File System Operations in Assembly Language

for Alice / MC-10

FILSYS = $0030

All file system operations are executed by a call to the FILSYS entry point ($0030). The operation to be performed is specified by an opcode parameter passed in accumulator B. A pointer to a File Control Block (FCB) is passed in the Index register (X). The Index register is always preserved across calls to FILSYS, but the A and B accumulators are not.

The File Control Block is a 34 byte structure. The first 8 bytes of this structure are used to pass parameters and return status information when making calls to FILSYS. The remaining bytes of the FCB are used internally by the file system and should not be modified while a file is open.

FOpen = $02
FOpenR = $03

Open a file with Read / Write access (**FOpen**) or Read-Only access (**FOpenR**) and set the current *File Position* to the beginning of the file (0).

FCB usage:

0	**Status**	<---	Status code upon return: 0 = Success 34 = I/O Error 38 = Bad Drive Number 40 = File Not Found 42 = File or Disk is Write Protected 44 = Bad File Name 46 = File System Error 52 = Already Open
1	**Opcode**	--->	Operation Code (set by value passed in accumulator B)
2	**Name**	--->	Pointer to a null-terminated file name string
4	**DefExt**	--->	Pointer to 3 character default extension (null = none)
6	**Buffer**	--->	Pointer to a 512 byte buffer for exclusive use by this FCB.

The **Name** field of the FCB must point a null-terminated string which identifies the file to be opened. A file name string has the following format:

```
drive:name.extension
```

The drive number is optional and may be either 0 or 1. If no drive number appears in the file name string then the current value of DEFDRV ($BDCF) will be used. The name must be from 1 to 8 characters in length. The extension is optional and can be from 1 to 3 characters in length. If no extension appears in the file name string and the **DefExt** field of the FCB is not null, then the 3 characters pointed to by **DefExt** will be used for the extension.

The **Buffer** field of the FCB must point to a 512 byte buffer which will be used exclusively by the FCB during the time that the file remains open. You must not modify this field while the file is open.

FCreate = $82

The **FCreate** operation is similar to the **FOpen** operation, except that the file will be created if it does not already exist.

FCB usage:

0	**Status**	<---	Status code upon return: 0 = Success 34 = I/O Error 38 = Bad Drive Number 42 = File or Disk is Write Protected 44 = Bad File Name 46 = File System Error 52 = Already Open 60 = Disk is Full
1	**Opcode**	--->	Operation Code (set by value passed in accumulator B)
2	**Name**	--->	Pointer to a null-terminated file name string
4	**DefExt**	--->	Pointer to 3 character default extension (null = none)
6	**Buffer**	--->	Pointer to a 512 byte buffer for exclusive use by this FCB.

See the description of **FOpen** for more information.

The example below opens a file named "`TESTFILE.DAT`" on drive 0, creating it if necessary:

```
          ldx     #FCB                point X at the File Control Block
          ldd     #FName              point D at the file name string
          std     2,x                 store pointer to name in FCB
          ldd     #0                  null pointer
          std     4,x                 no default extension
          ldd     #FBuffr             point D at the file buffer
          std     6,x                 store buffer address in FCB
          ldab    #FCreate            the operation code
          jsr     FILSYS              call File System
          bcs     doErr               branch if an error occurred
          ...

doErr     ldab    0,x                 return error code in B
          rts

FName     fcc     '0:TESTFILE.DAT'    File name string
          fcb     0                   Null terminator
FCB       rmb     34                  File Control Block
FBuffr    rmb     512                 File Buffer
```

FClose = $00

The **FClose** operation closes a file that was previously opened successfully using either the **FOpen**, **FOpenR** or **FCreate** operations.

FCB usage:

0	**Status**	<---	Status code upon return: 0 = Success 34 = I/O Error 46 = File System Error 54 = File is Not Open
1	**Opcode**	--->	Operation Code (set by value passed in accumulator B)

Closing a file causes any changes that are cached in memory to be written to the disk. This may involve writing data to the file and/or updating the directory.

Once a file has been closed, it is safe to reuse the memory holding the FCB structure and the file buffer for other purposes.

FRead = $06

The **FRead** operation reads data into memory from an open file.

