```
>834C    Unit #   |  Access code
>834E     Pointer to file name
>8350 (X)  Additional info
```

The meaning of each entry is:

Unit # - Indicates the disk drive on which the operation is to be performed. This entry has to be either 1, 2 or 3.

Access code - An access code is used to indicate which Function is to be performed, since this subprogram combines multiple functions. The following codes are used!

0 Transfer file parameters. This will transfer Level 2 parameters to the additional information area (six bytes). It also passes the number of AUs allocated for the file.

N when N is not equal to zero, this indicates the number of AUs to be read from the given file, starting at the AU indicated in the additional information block.

After the READ is complete, this entry contains the actual number of AUs read. If all AUs have been read the entry will be zero.

Pointer to file name. - Contains a pointer to the first character of a 10 character filename, possibly padded to the right with spaces. The filename is NOT checked by the disk software.

Additional information - Points to a 10 byte location in CPU ram containing additional information for direct disk access:

```
X        VDP Buffer Start Address
X+2          # of first AU
X+4    Status flags   |   # records/AU
X+6    EOF offset     | logical rec Size
X+8     # of Level 3 rec's allocated
```

The VDP Buffer start address indicates where the information read from the disk can be stored. The buffer has to be able to store at least N * 256 bytes, in which N is the access code
The # of first AU entry indicates the AU number at which the read should begin. If the access code = 0 (parameter passing) the total number of AUs allocated for the File will be returned.

The remaining six bytes are explained previously when giving information about the the File Descriptor records (Sector 1 on the disk).

The user must be very careful when changing these bytes, since they directly affect Level 3 operation. If the information in these 6 bytes is not modified consistently then unpredictable results may (WILL if you don't want them to!) occur.

Error codes are returned at location >8350 in CPU RAM

Direct File Output - Subprogram >15

The transfer block for the subprogram is:

```
>834C     Unit #   |   Access code
>834E       Pointer to file name
>8350 (x) Additional info
```

The meaning of each entry is:

Unit # - Indicates the disk drive on which the operation is to be performed. This entry has to be either 1, 2, or 3

Access code - An access code is used to indicate which function is to be performed, since this subprogram combines multiple functions. The following codes are used:

0 - Create file and copy Level 3 parameters from the additional information area

N - when N is not equal to zero, this indicates the number of AUs to be written to the given file, starting at the AU indicated in the additional information block

Pointer to file name. - Contains a pointer to the first character of a 10 character filename, possibly padded to the right with spaces. The filename is NOT checked by the disk software.

Additional information - Points to a 10 bytes location in CPU ram containing additional information For
direct disk access:

```
X      VDP Buffer Start Address
X+2       # of first AU
X+4  Status flags  |   # records/AU
X+6  EOF offset    |  logical recd Size
X+8   # of Level 3 records allocated
```

The VDP Buffer start address indicates where the information to be written to the disk is stored. The buffer must of course contain at least N * 256 bytes, in which N is the access code

The # of first AU entry indicates the AU number at which the write should begin. If the access code = 0 (parameter passing) the total number of AUs allocated for the file has to be indicated. (automatically passed if a read has been done first).

The remaining six bytes are explained in a previous section giving information about the the File Descriptor records (Sector 1 on the disk).

The user must be very careful when changing these bytes, since they directly affect Level 3 operation. If the information in these 6 bytes is not modified consistently then unpredictable results may (WILL if you don't want them to!) occur.

Error codes are returned at location >8350 in CPU RAM

======= end ============

Supplemental information from the Editor Assembler Manual:

Section 18: File Management - page 291 onwards:

18.1 FILE CHARACTERISTICS

A file consists of a collection of data groupings called logical records. These records do not necessarily correspond with the the physical divisions of the data in the file.
For example, a logical record often does not correspond to a sector on a diskette.
File input and output (I/O) are done on a logical record basis. Manipulation of physical records is handled by the DSR.

The records on sequential files can only be read from, or written to, in seAuential order. This is appropriate for printers, modems, cassettes, and some kinds of data files. The records on sequential files can be of either fixed or variable length.

The records on relative files can be read from, or written to, in either sequential order or in random order. You can only use relative files on diskettes. The records on relative files are of fixed length.

Each record on a file has a number from zero up to one less than the number of records in the file. You use these record numbers to specify which record to access on relative files.

