For Superboard 600 and C1P

YE-OSI DOS 3.54

1984 by TB

Revised 2024

YE-OSI

*** DISK OPERATING SYS FOR SB600 & C1P ***

*** WRITTEN IN 1984 BY TB ***

*** UPDATE VERSION FOR STD ROM'S ***

****

To run YE-OSI DOS 3.54, it is mandatory to

- have a minimum of 32kB of RAM memory

- add a disk controller board from ELEKTOR or an OSI 610 Floppy board

Main memory will be permanently occupied from 0x6800 up to 0x7FFF

- do minor modifications on OSI 610 board to allow 3.5 & 5.25 inch drives

YE-DOS supports 35 and 80 track, single or double sides drives

With version 3.54, it is recommended to remove Drive Select Line resistors R43 and R44 and the Write Enable Line resistor R41 at PB0 on the 610 board. There must be a pull-up resistor installed/enabled on one of the drives attached!

This will prevent data corruption on inserted diskettes, when, when power is turns on or off.

YE-DOS starts in 2 phases. First the Boot sector and afterwards DOSSUP.COM, a BASIC command add-on is loaded into memory.

With DOSSUP, you will get additional BASIC commands as:

PAGE, **SET**, **CALL** for general purpose

DLOD, **ERR**, **DISK**, **DOS**, **ASS** for rudimental disk management

DIR, **SEL**, **DSAV**, **DFMT**, **DCHK**, **DREN** for disk file management

EOF, **SEQS**, **SEQW**, **VER** for disk data management

CLG, **GDIS**, **SCR** for Text and Low-resolution graphics

PTR for general purpose variable management

DOSSUP will be loaded into memory at location $7040 to 0x79FF

It is possible to run a rudimental system DOS in conjunction with code from the Boot sector of the disk. But it is recommended to make use of DOS Support routines. All BASIC routines are located in DOSSUP and are automatically loaded at startup from the disk in drive 0.

Boot sequence selecting "D":

Like the standard OSI System ROM, the boot sector on drive 0 is loaded on request into memory. This is done by selection **D** after pressing the RESET.

From here, you have to select

C) COLD START (will clear all memory)

W) WARM START or E)

If file DOSSUP is present on the boot drive, it will be loaded automatically.

DOSSUP stands for DOS Supplement. Extended BASIC commands will be available.

DOS Memory Map:

	Address	Content
ROM EPROM1_V54.ROM	0xFFFF 0xF800	Modified SYSTEM ROM
OSI I/O	0xF000	Serial ACIA 0xF000 ACIA 1 Option 0xF400 ACIA 2
OSI I/O	0xDF00	Polled Keyboard
OSI DISPLAY RAM	0xD7FF 0xD000	Display RAM (2kB)
FDC I/O	0xC000 0xC010	6822 PIA Port and ACIA Floppy Disk
OSI BASIC ROM	0xBFFF 0xA000	8K Microsoft Basic
OPTIONAL HIRES DISPLAY	0x9FFF 0x8000	Hires 265x265 pixel or additional RAM
USER RAM	0x7FFF 0x0200	32k min YE-DOS starting at 0x6800 BASIC Code start at 0x0300 up to 32K(40k) User RAM
STACK ZEROPAGE	0x01FF 0x0000	

********* RAM for DOS Extension **********

OPERATIONG SYSTEM RAM:

$6800-6FFF = 2k used by YE-OSI DOS 3.54 (Boot Sector Code)

$7000-703F = DOS TEMP Memory for STACK and Zero page (2x32 Bytes)

$7040-79FF = 2.5k used by DOS SUPPLMENT DOSSUP

I/O:

$F000-F0FF = OSI ACIA I/O

$F100-F1FF = 2nd ACIA (optional Microsoft serial Mouse Port @ 1200 baud)

$C000 = Disk PIA DATA A

$C002 = Disk PIA DATA B

$C001 = Disk PIA CTRL A

$C003 = Disk PIA CTRL B

$C010 = Disk ACIA Control Port

$C011 = Disk ACIA Data Port

ADDITIONAL DOS RAM:

$7A00-7AFF = (256 Bytes Free RAM, (BASIC DISK OR FILE COPY or NMI Routine)

$7B00-7BFF = TEMPORARY memory for building Track/Sector list

$7C00-7FFF = DOS FAT memory location (RAM)

- $7C00 SECTOR USAGE INFORMATION (Address Vector in 682D)
- $7C50 DISK TITLE 16 bytes (Address Vector in 682B)
- $7C60 START OF FILE DIRECTORY (Address Vector in 682F)

IMPORTANT REMARK:

YE-DOS will work on double (max 2) or single sided (max 4) drives.

In case of double-sided drives, each side is accessed separately, like on single drives, but YE-DOS takes care, that each side/head walks synchronous.

This is important on disk format or copy operations.

****

********** QUICK START GUIDE **********

****

DOSSUP provides some of the following BASIC commands to manage disk drives

EXAMPLES:

Show file directory of currently selected disk drive

➔ **DIR**

Select the second disk drive 2, Boot drive is drive 0

➔ **SEL 2**

Save current Basic program to disk drive 0

➔ **DSAV "TEST",0,0,0**
will save under filename "TEST" onto drive 0 as a read write basic program.

Load back a Basic program from currently selected disk drive

➔ **DLOD "TEST"**
The BASIC program "TEST" will be loaded into memory

Rename the Basic program "TEST" from disk drive 0 into "HELLO"

➔ **DREN "TEST","HELLO",0**
The BASIC program "TEST" will be renamed to "HELLO" on disk 0

Delete a Basic program from disk drive 0

➔ **DREN "HELLO","",0**
The BASIC program "HELLO" will be deleted from disk 0

Writing Bootsector only to disk 0

➔ **DFMT 0,1,0**
The second parameter specifies "Bootsector only"
If you have double sided disk drives, you may update the Bootsector for these specific drives with (last parameter):

➔ **DFMT 0,1,1**

New BASIC commands summary :

**** DIR ********** COMMAND:

DIR ["String"]

DIR will list the current active file directory. If no "String" is entered all files will be displayed. The length is shown in 256-byte sectors. The listing will pause after 9 file names. To continue press, ENTER, any other key will end the directory listing.

DIR "String" will list all files starting with the "String" letters. For example: DIR"DO" will list all files starting with letters "DO..".

The file type is specified as:

BAS = BASIC Token Memory loads typically to $0300

COM = MCODE Binary Code

SEQ = SEQUENCIAL Data values separated by comma

VAR = VARIABLE Other type of data

Protection status:

RWn = Read/Write normal

RWa = Read/Write autorun

R n = Read Only normal

R a = Read Only autorun

***** SEL ********* COMMAND:

SEL DRIVE

will select "DRIVE" number 0...3. If FAT was changed, FAT is saved before.

IMPORTAT !

Be careful removing and inserting disk into the drive during operation. YE-DOS will not automatically detect, when a new disk is inserted!!

To reload the Disk FAT directory, always type SEL 0..3, to get the current disk content. Otherwise, the disk content may get corrupted.

Only inserted disk will be detected as valid drives. Use DISK command to refresh the drive valid information.

Physical Drive 1

Side A: >Drive number 0

Side B: >Drive number 1

Physical Drive 2

Side A: >Drive number 2

Side B: >Drive number 3

Remark: Emulation supports drive number 0/1 first and 2/3 for second.

****** DLOD ******* COMMAND:

DLOD "Filename"

Loads a program from currently selected drive to memory

DLOD command with “*” like (DLOD"*") will load first file in the directory

Filename are max 6 characters long. Additional characters are ignored.

You may enter less characters and YE-OSI DOS will load the first matching

filename into memory. For example, DLOD "EXT will load the file “EXTMON”.

DLOD reads the content from the currently selected Drive (0 after boot)

IMPORTANT! Any data retrieved with DLOD will be stored to the same memory location, as it came from! Loading BASIC programs will overwrite existing BASIC code.

**** DSAV ********** Command:

DSAV "FILNAME", DRIVE, TYPE, PROTECTION

> 1st VERSION <

DSAV stands for Disk Save File and will write a BASIC program, Binary data or other data to disk. "FILENAME" are max 6 characters, longer names are ignored.

When DSAV has been executed, active drive will change to the "DRIVE" number. Check the variable ERR, if any Error occurred.

For the file attributes Type and Protection, see the following valid codes.

***** TYPE codes:**

BAS=0, COM=1, SEQ=2, VAR=3

***** PROTECTION codes:**

RWn=0 RWa=1 ROn=2 ROa=3

Example: DSAV "TEST",2,0,0 will save a BASIC program in memory with the filename "TEST". BASIC programs require no address range information.

If a file already exists and file protection is "Read Only", like 2 or 3, DSAV will fail. In such a case, you have to remove the file protection with DREN (Disk File Rename) first. For example, DREN “TEST”,”TEST”,0,0,0

***** DSAV ********* Command:

DSAV "FILNAME", DRIVE, TYPE, PROTECTION, START, END

> 2nd VERSION <

Types COM, SEQ and VAR are saved in the same way. These file types (Binary Code, Sequential or Variable) are written to the disk like binary data, but with its specific Type identification.

Example: DSAV "TEST", 0, 1, 1, 32768, 32768+1023 will write 1kb binary data to drive 0 as an autorun RW file (RWa). Execution will start after loading the file back at address 32768 in this example.

**** SEQS ********** COMMAND:

SEQS Address

SEQS or Sequence Set Read file pointer will set the pointer to the "Address" in memory.
The purpose is to READ strings or numbers from a given memory location.
These data elements have to be "comma" separated.
The next READ operation will get the stored strings and numbers in a typical DATA read operation.

***** SEQW ********* COMMAND:

NewAddress = SEQW Address, Parameter1, P2, ..

SEQW or Sequence Write data, will put strings or numbers to the Address

Pointer. The Command will return the new Address pointing to the next Input.

SEQS and SEQW are used to store string or variables in memory that can be saved or loaded to or back from disk. Memory space selection and pointer control has to be done by software.

***** EOF ********* COMMAND:

[Value=] EOF

EOF will return TRUE after a READ operation, if more data is available.

Example in BASIC:

```
 10 AN=31232:EN=AN+256-20
 20 RESTORE:PAGE
 25 A$="QWERTY"
 30 LE=SEQW AN:REM SET START POINTER
 35 PRINT"GENERATE SEQ DATA AT $7A00"
 40 LE=SEQW LE,A$,LE,-1
 50 IFLE<ENTHEN40
 60 LE=SEQW LE:REM GET END POINTER
 70 SL=LE:LE=AN
 80 SEQS LE
 90 READB$,AD,F
 110 PRINT B$;AD;F
 120 IF EOF THEN90
 130 PRINT
 140 PRINT"SEQ DATA SIZE";SL-AN;" BYTES"
 150 F$="DATA":DSAV F$,0,2,0,AN,SL
```

```
 155 IF ERR<>0 THEN PRINT"ERROR";ERR:STOP
 160 PRINT"DATA SAVED"
 170 DLOD F$
 180 IF ERR<>0 THEN PRINT"ERROR";ERR:STOP
 190 SEQS AN:REM READ POINTER TO START
 200 READB$,AD,F
 210 PRINT B$;AD;F
 220 IF EOF THEN200
 230 PRINT"DATA LOADED BACK"
 240 DREN F$,"",0:REM DELETE FILE
 250 IF ERR<>0 THEN PRINT"ERROR";ERR:STOP
```

This program will generate a data parameter stream of a string and two numbers at $7A00 (Line 25..50). The sequential data stream is than stored as "DATA" SEQ file to disk. Afterwards read back and displayed using the BASIC READ statement (Line 190..220)

***** VER ********* COMMAND:

VER

VER will return the disk DOS version of the currently selected drive
For Example, VER will return "**YE-OSI DOS 3.54**" on the current revision.

****** DFMT ******** COMMAND:

DFMT DRIVE, SECTION, DISKTYPE

DFMT stands for Disk Format. The Command will format a disk "DRIVE".

DFMT will be executed without further prompt or question.

Please make sure, you have no valuable data on the disk to format.

"DRIVE" number has to be between 0...3.

"SECTION" defines, if the whole disk (0) or only the disk BOOT sector (1)

has to be formatted.

"DISKTYPE" defines, if we have a single (0) or double-sided disk (1).

IMPORTANT !

DFMT will only create "blank" diskettes, without content. Use the Basic

program FORMAT.BAS, to create fully bootable diskettes. DFMT will immediately start, there will be no warning. Both sides on double-sided will be formatted.

EXAMPLE: DFMT 2,0,0 will format disk 2 as single sided.

FORMAT.BAS program example:

```
10 REM DISK FORMAT UTILITY
15 GOSUB900
20 TA=64768:PAGE:PRINT"UTILITY FOR 40/80 TRACK DRV":PRINT
25 PRINT"FORMAT DRIVE NUMBER ?":T=CALLTA,0:IFT<48 OR T>51THEN END
30 PRINT:PRINT"INSERT DISK IN DRIVE ";CHR$(T)
35 DR=T-48:GOSUB800:PRINT
40 PRINT"PRESS(Y),WHEN READY:":T=CALLTA,0:IFT<>89THEN END
50 PRINT:PRINT"FORMATTING DISK";DR
60 DFMT DR,0,SS:REM FULL FORMAT DISK
80 IF ERR>0 THEN PRINT"FORMAT FAILED, ERROR NUMBER ";ERR
90 REM ---------------------
100 PRINT"TRANSFER DOS TOOLS TO DRIVE";DR
105 PRINT"PRESS(Y),WHEN READY:":T=CALLTA,0:IFT<>89THEN160
110 SELDR:PRINT:PRINT "TRANSFER DOS EXTENSION TO DRV";DR
```

```
115 IFYE=1 THEN DSAV"DOSSUP",DR,1,3,BA+2304,BA+4095
116 IFYE=2 THEN DSAV"DOSSUP",DR,1,3,BA+2112,BA+4607
120 IF ERR>0 THEN PRINT "TRANSFER FAILED WITH ERROR";ERR
130 PRINT:PRINT "TRANSFER FORMAT.BAS TO DRV";DR
140 DSAV "FORMAT",DR,0,1:PRINT:REM TYPE BASIC,R/W AUTORUN
150 IF ERR>0 THEN PRINT "TRANSFER FAILED WITH ERROR";ERR
160 DIR:IF ERR>0THEN PRINT"DOS FORMAT FAILED"
170 SEL0:IF ERR>0THEN PRINT"DOS DRIVE 0 FAILURE":END
200 END
800 REM CHECK FOR SS OR DS DRIVES
810 SS=0:IF DR<2 THEN 840
820 IF PEEK(BA+23)<>255 THEN SS=1
830 GOTO 850
840 IF PEEK(BA+21)<>255 THEN SS=1
850 IF SS<>1 THEN RETURN
860 PRINT:PRINT"SINGLE(S) OR DOUBLE(D) SIDED DRIVES ?":T=CALLTA,0
870 IFT=68 OR T=83THEN880
875 GOTO860
880 IFT=83 THEN SS=0
885 RETURN
900 REM CHECK YE VERSION
910 YE=-1:BA=0:IF PEEK(190)>223 THEN YE=1
920 IF PEEK(190)<160 THEN YE=2
930 IF PEEK(190)=208 THEN YE=0
940 IFYE=1 THEN BA=57344
950 IFYE=2 THEN BA=26624
960 IF BA=0 THEN PRINT"YE-DOS NOT PRESENT":END
970 RETURN
```

***** DCHK ********* COMMAND:

DCHK "FILENAME"

DCHK stands for Disk Check/Verify. The Command will Verify a saved file "FILENAME" with its original location in memory.
It will also check, if the file exists on the disk.
If the filename is not on the disk, ERR number 9 is returned.
In case the filename exists, the file content is compared to memory.
If this verification fails, ERR number 11 is returned. Elsewhere ERR 0 is returned.
Verification or Check is done on the current selected drive only.

**** DREN ********** COMMAND:

DREN "FILENAME", "NEW FILENAME", DRIVE [, TYPE, PROTECTION]

DREN stands for Disk File Rename or Delete. The Command will rename a saved file "FILENAME", change its type or protection status. With an empty new filename, the file will be deleted.
Keep in mind, that Read Only protected files cannot be deleted, before the protection has been changed.

DREN Version 1 (5 parameters):
DREN "FILNAME", "NEW FILENAME", DRIVE, TYPE, PROTECTION
This will change Filename and/or file attributes. Using the same filename will only change attributes. For example, changing DOSSUP from RO to RW:

DREN"DOSSUP","DOSSUP",0,1,0 (Drive 0, COM Type, RW Protection)

DREN Version 2 (3 parameters):

DREN "FILENAME", "", DRIVE

This will delete the file "FILENAME" on drive "DRIVE". Data is still on the disk, but the directory entry filename is cleared.

File recovery is possible.

**** SCR ********** COMMAND:

SCR X (0...31/63), Y(0...15/31), DATA, [DATA,...]

SCR will write DATA (Strings or Variables) to the screen at position X,Y.

The left bottom corner of the screen is at SCR 0,0,"x". Range depends on machine graphic capabilities like 32x32, 64x16 or 64x32 characters.

***** PTR ********* COMMAND:

VAL=PTR(VARIABLE)

PTR will return the Pointer to the variable content as a 16bit address value. VARIABLES can be a numeric variable like AD=PTR(A) or a string variable AD=PTR(A$) or a pointer to an array like AD=PTR(M(0)). PTR(M(0)) will return a pointer direction to the first byte of array M().