FCB usage:

0	**Status**	<---	Status code upon return: 0 = Success 34 = I/O Error 46 = File System Error 48 = Read past End of File 54 = File is Not Open
1	**Opcode**	--->	Operation Code (set by value passed in accumulator B)
2	**DatPtr**	--->	Pointer to location where data will be stored
4	**Count**	--->	Number of bytes to read

Data is read from the file starting at the current *File Position*. You can use the **FSeek** operation to change the current file position before using **FRead**. If the number of bytes requested by **Count** is greater than the number remaining in the file, then all remaining bytes will be read and the status code for *Read past End of File* will be returned. The file position will be moved ahead by the number of bytes that were read.

The **Count** parameter is an unsigned 16 bit integer allowing you to read up to 65535 bytes per request (although such large requests are not practical on the Alice and MC-10).

The example below loads 4K of data at $5000 from a previously opened file:

```
        ...
        ldd     #$5000          location to load the data
        std     2,x             store load address in FCB
        ldd     #4*1024         number of bytes to load (4K)
        std     4,x             store byte count in FCB
        ldab    #FRead          the operation code
        jsr     FILSYS          call File System
        bcs     doErr           branch if an error occurred
        ...
```

FWrite = $0C

The **FWrite** operation writes data from memory to a file opened with Read/Write access.

FCB usage:

0	**Status**	<---	Status code upon return: 0 = Success 34 = I/O Error 36 = File has Read-Only access 42 = Disk is Write Protected 46 = File System Error 54 = File is Not Open 60 = Disk is Full
1	**Opcode**	--->	Operation Code (set by value passed in accumulator B)
2	**DatPtr**	--->	Pointer to the data to be written
4	**Count**	--->	Number of bytes to write

Data is written to the file starting at the current *File Position*. You can use the **FSeek** operation to change the current file position before using **FWrite**. The file position will be moved ahead by the number of bytes that were written. If necessary, the file size will be increased to accommodate the data.

The **Count** parameter is an unsigned 16 bit integer allowing you to write up to 65535 bytes per request (although such large requests are not practical on the Alice and MC-10).

The example below writes 32 bytes of data from $7800 to a previously opened file:

```
...
ldd     #$7800          location of data to write
std     2,x             store data pointer in FCB
ldd     #32             number of bytes to write
std     4,x             store byte count in FCB
ldab    #FWrite         the operation code
jsr     FILSYS          call File System
bcs     doErr           branch if an error occurred
...
```

FSeek = $08

Changes the current file position.

FCB usage:

0	**Status**	<---	Status code upon return: 0 = Success 48 = Seek past End of File 54 = File is Not Open
1	**Opcode**	--->	Operation Code (set by value passed in accumulator B)
2		----	Unused
3	**NewPos**	--->	Absolute seek position (24 bits)

The **NewPos** parameter is an unsigned 24 bit integer (3 bytes) which specifies the absolute position in the file where the next Read or Write operation should occur. If the new position is greater than the current file size then the position will be set to the end of the file and the status code for *Seek past End of File* will be returned.

The example below sets the current file position to 64:

```
        ...
        ldd     #64           seek position
        std     4,x           store low-order 16 bits in FCB
        clr     3,x           hi-order 8 bits = 0
        ldab    #FSeek        the operation code
        jsr     FILSYS        call File System
        bcs     doErr         branch if an error occurred
        ...
```

FSetEnd = $0A

Change the file size to equal the current file position.

FCB usage:

0	**Status**	<---	Status code upon return: 0 = Success 36 = File has Read-Only access 54 = File is Not Open
1	**Opcode**	--->	Operation Code (set by value passed in accumulator B)

The **FSetEnd** operation can be used to reduce the size of a file, but not extend it. You may first use the **FSeek** operation to move the current *File Position* and then use **FSetEnd** to make that position the new *End of File*. The file must be open with Read/Write access in order to change its size.

FFlush = $04

Commit any cached file data to the disk and update the directory if necessary.

FCB usage:

0	**Status**	<---	Status code upon return: 0 = Success 34 = I/O Error 42 = Disk is Write Protected 46 = File System Error 54 = File is Not Open
1	**Opcode**	--->	Operation Code (set by value passed in accumulator B)

Changes made to a file by the **FWrite** and **FSetEnd** operations may be cached in memory. The **FFlush** operation forces those changes to be written out to the disk. You do not normally need to use this operation since closing the file has the same effect.