When a file is created, its characteristics must be defined. Most of these characteristics cannot be changed later in the file's existence. The characteristics of files are discussed below.

18.1.1 File Type--DISPLAY or INTERNAL

The file type attribute specifies the format of the data in the file.

DISPLAY sets the file type to contain displayable or printable character strings. Each data record corresponds to one print line.
INTERNAL sets the file type to contain data in internal machine format.

The file type attribute is not significant to the DSR. It is merely passed without affecting the actual data stored.

18.1.2 Mode of Operation--INPUT, OUTPUT, UPDATE, or APPEND

A file is opened for a specific mode of operation.

INPUT specifies that the contents of the file can be read from but not written to.
OUTPUT specifies that the file is being created. Its contents can be written to but not read from.
UPDATE specifies that the contents of the file can be both written to and read from.
APPEND specifies that data can be added to the end of the file but data cannot be read.

The DSR determines whether a specific mode for an I/O operation can be accepted by the given device. For example, the TI Thermal Printer can only be opened in OUTPUT mode.

18.2 PERIPHERAL ACCESS BLOCK (PAB) DEFINITION

DSRs are accessed through a Peripheral Access Block (PAB). The format of the PAB is the same for every peripheral. In a program that you write, the only difference between peripherals is that some of them do not allow every option provided for in the PAB. An example of using a PAB is given in Section 18.3.

The PABs are in VDP RAM. They are created before an OPEN statement and are not released until the I/O for their corresponding peripheral has been closed.

The following describes the bytes which make up a PAB.

Byte	Bit	Contents	Meaninq
0	All	I / O Op-code	The op-code for the current I/O call. See Section 18.2.1 for a description of the op-codes.
1	All	Flag/Status	All information the system needs about the file type, mode of operation, and data type. The meaning of the bits is described below.
	0-2	Error code	No error is 0 Other errors are indicated in combination with the I/O op-code. The error codes are discussed in Section 18.2.2.
	3	Record type	"Fixed length records" are 0 and "variable length records" are 1 .
	4	Datatype	DISPLAY is 0 and INTERNAL is 1
	5,6	Mode of operation	UPDATE is 00, OUTPUT is 01, INPUT is 10, and APPEND is 11
	7	File type	"Sequential file" is 0 and "relative file" is 1.
2,3	All	Data Buffer	The address of the data buffer that the data Address must be written to or read from in VDP memory.
4	All	Logical Record Length	The logical record length for fixed length records or the maximum length for a variable length record.
5	All	Character Count	The number of characters to be transferred for a WRITE op-code or the number of bytes actually read for a READ op-code.
6,7	All	Record Number	(Only required for a relative record type file.) The record number on which the current I/O operation is performed. The most-significant bit is ignored, so this number can be from 0 through 32767.
8	All	Screen Offset	The offset of the screen characters with respect to their normal ASCII value. This is used only by the cassette interface, which must put prompts on the screen.
9	All	Name Length	The length of the file descriptor, which starts in byte 10
10+	All	File Descriptor	The device name and, if required, the filename and options. The length of this descriptor is given in bytes.

.
The following figure summarizes the bytes which make up a PAB.

```
       0     I/O Opcode            |  Flag/Status        1
       2               Data Buffer Address               3
       4   Logical Record Length   |  Character Count    5
       6                   Record Number                 7
       8   Screen Offset           |  Name Length        9
      >A....          File Descriptor....
```

Errors that occur in input/output calls are returned in byte 1 (Flag/Status) of the PAB.

18.2.1 Input/Output Op-codes

The following describes the op-codes which can be used in byte 0 (I/O Op-code) of
the PAB.

18.2.1.1 OPEN--0

The OPEN operation must 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 must be given in byte 1 (Flag/Status) of the PAB. Changing the mode of operation after an OPEN causes unpredictable results.

If a record length of 0 is given in byte 4 (Logical Record Length) of the PAB, the assigned record length (which depends on the peripheral) is returned in byte 4. If a non-zero record length is given, it is used after being checked for correctness with the given peripheral.

18.2.1.2 CLOSE--1

The CLOSE operation closes the file. If the file was opened in OUTPUT or APPEND mode, an End of File (EOF) record is written to the device or file before closing the file.

After the CLOSE operation, you can use the space allocated for the PAB for other purposes.

18.2.1.3 READ--2

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 byte 4 (Logical Record Length) of the PAB. The actual number of bytes stored is specified in byte 5 (Character Count) of the PAB.
If the length of the input record exceeds the buffer size, the remaining characters are discarded.