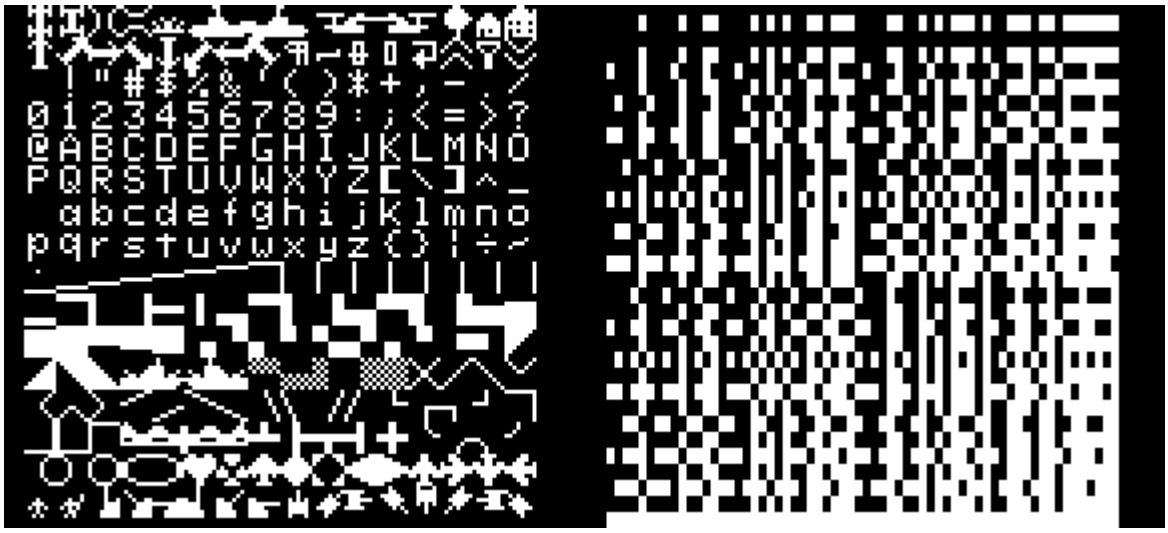
This can be used to reserve memory space and to place code or data into the array to peek or poke or read or write to the disk.

REMARK: Each array element occupies 4 bytes. DIM M(255) will reserve 1kB.

***** CLG ********* COMMAND:

CLG NUMBER

CLG will clear text or low-resolution graphics or enable/disable text/low resolution mode. Low resolution mode is a 128x32 pixel graphic displayed in the upper half of the text screen as a kind of split screen.
This was done by a modified character ROM of 4kB instead of 2kB. The graphic part of the ROM is enabled by the ACIA RTS line and the upper half display interval.



C1P lower 2kB character ROM Upper Low-Res 2kB character ROM

placed into a pin compatible 4kB EPROM with one gate logic chip.

CLG 0: DISABLE LOW-RES MODE (same as CLG without parameter)

- Standard text mode (RESET (F12) will do the same)

CLG 1: ENABLE LOW-RES MODE

- TOP part of the low-res display (128x32 or 128x64 pixel) in half screen mode

CLG 2: Clear BOT half with "20"

- Clear text part

CLG 3: Clear TOP half with "00"

- Clear graphic part

also see PAGE command

- Clear entire text screen, when back in text mode

***** GDIS ********* COMMAND:

GDIS X (0...127), Y(0...16/32/63), MODE (depending on display mode)

When the low-resolution graphics mode is enabled, GDIS will plot dots or

lines on the screen or will clear the same if required.

The Y coordinate of low-res section starts at Y=0 or Y=32 (upper half of screen, depending on display mode).

A graphic section of 128x32 or 128x64 is not much, but it is a simple add-on to allow fast pixel graphics in combination with text output on the C1P machine.

And it uses only the standard screen memory.

The pixel origin is at the left bottom corner of the upper low-res screen.

GDIS X, Y, 1 - Plot at X,Y a white dot

GDIS X, Y, 2 - Draw a line from the last coordinate to this one.

GDIS X, Y, 0 - Plot at X,Y a black dot (clear)

GDIS X, Y, 3 - Draw a black line (clear) to the new coordinate.

See GRDEMO.BAS program example:

```
5 OY=32:IFPEEK(65506)<>0THENOY=0
6 F=1:IFPEEK(65506)<>0THENF=2
7 OM=OY+16*F:OX=OY+32*F-1
8 CLG1:CLG3
9 CLG2:SCR5,5,"LINE SET&RESET"
10 GDIS0,OM-1,1:GDIS127,OM-1,2
11 FOR L=0 TO 1
12 FOR R=0 TO 127 STEP 2
15 IFL=0 THEN GDISR,OX,1:GDISR,OY,2
16 IFL=1 THEN GDISR,OX,0:GDISR,OM,3
17 IFL=1 THEN GDISR,OM-2,0:GDISR,OY,3
18 NEXT R:NEXT L
20 CLG3:IFPEEK(57088)=222 THEN 200
25 CLG2:SCR5,5,"LINE MESH"
30 FOR R=4TO 127 STEP 15
40 FOR S=4 TO 127 STEP 15
50 GDISR,OX,1:GDISS,OY,2
60 NEXT S:NEXT R
70 CLG3:IFPEEK(57088)=222 THEN 200
110 CLG2:SCR5,5,"BOXES"
112 F=1:IFPEEK(65506)<>0THENF=2
115 FOR L=0 TO 3:FOR S=0 TO 1
120 FOR R=1 TO 10*F STEP 2*F
130 GDIS64-7*R/F,OM-R,1-S:GDIS64+7*R/F,OM-R,2+S
140 GDIS64-7*R/F,OM+R,1-S:GDIS64+7*R/F,OM+R,2+S
150 GDIS64-7*R/F,OM-R,1-S:GDIS64-7*R/F,OM+R,2+S
160 GDIS64+7*R/F,OM+R,1-S:GDIS64+7*R/F,OM-R,2+S
190 NEXT R:NEXT S:NEXT L
195 CLG3:IFPEEK(57088)<>222 THEN 7
200 CLG0:PAGE:PRINT"READY"
```

In this program example, placing text by the SCR command and drawing and removing low-res lines by the GDIS command is shown.

****** PAGE ******* COMMAND:

Will clear text screen ($D000-D3FF or $D000-D7FF) with $20 (Space)

******* SET ****** COMMAND:

SET Number

SET will place the READ pointer to the given line number in Basic.

The next READ operation will take the first parameter from that line.

****** CALL ******* COMMAND:

[Val=] CALL, Address, Parameter

Call will call a subroutine at "Address" with the "Parameter" in ACCU

When ending the subroutine with RTS, the ACCU will be returned as Val.

Example: K=CALL64768,0 will return the keypress in K

****** ERR ******* COMMAND:

[Val=] ERR

ERR will return the last DOS Error number. If now Error occurred, ERR returns zero. Here a list of Error numbers and explanation.

ERROR MESSAGES:

Returns last DOS Error value from DOS parameter $E027

ERR 0: No Error

ERR 1 : Sync byte not found

ERR 2 : Sync byte at start sector no found

ERR 3 : Searching track error, not found

ERR 4 : Track or Sector out of range

ERR 5 : Drive not found

ERR 6 : Data to long (>32k) to be saved, not enough free space on disk

ERR 7 : Checksum not correct

ERR 8 : DRIVE not valid/existing

ERR 9 : File name not found

ERR 10: Disk Full Error

ERR 11: Verify failed or Sync byte F7 not found

ERR 12: Track zero not found

ERR 13: FAT Checksum Error

ERR 14: DISK IS WRITE PROTECTED

ERR 15: FILE is WRITE PROTECTED

***** DOS ****** COMMAND:

DOS (identical to DLOD "DOSSUB")

Loads program called "DOSSUP" from disk, if available.

The program will be placed at $E900-EFFF and provide additional DOS Basic commands.

IMPORTANT!

In case of a "RESET", the DOSSUP Extension is disabled. Type "DOS" to re-enable.

***** ASS ****** COMMAND:

ASS (identical to DLOD "EXTMON")

Loads program called "EXTMON" from disk, if available.

The program will be placed at $1500 to $1EFF

This may destroy BASIC code that’s located at this RAM section.

EXTMON.COM may be replaced by any other utility of your choice.

REMARKS:

DOSSUP may be replaced by newer DOS Supplement versions or other tools.

When using only the minimal Disk Basic extensions you have to poke and peek some memory location to get additional functions.

For example:

- Drive selected by $6820 (from 0 to 3)

- Single/Double by $681E

- After loading a file: $F0..F1= Start ADR , $F2..F3= End ADR of loaded data

- and so on

***** DISK ****** COMMAND:

DISK

Will load and run Boot sector of Disk 0 to load YE-OSI DOS routines to $6800

Like OSI Boot ROM, you have to select afterwards

C) COLD START (clears all memory)

W) WARM START or E) for EXTMON

IMPORTANT: If you have changed a disk in the drive, or you have added a disk that is not recognized, enter "DISK" so DOS can rescan all drives for presents (same as DISK 4) Otherwise, you can also enter "SEL {drive number] and DOS will read the disk directory of the chosen drive.

***** DISK ****** COMMAND:

DISK [Number 0...7]

This will call the YE-OSI DOS routines. Keep in mind, that this requires

to set up the DOS parameter table first!

For example

DOS Parameters:

$6820 Drive to be selected

$6827 returns last DOS Error value

(DRV 1: side A(0)/ B(1), DRV 2: side A(2)/side B(3)) for 3,5 inch disk drives. (Emulation only supports single sided disk 0 & 2)

IMPORTANT:

Emulation supports drives 0/1 for first and 2/3 for second single or double sided disks.

******* General:**

Usage of SS or DS 3.5 and 5.25 inch disk drives with 35 or 80 Tracks.

(35 Track drives require a different boot sector version)

DS SD (160k capacity per side @ 125kbit FM coded in 8N1)

Physical Drive 1

Side A: >Drive number 0

Side B: >Drive number 1

Physical Drive 2

Side A: >Drive number 2

Side B: >Drive number 3

Max File length <=32k

Max 71 FAT Directory entries/ files on a disk

Sector 0 and 1 are used by DOS (BOOT and FAT sectors)

********** **Supporting Programs on disk**

DCOPY.BAS

This BASIC program will copy all tracks of a diskette in Drive 1

(single sided only) to Drive 2.

```
4  PAGE:PRINT"* DISK COPY UTILITY *":PRINT"DRIVE 0 TO 1,2 OR 3":PRINT
5  GOSUB900:FA=BA+5216:ID=BA+5120:E=0
6  TM=BA+4608:TA=64768
7  PRINT"TARGET DRIVE NUMBER ?":T=CALLTA,0:IFT<49 OR T>51THEN END
8  DR=T-48:GOSUB800:PRINT
10 PRINT"INSERT SOURCE DISK IN DRIVE 0":PRINT
15 PRINT"PRESS (Y), WHEN READY":T=CALLTA,0:IFT<>89THEN END
20 SEL0:IF ERR>0 THEN PRINT "DRIVE 0 NOT READY":END
25 SELDR:IF ERR>0 THEN PRINT "DRIVE";CHR$(DR+48);" NOT READY":END
30 FOR R=FA TO FA+(70*13) STEP 13: REM CHECK IF DRV IS EMPTY
40 IF PEEK(R)<>0 THEN E=1
50 NEXT R
60 IF E=0 THEN 100
70 PRINT:PRINT ">> DRIVE ";CHR$(DR+48);" IS NOT EMPTY"
80 PRINT"PRESS (Y), TO FORMAT DRIVE":T=CALLTA,0:IFT<>89THEN END
90 PRINT"FORMATTING, PLS WAIT":DFMT DR,0,0
100 REM ****** COPY DISK 0 TO TARGET DISK *****
110 SEL0:IF ERR>0 THEN PRINT "DRIVE 0 NOT AVAILABLE":END
130 FOR R=2 TO 79: REM SEC_T,TRK_T first, E022/E023 second
140 IF PEEK(ID+R)=0 THEN 310: REM TRACK NOT USED
150 PRINT"T="R;"-";
155 BI=1:FOR S=0 TO 7
160 POKEBA+37,255: REM Finish with E022/23
170 POKEBA+36,0: REM TAKE NEXT SECTOR
180 POKEBA+32,0: REM DRIVE 0 SOURCE
190 POKEBA+30,SS: REM SINGLE SIDED
200 POKE240,0:IFYE=1THENPOKE241,242: REM ADR=$F200
```

```
205 IFYE=2THENPOKE241,122: REM ADR=$7A00
210 POKEBA+28,1: REM LENGHTH=256 Bytes
220 POKE236,R:POKE237,S: REM Start TRK2..79,SEC
225 IF (PEEK(ID+R) AND BI)=0 THEN 240:REM SKIP EMPTY SECTORS
230 DISK1:IF ERR>0 THEN PRINT "READ SECTOR FAILED":END
240 POKE236,R:POKE237,S: REM Start TRK,SEC
250 REM TRK2,SEC0 info is given by last READ operation
260 POKE240,0:IFYE=1THENPOKE241,242: REM ADR=$F200
265 IFYE=2THENPOKE241,122: REM ADR=$7A00
270 POKEBA+28,1: REM LENGHTH=256 Bytes
280 POKEBA+32,DR: REM DRIVE DESTINATION
285 IF (PEEK(ID+R) AND BI)=0 THEN 305:REM SKIP EMPTY SECTORS
290 DISK2: IF ERR>0 THEN PRINT "SECTOR SAVE FAILED":END
300 PRINT CHR$(48+S);" ";
305 BI=BI*2:NEXT S:PRINT
310 NEXT R
500 DISK6:IF ERR>0 THEN PRINT "UPDATE FAT TO TARGET FAILED":END
510 POKEBA+37,0: REM Finish with 00 00
520 POKEBA+36,255: REM Take next free sector
530 SELDR:IF ERR>0 THEN PRINT "DRIVE"DRIVE ";CHR$(DR+48);" FAILED":END
540 SEL0:IF ERR>0 THEN PRINT "DRIVE 0 NOT AVAILABLE":END
600 PAGE:PRINT"COPY BOOT SECTOR AND FAT"
610 POKEBA+38,255: REM BOOT SECTOR FORMAT DISK
620 POKEBA+32,DR: REM TARGET DRIVE
630 POKEBA+30,SS: REM SINGLE SIDED
640 DISK3: REM WRITE BOOT SECTOR
650 IF ERR>0 THEN PRINT "BOOT SECTOR FAILED":END
660 SEL0:IF ERR>0 THEN PRINT "DRIVE 0 NOT AVAILABLE":END
670 PRINT "READY"
700 SELDR:DIR:IF ERR>0 THEN PRINT "DOS FORMAT FAILED"
710 SEL0:IF ERR>0 THEN PRINT "DOS DRIVE 0 FAILURE":END
720 END
800 REM CHECK FOR SS OR DS DRIVES
810 SS=0:IF DR<2 THEN 840
820 IF PEEK(BA+20+3)<>255 THEN SS=255
830 GOTO 850
840 IF PEEK(BA+20+1)<>255 THEN SS=255
850 IF SS<>255 THEN RETURN
860 PRINT:PRINT"SINGLE(S) OR DOUBLE(D) SIDED DRIVES ?":T=CALLTA,0
870 IFT=68 OR T=83THEN880
875 GOTO860
880 IFT=83 THEN SS=0
885 RETURN
900 REM CHECK YE VERSION
910 YE=-1:BA=0:IF PEEK(190)>223 THEN YE=1
920 IF PEEK(190)<160 THEN YE=2
930 IF PEEK(190)=208 THEN YE=0
940 IFYE=1 THEN BA=57344
950 IFYE=2 THEN BA=26624
960 IF BA=0 THEN PRINT"YE-DOS NOT PRESENT":END
970 RETURN
```

********** Supporting Programs on disk

FCOPY.BAS

This BASIC program will copy a single file form diskette in Drive 1

to Drive 2. Read Only files will be transferred, if confirmed.