FKill = $10

Delete a file from the disk

FCB usage:

0	**Status**	<---	Status code upon return: 0 = Success 34 = I/O Error 38 = Bad Drive Number 40 = File Not Found 42 = File or Disk is Write Protected 44 = Bad File Name 46 = File System Error 52 = File is Already Open
1	**Opcode**	--->	Operation Code (set by value passed in accumulator B)
2	**Name**	--->	Pointer to a null-terminated file name string

The **Name** field of the FCB must point a null-terminated string which identifies the file to be deleted. If no drive number appears in the file name string then the current value of DEFDRV ($BDCF) will be used. The file name string must include any extension associated with the file (there is no provision for a default extension using this operation).

The example below deletes the file named "`TESTFILE.DAT`" in drive 0:

```
          ldx     #FCB                point X at the File Control Block
          ldd     #FName              point D at the file name string
          std     2,x                 store pointer to name in FCB
          ldab    #FKill              the operation code
          jsr     FILSYS              call File System
          bcs     doErr               branch if an error occurred
          ...

FName     fcc     ’0:TESTFILE.DAT’    File name string
          fcb     0                   Null terminator
FCB       rmb     34                  File Control Block
```

FRename = $0E

Changes the name of an existing file.

FCB usage:

0	**Status**	<---	Status code upon return: 0 = Success 34 = I/O Error 38 = Bad Drive Number 40 = File Not Found 42 = File or Disk is Write Protected 44 = Bad File Name 46 = File System Error 52 = File is Already Open 58 = A File with the New Name Already Exists
1	**Opcode**	--->	Operation Code (set by value passed in accumulator B)
2	**Name**	--->	Pointer to a null-terminated string identifying the file
4	**NewName**	--->	Pointer to a null-terminated string for the file's new name

The **Name** field of the FCB must point a null-terminated string which identifies the file to be renamed. If no drive number appears in the file name string then the current value of DEFDRV ($BDCF) will be used. The file name string must include any extension associated with the file (there is no provision for a default extension using this operation).

The **NewName** field must point a null-terminated string which represents the new name to be assigned to the file. If a drive number appears in the new file name string then it must match the drive number of the target file, otherwise the status code for *Bad Drive Number* will be returned.

The example below renames the file "`NAMES.DAT`" to "`NAMES.BAK`":

```
          ldx     #FCB              point X at the File Control Block
          ldd     #OldFNam          point D at current name string
          std     2,X               store pointer to current name in FCB
          ldd     #NewFNam          point D at new name string
          std     4,X               store pointer to new name in FCB
          ldab    #FRename          the operation code
          jsr     FILSYS            call File System
          bcs     doErr             branch if an error occurred
          ...

OldFNam   fcc     ’0:NAMES.DAT’     Original name string
          fcb     0                 null terminator
NewFNam   fcc     ’NAMES.BAK’       New name string
          fcb     0                 null terminator
FCB       rmb     34                File Control Block
```

FAttrib = $12

Set file attributes.

FCB usage:

0	**Status**	<---	Status code upon return: 0 = Success 34 = I/O Error 38 = Bad Drive Number 40 = File Not Found 42 = Disk is Write Protected 44 = Bad File Name 46 = File System Error 52 = File is Already Open
1	**Opcode**	--->	Operation Code (set by value passed in accumulator B)
2	**Name**	--->	Pointer to a null-terminated string identifying the file
4	**Mask**	--->	Mask of attributes to be retained
5	**Value**	--->	Value of attributes to be changed

The **Name** field of the FCB must point a null-terminated string which identifies the file to be renamed. If no drive number appears in the file name string then the current value of DEFDRV ($BDCF) will be used. The file name string must include any extension associated with the file (there is no provision for a default extension using this operation).

The **Mask** field is a byte in which each bit that is set corresponds to an attribute that will remain unchanged. The file system uses this mask to clear the attributes which will be changed (using an AND operation) before ORing in the new attribute settings from the **Value** field.