18.2.1.4 WRITE--3

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

18.2.1.5. RESTORE/REWIND--4

The RESTOREIREWIND operation repositions the file read/write pointer 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.

The RESTORE/REWIND operation can only be used if the file was opened in INPUT or UPDATE mode. For relative record files, you can simulate a RESTORE in any 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 operation then uses the indicated record.

18.2.1.6. LOAD--5

The LOAD operation loads a memory image of a file from an external device or file into VDP RAM. The LOAD operation is used without a previous OPEN operation.
N o t e that the LOAD operation requires as much buffer in VDP RAM as the file occupies on the diskette or other device.

For a LOAD operation, the PAB needs the op-code in byte 0 (I/O Op-code), the starting address of the VDP RAM memory area into which the file is to be copied in bytes 2 and 3 (Data Buffer Address), the maximum number of bytes to be loaded in bytes 6 and 7 (Record Number), the name length in byte 9 (Name Length), and the file descriptor information in bytes 10+ (File Descriptor).

For related information, see the explanation of the RUN PROGRAMFILE option from the Editor/Assembler selection list in Section 2.5.(refer to full Editor Assembler Manual)

18.2.1.7 SAVE--6

The SAVE operation writes a file from VDP RAM to a peripheral. The SAVE operation is used without a previous OPEN operation. Note that the SAVE operation copies the entire memory image from the buffer in VDP RAM to the diskette or other device.

For a SAVE operation, the PAB needs the op-code in byte 0 (I/O Op-code), the starting address of the VDP RAM memory area from which the file is to be copied in bytes 2 and 3 (Data Buffer Address), the number of bytes to be saved in bytes 6 and 7 (Record Number), the name length in byte 9 (Name Length), and the file descriptor information in bytes 10+ (File Descriptor).

For related information, see the explanation of the SAVE utility in Section 24.5.

18.2.1.8 DELETE--7

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

18.2.1.9 SCRATCH RECORD-8

The SCRATCH RECORD operation removes the record specified in bytes 6 and 7 (Record Number) from the specified relative record file. This operation causes an error for peripherals opened as sequential files.

The status is in byte 8 (Screen Offset) of the PAB. The status byte returns the status of a peripheral and can be examined at any time. All of the bits have meaning if the file is currently open. Bits 6 and 7 only have meaning for files that are currently open. Otherwise, they are reset. The bits return the information shown below.

Bit	Information
0	If set, the file does not 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	If set, the file i s protected against modification. If reset, the file is not protected.
2	Reserved for possible future use. Fixed to 0 by the current peripherals.
3	If set, the data type is INTERNAL. If reset, the data type is DISPLAY or the file is a program file.
4	If set, the file is a program file. If reset, the file is a data file.
5	If set, the record length is VARIABLE. If reset, the record length is FIXED.
6	If set, the file is at the physical end of the peripheral and no more data can be written.
7	If set, the file is at the end of its previously created contents. You can still write to the file (if it was opened in APPEND, OUTPUT, or UPDATE mode), but any attempt to read data from the file causes an error.

18.2.2. Error Codes

Errors are indicated in bits 0 through 2 of byte 1 (Flag/Status) of the PAB. An error code of 0 indicates that no error has occurred. However, an error code of 0 with the COND bit (bit 2) set in the STATUS byte at address >837C indicates a bad device name.

The table below shows the possible error codes and their meanings.

Error Code	Meaninq
0	Bad device name.
1	Device is write protected.
2	Bad open attribute such as incorrect file type, incorrect record length, incorrect I/O mode, or no records in a relative record file.
3	Illegal operation; i.e., an operation not supported on the peripheral or a conflict with the OPEN attributes.
4	Out of table or buffer space on the device.
5	Attempt to read past the end of file. When this error occurs, the file is closed. Also given for non-extant records in a relative record file.
6	Device error. Covers all hard device errors such as parity and bad medium errors.
7	File error such a s program/data file mismatch, non-existing file opened in INPUT mode, etc.

18.2.3. Device Service Routine Operations

Device Service Routines (DSRs) react in specific ways to various operations and conditions. These reactions are described in the following sections.

18.2.3.1 Error Conditions

If a non-existent DSR is called, the File Management System returns with the COND bit (bit 2) set in the STATUS byte at address >837C.