Basic Code:

```
5 REM FILE COPY UTILITY TO OTHER DRIVE
10 CLEAR:GOSUB900:DR=PEEK(BA+32):IF DR=0 THEN DN=2
20 IF DR=2 THEN DN=0:REM GET OTHER DRIVE NUMBER
30 PAGE:PRINT"FILE COPY UTILITY FROM";DR;"TO";DN
40 INPUT"ENTER FILENAME ";NA$:IF NA$="" THEN 40
50 FA=BA+5216:ID=BA+5120:E=0
55 TM=BA+4608:TA=64768:AD=245
60 DCHK NA$:REM TEST, IF FILENAME EXIST
65 FE=PEEK(AD)+256*PEEK(AD+1)
70 IF ERR=9THEN PRINT"FILENAME NOT FOUND, TRY AGAIN":PRINT:GOTO40
75 SEL DN:IF ERR>0 THEN PRINT "DRIVE";DN;"IS NOT AVAILABLE":END
80 DCHK NA$: IF ERR=9 THEN 110
90 PRINT"FILE EXIST,OVERWRITE IT (Y/N)?": T=CALL TA,0
100 IFT<>89THEN PRINT"QUIT":END
110 SEL DR:IF ERR>0 THEN PRINT "DRIVE";DR;"IS NOT AVAILABLE":END
120 S1=PEEK(FE+8):S2=PEEK(FE+9):S=S1+256*S2
130 E1=PEEK(FE+10):E2=PEEK(FE+11):E=E1+256*E2
135 TP=PEEK(FE+12):Z=FRE(0):IF Z<0 THEN Z=65536+Z
140 IF (E-S+255)<Z THEN 150
145 PRINT"NOT ENOUGH FREE MEMORY TO COPY FILE !":GOTO380
150 DIM M((E-S+256)/4)
152 MS=PTR(M(0)):ME=MS+(E-S):REM GET MEMORY ADDRESS
155 POKE(FE+8),MS AND 255:POKE(FE+9),INT(MS/256)
160 POKE(FE+10),ME AND 255:POKE(FE+11),INT(ME/256)
165 POKE(FE+12),16: REM SIMPLE BINARY FILE
170 DLOD NA$
180 POKE(FE+8),S1:POKE(FE+9),S2
190 POKE(FE+10),E1:POKE(FE+11),E2
200 POKE(FE+12),TP: REM RESTORE ORIGINAL
210 IF ERR>0 THEN PRINT "LOADING FILE ERROR":GOTO380
220 SEL DN:IF ERR>0THEN PRINT"DRIVE";DN;"IS NOT AVAILABLE":END
300 IF (TP AND 15)>=2 THEN DREN NA$,NA$,DN,1,0
305 DSAV NA$,DN,1,0,MS,ME
310 IF ERR>0THEN PRINT"SAVING FILE FAILED, ERROR";ERR:END
320 DCHK NA$:IF ERR>0 THEN PRINT "VERIFY FAILED, ERROR";ERR:END
330 FE=PEEK(AD)+256*PEEK(AD+1)
340 POKE(FE+8),S1:POKE(FE+9),S2
350 POKE(FE+10),E1:POKE(FE+11),E2
360 POKE(FE+12),TP: REM RESTORE ORIGINAL
370 POKE BA+31,1:DISK 6:REM FORCE SAVING FAT
380 SEL DR:CLEAR
390 END
900 REM CHECK YE VERSION
910 YE=-1:BA=0:IF PEEK(190)>223 THEN YE=1
920 IF PEEK(190)<160 THEN YE=2
930 IF PEEK(190)=208 THEN YE=0
```

```
940 IFYE=1 THEN BA=57344
950 IFYE=2 THEN BA=26624
960 IF BA=0 THEN PRINT"YE-DOS NOT PRESENT":END
970 RETURN
```

******* Disk Controller Interface**

DISK CONTROLLER BOARD FROM ELEKTOR (Almost identical to OSI 610 BOARD)

PIA DATA A : C000

PIA DATA B : C002

FCD Connector PIN layout (on a 610 Floppy controller board):

<$C002> <PIN>,<PORT> <COMMENT>

HEAD LOAD 1,PB7 (ELEKTOR combined HL and Step (to disable drive selector)

MOTOR ON 2,PB6 (ELEKTOR not used) -> **ONLY on modified 610 board**

DRIVE SEL0 3,PB5 (Drive1 :PB5=1,PA6=0)

SIDE SEL 4,PB4 (ELEKTOR option) -> **ONLY on modified 610 board**

STEP 5,PB3

DIR 6,PB2

Not used 7,PB1 (ELEKTOR not used) ERASE Enable (TRIM ERASE)

WE 8,PB0 For YE-DOS, signal has to be inverted for the drive!!!

WD 9,ACIA Write Data to Disk Drive (FM coded)

RXC 10,ACIA Receive Clock

RD 11,ACIA Read Data

POWER 12,13 are GROUND, 14 is +5V -> **ONLY on modified 610 board**

<$C000>

INDEX 17,PA7

DRIVE SEL1 18,PA6 (Drive2 :PB5=0,PA6=1) -> **ONLY on modified 610 board**

WPROTECT 19,PA5

READY1 20,PA4 (ELEKTOR PA4=GND) (MY BOARD DRV RDY if available)

SECTOR 21,PA3 (ELEKTOR PA3=5V) (not used)

FAULT 22,PA2 (ELEKTOR PA2=5V) (not used)

TRK00 23,PA1 (ELEKTOR TRK00)

READY0 24,PA0 (ELEKTOR PA0=GND) (MY BOARD DRV RDY if available)

Serial Disk Data port:

ACIA CONTROL: C010

ACIA DATA : C011

******* YE-OSI DOS VECTOR/PARAMETER TABLE:**

6800: JUMP SEARCH FILE (0)

6802: JUMP READ FILE OR DELETE (1)

6804: JUMP WRITE FILE (2)

6806: JUMP FORMAT OR WRITE BOOT SECTOR (3)

6808: JUMP CHECK DRIVES ATTACHED AND LOADS FAT (4)

680A: JUMP READ SELECTED FILE (5)

680C: JUMP WRITE DISK FAT (6)

680E: JUMP LOAD DISK FAT (7)

DOS INITIAL DISK PARAMETER TABLE

6810: COPY OF START/END ADRESS OF BASIC 2x2

6814: DRIVE FLAGS 4x

FF= Drive not available

00= Drive OK

6818: Last Drive Index

6819: Step delay in ms (24)

681A: $C002 PIA Port Mirror (FE)

681B: PIA PORT B MASK (FE)

681C: ACTUAL TRACK ON READING / SECTOR COUNTER FOR WRITING

681D: Used space sector counter High

681E: Drive Double sided (FF), default single sided (00)

681F: FAT has changed if >00

6820: Selected Drive (0=A side 0, 2=B side 0)

6821: Read or Delete flag (00 = READ)

6822: Low FAT File Name Pointer / Free sector count LOW

6823: High FAT File Name Pointer

6824: USER Define: Search free (FF) or take next (00) sector

6825: USER defined: FAT Single Sector flag 6825, 00(default) or single with zero or FF with 6822/32

6826: READ ($FF) Bit or VERIFY / FULL FORMAT ($00)

6827: Error Code ($00)

6828: DOS BOOT Start entry

682B: DISK ID Vector Address

682D: DISK FAT Vector Address

682F: DISK TRK/SEC MAP Vector Address

A2/A3: Search Filename Pointer

9F: Length of Filename

****** DISK CALLS in detail ******

******* DISK 1**

READ FILE/SECTOR

Start sector will be TRK_T ($EC) and SEC_T ($ED)

Flag $6826: Verify (00) or default Read Data (FF)

Val $681C: Length of data file in sectors

Data Adr : Data pointer to memory DATA_S (F0-F1)

Start : FDC_T pointer Start Track, Start Sector (EE-EF)

Next : $6822/23 TRK/SEC will show next Sector in chain

******* DISK 2**

WRITE FILE

File length is max. 128 sectors or 32kB

Num $681C: Number of sectors (1...128)

Num $6820: Selected Drive (0=1 side A, 2=2 side A)

Flag $6824: Search free default (FF) or take next (00) sector for file

If (00), start sector will be TRK_T ($EC) and SEC_T ($ED)

and all following sectors will be incremented (FAT bits are set)

If (00), Number of sectors will be occupied in any case (if used or not)

Flag $6825: FILE FAT LIST default (00) will end with (00 00) or (FF) by $6822/23 TRK/SEC

Adr $E0 : Data pointer to memory DATA_S (F0-F1)

******* DISK 3**

FORMAT OR WRITE BOOT SECTOR

Flag $6826: "00" will clear and format entire disk

"FF" (default) , Format only Boot sector, disk content will remain.

Flag $6820: Selected Drive (0=1 side A, 2=2 side A)

Flag $681E: Drive Double sided (FF), default single sided (00)

EXAMPLE: To format a "blank" 160k disk in drive 2 you have to:

Set $6826=0 , $6820=2 , $681E=0, "DISK 3" , $6826=255

IMPORTANT:

In this DOS version, format will only work, if DOSSUP is loaded correctly. There is a vector pointing to the memory area within DOSSUP.

******* DISK 4**

CHECK DRIVES ATTACHED AND LOADS FAT (4)

Will Check for available drives and reload FAT from drive 0 or lowest attached drive

******* DISK 5**

READ FILE FROM FAT POINTER

File will be FAT DATA POINTER (F5/F6) to FAT text entry

Flag $6826: Verify (00) or default Read Data(FF)

Data Adr : Data pointer to memory DATA_S (F0-F1)

Start   : FDC_T pointer Start Track, Start Sector (EE-EF)

******* DISK 6**

WRITE DISK FAT (6)

Will write FAT data from $7C00.. to currently active drive, if FAT data has been changed

$681F indicates FAT Changes if >00

******* DISK 7**

Will load FAT data from currently active drive to memory $7C00..

******* YE-OSI DOS FAT structure in memory:**

SECTOR Table ($50 bytes), Starts at $7C00, BIT 0=Sector 0, BIT 1=Sector 1, ...

DOS VERSION INFO ($10 bytes), Starts at $7C50. Should end with "00"

MAX 71 File Entries in FAT, Starts at $7C60 (.. $7FFB), each 13 bytes in size.

Directory table 13 bytes each -

6 Bytes for	File name
2 Bytes for st,ss	Start Track, Start Sector
2 Bytes for Ls,Hs	Low, High Start address of data
2 Bytes for Le,He	Low High End address of data
1 Byte for ft	File Type and protection status
File type example:	13(SYSTEM),10(BINARY),00(BASIC)

<$10 -> DATA FILE

>=$10 -> EXEC FILE

>=$20 -> OTHERS

BIT 0=0 -> NORMAL

BIT 0=1 -> EXECUTABLE

ASS has $13 (executable)

DATA TRACKS 2...79 or 2...39 / 2... 34

Each track includes 8 Sectors with DATA, GAP and Lead In (Pre-Formatted). This will allow to read / write single sectors without reading the whole track before.

IMPORTANT:

This DOS will only run on 1Mhz machines, actually on an average CPU clock of an C1P and UK101 (about 0,991 Mhz). Modified C1P's should clock at a max. CPU clock of 1.0 Mhz, to guarantee correct floppy data rates and timing. The Emulator will work at any selected CPU speed.

YE-DOS has been tested on newer 1.44MB drives as well as old Shugart 400 5.25 drives and works well. On some early 3.5 floppy drives it may fail, when the time of switching from WE active to Read valid data takes more than 800 usec.

**** File Type and protection status:

BAS=0 RWn (BASIC Token Memory loads typically to $0300)

BAS=1 RWa

BAS=2 R n

BAS=3 R a

COM=16 RWn (MACHINE CODE Binary Code)

COM=17 RWa

COM=18 R n

COM=19 R a

SEQ=32 RWn (SEQUENCIAL comma separated data, same as binary data)

SEQ=33 RWa

SEQ=34 R n

SEQ=35 R a

VAR=48 RWn (VARIABLE , sane as binary data)

VAR=49 RWa

VAR=50 R n

VAR=51 R a

Protection status:

RWn +0 Read Write normal

RWa +1 Read Write autorun

R n +2 Read Only normal

R a +3 Read Only autorun

******* YE-OSI DOS Track structure:**

Track 0 (@ $0000 of IMG file)

xx,yy High, Low Start address (6800)

zz Cluster number of 256 bytes (08)

DOS Start code 6800-6FFF (2 kBytes)

Track 1 (@ $0900 of IMG file)

Sector table, Directory, and duplicate Sector table, Directory

$0900: Sector table, 1 bit = 1 Sector starting with highest bit (1 byte = 1 track) max 80/40/35 tracks or 640 sectors or 160kB / 80kB

(First 2 FAT bytes are FF always used for TRK00 and TRK01)

Followed by Directory name table 13 bytes each (max. 71 entries)

End of Directory with Checksum

Followed by copy of directory table

Track 2 (@ $1200 with length of 0900)

Track 2...79/39/35 with Sector 0...7

FC=Sync ID

FE=Track Sector ID

F7=Chksum ID

FB=Data ID

FF=Timing filler and Read/Write change zones

Each sector starts with track sync ID(FC):

Sector ID FC followed by physical Sector gap

FE Sector Info --- Track number, Sector number, Next track, Next sector, F7 Checksum ID, Sum of Sector info

FB Sector Data ID --- Sector data: 256 data bytes

F7 Checksum ID --- Sum of sector data

followed by physical Sector Write runout gap of about 1.0ms

Track Structure:

Track Header:

1 - START OF INDEX PULSE (1 to 0) plus 5ms delay to start reading

2 - 3 Bytes FF (to sync controller)

3 - 3 Bytes track Sync ID (Space "FF FF FC")

4 - 9 Bytes PRE-Sector header (Header "00 01 02 03 04 05 06 07 08")

Sector Header:

5 – 7..13 Bytes Inter-Sector GAP (runout for floppy +-1.5% speed tolerances)

6 - 3 Bytes Sync ID (Space "FF FF FC")

7 - 3 Bytes R/W switching zone ("FF FF FF")

8 - 3 Bytes Sector Start Info ID (Space "FF FF FE")

9 - 4 Bytes Sector Info ("TRK SEC NEXT_TRK NEXT_SEC")

10 - 2 Bytes CHECKSUM ID ("F7, CHECKSUM")

11- 257 Bytes Data ID plus Data("FB, 256x DATA....")

12- 2 Bytes CHECKSUM ID ("F7, CHECKSUM")

Next Sector Header:

11 - 12 Bytes Inter-Sector GAP

.....

Remark:

Due to the Inter-Sector GAPs, single sectors can be written without reading the entire track before (direct sector access method).

CPU cycle timing is not critical as the sync ID's(FC) are fixed to allow this Sector insertion method. Will run only on unmodified C1P and UK101 machines in real. (2 Mhz machines may work at double Floppy controller frequencies, this has not been verified)

Floppy step rate is set by default to 24ms (becomes 12ms on 2Mhz).

Within Emulation, the CPU clock speed does not cause changes nor problems.