The supported file attributes are:

Read Only	$01
Hidden	$02
Archive	$20

The example below removes the *Archive* attribute from the file named "TESTFILE.DAT" in drive 0:

```
          ldx     #FCB                point X at the File Control Block
          ldd     #FName              point D at the file name string
          std     2,X                 store pointer to name in FCB
          ldaa    #~$20               mask to retain all except 'Archive'
          clrb                        do not set any attributes
          std     4,x                 store mask and value in FCB
          ldab    #FAttrib            the operation code
          jsr     FILSYS              call File System
          bcs     doErr               branch if an error occurred
          ...

FName     fcc     '0:TESTFILE.DAT'    File name string
          fcb     0                   Null terminator
FCB       rmb     34                  File Control Block
```

The Low Level Disk Driver

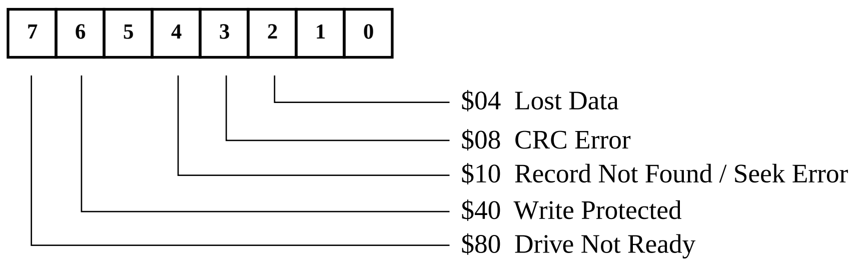
DSKCON = $0020

The low level floppy disk driver can be called by a JSR to the DSKCON entry point. Parameters are passed in the disk driver parameter block at $BDC0-BDC5 and the status result is returned in DCSTA at $BDC6. The carry flag is also set upon return if the status result is not zero.

The parameter values at $BDC0-BDC5 are not modified by the disk driver.

The index register (X) and both accumulators are preserved across calls to DSKCON.

$BDC0	**DCOPC**	--->	Driver operation code: 0 = Restore head to track zero 1 = Get Write Protect status 2 = Read sector 3 = Write sector 4 = Step in 5 = Format the current track
$BDC1	**DCDRV**	--->	Drive number (0 or 1)
$BDC2	**DCCYL**	--->	Cylinder number in bits 0..6, Side in bit 7
$BDC3	**DCSEC**	--->	Sector number
$BDC4-5	**DCBUF**	--->	Buffer address
$BDC6	**DCSTA**	<---	Driver status result:



This example will read sector 3 on track 14, side 0 from the disk in drive 0. Data is loaded at $5000.

```
          ldaa     #2             opcode for Read Sector
          clrb                    drive 0
          std      DCOPC          set opcode and drive number
          ldaa     #14            track 14, side 0
          ldab     #3             sector 3
          std      DCCYL          set cylinder and sector
          ldx      #$5000         address where sector will be loaded
          stx      DCBUF          set the buffer address
          jsr      DSKCON         call the disk driver
          bcs      doErr          branch if error
          rts                     return

doErr     ldaa     DCSTA          get driver status flags
          ...                     handle error
          rts                     return
```

Logical Sector I/O

RDSEC = $0023

Reads one sector, identified by *Logical Sector Number*, from disk. The drive number (0..1) is specified in the DCDRV parameter ($BDC1). The logical sector number (0..1439) is passed in the D accumulator. The buffer address for the sector data is passed in the index register (X).

The disk driver status flags are returned in DCSTA at $BDC6. The carry flag is also set upon return if the status result is not zero.

The index register (X) and both accumulators are preserved.

This example reads logical sector 184 from the disk in drive 0. Data is loaded at $5000.

```
clr      DCDRV         drive 0
ldd      #184          logical sector number
ldx      #$5000        address where sector will be loaded
jsr      RDSEC         call the Read Sector subroutine
bcs      doErr         branch if error
...
...
```

WRSEC = $0026

Writes one sector, identified by *Logical Sector Number*, to disk. The drive number (0..1) is specified in the DCDRV parameter ($BDC1). The logical sector number (0..1439) is passed in the D accumulator. The address of the data to be written is passed in the index register (X).

The disk driver status flags are returned in DCSTA at $BDC6. The carry flag is also set upon return if the status result is not zero.

The index register (X) and both accumulators are preserved.

This example writes data from $7400 to logical sector 18 on the disk in drive 0.

```
clr      DCDRV         drive 0
ldd      #18           logical sector number
ldx      #$7400        data is at $7400
jsr      WRSEC         call the Write Sector subroutine
bcs      doErr         branch if error
...
...
```