If the DSR detects an error, it indicates the error in bits 0 through 2 of byte 1 of the PAB. Therefore, your assembly language program must clear these bits before every I/O operation and check them after every I/O operation.

18.2.3.2 Special Input/Output Modes

The DSR uses only the first part of the file descriptor in its search for the requested peripheral. The remainder of the descriptor can be used to indicate special device-related functions such as transmission rate , print width, etc. The DSR ignores descriptor portions that it does not recognize.

An example of a special I/O mode descriptor that sets values for the RS232 Interface is:

```
    RS232.BAUDRATE=1200.DATABITS=7.CHECKPARITY.PARITY=ODD
```

18.2.3.3. Default Handling

The DSR has certain defaults that are used if no values are specified. The following shows these defaults.

Possibilities	Default
Sequential or relative	Sequential.
UPDATE, OUTPUT , INPUT, or APPEND	UPDATE.
DISPLAY or INTERAL	DISPLAY.
Fixed or variable length	Fixed if relative and variable if sequential.
Logical record length	Depends on the specific peripheral.

18.2.4 Memory Requirements

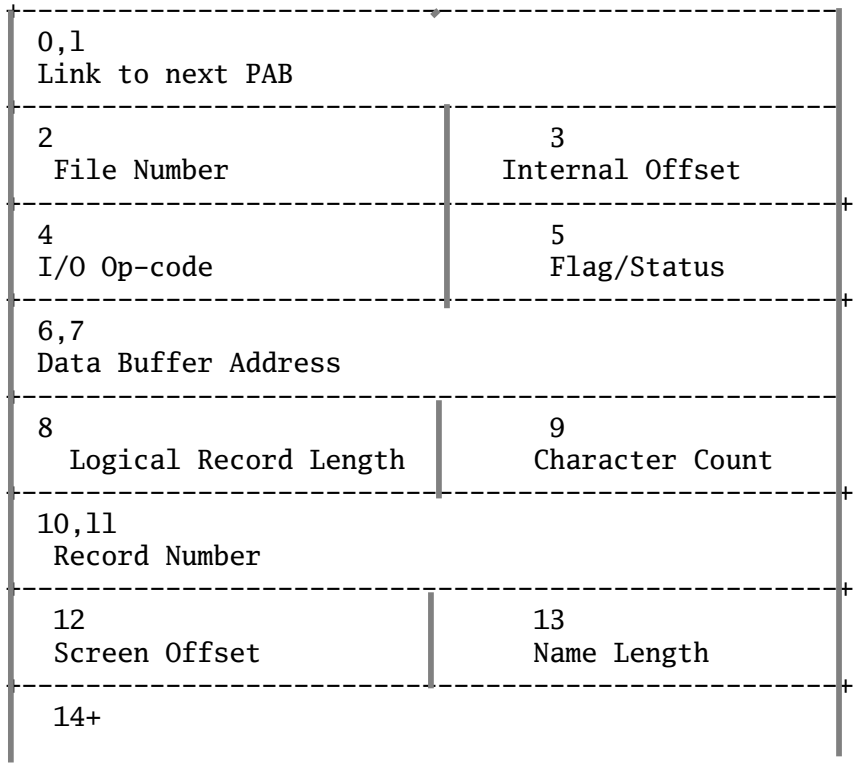
The DSR uses Registers 0 through 10 of the calling Workspace and addresses >834Athrough >836D. If the DSR is calle d in a non-interrupt driven mode (for example , through a standard DSR entry) , addresses >83DA through >83DF are used. Also used are PAD (See Section 24.3.1) and VDP RAM .

18.2.5 Linkage to TI BASIC

When using TI BASIC, the PAB is modified by the addition of four bytes at the beginning of the PAB. The list below describes the bytes which make up a PAB when it is called from TI BASIC.