YE-DOS

Listing

```
;
;                       (c) Copyright TB 2024
;
;           Date:               March 1 2024
;
;           CPU:                MOS Technology 6502 (MCS6500 family)
;
;    VERSION:       Version X3.54_35/80 - Updated for 35/80 Track drives
;                                HEAD LOAD will be present with MOTOR ON,
;                                Resting on TRACK00 to protect disk errors during POWER cycles
;                                Runs on old 5 1/4 Shugart drives as well as 3.5 inch 1.44Mb drives
;                                Will not work on some older 3.5 inch drives with longer delay at end of WE
;                                Disk drives "must" have internal pull-up resistors
;                                Relocated to $6800
;                                DOS=$E000, DOSSUP=$E900, RAM=$F200, FAT=$F400, ROM=$F800
; moved to                       DOS=$6800..6FFF, DOSSUP=$7140+$0200, RAM=$7A00, FAT=$7C00, ROM=standard SYM600
;                                TRACK00 in 8E1 format, all other are 8N1 !!!
;                                WITHOUT DOSSUP, YE-DOS is not present/uncomplete/cleared by Cold start

ORG_POS     = $6800

DISK_TYPE = 0               ; Compiler option: Disk type 0=35 tracks, 1=80 tracks to be set
WE_TYPE         = 0                 ; 0=active low or 1=active high

D_VERS          = 54                ; Version number
ORG_SUP         = $7040             ; DOSSUP start address
DOS_POS         = $6800  ; BOOT sector DOS start address
FATVER          = FAT_D+$50         ; VERSION TEXT STRING
FATCHANGE       = ORG_POS+$1F       ; FAT Change info flag
PAR_STOR        = $0230             ; Storage if command parameters (7 Bytes)
BASIC_EXT       = $022C             ; Basic extension Vector


ACIA_S          = $F000             ; SERIAL ACIA Control Port for turning low res graph on/off

BASIC_16_FLOAT = $AFC1              ; Convert Fixed Point to Floating Point
BASIC_OUT       = $A8E5             ; BASIC Character output
BASIC_CRLF      = $A86C             ; BASIC LINE Return
BASIC_LPT       = $BC               ; Basic Line Processing Vector in Zero-Page
BAS_GET_CR      = $00C2             ; BASIC GET CURREMT CHAR FROM BASIC LINE
BASIC_ALPHA     = $AD81             ; BASIC CHECK CHARACTER FOR ALPHA
BASIC_FCERR     = $AE88             ; BASIC FC ERROR
BASIC_EVAL      = $AAC1             ; BASIC EVALUATE EXPRESSION
BASIC_B2B6      = $B2B6             ; BASIC Free Temp String
BASIC_CLEAR     = $A47A             ; BASIC CLEAR
BASIC_RUN       = $A5FC             ; BASIC RUN
BASIC_A477      = $A477             ; INITIALIZE, KIND OF BASIC CLEAR
BASIC_POKE_PARM = $B3FC             ; BASIC Evaluate like POKE (adress, value)
BASIC_FINDL     = $A432             ; BASIC SEARCH BASIC LINE NUMBER, ADR in AA-AB
BASIC_G16B      = $AAAD             ; BASIC GET 16BIT ARG FROM BASIC LINE
BASIC_RSTOR     = $A621             ; BASIC FINALIZE RESTORE
BASIC_B408      = $B408             ; BASIC Convert FLOAT to INT, Result in 11-12
```

```
DISK_BOOT       = $FC00         ; Entry to disk boot in ROM

HORZ_SIZE       = $FFE1         ; <32 or >32 will indicate horizontal screen resolution (32 or 64)
VERT_SIZE       = $FFE2         ; 0 will indicate 2k, 1 is 4k Screen memory size

VIDEO_RAM       = $D000


FAT_D           = ORG_POS+$1400 ; FAT Memory area $F400
FAT_S           = FAT_D+$60     ; FAT start of name are
FAT_E           = FAT_D>>8+4    ; FAT Memory wnd $F8
FAT_ID          = FAT_D+$50     ; DISK Title 16 Bytes
STACKS          = ORG_POS+$0900 ; DOS TEMP Stack Area

FreeM           = FAT_D-$0100   ; Free Memory area (moved to $F300 up)
Unused          = FAT_D-$0200   ; Unused 256 Bytes

LE8C0           = ORG_POS+$0800 ; Stack Storage change for standard ROMs
LE8E0           = ORG_POS+$0820 ; Zero Page Storage

DOS_PARAM       = ORG_POS+$10   ; DOS Parameter table for BASIC Start and End


STOP  = 3               ; DEBUGGING STOP CODE
PIA_PA = $C000          ; PIA PORT A
PIA_PB = $C002          ; PIA PORT B
PIA_DA = $C001          ; PIA DIR A
PIA_DB = $C003          ; PIA DIR B

ACIA_C = $C010          ; ACIA Control Port
ACIA_D = $C011          ; ACIA Data Port

ZEROP  = $0000          ; Zero Page Start Address

        .IF DISK_TYPE==0
TRK_M = $22             ; Max number of tracks 40/35, changed to 35 to work with old SHUGART 400L drives
        .ELSE
TRK_M = $4F             ; Max number of tracks 80
        .ENDIF

STEP_D = 24             ; 24ms TRK TO TRK delay
MOT_S  = 100            ; 500ms Motor Start delay in 5ms times x in ms
TRK_D  = 20             ; Track settle time in ms after last step

DEL_1  = $0D            ; Delay about 1.35 byte (107 usec) (13*7 +13 +3) , also used at end of WE acive to transfer
last byte
DEL_2  = $35            ; Delay to bridge 6xFF (380 usec + FC find delay of 80 usec at end of sector)

PRE_GAP= 5              ; GAP before Data block (bytes) increase form 4 to 5
POS_GAP= 12             ; Post GAP at the end of sector (bytes)  ** reduce from 14 to 12 for Mark's emulator
ID_GAP = 7              ; GAP after Track ID (bytes)
TRK_GAP= 3              ; GAP Lead In (bytes) at start of Track


ROMOUT_V = $FFEE        ; ROM Output Vector
ROMINP_V = $FFEB        ; ROM Input Vector
```

```
BASIC_WARM = $0000      ; BASIC WARMSTART
BASIC_INI = $BDF6       ; INI BASIC with Start Vector in X,Y
BASIC_COLD = $BD11      ; BASIC COLD START asdress

MON_ROM  = $FE00        ; ROM MONITOR ENTRY

                        ; Zero Page Parameter
File_L = $94            ; File Name Lenghth
X00A2 = $A2             ; A2-A3 DOS FILNAME TEXT VECTOR
ErrCnt = $E0            ; Error counter (max 4)
XTEMP = $E1
TS_IDX = $E2            ; Track/Sector Index to EF00
Side_T = $E3            ; Side temporary variable
DSum  = $E4             ; Data Sum Value
YTEMP = $E5

TRK_T = $EC             ; Track Temp
SEC_T = $ED             ; Sector Temp

                        ; File discriptor block
FDC_TS = $EE            ; EE-EF  File Descriptor Temp
DATA_S = $F0            ; F0-F1  Data Pointer
DATA_E = $F2            ; F2-F3  DATA PINTER END ADRESS
TYPE  = $F4             ; F4 DATA TYPE
FAT_P = $F5             ; F5-F6  FAT DATA POINTER OR OTHERS

        .ORG    ORG_POS-3
        .db     ORG_POS>>8, ORG_POS&255         ; BOOT VECTOR
        .db     $08                             ; BOOT LENGTH for 8E1 TRK 00

        .ORG    ORG_POS
;
;*********************************************************************
;               DOS COLD START
;*********************************************************************

LE000:
   bne  BOOT_S                      ; LATER JUMP SEARCH FILE (0)


   ; DOS VECTOR LIST
LE002: .db   LE4E4&255, LE4E4>>8     ; JUMP READ FILE OR DELETE (1)
LE004: .db   LE59B&255, LE59B>>8     ; JUMP WRITE FILE (2)
LE006: .db   LE389&255, LE389>>8     ; JUMP FORMAT OR WRITE BOOT SECTOR (3) >> placed in DOSSUP
LE008: .db   LE335&255, LE335>>8     ; JUMP CHECK DRIVES ATTACHED AND LOADS FAT (4)
LE00A: .db   LE4DB&255, LE4DB>>8     ; JUMP READ SELECTED FILE (5)
LE00C: .db   LE5F4&255, LE5F4>>8     ; JUMP WRITE DISK FAT (6)
LE00E: .db   LE66E&255, LE66E>>8     ; JUMP LOAD DISK FAT (7)

        ; DOS INITIAL DISK PARAMETER TABLE

LE010: .DB   $01, $03
        .DB     $01, $03   ; COPY OF START/END ADRESS OF BASIC

LE014:  .db   $00, $FF      ; DRIVE FLAGS
        .db   $FF, $FF      ; FF= Drive not available
                            ; 00= Drive OK
```

```
                                    ; 32= Normal
LE018:  .db   $00                   ; Last Drive Index
LE019:  .db       STEP_D            ; Step delay in ms

                  .IF WE_TYPE==1
LE01A:  .db       $FE               ; PIA PORT B LAST VALUE (typical: SEL, SIDE, MOTOR & HEAD LOAD)
LE01B:  .db   $FE                   ; PIA PORT TEMP B MASK for WE(PB0), DIR(PB2),STEP(PB3)
                  .ELSE
LE01A:  .db       $FF               ; PIA PORT B LAST VALUE (typical: SEL, SIDE, MOTOR & HEAD LOAD)
LE01B:  .db   $FF                   ; PIA PORT TEMP B MASK for WE(PB0), DIR(PB2),STEP(PB3)
                  .ENDIF

LE01C:  .db       $00               ; ACTUAL TRACK ON READING / SECTOR COUNTER FOR WRITING
LE01D:  .db       $01               ; Used space sector counter HIGH/ MotorOn/Headload flag (00)=dont reset
LE01E:  .db       $00               ; Drive Double sided (FF), default single sided (00)
LE01F:  .db   $00                   ; FAT Changes if >00
LE020:  .db   $00                   ; Selected Drive (0=A side 0, 2=B side 0)
LE021:  .db   $00                   ; Read or Delete flag (00 = READ)
LE022:  .db   $00                   ; Low FAT File Name Pointer or Free sector count LOW
LE023:  .db   $00                   ; High FAT File Name Pointer
LE024:  .db   $FF                   ; USER Define: Search free (FF) or take next (00) sector
LE025:  .DB       $00               ; USER DEF:FAT Single Sector flag LE025, 00(default)  or single with zero or FF
with E022/23
LE026:  .db       $FF               ; READ ($FF) Bit or VERIFY/ FULL FORMAT ($00)
LE027:  .db   $00                   ; Error Code ($00)


BOOT_S:                             ; DOS Boot Routine LE028
        JMP   LE74B
LE02B:  .DB       FAT_ID&255, FAT_ID>>8   ; DISK ID Vector
LE02D: .DB        FAT_S&255, FAT_S>>8     ; DISK FAT Vector
LE02F: .DB        FreeM&255, FreeM>>8     ; DISK TRK/SEC MAP Vector

;*******************************************************************************

;                                   ; **** USE WITH CARE !! *****
LE028:                              ; ***** Store ZERO PAGE and STACK ****
        php                         ; Save $00E0 to $E8E0 (ZERO PAGE)
        sei                         ; Save Stack to $E8C0 (STACK)
        tsx                         ; for 32 Bytes
        txa                         ; inccludes Status and Stack pointer as well !!
        pha
        ldx       #$E0
LE02FN:
        lda       ZEROP,x           ; Save $E0..FF
        sta       LE8E0-$E0,x
        pla                         ; Save 32x Stack values
        sta       LE8C0-$E0,x       ; Top stack lands in $E8C0... (stack pointer)
        inx
        bne       LE02FN
        lda       LE8C0+$03         ; Calling Stack pointer
        pha
        lda       LE8C0+$02
        pha
        rts                         ; Will on restore return to previous caller

;*******************************************************************************
```

```
;
LE044:                                  ; **** USE WITH CARE !! *****
        lda     LE8C0                   ; ***** Restore ZERO PAGE and STACK ****
        clc                             ; from $E8C0/$E8E0
        adc     #$1F
        tax
        txs                             ; Restore Stackpointer before call
        ldx     #$1F
LE04E:
        lda     LE8E0,x
        sta     $E0,x
        lda     LE8C0,x
        pha
        dex
        bpl     LE04E
        pla                             ; old saved Stackpointer
        plp                             ; Restore old status
        pla                             ; remove old calling adress (from store)
        pla                             ; remove old calling adress (from store)
        rts                             ; Return to previous Return adress


;*******************************************************************************
;
                                ; *** Clear Checksum, Error and Head Load FDC plus settle time ***

LE075:
        jsr     LE15C           ; Clear Error and Checksum
        jsr     LE0DC           ; HEAD LOAD ON and WE OFF
        jmp     LE0CD           ; Delay Settle time


;*******************************************************************************
;
LE097:                          ; ****** SET DISK WRITE MODE ******

        lda     LE01A           ; Port B Setup
        .IF WE_TYPE==1
        ora     #$01            ; MASK WE PB0=1 to Port B
        .ELSE
        and     #$FE            ; MASK WE PB0=0 to Port B
        .ENDIF
        sta     PIA_PB          ; ENABLE WE
        sta     LE01B           ; Save TEMP PORT B value
LE0AC:
        rts

TEST_PROT:                      ; ****** Check Write protection ********
        lda     #$20
        bit     PIA_PA          ; Check WRITE PROTECT PA5
        bne     LE0AC
        lda     #$0E
        jmp     LE380           ; ERROR 14 DISK IS WRITE PROTECTED

;*******************************************************************************
;
LE0A9x:                         ; *** 5 msec Delay ***
```

```
        ldx     #$05            ; Load 5 msec delay MOD
;*******************************************************************************

LE0A9:                          ;*** 1ms  DELAY (1ms times X) ***
        ldy     #$C6
LE0AB:
        dey
        bne     LE0AB           ; Loop back
        nop
        nop
        dex
        bne     LE0A9           ; Loop back
        rts


;*******************************************************************************
;
LE0C9:
        ldx     #MOT_S          ;***** Delay Motor Start 256ms + MOT_S msec *******
        jsr     LE0A9
        beq     LE0A9           ; Always jump to 1ms  DELAY (1ms times X)

LE0CD:
        ldx     #TRK_D          ;***** Delay Track settle time msec *******
        bne     LE0A9           ; Always jump to 1ms  DELAY (1ms times X)

;*******************************************************************************
;
LE0CF:                          ;***** WRITE(DATA_S) 256-(Y) bytes to FDC + checksum *****
        lda     (DATA_S),y
        jsr     LE124           ; Sum to Checksum and Write to Disk
        iny
        bne     LE0CF
        inc     DATA_S+1
        lda     DATA_S+1
        rts

;*******************************************************************************
;

LE0DC:                          ;*** HEAD LOAD ON , WE OFF ****
        lda     #$02
LE0DE:
        bit     ACIA_C          ; Wait until all write bytes have passed Accia
        beq     LE0DE

        jsr     Delay_1         ; 1 Byte additional delay for ACIA to complete sending byte

        lda     LE01A           ; Port B Setup (WE is off)
        sta     LE01B
        and     #$3F            ; Keep HL and Motor active
        sta     PIA_PB
        rts
;
LE0E5:
        .db     $01, $02, $04, $08, $10, $20, $40, $80
;
```

```
;*******************************************************************************
;
LE0EE:                          ;*** READ byte and add to Checksum ***
        jsr     LE119           ;READ single FDC byte
LE0F1:
        pha
        clc
        adc     DSum            ;Add to Checksum
        sta     DSum
        pla
        rts
;*******************************************************************************



Delay_1:
        lda     #DEL_1          ;Short <1 byte delay ()

LE0FF:
        sec                     ;(2)
        sbc     #$01            ;(2)
        bne     LE0FF           ;(3) Total 7
        sec
        rts                     ;(13) JSR Extras
;*******************************************************************************
;
LE106:                          ;**** WRITE CHECKUM Marker plus Checksum to FDC ****
        lda     #$F7            ;Checksum Marker

LE108:                          ;**** Write Marker to FDC and clear Checksum ****
        jsr     LE127           ;WRITE byte to FDC

        cmp     #$F7            ;Was Checksum ?
        bne     LE114           ;if not, jump to Clear Checkum
        lda     DSum            ;Write Checkum value
        jsr     LE127           ;WRITE byte to FDC

LE114:                          ;**** Clear Checkum ****
        lda     #$00
        sta     DSum            ;Clear Checkum
        rts

;*******************************************************************************
;
LE119:                          ;**** READ single FDC byte ****
        lda     #$01
LE11B:
        bit     ACIA_C
        beq     LE11B
        lda     ACIA_D
        rts
;*******************************************************************************
;
LE124:                          ;*** WRITE byte and add to Checksum ***
        jsr     LE0F1           ;Add Accu to Checksum

LE127:                          ;*** WRITE byte to FDC ***
```

```
        pha
        lda     #$02
LE12A:
        bit     ACIA_C
        beq     LE12A
        pla
        sta     ACIA_D          ; Write byte to FDC
LE133:  rts

;*******************************************************************************
;
                                ; *** READ and check for FF and SYNC byte ***
LE144_FCX:
                                ; *** Bridge xx bytes SECTOR START GAP for reading or writing, with reset ***
        jsr     Delay_1         ; Give some extra delay in GAP (about 80 usec)
        jsr     LE227           ; Reset ACIA
LE144_FC:
        lda     #$FC            ; *** Find fixed SECTOR FC SYNCRON without reset or delay (Fast) ***
        bne     LE144_S

LE144_FEX:                      ; *** Bridge xx bytes SECTOR GAP for reading with reset ***
        jsr     Delay_1     ; Give some extra delay in GAP (about 50usec)
        jsr     LE227           ; Reset ACIA
        lda     #$FE            ; *** Bridge xx bytes SECTOR GAP for reading without reset or delay (fast) ***

LE144_S:
        sta     XTEMP           ; Counter preset (max 3 full loops for FF FF FC or 1 loops for FF FF FE)
        sta     YTEMP
LE144:
        inc     XTEMP
        beq     LE13F           ; ERROR 2 (Sync not found after few attemps)
LE148:
        jsr     LE119           ; read next FDC byte
        cmp     #$FF
        bne     LE148           ; wait for 1st FF
LE14F:
        jsr     LE119           ; read 2nd FF byte
        cmp     #$FF
        bne     LE148           ; Go back and wait for next two FF's
LE14FX:
        jsr     LE119           ; read next FDC byte
        cmp     #$FF
        beq     LE14FX          ; Go back and wait for not equal FF

        cmp     YTEMP           ; Check for FC or FE
        bne     LE144           ; Start all over with counter +1
        rts
LE13F:
        lda     YTEMP
        cmp     #$FE
        beq     LE140           ; ERROR 2
        lda     #$01            ; Sync byte FC no found ERROR 1
        bne     LE142
LE140:
        lda     #$02            ; Sync byte FE at start sector no found
LE142:
        jmp     LE380
```

```
;*******************************************************************************
;
LE14E:                          ;******** INCREMENT Error count ********
        inc     ErrCnt
        lda     ErrCnt
        cmp     #$04
        bcc     LE169           ;More than 3 FDC error reads ?
        lda     #$03
        jmp     LE380           ;ERROR 3 Searching track error, not found

;*******************************************************************************
;
LE15C:                          ;*** Clear Error and Checksum ***
        lda     #$00            ;*** CLR and READ all 00 until SYNC byte ***
        sta     ErrCnt          ;Clear Error counter
        beq     LE114           ;jump always to Clear Checksum and return

;*******************************************************************************
;
LE163:                          ;*** READ Y- FDC bytes ****
        jsr     LE119           ;read next FDC byte
        dey
        bne     LE163           ;Loop
LE169:
        rts
;
LE16AX:                         ;*** WRITE Y-FF FDC bytes + checksum ****
        lda     #$FF            ;ADDED $FF option for FAT Format
        bne     LE16C           ;Skip next two bytes

LE16A:                          ;*** WRITE Y-FF FDC bytes + checksum ****
        lda     #$00            ;$00 option for FAT Format
LE16C:
        jsr     LE124           ;Sum to Checksum and Write to Disk
        dex
        bne     LE16C
        rts

LE16D:                          ;Fast WRITE $FF version x-times
        lda     #$02
LE16E:
        bit     ACIA_C
        beq     LE16E
        lda     #$FF
        sta     ACIA_D          ;Write byte to FDC
        dex
        bne     LE16D
        rts


;*******************************************************************************
;
LE0B4:                          ;*** Correct Index, if double sided ***
                                ;will update track position on second side as well
        pha
```

```
        bit     LE01E
        bpl     LE0BE           ; jump if single sided (00)
        txa
        eor     #$01            ; Set X for second FDC side
        tax
LE0BE:
        pla
        sta     LE014,x         ; Store new track number
                                ; next GET Track Position of Drive (E018) and store

;*******************************************************************************
;
LE18B:
        ldx     LE018           ; *** GET Track Position of Drive **
        lda     LE014,x
        sta     FDC_TS          ; Store to Track
        lda     #$00
        sta     FDC_TS+1        ; Sector=0
        rts

;*******************************************************************************
;
LE198:                          ; *** READ SECTOR dummy and FDC_TS+1=Sec count Sector 0 ***
        jsr     LE144_FCX       ; Bridge xx bytes SECTOR START GAP for reading or writing, with delay and reset
LE19C:
        jsr     LE144_FEX       ; Delay and bridge xx bytes SECTOR START GAP for reading with reset
        ldy     #09             ; 9 Bytes
        jsr     LE163           ; READ 9 bytes
        jsr     LE163           ; READ 256 Bytes (Y was 00) (incl Checksum)
        inc     FDC_TS+1        ; Increment Sector counter
        lda     #DEL_2          ; Bridge several bytes at end of sector
        jmp     LE0FF           ; Delay Subroutine x times 7usec plus 13 and return

;*******************************************************************************
;                               **** SELECT DRIVE ACCORDING TO E020 ****
LE1ADX:
        lda     LE020           ; Selected drive number
        sta     LE018           ; Store Drive number index

LE1B0:                          ; *** Set Drive Port A/B depending on actual Drive Index ***

        lda     LE01A           ; Check if HL and Motor already acive or not
        pha
        lda     LE018           ; Get Drive Offset (Index)
        and     #$03            ; Only 0..3 allowed
        tax
        lda     LE1C3,x         ; Get PORT A MASK (Drive Number)
        sta     PIA_PA          ; Store to A (just PA6 counts for Drive 0/1 or 2/3)
        .IF WE_TYPE==1
        and     #$3E            ; Turn Motor on PB6=0 , Head Load PB7=0, WE OFF PB0=0, DIR=1
        .ELSE
        and     #$3F            ; Turn Motor on PB6=0 , Head Load PB7=0, WE OFF PB0=1, DIR=1
        .ENDIF
                                ; All others are OFF (STEP, ERASE ENABLE, FAULT)
        sta     LE01A           ; Store new default value
        sta     PIA_PB          ; Store to B PB6 (Low current) must be HW modified to become Motor ON
        pla
```

```
        and     #$40            ; Check if MOTOR (PB6) was on before
        beq     LE1C2           ; Motor has been active     , no more delay, just return

LE1BF:                          ; **** Motor startup Delay or leave on RDY=0 ****
        ldy     #MOT_S
LE1C1:                          ; WAIT FOR FDC READY (will become low, when drive spon up)
        tya                     ; Ready Drive 1&2 on PA0 (Pin 34 FDC) PA0 and PA4 to be connected
        jsr     LE0A9x          ; Delay 5ms (A is not changed)
        tay
        dey
        beq     LE1C2           ; Return after Motor delay, ignore RDY
        lda     PIA_PA          ; Check PA0 READY (has to be 0)
        lsr
        bcs     LE1C1           ; Wait for FDC READY (must be available)
LE1C2:
        rts                     ; Ready became 0

;*******************************************************************************
;
LE1C3:
    .DB   $FF, $DF, $BF, $9F    ; Port A6/B5 mask (0..3) corrected

;*******************************************************************************
;
LE1C8:
        sta     PIA_PB          ; *** Store A to PORT B plus STEP ****
        ldx     #$01            ; Load 1 msec pre-delay
        jsr     LE0A9           ; Return with Delay in ms of Step Rate
        and     #$F7            ; SET PB3 STEP to 0
        sta     PIA_PB          ; Store B
        ora     #$08            ; SET PB3 Step to 1
        sta     PIA_PB          ; Store B
        ldx     LE019           ; Load Step Rate
        dex                     ; minus pre-delay
        jmp     LE0A9           ; Return with Delay in ms of Step Rate

;*******************************************************************************
;
LE1DB:                          ; *** Update Drive present & GO to TRK00 ***
        jsr     LE1B0           ; Set Drive Port A/B depending on Drive Index (is in X)

LE1DE:
        lda     #$02            ; *** Check and find TRACK 00 ***
        bit     PIA_PA
        beq     LE1F1           ; IF TRACK 00 jump
        lda     LE01A           ; Load Default MASK B and WE Off
        ora     #$04            ; SET PB2 to 1 DIR OUT towards TRK00
        jsr     LE1C8           ; Store A to PORT B plus do STEP
        beq     LE1DE           ; Always jump

LE1F1:
        lda     #$00            ; Track 00 found

LE1F3:                          ; *** Store A to Drive Track Present Flags ***
        ldx     LE018           ; Get Last Drive Index
        sta     LE014,x         ; Store new Track number to Flag
        jmp     LE0B4           ; Correct Index, if double sided and return
```

```
;*******************************************************************************
;
LE1FC:                          ;*** SET Read MODE Wait for INDEX  ***
                                ;*** plus delay for gap        ***

                                ;****** RAW FORMAT ENTRY ******
        jsr     LE0CD           ; Delay Settle time (previous action delay)
        jsr     LE18B           ; GET Track Position of Drive and set FDC_TS to actual

LE207:                          ;**** SET WE and wait for INDEX, 5/10ms delay  ****
        jsr     LE0CD           ; NEW:Delay Settle time (previous action delay)
LE20D:
        lda     PIA_PA
        bpl     LE20D           ; Wait for INDEX become "1" PA7=1
LE212:
        lda     PIA_PA
        bmi     LE212           ; Wait for INDEX become "0" >> Start processing, when INDEX becomes "0" (early
trigger)
                                ; Start OF INDEX

        lda     LE01B           ; Get TEMP Port B Mask
        sta     PIA_PB          ; Set Port B
                                ; (will set READ or WRITE MODE)

        lsr                     ; New Start Delay, READ (5 ms) Write (10 ms)
        .IF WE_TYPE==1
        bcc     LE224           ; jump on  PB0 WE = "0" (READ)
        .ELSE
        bcs     LE224           ; jump on  PB0 WE = "1" (READ)
        .ENDIF
        jsr     LE0A9x          ; Delay 5ms extra for Write

LE224:                          ;******* GAP plus 5 ms and Return ******
        jmp     LE0A9x          ; Delay 5ms

;*******************************************************************************
;
LE227:
        lda     #$03            ;******* Rest ACIA ********
        sta     ACIA_C
        lda     #$54            ; YE-DOS needs 8N1 to fit on track
        sta     ACIA_C
        rts


;*******************************************************************************
;
LE232:                          ;**** Goto Track (Y) ****
        sty     TRK_T
        cpy     #TRK_M+1
        bcc     LE23B           ; Goto Track (TRK_T)
        lda     #$04            ; ERROR 4 Track or Sector out of range
        jmp     LE380           ; Jump to ERROR
;
LE23B:                          ;**** Goto Track (TRK_T) ****
```

```
        ldx     LE018           ; Get Drive Index
        lda     LE014,x         ; Check if Drive is on Search track ?
        cmp     TRK_T
        beq     LE262           ; Jump if track number is present

        lda     LE01A           ; Get Port B mask (DIR is 1)
        bcs     LE251           ; Jump if Search Track< actual track (carry set)
        and     #$FB            ; Set PB2 DIR to 0 (Inwards)
        inc     LE014,x         ; Add 1 to actual track
        bne     LE254           ; Always jump

LE251:                          ; Search Track is less than actual track
        dec     LE014,x         ; Sub 1 to actual track
LE254:
        jsr     LE1C8           ; Store A to PORT B plus STEP FDC and Delay
        ldx     LE018           ; Get Last Drive Index
        lda     LE014,x         ; Get Drive status / Track position
        jsr     LE0B4           ; Correct Index, if double sided
        beq     LE23B           ; Always loop back until track found

LE262:                          ; Track found
        jmp     LE0CD           ; Delay Track settle time and return

;*******************************************************************************

LE273:                          ; **** WRITE SECTORS according Sector table coming form write file command
****
                                ; OR SINGLE SECTOR ACCORDING LE025

        jsr     LE144_FC        ; Changed: READ all FF until SYNC FC byte, now GAP follows
LE276:
        jsr     LE097           ; SET DISK WRITE MODE
        ldx     #PRE_GAP-1      ; GAP before data block
        jsr     LE16D           ; Fast Write "FF"

        ldy     TS_IDX          ; **** will write a sequence of sectors ****
        iny                     ; TS_IDX is index to table at F300
        iny                     ; showing the sector/track  list (TRK/SEC)

        lda     #$FE
        jsr     LE108           ; Write FDC "FE" and set Checksum =0
        ldx     LE018           ; Drive number offset
        lda     LE014,x         ; Get Drive Status, here Track number
        jsr     LE124           ; Sum to Checksum and Write to Disk
        lda     FDC_TS+1        ; Get sector
        jsr     LE124           ; Sum to Checksum and Write to Disk
        inc     FDC_TS+1        ; increment Sector

        lda     FreeM,y         ; Get next Track Data from F300 Sector Data area
        sta     TRK_T           ; Store to Track (next target)
        jsr     LE124           ; Sum to Checksum and Write to Disk
        lda     FreeM+1,y       ; Get next Sector from F300 Sector Data area
        sta     SEC_T           ; Store to Sector (next target)
        jsr     LE124           ; Sum to Checksum and Write to Disk

        jsr     LE106           ; WRITE CHECKUM Marker plus Checksum to FDC Clear Checksum
        lda     #$FB            ; Data Block start
```

```
        jsr     LE108           ; Write FDC "FB" and Checksum =0
        sty     TS_IDX          ; Save last TS_IDX index

        ldy     #$00
LE2B0:
        lda     (DATA_S),y      ; Write 256 bytes data block from pointer F0/F1 to disk
        jsr     LE124           ; Sum to Checksum and Write to Disk
        iny
        bne     LE2B0
        inc     DATA_S+1        ; Inc data pointer by 256

        bit     LE025           ; Check FAT List finish flag LE025, 00 finsh with zero or FF with E022/23
        bpl     LE2C5           ; Jump on finish with 00 00
        lda     #$00            ; Reset Track and Sector to 00 (never ENDING SECT WRITE)
        sta     TRK_T
        sta     SEC_T
LE2C5:
        jsr     LE106           ; WRITE CHECKSUM Marker plus Checksum to FDC, Clear Checksum
        jsr     LE0DC           ; HEAD LOAD ON and WE OFF (END of SECTOR)
        lda     FDC_TS
        cmp     TRK_T           ; Track found ?
        bne     LE2D6
        lda     FDC_TS+1
        cmp     SEC_T           ; Sector found ?
        bne     LE2D6
        jsr     LE144_FCX       ; Delay and bridge xx bytes SECTOR START GAP for reading or writing with reset
        beq     LE276           ; always jump
LE2D6:
        rts

;*******************************************************************************
;
LE2D7:                          ; ***** Search for next available sector *****
        lda     #$FF            ; Returns Sector in (DSum) and TRK in (Y)
        tay                     ; Start Track is 00
LE2DA:
        iny
        cmp     FAT_D,y
        bne     LE2E9           ; Jump, if Free sector on track found (not equal FF)
        cpy     #TRK_M+1
        bcc     LE2DA           ; Loop back, if not at the end
        lda     #$0A            ; ERROR 10 Disk Full Error
        jmp     LE380
;
LE2E9:                          ; Free sector found
        lda     #$00
        sta     DSum            ; DSum used for sector counter
        lda     #$01
LE2EF:
        and     FAT_D,y
        beq     LE2F9           ; Jump, if empty sector(bit) found
        inc     DSum
        asl                     ; Shift sector bit
        bcc     LE2EF           ; loop back, for sector mask 01..40/80
LE2F9:
        rts
```

```
;*******************************************************************************
;
LE2FA:                          ;** GET FAT SECTOR MASK Bit in (A) and Track in (Y) **
                                ; Changed, was too long for verify
        ldy     SEC_T
        lda     LE0E5,y         ; LOAD FAT Mask (Sector number)
        ldy     TRK_T           ; FROM $EC/ED
        sty     TS_IDX          ; Return with Track number in Y
        rts

;*******************************************************************************
;
;               PIA_PA = $C000          ; PIA PORT A
;               PIA_PB = $C002          ; PIA PORT B
;               PIA_DA = $C001          ; PIA CTRL A
;               PIA_DB = $C003          ; PIA CTRL B


LE314:                          ;**** Sub RESET PIA Ports A & B ****
        ldy     #$00
        sty     PIA_DA          ; Select DDR A
        lda     #$40            ; PA6 Output, All other Input Ports
        sta     PIA_PA          ; Sett DDR A
        ldx     #$04
        stx     PIA_DA          ; Select PORT A Register PA6 keeps undefined (probably high)
        stx     PIA_DB          ; Select PORT B Register (first set data)
        tya
        .IF WE_TYPE==1
        ldy     #$FE            ; A=$00, Y=$FE
        .ELSE
        ldy     #$FF            ; A=$00, Y=$FF
        .ENDIF
        sty     PIA_PB          ; Set Port B to $FE (PB5 is high, WE is OFF)
        sta     PIA_DB          ; Select DDR B
        .IF WE_TYPE==1
        iny                     ; back to FF
        .ELSE
        nop
        .ENDIF
        sty     PIA_PB          ; Set All Port B to Output Ports
        stx     PIA_DB          ; Select PORT B Register
        rts

;*******************************************************************************
;
; JUMP 4                ;**** CHECK DRIVES ATTACHED AND LOADS FAT (4) ****
LE335:
        jsr     LE028           ; Store ZERO PAGE and STACK
        jsr     LE314           ; Sub RESET PIA Ports A & B
        ldx     #$03            ; Check start value
LE347:
        stx     LE018           ; Drive Offset (Index) starts with Drive 3
LE34A:
        jsr     LE1B0           ; Set Drive Port A/B depending on Drive Index
        jsr     LE075           ; Head Load FDC   plus delay
        lda     #6              ; Loops to detect any index activity
LE354:
```

```
        ldy     #$00
LE355:
        ldx     #40
LE357:
        bit     PIA_PA          ; Index is PA7
        bpl     LE365           ; Check for INDEX PULSE becomes 0
        dex
        bne     LE357           ; Inner loop 11 usec * 40 = 440usec
        dey                     ;
        bne     LE355           ; x 256 = inner loop 112ms
        sec
        sbc     #1
        bne     LE354           ; outer loop 6*110 msec (4+ disk rotations)
        beq     LE36E           ; Jump if no index detected
LE365:                          ; Index pulse found !
        jsr     LE1DB           ; Update Drive present & GO to TRK00, return drive in X
        dex
LE369:
                                ; Next Drive number is in (X)
        bpl     LE347           ; Loop back to test next drive
        bmi     LE37B           ; Alway jump to ready and found a drive

LE36E:                          ; No index found at drive (X)
        lda     #$FF            ; STORE "Drive not present" to Drive Index
        jsr     LE1F3           ; Store to Drive Table E014 and update TRK/SEC
        dex
        bpl     LE369           ; Loop back to test next drive
        lda     #$05            ; ERROR 5, no DRIVE found
        bne     LE380X
;
LE37B:                          ; Ready and found at least one drive
        jsr     LE632           ; LOAD FAT

;*******************************************************************************