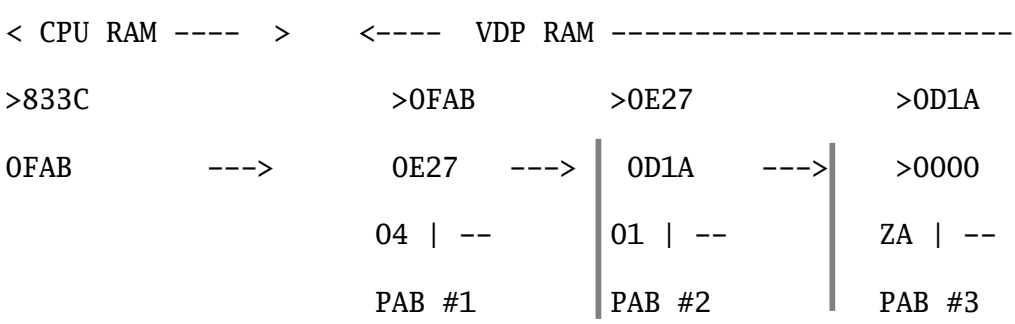
Byte	Contents	Meaning
0,1	Link to next PAB	The address of the next P A B in the chain of PABs used by T I BASIC. The last P A B in the chain has a value of >0000 in these bytes.
2	File Number	The number assigned to the file by T I BASIC.
3	Internal Offset	If 0, there is no effect. If non-zero, it is the value to be added to the start address of the data buffer before the next PRINT or INPUT operation. This is only used if the previous PRINT operation ended in a semicolon(;) or comma (,) or if the previous INPUT operation ended in a comma (,).
4	I/O Op-code	Same as byte 0 in the P A B described in Section 18.2.
5	Flag/Status	Same as byte 1 in the P A B described in Section 18.2.
6,7	Data Buffer Address ..	Same as bytes 2,3 in the P A B described in Section 18.2
8	Logical Record Length	Same as byte 4 i n the P A B described in Section 18.2.
9	Character Count	Same as byte 5 in the P A B described in Section 18.2.
10,11	Record Number	Same as bytes 6,7 in the P A B described in Section 18.2
12	Screen Offset	Same as byte 8 in the P A B described in Section 18.2.
13	Name Length	Same as byte9 in the P A B described in Section 18.2.
14+	File Descriptor	Same as bytes 10+ in the P A B described in Section 18.2.

The following figure summarizes the bytes which make up a PAB.



File Descriptor

The following shows how three PABs might be linked in T I BASIC.



==========

18.3 EXAMPLE OF FILE ACCESS

The following program opens a fixed 80 file called DSK1.DATA, reads a record from it, waits for you to press a key, closes the file, and returns to the calling program.

```
               DEF     DSR
               REF     DSRLNK,VMBW,VMBR,VSBW,KSCAN
     PABBUF    EQU     >1000
     PAB       EQU     >F80
     *
     STATUS    EQU     >837C
     PNTR      EQU     >8356
     *
     SAVRTN    DATA    0
     PDATA     DATA    >0004,PABBUF,>5000,>0000,>0009
               TEXT    'DSK1.DATA'
               EVEN
     READ      BYTE    >02
     CLOSE     BYTE    >01
     *
     MYREG     BSS     >02
     BUFFER    BSS     80
     *
               MOV       R11,@SAVRTN     Save return address.
               LWPI      MYREG           Load own registers.
               LI        R0,PAB
               LI        R1,PDATA
               LI        R2,>20
               BLWP      @VMBW           Move PAB data into PAB in VDP RAM.
               LI        R6,PAB+9        Pointer to name length.
               MOV       R6,@PNTR        Store pointer to name length in
                                         >8356.
               BLWP      @DSRLNK         Open file.
               DATA      8
               MOVB      @READ,R1
               LI        R0,PAB
               BLWP      @VSBW           Change I/O op-code to read.
```

```
     MOV  R6,@PNTR    Restore pointer to name.
     BLWP @DSRLNK     Read one record.
     DATA 8
     LI   R0,PABBUF
     LI   R1,BUFFER
     LI   R2,80
     BLWP @VMBR       Move to C P U buffer.
     LI   R0,>102     Specify beginning screen location.
     LI   R1,BUFFER
     LI   R2,80
     BLWP @VMBW       Move line to screen.
*
LOOP
     BLWP @KSCAN      Wait for key press.
     MOVB @STATUS,RO
     JEQ  LOOP
*
OVER MOVE @CLOSE,Rl
     LI   R0,PAB
     BLWP @VSBW       Change I/O op-code to close.
     MOV  R6,@PNTR    Restore pointer to name.
     BLWP @DSRLNK     Close file.
     DATA 8
     CLR  RO
     MOVB RO,@STATUS   So that no error is reported.
     MOV  @SAVRTN,Rll  Saved return address.
     RT                Return to calling routine.
     END
```

==========END==========