LE37E:                          ; **** Leave DOS without Error ****
        lda     #$00
        sta     LE027           ; Store ERROR Flag
        lda     LE01A
        ldy     LE01D
        beq     KEEP_ON
TURN_OFF:
        ora     #$C0            ; Turn Motor and HL off
        sta     LE01A           ; Remember last status
KEEP_ON:
        sta     PIA_PB
        jmp     LE044           ; Restore ZERO PAGE and STACK and return

;*******************************************************************************

LE380:                          ; **** LEAVE DOS WITH ERROR NUMBER (A) ****
        pha
        lda     LE01F           ; Check if FAT has been changed ?
        beq     LE380Y          ; Jump, if not changed
        jsr     LE632           ; Added: Re-LOAD FAT (Changes will not be saved)
LE380Y:
        pla
```

```
LE380X:
        sta     LE027           ; Store ERROR Flag
        lda     #$01
        sta     LE01D           ; HL and Motor Off mode
        lda     LE01A
        jmp     TURN_OFF

;
; *** JUMP 3 moved to DOSSUP **
;
; ******************************

LE471:                          ; *** Add sector (free or next) to table
        bit     LE024           ; Search free (FF) or take next (00) sector
        bpl     LE47F           ; jump, if take next of TRK_T and SEC_T

LE476:
        jsr     LE2D7           ; Search for next available sector on disk
                                ; Dsum=sector, Y=track
        lda     DSum            ; Get sector number
        STA     SEC_T
        bpl     LE48E           ; Always jump
LE47F:
        ldy     TRK_T           ; Flag was "next sector"
        inc     SEC_T           ; take next sector
        lda     #$08
        cmp     SEC_T
        bne     LE48E           ; Jump, if sector number is 0...7
        lda     #$00            ; Set sector to 0
        sta     SEC_T
        iny                     ; and increment Track
LE48E:
        sty     TRK_T           ; Store Track number
        jmp     LE49A           ; Jump to Add entry to track/Sector table


;                               ; ********************************************
LE493:                          ; *** Add xx sectors (free or next) to table ***
                                ; Amount xx in LE01C. List finish flag LE025

        ldx     #$00            ; Table index starts at 00
        bit     LE024           ; Check Search free (FF) or take next (00) sector
        bmi     LE476           ; jump if search next free sector
;
LE49A:                          ; Add track/Sector to table
        lda     SEC_T
        sta     DSum
        ldy     TRK_T
        tya
        sta     FreeM,x         ; Track number to table at $EF00..
        lda     DSum
        sta     FreeM+1,x       ; Sector number to table at $EF01..
        jsr     LE2FA           ; GET FAT SECTOR MASK Bit in (A) and Track in (Y)
        ora     FAT_D,y         ; Mask sector as occupied
        sta     FAT_D,y         ; Update FAT Entry
        inx                     ; Increment table index by 2 (X)
        inx
```

```
        bne     LE4BB           ; Less than 128 entries (32k)
        lda     #$06            ; ERROR 6 Data to long to be saved, not enough free space on disk

LE4A8:                          ; ***** Error with re-load FAT *****
        inc     LE01F           ; Force Re-Load FAT
        jmp     LE380
;
LE4BB:
        dec     LE01C           ; Sector Counter -1
        beq     LE4C3           ; Finished with all sectors needed ?
        bne     LE471           ; LOOP back to Add sector (free or next) to table
;
LE4C3:                          ; All Sectors needed are finished
        bit     LE025           ; Check List finish flag
        bpl     LE4D5           ; Jump, if LE025<128 finisch with 00 00
        lda     LE022           ; Store single sector data from E022/23
        sta     FreeM,x         ; to table
        lda     LE023
        jmp     LE45DA          ; finish table with E022/23
;
LE4D5:                          ; Sector table finish with "00 00"
        lda     #$00
        sta     FreeM,x
LE45DA:
        sta     FreeM+1,x
        inc   LE01F             ; Mark FAT change and return
        rts

;*******************************************************************************
;
; JUMP 5                ;******** READ CURENT SELECTED FILE ******
LE4DB:
        jsr     LE028           ; Store ZERO PAGE and STACK
        jsr     LE6A0           ; Get current FAT discriptor plus sector count
        jmp     LE4E7           ; to Read File/Sector

        ; FDC_TS/+1 are FAT start values TRK/SEC
        ; TRK_T and SEC_T are the search targets

; JUMP 1                ;********* READ FILE/SECTOR ***********
LE4E4:
        jsr     LE028           ; Store ZERO PAGE and STACK

LE4E7:
        jsr     LE1ADX          ; SET Ports for selected drive number
        jsr     LE075           ; Clear and Head Load FDC , WE Off

LE4F9:
        ldy     TRK_T           ; **** Loop for next TRK ****
        jsr     LE232           ; Goto Track (Y)
LE4FC:
        jsr     LE684           ; Wait for index pulse and READ first Sync FC and Track ID
                                ; Now, FF FF FC sync follows with Sector GAP
                                ; FDC_TS/+1 is also set
```

```
LE4FF:                          ; **** Loop for next SEC ****
        lda     TRK_T
        cmp     FDC_TS          ; Check Track still the same ?
        beq     LE505
        bne     LE4F9           ; Always Loop for next TRK

LE505:
        lda     SEC_T           ; Check Sector correct ?
        cmp     FDC_TS+1        ; Compare to FAT target sector
        beq     LE516           ; Jump, if next sector is found

        lda     FDC_TS+1        ; Track OK but sector different ....
        cmp     #$08            ; Sector >=8 , target sector must be lower
        bcs     LE4FC           ; Start again at sector 0 with next index pulse

        jsr     LE198           ; Read Dummy Sector data of 256 bytes
                                ; FF FF FC sync follows with Sector GAP
                                ; This will increment FDC_TS+1 (Sector)
        bcs     LE505           ; Always Loop back for next SEC

LE516:                          ; Sector found...
        jsr     LE144_FCX       ; Bridge xx bytes SECTOR START GAP for reading or writing, fast

        jsr     LE144_FEX       ; Delay and bridge xx bytes SECTOR START GAP for reading with reset

        jsr     LE114           ; Clear Checksum
        jsr     LE0EE           ; READ track info and add to Checksum
        cmp     TRK_T           ; Correct track ?
        bne     LE52F           ; Count fail
        jsr     LE0EE           ; READ sector info and add to Checksum
        cmp     SEC_T           ; Correct sector ?
        beq     LE534
LE52F:
        jsr     LE14E           ; INCREMENT Error count (evt leave with ERROR 3
        bcc     LE4F9           ; Go all way back to Search Track again
LE534:
                                ; **** Correct sector found on disk
        bit     LE021           ; Check if delete file = FF (CLEAR FAT)
        bpl     LE54B           ; Jump, if only READ file

        jsr     LE2FA           ; GET FAT SECTOR MASK Bit in (A) and Track in (Y)
        eor     #$FF
        and     FAT_D,y         ; Clear Track/Sector occupied byte
        sta     FAT_D,y
        inc     LE01F           ; Make FAT invalid/changed
LE54B:
        jsr     LE0EE           ; READ next track info and add to Checksum
        sta     TRK_T           ; Store Next Track

        jsr     LE0EE           ; READ next sector info and add to Checksum
        sta     SEC_T           ; Store Next sector
        jsr     LE587           ; Check F7 and Checksum

        jsr     LE119           ; Read single byte
        cmp     #$FB            ; Check for FB Start of Data block ?
        beq     LE55F           ; Jump , if no Error
LE55A:
```

```
        lda     #$01            ; ERROR 1 Sync not found
        jmp     LE380
LE55F:
        jsr     LE114           ; Clear Checksum
        ldy     #$00
LE564:
        jsr     LE0EE           ; READ byte and add to Checksum

LE7C5:                          ; **** Read or Verify (E026 flag) to Data Pointer F0/F1 ***
        bit     LE021           ; Check Delete Flag (FF)
        bmi     LE7D5           ; On Delete jump an do nothing
        bit     LE026           ; Check Verify or Read Data(FF)
        bmi     LE7D3           ; Jump, if read (FF) data from disk
        cmp     (DATA_S),y      ; Compare data to memory
        beq     LE7D5
LE7CE:
        lda     #$0B            ; ERROR 11, Verify failed
        jmp     LE380
;
LE7D3:
        sta     (DATA_S),y      ; Write data to memory
LE7D5:
        iny
        bne     LE564           ; Loop for 256 bytes

        inc     FDC_TS+1        ; Increment Sector. Quick point to next Sector
        inc     DATA_S+1        ; Increment Data Pointer (H)                                    ; This is for
speed up next sector reads

        jsr     LE587           ; Check F7 and Checksum
        lda     TRK_T           ; Check next track is 00 (end of file) ?
        beq     LE57C           ; Jump, if next Track is 00 ?
        dec     LE01C           ; Decrement Length of data file
        beq     LE57C           ; Jump if no more data to read
        jmp     LE4FF           ; Loop back for next SEC

LE57C:
        sta     LE022           ; End of file, E022=0 // Store next Trk/Sec or 00
        lda     SEC_T
        sta     LE023           ; Last sector to E023
        jmp     LE37E           ; End with ERROR 0

;*******************************************************************************************
;
LE587:                          ; ***** Check F7 and Checksum **********

        jsr     LE119           ; READ single FDC byte
        cmp     #$F7
        bne     LE7CE           ; ERROR 11, Verify failed/ ID is missing
        jsr     LE119           ; READ single FDC byte
        cmp     DSum
        bne     LE58E           ; Jump, if checksum value is not correct
        rts

LE58E:
        lda     #$07            ; ERROR 7 , Checksum wrong
        jmp     LE380
```

```
;
;*****************************************************************************************
;
LE594:                                  ;**** Write Space FF FF FC ****
        lda     #$FF
        jsr     LE127                   ; Write Byte to FDC
        jsr     LE127                   ; Write Byte to FDC
        lda     #$FC
        jmp     LE127                   ; Write Byte to FDC and RETURN

;
;*****************************************************************************************
;
; JUMP 2                        ;************ WRITE File **************
LE59B:                                  ; E025 = FF WILL WRITE ONLY SINGLE SECTOR
        jsr     LE028                   ; Store ZERO PAGE and STACK
        jsr     LE1ADX                  ; SET Ports for selected drive number
        jsr     TEST_PROT               ; Test disk protection
        jsr     LE493                   ; (only here) Add xx sectors (free or next) to table
        lda     FreeM                   ; Get first TRK/SEC from table
        sta     TRK_T                   ; to TRK_T and SEC_T
        lda     FreeM+1
        sta     SEC_T
        jsr     LE075                   ; Clear and Head Load FDC, WE OFF
        stx     TS_IDX                  ; TS_IDX = 0 (TRK/SEC table index)

                                        ;***** WRITE LOOP
LE5B9:
        jsr     LE23B                   ; Goto Track (TRK_T)
        jsr     LE5E0                   ; Sub Wait for Index and Write Sectors
        bit     LE025                   ; Check SINGLE SECTOR (FF) flag
        bmi     LE5DD
        lda     TRK_T                   ; Target Track is "00" (finish) ?
        bne     LE5B9                   ; Loop back, if more tracks to write
LE5DD:
        jsr     LE0DC                   ; Head Load ON and WE OFF for next track
        jmp     LE37E                   ; Leave DOS without Error



LE5E0:                                  ;****** Wait for Index and WRITE Sectors *****
        jsr     LE15C                   ; Clear Error and Checksum
        jsr     LE684                   ; Wait for index pulse and READ first Sync FC and Track ID
                                        ; Now, FF FF FC sync follows with Sector GAP
                                        ; FDC_TS/+1 is also set
        lda     SEC_T
        bne     LE5D9                   ; Jump if Sector <> "0", so we need Dummy reads before
LE5D0:
        jmp     LE273                   ; Write Sectors according table at F300 / RTS return point

LE5D9:                                  ;***** Dummy read , because first write Sector is > "0"
        sec
        sbc     FDC_TS+1                ; Subtract FDC Sector form SEC_T
        beq     LE5E0                   ; If already too far, jump back to start of track
```

```
LE5E8:                          ; Skip and READ (A) times Dummy sector
        tax
LE5E9:
        jsr     LE198           ; Read dummy sector sector
LE5EF:                          ; This will increment FDC_TS+1 (Sector)
        dex
        beq     LE5D0           ; Done, Loop back and write sector
        bne     LE5E9           ; Loop (X) times


;*******************************************************************************
;
; JUMP 6                    ; ************** WRITE FAT (6) **************

FAT_DL = FAT_D&255
FAT_DH = FAT_D-5
FAT_END = FAT_D + $03FB

LE5F4:
        jsr     LE028           ; Store ZERO PAGE and STACK

        lda     LE01F           ; Check FAT Changed Flag (>0)
        beq     LE62F           ; Jump, if not changed
        lda     #$00            ; Clear FAT change flag
        sta     LE01F           ; Store to Flag

        jsr     LE1ADX          ; Select Drive and Set Port A/B
        jsr     TEST_PROT       ; Test disk protection
        jsr     FAT_INDEX       ; FAT SEARCH AND CHECK INDEX SYNCRON
        lda     #$02            ; FAT counter to 2
        sta     XTEMP
        jsr     LE097           ; PRESET DISK WRITE MODE
LE60E:
        ldx     #PRE_GAP-1
LE610:
        jsr     LE16D           ; Fast Write GAP "FF" before data block

        lda     #$FE
        jsr     LE108           ; Write FDC "FE" and set Checksum =0

        lda     #FAT_DH>>8      ; Setup Pointer to FAT
        sta     DATA_S+1
        lda     #FAT_DL-5
        sta     DATA_S


        ldy     #$05
LE622:
        jsr     LE0CF           ; WRITE 256-(Y) bytes to FDC + checksum
        cmp     #FAT_END>>8     ; All written
        bne     LE622           ; Loop back

        jsr     LE106           ; WRITE CHECKUM Marker plus Checksum to FDC, Clear Cheksum
        dec     XTEMP
        beq     LE62C           ; Leave after 2nd FAT part
        ldx     #POS_GAP        ; Post GAP for following sectors
```

```
        jsr     LE16D
        jsr     LE594           ; Write Space FF FF FC
        ldx     #PRE_GAP
        bne     LE610           ; Loop back

LE62C:
        jsr     LE0DC           ; Head Load ON and WE OFF
        jsr     LE1DE           ; Goto to Track00 to protect FAT
LE62F:
        jmp     LE37E           ; FAT OK and Leave DOS without Error

;*******************************************************************************
;
FAT_INDEX:                      ; **** FAT SEARCH AND CHECK INDEX SYNCRON ****
        jsr     LE075           ; Clear Checksum, error and Head Load FDC and delay
        jsr     LE1DE           ; FIND TRACK 00
        ldy     #$01
        jsr     LE232           ; Goto to Track (Y)
        jsr     LE684           ; Wait for index pulse and READ first Sector Sync
        jmp     LE144_FC        ; Bridge xx bytes SECTOR START GAP for reading or writing, fast

;*******************************************************************************
;
LE632:                          ; ******** LOAD FAT ******

        lda     #$00
        sta     LE01F           ; Added: Clear FAT change

        jsr     LE1ADX          ; Select Drive and Set Port A/B
        jsr     FAT_INDEX       ; FAT SEARCH AND CHECK INDEX SYNCRON
        jsr     LE144_FEX       ; Delay and bridge xx bytes SECTOR START GAP for reading with reset

        lda     #FAT_DH>>8      ; Setup Pointer to FAT
        sta     FAT_P+1
        lda     #FAT_DL-5
        sta     FAT_P

        ldy     #$05
LE657:
        jsr     LE0EE           ; READ byte and add to Checksum
        sta     (FAT_P),y       ; Save to FAT table $F400...
        iny
        bne     LE657           ; Loop for all $03FB FAT bytes
        inc     FAT_P+1
        lda     FAT_P+1
        cmp     #FAT_END>>8     ; All uploaded ?
        bne     LE657           ; Loop until F7xx range is reached
        jmp     LE587           ; Check F7 and Checksum and RETURN

;*******************************************************************************
;
; JUMP 7                ******* LOAD DISK FAT *******
LE66E:
        jsr     LE028           ; Store ZERO PAGE and STACK
        ldx     LE020           ; Get Selected Drive
        lda     LE014,x         ; Load Drive Status
        bpl     LE67E           ; Jump if exist (value <$80)
```

```
        lda     #$08            ; ERROR 8 DRIVE not valid/existing
        jmp     LE380X
;
LE67E:                          ; READY for READ FAT Sector
        jsr     LE632           ; Load FAT
        jmp     LE62C           ; Leave in FAT Load section

;*******************************************************************************
;
LE684:          ; *** Wait for index pulse and READ first Sync FC and Track ID *


        jsr     LE1FC           ; SET READ MODE Wait for INDEX, FDC_TS is set
        jsr     LE144_FC        ; Bridge xx bytes SECTOR START GAP for reading or writing, fast
        lda     #$00
LE689:
        sta     XTEMP           ; Set Temp Counter variable

LE68B:
        jsr     LE119           ; READ next FDC byte (Track ID 00 ... 08)
        cmp     XTEMP
        beq     LE697           ; Jump if value equals Counter

        jsr     LE14E           ; INCREMENT Error count (max 3) evt. leave with ERROR 3
        bcc     LE684           ; Always loop back and wait for next index

LE697:                          ; Counter found
        inc     XTEMP           ; Increment Counter
        lda     XTEMP
        cmp     #$09            ; Is Counter already 9
        bcc     LE68B           ; Loop, if smaller 9
        rts

;*******************************************************************************
;
F_DISC = FAT_P-13               ; (E8) Adress (File Descriptor-length of)


LE6A0:                          ; **** Copy FAT discriptor calculate sector count ***
        ldy     #$0C
LE6A2:
        lda     (FAT_P),y       ; Move FAT Decriptor data of Temp
        sta     F_DISC,y; File Descriptor Temp target EE..F4
        dey
        cpy     #$05
        bne     LE6A2
                                ; F_DISC:
                                ; FDC_TS = $EE-EF  Start Track, Start Sector
                                ; DATA_S = $F0-F1  Low, High Start address of data
                                ; DATA_E = $F2-F3  Low High End of data
                                ; TYPE   = $F4    File Type and protection status
        lda     FDC_TS
        sta     TRK_T           ; Target
        lda     FDC_TS+1
        sta     SEC_T           ; Target
        sec
        lda     DATA_E+1
```

```
        sbc     DATA_S+1
        sta     LE01C           ; Save Sector count
        lda     DATA_S
        cmp     DATA_E
        bcs     LE6C5
        inc     LE01C           ; Add 1 to Sector count
LE6C5:
        rts

;*******************************************************************************
;
; JUMP 0        ; ***************+ SEARCH FILE ****************

                                ; Search "*", pointer (A2/A3) Length (94)
                                ; if length is zero, free space on disk is calculated in E022 (0..255)
LE6C6:
        jsr     LE028           ; Store ZERO PAGE and STACK
        lda     #FAT_S&255      ; Set Pointer F0/F1 to start of FAT
        sta     DATA_S
        lda     #FAT_D>>8
        sta     DATA_S+1
        lda     File_L          ; File Name has been valid ?
        bne     LE6F9           ; Jump, if valid

                                ; **** On name length=0, free disk space is calculated *****
        sta     LE01D           ; File length = 0 >> Calculate Free Space on disk in E01D & E022
        sta     DATA_S          ; F0/F1 Data (F0=used for Free sectors)
        ldy     #TRK_M          ; Max Tracks of 39 or 79
LE6DC:
        ldx     #$08            ; 8 sectors per track/byte
        lda     FAT_D,y         ; Get FAT content starting at $F44F (Last byte in Sector occumpied table)
LE6E1:
        asl
        bcs     LE6EB           ; Jump if sector used on disk
        inc     DATA_S          ; (F0) increment free sector count
        bne     LE6EB           ;
        inc     LE01D           ; Used space sector high counter
LE6EB:
        dex
        bne     LE6E1
        dey
        bpl     LE6DC           ; FAT table complete ?

        lda     #$F8            ; Value $F8 for high DOS FAT Pointer (out of FAT position)
        bne     LE71C           ; Leave (and pretend nothing found)

LE6F9:
        ldy     #$00            ; **** Continue searching valid file name ****
        ldx     File_L
LE6FD:
        lda     (X00A2),y       ; Search Name Pointer A2/A3
        cmp     #$2A            ; is it a "*" ?
        beq     LE71A           ; Found "*" in name and end search
        cmp     (DATA_S),y
        beq     LE727           ; Char in Filename is equal ?
        lda     #$0D
        clc
```

```
        adc     DATA_S          ; Add 13 to Pointer to next FAT entry
        sta     DATA_S
        bcc     LE712
        inc     DATA_S+1
LE712:
        lda     DATA_S+1
        cmp     #FAT_E          ; All FAT entries (F460...F7FF) checked ?
        bne     LE6F9           ; Loop search


LE71A:
        lda     DATA_S+1        ; Found or "*" or end of FAT
LE71C:
        sta     LE023           ; High DOS FAT Pointer
        lda     DATA_S          ; F0  Data Pointer/Counter Free sector
        sta     LE022           ; Low DOS FAT Pointer
        jmp     LE044           ; Restore ZERO PAGE and STACK and return
;
LE727:
        inx
        iny
        cpy     File_L          ; Length of name reached ?
        bcs     LE71A           ; jump if Y >= name lentgh
        cpy     #$06            ; Max name length of 6 reached ?
        bcc     LE6FD           ; continue compare name
        bcs     LE71A           ; Jump to Name found

;*******************************************************************************
;

DOS_CFG0:
        .db   LE6C6&255, LE6C6>>8       ; JUMP SEARCH FILE (0)

LE74B:                          ; NEW DOS COLD START ROUTINE

        lda     DOS_CFG0        ; Restore DOS JUMP (0) vector
        sta     LE000
        lda     DOS_CFG0+1
        sta     LE000+1
        jsr     LE227           ; Reset ACIA to 8N1
        jsr     LE335           ; CHECK DRIVES ATTACHED AND LOADS FAT (4)
        jsr     LOADDOS                 ; Load DOS
        jsr     LE795           ; Framed Text Output
LE768:
        jsr     ROMINP_V
        cmp     #'C'            ; C) BASIC COLD START
        bne     LE776
LE768X:
        lda     LE027           ; Check for ERRORs
        bne     JUSTCOLD
        lda     $0218
        sta     XTEMP
        lda     $0219
        sta     YTEMP
        lda     #COLD&255
        sta     $0218
        lda     #COLD>>8
```

```
        sta     $0219
JUSTCOLD:
        jmp     BASIC_COLD      ; Here we will reduce memory size


;
LE776:
        cmp     #'W'
        bne     LE784
        lda     $00
        cmp     #$4C            ; Check if Basic was already initialized
        bne     LE768X
        jmp     BASIC_WARM      ; b) EXTENDED PROGRAM WARM START

LE784:
        cmp     #'E'
        bne     LE768
        lda     LE027           ; Check for ERRORs
        bne     LE768
        jmp     ASS_C           ; c) Extended MONITOR
;


COLD:
        pla
        pla
        pla
        pla
        pla
        pla
        jsr     BASIC_CRLF
        lda     XTEMP           ; Restore ROM input vector
        sta     $0218
        lda     YTEMP
        sta     $0219

        lda     #$4C            ; Support Routine
        sta     BASIC_LPT       ; Copy EX_BASIC vector to BASIC_LPT $BC
        lda     #EX_BASIC & 255
        sta     BASIC_LPT+1
        lda     #EX_BASIC >> 8
        sta     BASIC_LPT+2

        lda     #ORG_POS&255    ; Set End of Memory
        ldy     #ORG_POS>>8
        jmp     $BDBA           ; continue Cold start

;*******************************************************************************
;                               ; **** Framed Text Output *****
LE795:

        ldx     #$00            ; Print Coded ENTRY DOS Screen
LE79A:
        sta     XTEMP
        lda     LE800,x
        beq     LE7AB
        cmp     #$E0
```

```
        bcs     LE7AC           ; PRINT (A) times value of $94 and return
        jsr     ROMOUT_V
LE7A8:
        inx
        bne     LE79A           ; Loop back
LE7AB:
        rts

LE7AC:                          ; PRINT (A) times value of $94
        pha
        lda     XTEMP
        jsr     ROMOUT_V
        pla
        clc
        adc     #$01
        bne     LE7AC           ; Loop back
        beq     LE7A8           ; Always jump back




DOSNAME:
        .DB     "DOSSUP", $00           ; Filename max 6 Char


; ********************************* LOAD DOSSUP *********************************

LOADDOS:
        lda     #$06
        sta     File_L
        lda     #DOSNAME&255
        sta     X00A2
        lda     #DOSNAME>>8
        sta     X00A2+1
        jsr     LE6C6                   ; Call DOS SEARCH FILE (0) OK
        lda     LE022                   ; Load DOS Vector for FILE POINTER
        sta     $F5
        lda     LE023
        sta     $F6

        cmp     #FAT_E                  ; Pointing outside FAT (Means Name not found)
        bne     DFOUND
        lda     #$09                    ; Load ERROR 9 (File not found)
        sta     LE027
LOADEND:
        rts
;
DFOUND:                                 ; File Name found
        jsr     LE4DB                   ; Call DOS READ DELETE (5) (Not working after warm start)
        lda     LE027
        bne     LOADEND                         ; Error loading file
        ldy     #$0C
        lda     ($F5),y                 ; Get File descriptor +12
        tax                             ; X= File Type
FDLP1:
```

```
        dey
        lda     ($F5),y                 ; Copy File disctriptor to $F0...$F3
        sta     $E8,y                   ; $F2..F3= Length, $F0..F1= Start ADR
        cpy     #$08
        bne     FDLP1                   ; Loop

        cpx     #$10
        bcc     LOADEND                         ; Type equals <16 (BASIC File)
        cpx     #$20
        bcs     LOADEND

        txa                             ; ***MCODE***
        lsr                             ; Check for Bit 0 (autorun bit =1)
        bcc     LOADEND                         ; Return if bit 0 equals zero

        jmp     ($F0)                   ; Autorun indirect to Start ADR (EXE FILE)
                                        ; Later, return will jump back to BASIC or Caller !


; ********************************** BOOT SCREEN DATA **********************************
;


LE800:
        .db     $0D, $CC, $83                           ; Startup String Data
    .db $EC, $CD, $0D, $0A, $8C, $20

    .db $EC, $8B, $0D, $0A, $8C, $20, $FF
    .db "<C> COLD START ", $FC
    .db $8B, $0A, $0D, $8C, $20, $FF
    .db "<W> WARM START ", $FC
    .db $8B, $0A, $0D, $8C, $20, $FF
    .db "<E> EXT MON ", $F9
    .db $8B, $0D, $0A, $8C, $20, $EC

    .db $8B, $0D, $0A, $CB, $84, $F7
        .db     "OSI DOS 84"
        .db     $84, $CE, $0D, $0A, $FC, $0D, $00


; ****************** BOOT SECTOR END
*****************************************************************************************************************
*******************************

BOOT_E: ; UNUSED AREA AND BOOT SECTOR END

; ****************** DOSSUP
*****************************************************************************************************************
***************************************

HERE_POS        .SET *
                .ORG ORG_SUP-$0043
DELTA           .SET HERE_POS - *
                .IF DELTA > 0
                .ERROR "*** A D D R E S S Conflict !! ***"
                .ENDIF
```

```
        .org    ORG_SUP

        jmp     L037D                   ; just RTS, nothing to do

        .DB DOS_C&255, DOS_C>>8         ; Call vectors for DOS and EXMON
        .DB ASS_C&255, ASS_C>>8

VER:    .DB "YE-OSI DOS 3."
        .DB 48+D_VERS/10
        .DB 48+D_VERS%10

FINAME: .DB $00, $00

STAR:   .DB "*", $00

CODETBL:
    .DB $42, $41, $53, $00  ;"BAS"
    .DB $43, $4F, $4D, $00  ;"COM"
    .DB $53, $45, $51, $00  ;"SEQ"
    .DB $56, $41, $52, $00  ;"VAR"
PROTTBL:
    .DB $52, $57, $6E, $00  ;"RWn"
    .DB $52, $57, $61, $00  ;"RWa"
    .DB $52, $20, $6E, $00  ;"R n"
    .DB $52, $20, $61, $00 ;"R a"


GDPIX:  .db $01, $02, $04, $08, $10, $20, $40, $80      ; 8 Bit 4x2 1st is left

TSPACE:
    .db "  ", $00                   ; SPACES
TDEV:  .db $0D, $0A, "DEVICE", $00   ; DEVICE
TSEC:  .db "SECTORS FREE", $00               ; SECTORS FREE
TNAME: .db $0D, $0A, $0A, "NAME   LENGTH  TYPE", $0D, $0A, $00          ; NAME  LENGTH  TYPE



DOSSUP:
LFA1D:

EX_BASIC:       ;********** DOSSUP BASIC ENTRY POINT *******
        pla
LFB01:
        cmp     #$30            ; Check if Basic calls from Token analysis
        pha
        bne     LFB13
        stx     $BF             ; Basic Extension Linkage
        ldy     #$01
        lda     ($C3),y
        jsr     BASIC_ALPHA
        bcs     LFB1C
LFB11:
        ldx     $BF             ; continue Basic Interpreter
LFB13:
        inc     $C3
        bne     LFB19
        inc     $C4
```

```
LFB19:
        jmp     BAS_GET_CR      ;Return to Adress $00C2
;
LFB1C:
        lda     $C3
        bne     LFB22
        dec     $C4
LFB22:
        dec     $C3
        ldy     #$FF
        ldx     #$00

L0311:
        iny
        dex
L0313:
        lda     TOKEN-256,x     ;BASIC TOKEN CHECK
        beq     L033F
        sec
        sbc     ($C3),y
        beq     L0311
        cmp     #$80
        beq     L032D
        ldy     #$00
L0323:
        dex
        lda     TOKEN-255,x
        bpl     L0323
        dex
        dex
        bne     L0313
L032D:
        jsr     $A70F           ;BASIC:
        pla
        pla
        pla
        pla
        lda     TOKEN-257,x     ;Vector to DOSSUP Basic token code
        pha
        lda     TOKEN-258,x
        pha

L0336:
        inc     $C3             ;Moved from ROM to here
        bne     L033C
        inc     $C4
L033C:
        jmp     $00C2           ;Return to Adress $00C2

;
L033F:
        ldy     #$01
        jsr     $A70F           ;BASIC:

        ldx     $BF             ;continue Basic Interpreter
        jmp     L0336
```

```
;                                       ; ************ PRINT 3x SPACES TO BASIC **********
L0347:
        lda     #TSPACE&255
L0349:                                  ; *** PRINT TEXT in same segment (A) ***
        ldy     #TSPACE>>8
        JMP     $A8C3                   ; BASIC PRINT 3x Space TEXT


;
L034E:                                  ; ************** SUB EVALUATE Byte/Word Parameters *****
        jsr     $AC01                   ; BASIC check Next parameter

L0351:                                  ; **** Evaluate single Integer parameter ***
        sty     $C0                     ; Pointer Counter to Parameter storage
        cpy     #$03                    ; Already 3 Paramenetrs
        BCS     L036D                   ; Jump, if 3 or more
                                        ; Evaluate Integers Parameters 1,2,3
        JSR     $B3AE                   ; BASIC ROM: EVALUATE 8BIT EXPRESSION and convert to byte
                                        ; Integer value in (X)
        ldy     $C0
L035C:
        txa
        sta     PAR_STOR,y              ; Store Integer Parameter
        iny
        jsr     $00C2                   ; Next BASIC value
        cmp     #$2C                    ; More Parameters ?
        bne     L037D                   ; Test for "," (jump and RETURN, if no more parameters)
        jsr     $00BC                   ; BASIC: Advance to next parameter
        bne     L0351                   ; Loop back for next parameter
L036D:
        jsr     $AAAD                   ; ROM BASIC: EVALUATE 16BIT EXPRESSION, MAKE SURE IT IS NUMERIC
        jsr     $B408                   ; CONVERT TO A 16-BIT VALUE
                                        ; Result in (A) and (Y)
        tax
        tya
        ldy     $C0
        sta     PAR_STOR,y              ; Store Lower Byte of 16Bit
        iny
        bne     L035C                   ; Always jump to store Higher Byte in (X)
L037D:
        rts
;
L037E:                                  ; ***** Get String parameters to Stack ******

        bne     L0383                   ; Command without parameters ?
L0380:

        jmp     $AE88                   ; STOP WITH ILLEGAL QUANTITY ERROR
;
L0383:
        pla                             ; Get caller Return address to $9C/9D
        sta     $9C
        pla
        sta     $9D                     ; Remember Return address

        jsr     $AAC1                   ; ROM BASIC, Evaluate expression
        bit     $5F                     ; Check for String
```

```
        bpl     L0380           ; ERROR of not a Sting

        jsr     $B2B3           ; ROM BASIC, Release string
                                ; Pointer in 71/72, length in A
                                ; Points to String in BASIC code or memory
L0398:
        sta     $BF
        pha                     ; Pushing A(lenth) must be <>0
        tya
        pha                     ; Pushing Y(High vector)
        txa
        pha                     ; Pushing X(Low vector)
        lda     $9D
        pha                     ; Restore Return Adr
        lda     $9C
        pha                     ; Restore Return Adr
        lda     $BF
        rts
;
L03A4:                          ; ***** SUB Process Parameter: Drive *****
        lda     #$00            ; Analyse input parameter for Drive number
        sta     LE027           ; Set ERROR to "0"
        lda     PAR_STOR        ; First Paramneter is DRIVE number
        cmp     LE020           ; Compare with actual Drive number
        beq     L03C3           ; Jump, if the same
        cmp     #$04            ; Check for range 0...3
        bcc     L03B8           ; Jump, if <4
        jmp     $AE88           ; FC-Error
;
L03B8:                          ; New valid Drive number selected
        pha
        jsr     L03CD           ; DOS_WRITE_FAT(6), if FAT needs update
        pla
        sta     LE020           ; Store new drive number
        jsr     L03C7           ; DOS_READ_FAT(7), from new drive
L03C3:
        lda     LE027           ; Hold Error value in (A)
        rts


; ************************** STANDARD BOOT UP DOS Parameter Set for Format ************************
;

DOS_CFG1:
        bne     DOS_JMP                         ; 40 Bytes Parameter Standard
    ; LATER JUMP SEARCH FILE (0)

        .db   LE4E4&255, LE4E4>>8       ; JUMP READ FILE OR DELETE (1)
    .db   LE59B&255, LE59B>>8       ; JUMP WRITE FILE (2)
    .db   LE389&255, LE389>>8       ; JUMP FORMAT OR WRITE BOOT SECTOR (3)
    .db   LE335&255, LE335>>8       ; JUMP CHECK DRIVES ATTACHED AND LOADS FAT (4)
    .db   LE4DB&255, LE4DB>>8       ; JUMP READ SELECTED FILE (5)
    .db   LE5F4&255, LE5F4>>8       ; JUMP WRITE DISK FAT (6)
    .db   LE66E&255, LE66E>>8       ; JUMP LOAD DISK FAT (7)

DOS_CFG2:
        .db   $01, $03
```

```
        .db     $01, $03   ; COPY OF START/END ADRESS OF BASIC

        .db   $00, $FF     ; DRIVE FLAGS
                           ; FF= Drive not available
                           ; 00= Drive OK
                           ; 32= Normal
    .db   $FF, $FF
        .db   $00          ; Last Drive Index ($03)
        .db     STEP_D     ; Step delay in ms
        .IF WE_TYPE==1
        .db     $FE        ; PIA B SEL, SIDE, MOTOR & HEAD LOAD default
        .db   $FE          ; PIA B WE(PB0), DIR(PB2),STEP(PB3) temp
            .ELSE
        .db     $FF        ; PIA B SEL, SIDE, MOTOR & HEAD LOAD default
        .db   $FF          ; PIA B WE(PB0), DIR(PB2),STEP(PB3) temp
            .ENDIF
        .db     $00        ; ACTUAL TRACK ON READING / SECTOR COUNTER FOR WRITING
        .db     $01        ; HIGH/ MotorOn/Headload flag (00)=dont reset
SSDD:   .db     $00        ; Double sided  ($00(single) or $FF(double))
        .db   $00          ; FAT Changes if >00

DOS_CFG3:
        .db   $00          ; Selected Drive (0=A side 0, 2=B side 0)
        .db   $00          ; Read or Delete flag (00 = READ)
        .db   $00          ; FAT Vector File Entry Pointer ($00, $00)
        .db   $00
        .db   $FF          ; USER DEF:Search free (FF default) or take next (00) sector
        .db     $00        ; USER DEF:FAT Single Sector flag LE025, 00(default)  or single with zero or FF
with E022/32
        .db     $FF        ; READ ($FF) Bit or VERIFY/ FULL FORMAT ($00)
        .db   $00          ; Error Code ($00)
DOS_JMP:


;*******************************************************************************
;
; JUMP 3                   ; ****** FORMAT OR WRITE BOOT SECTOR ******
;                          ; LE026: Flag for Format all tracks

PAR_T = BOOT_S-ORG_POS     ; Length of DOS parameter table
B_OFF = $0800+ORG_POS-BOOT_E
Y_OFF = B_OFF+PAR_T        ; BOOT Correction position
BC_ST = BOOT_S-Y_OFF       ; Pre calculate pointer

                           ; ************ FORMAT TRACKS **************
LE389:
        jsr     LE028      ; Store ZERO PAGE and STACK
        jsr     TEST_PROT  ; Test disk protection
        lda     LE020      ; Selected drive number
        bit     LE01E      ; Check single or double
        bpl     LE389X     ; Jump on single sided
        and     #$02       ; only allow Drive 0 or 2 on double sided disk
LE389X:
        sta     LE018      ; Store Drive number index
        lda     LE01E      ; Get Double sided (FF) / Single sided (00) flag
        sta     SSDD       ; prepare STD DOS PARAETER BLOCK
```

```
LE38F:                          ; (also entry for second side of disk)
        sta     Side_T          ; Store to Side Temp Double/Single
        jsr     LE1B0           ; Changed: SET Ports for Last Drive number
        jsr     LE1DE           ; FIND TRACK 00
        jsr     LE097           ; SET DISK WRITE MODE (Head load plus Mask B prepared)
        lda     #$03            ; Reset ACIA to 8E1 for TRK00
        sta     ACIA_C
        lda     #$58
        sta     ACIA_C
        jsr     LE207           ; Wait for INDEX plus 10ms delay

        lda     #ORG_POS>>8
        jsr     LE127           ; WRITE byte to FDC (Checksum is not relevant for TRK 00)
        lda     #ORG_POS&255
        jsr     LE127           ; WRITE byte to FDC
        lda     #BC_ST>>8       ; WRITE BOOT CODE Start to pointer
        sta     DATA_S+1
        lda     #BC_ST&255      ;
        sta     DATA_S
        lda     #$08            ; only 2k on 8E1
        jsr     LE127           ; WRITE Length byte to FDC

        ldy     #B_OFF          ; WRITE STD DOS PARAETER BLOCK TO DISK
LE39x:
        lda     DOS_CFG1-B_OFF,y ; Compensate for shorter BOOT SECTOR
        jsr     LE127           ; WRITE byte to FDC
        iny
        cpy     #Y_OFF          ; Parameter Block length (40 bytes)
        bne     LE39x
LE3C4:
        jsr     LE0CF           ; WRITE 256-(Y) bytes to FDC + checksum
        cmp     #BOOT_E>>8
        bne     LE3C4           ; WILL NOT WRITE INTO INDEX AREA

        jsr     LE0DC           ; Head Load ON and WE OFF
        jsr     LE227           ; Track 01... to 8N1
        ldy     #$01            ; Format rest from TRK 1....39/79
        bit     LE026           ; Flag for Format all tracks
        bpl     LE3DB           ; Jump, if Flag =00 (FULL FORMAT), FF (BOOT SEC FORMAT)
        jmp     LE460           ; Leave to Double sided disk check
;

LE3DB:                          ; *** FULL FORMAT section starting at TRK Y ***
        jsr     LE232           ; Goto to Track (Y)
        jsr     LE15C           ; Clear Error and Checksum
        jsr     LE097           ; PRESET DISK WRITE MODE (Head load plus Mask B prepared)
        jsr     LE207           ; Wait for INDEX, 10ms delay

        ldx     #TRK_GAP
        jsr     LE16D           ; FastWrite "FF"
        stx     SEC_T           ; Preset Sector Temp=0
        jsr     LE594           ; Write Sync FF FF FC
        ldy     #$02            ; 2 FAT Blocks
LE3DB0:
        txa
        jsr     LE127           ; WRITE byte to FDC 00...08
        inx
```

```
        cpx     #$09
        bcc     LE3DB0          ; Loop Sector 00 01 02 .... header
        ldx     #ID_GAP         ; ID Bytes Gap for first sector
        bne     LE3DB11

LE3DB1:

        ldx     #POS_GAP        ; Post GAP Bytes for following sectors
LE3DB11:
        jsr     LE16D           ; Fast Write "FF" (Write Run-Out gap after Sec 1...6)

LE3DB2:
        jsr     LE594           ; Write Sync FF FF FC

        ldx     #PRE_GAP        ; Pre GAP bytes before data block
        jsr     LE16D           ; Fast Write "FF" (Write Start GAP)

        lda     TRK_T           ; Check if Track is 1 (FAT)
        cmp     #$01
        bne     LE428           ; Jump and go on with Track 2....

                                ; **** TRACK 1 FORMAT (FAT) ****
        lda     #$FE
        jsr     LE108           ; Write FDC "FE" and set Checksum =0
        ldx     #$02            ; 2x "FF"  (FAT TRK0/1 used)
        jsr     LE16AX          ; Write X times FF + checksum
        ldx     #$F9            ; F9x "00"
        jsr     LE16A           ; Write X times 00 + checksum
        jsr     LE16A           ; Write X times 00 + checksum
        jsr     LE16A           ; Write X times 00 + checksum
        jsr     LE16A           ; Write X times 00 + checksum
        jsr     LE106           ; WRITE CHECKUM Marker plus Checksum to FDC Clear Checksum
        dey                     ; FAT Blocks counter -1
        bne     LE3DB1          ; Second FAT part
        jsr     LE0DC           ; Head Load ON and WE OFF
        ldy     #$02            ; Go on with Track 2
LE3DBB:
        bne     LE3DB           ; Always loop back

LE428:                          ; **** TRACK 2+ FORMAT ****
        ldx     SEC_T           ; Start Sector
        ldy     TRK_T           ; Actual Track

        lda     #$FE            ; Write empty Sector header (X,Y (track)
        jsr     LE108           ; WRITE byte to FDC and clear checksum
        tya                     ; Write TRK
        jsr     LE124           ; WRITE byte to FDC + checksum
        txa                     ; Write SEC
        jsr     LE124           ; WRITE byte to FDC + checksum
        inx
        cpx     #$08
        bne     LASTSEC
        ldx     #$00
        ldy     #$00
LASTSEC:
        tya
        jsr     LE124           ; WRITE byte to FDC + checksum
```

```
        txa
        jsr     LE124           ; WRITE byte to FDC + checksum
        jsr     LE106           ; WRITE CHECKUM Marker plus Checksum and RETURN

        lda     #$FB
        jsr     LE108           ; Write FDC "FB" and set Checksum =0
        ldx     #$00
        lda     #$F6            ; DISK DATA FILLER
        jsr     LE16C           ; Write 256 times F6+ checksum
        jsr     LE106           ; WRITE CHECKUM Marker plus Checksum

        inc     SEC_T
        ldx     SEC_T
        cpx     #$08
        bcc     LE3DB1          ; Loop back Sectors

        jsr     LE0DC           ; Head Load ON and WE OFF
        ldy     TRK_T
        iny
        cpy     #TRK_M+1
        bcc     LE3DBB          ; Loop back Tracks

        jsr     LE1DB           ; Update Drive present & GO to TRK00
LE460:
        bit     Side_T          ; ****** Check for Double Sided disk ******
        bpl     LE46C           ; Jump and leave on single sided

        inc     LE018           ; Last Drive Index +1 (0>1, 2>3)
        lda     #$00
        jmp     LE38F           ; Do second side of disk with Side_T=0 (single sided but other side)
LE46C:
        jmp     LE37E           ; On Single sided, leave DOS without Error



;                               ****************** DIR ******************
DIC:    bne     DIC1            ; Jump on search name
        lda     #$01
        pha                     ; Pushing A(lenth)
        lda     #STAR>>8
        pha                     ; Pushing Y(High vector)
        lda     #STAR&255
        pha                     ; Pushing X(Low vector)
        jmp     DIC2
DIC1:
        jsr     L037E           ; Get String parameters to Stack
DIC2:
        lda     #TDEV&255       ; Print DEVICE
        jsr     L0349           ; PRINT TEXT in same segment (A)
        ldx     LE020
        lda     #$00
        jsr     $B95E           ; BASIC Print value in X,A (device no)
        jsr     $A86C           ; BASIC: Do PRINT CR,LF ?

        lda     #TSEC&255       ; PRINT SECTOR
        jsr     L0349           ; PRINT TEXT in same segment (A)
```

```
        lda     #$00            ; SET lenth to 0 (FREE SPACE function)
        sta     $94
        jsr     L03E0           ; ROM: DOS SEARCH FILE returns disk free space
        ldx     LE022           ; Free Space into X,A
        lda     LE01D
        jsr     $B95E           ; BASIC Print value in X,A (sectors free no)

;       jsr     L0347           ; Print 3x SPACES

        lda     #TNAME&255      ; PRINT LF NAME
        jsr     L0349           ; PRINT TEXT in same segment (A)

        inc     LE01D           ; Correct Preset for HL and Motor off

        lda     #FAT_S&255      ; Start of FAT ($F460)
        sta     $97
        lda     #FAT_S>>8
        sta     $98             ; to pointer FAT address pointer $97/98
        lda     #$08
        sta     $C0
        pla                     ; ex: $FD / $FF
        sta     $9E
        pla                     ; ex: $7F / $7F / $77
        sta     $9F             ; Pointer to DIR string to $9E/9F
        pla                     ; String length >0 ?
        sta     $BF             ; Remember length of DIR string
        bne     L041B
L041A:
        rts                     ; BASIC: String "", just return
;
L041B:
        ldy     #$00
L041D:
        lda     ($9E),y
        cmp     #$2A            ; is "*"
        beq     L043F           ; jump to matching name in FAT
        cmp     ($97),y         ; Compare first char of FAT name
        bne     L0427           ; jump to not matching name
        iny
        cpy     $BF
        bcc     L041D
        bcs     L043F           ; jump to matching name in FAT
L0427:
        lda     #$0D            ; Next FAT entry + 13
        clc
        adc     $97             ; Add 13 to FAT address pointer
        sta     $97
        bcc     L0432
        inc     $98
L0432:
        cmp     #$FB            ; END OF FAT REACHED ?
        bne     L041B           ; Loop back search name
L0436:
        ldx     $88             ; Get ??
        inx
        bne     L041A           ; Jump, if was <> FF ,return
```

```
        jmp     $A86C           ; BASIC: Do PRINT CR,LF ? and return
;
L043F:
        ldy     #$00            ; Matching first Character in FAT
L0441:
        lda     ($97),y         ; Load last name char from FAT
        beq     L0427           ; Empty FAT entry found with "*", loop back

        jsr     OUTVEC          ; ROM Output
        iny
        cpy     #$06
        bcc     L0441
        jsr     L0347           ; Print 3x spaces
        ldy     #$0B
        lda     ($97),y         ; Calculate file length
        ldy     #$09
        sec
        sbc     ($97),y
        TAX
        INX
        pha                     ; Sector value to stack
        LDA     #$00
        JSR     $B95E           ; PRINT Length of (A) and (X)
        PLA
        CMP     #$09            ; Value >=10
        BCS     L043FX
        lda     #$20            ; PRINT Single Space
        jsr     BASIC_OUT       ; PRINT TEXT in same segment (A)

L043FX:
        jsr     L0347           ; 3x SPACE
        ldy     #$0C
        lda     ($97),y         ; Get File Type
        pha
        lsr
        lsr
        clc
        adc     #CODETBL&255
        jsr     L0349           ; PRINT TEXT in same segment (A)
        jsr     $A8E0           ; PRINT LENGTH
        pla
        and     #$03
        asl
        asl
        clc
        adc     #PROTTBL&255
        jsr     L0349           ; PRINT TEXT in same segment (A)
        jsr     $A86C           ; BASIC Some kind of Print Return
        dec     $C0
        bpl     L0488           ; END of FAT ?
        jsr     INVEC           ; ROM Get Key every 8 lines
        cmp     #$0D
        bne     L0436
        sta     $C0
L0488:
        jmp     L0427
;
```

```
L048B:                          ; ********* Copy FILE NAME to FAT ******
        inc     FATCHANGE       ; Mark Change drive
        lda     PAR_STOR+1      ; Get TYPE
        pha
        asl
        asl
        asl
        asl
        ora     PAR_STOR+2      ; OR with PROTECTION
        ldy     #$0C
        sta     ($F5),y         ; STORE TO FAT
        ldy     #$05
L049F:
        cpy     $94             ; Compare to length
        bcc     L04A6           ; Jump if smaller
        lda     #$20            ; Fill Name wit SPACE
        bne     L04A8
L04A6:
        lda     ($A2),y         ; If not, copy string name to FAT
L04A8:
        sta     ($F5),y
        dey
        bpl     L049F           ; Loop for 6 Parameters
        pla                     ; Get back File Type
        rts


;                               ; ************ DSAV STORE **************
STR:    jsr     L037E           ; Get String parameters to Stack
        jsr     L034E           ; EVALUATE Byte/Word Parameters
        jsr     L03A4           ; Process Parameter: Drive
        beq     L04D5           ; Jump, if NO ERROR
L04D1:
        pla                     ; Clear Stack form String Discriptor
        pla
        pla
        rts                     ; and return
;
L04D5:
        lda     $C0             ; Number of parameters found of min 3
        cmp     #$02            ; String plus 2 parameter minimum.
        bcs     L04DE           ; jump if >=3
L04DB:
        jmp     $AE88           ; F-Error
;
L04DE:
        ldx     PAR_STOR+1      ; 2nd parameter (File Type)
        BEQ     L04E9           ; Jump, if Command type is BASIC (0)?

        cmp     #$03            ; Check number of parameters found for Type 1,2,3,..
        BCC     L04DB           ; Jump if number of parameters found <4 to Error
                                ; String plus 3 parameter minimum (drive,type,prot)
L04E9:
        pla                     ; Continue with 4 or more parameters or BASIC
        sta     $9E             ; String name adress to $9E/9F
        tay
        pla
```

```
        sta     $9F
        sta     $A3             ; Store High address to A3
        pla
        sta     $C0             ; String length to $C0
        jsr     L07FA           ; FAT name Search (1 sector)
        beq     L050C           ; Check if found, jump if name exist in FAT

        lda     #FINAME>>8      ; Empty file name vector
        sta     $A3             ; *** SEARCH FOR "00" NAME (EMPTY)? ***
        ldy     #FINAME&255
        lda     #$01            ; Single byte is enough to find empty entry
        jsr     L07FA           ; FAT name Search (1 sector)
        beq     L0519           ; Check if found, jump if name exist in FAT
L0504:
        sta     LE027           ; Remember ERROR 9
        rts                     ; Return if not found (no Free entry ?)
;
L0508:                          ; Leave, if READ ONLY FILE FOUND
        lda     #$0F
        bne     L0504           ; ERROR 15 - FILE is WRITE PROTECTED
;
L050C:                          ; *** FILE NAME ALREADY EXIST IN FAT ***
        ldy     #$0C
        lda     ($F5),y         ; Get File Type
        and     #$03
        cmp     #$02            ; Check for READ ONLY
        bcs     L0508           ; Jump if >=2 (means READ ONLY)

        lda     #$00
        sta     LE01D           ; HL and Motor keep on
        jsr     L05D2           ; DELETE CURRENT SELECTED FILE
                                ; Continue and overwrite same name.
L0519:
        lda     #$00
        sta     LE01D           ; HL and Motor keep on

        jsr     L05C2           ; COPY NAME and TYPE to FAT
        bne     L056F           ; Check if Type is "BASIC (0)"
        ldx     $7A             ; Type is BASIC
        ldy     $79             ; Get Basic Start -1
        bne     L0525
        dex
L0525:
        dey                     ; Basic -1 is in (X),(Y)
        tya
        ldy     #$08
        sta     ($F5),y         ; Writes to FAT Directory Adress parameter
        txa
        iny
        sta     ($F5),y         ; Start / End Adress
        lda     $7B
        iny
        sta     ($F5),y         ;
        lda     $7C
        iny
        STA     ($F5),y         ;
```

```
L0539:                          ; *** and write file to disk
        ldy     #$0C            ; Pointer to Type
L053B:
        lda     ($F5),y         ; Copy all to File discriptor block EE-F4
        sta     $E8,y
        dey
        cpy     #$07
        bne     L053B
        sec
        lda     $F3
        sbc     $F1
        sta     LE01C           ; Length Calc
        lda     $F0
        cmp     $F2
        bcs     L0556
        inc     LE01C           ; +1
L0556:
        lda     #$FF
        sta     LE024           ; Search free (FF default) for storage
        jsr     L03CA           ; DOS Write File
        ldy     #$06
        sty     LE024           ; Search free back to normal

        jsr     L0581           ; Check for ERROR, Read FAT back

        lda     FreeM           ; Get fist Value form Table
        sta     ($F5),y         ; Store Start Track to FAT
        lda     FreeM+1
        iny
        sta     ($F5),y         ; Store Start sector to FAT

        inc     LE01D           ; and back to HL and motor off

        jmp     L03CD           ; Write FAT (6) after file save

;
L056F:
        cmp     #$04            ; Type is "Others" 1,2,3..
        bcs     L0581           ; Return if Type >=4
        ldy     #$08
L0575:
        lda     PAR_STOR-5,y    ; Get Start Address and End Address
        sta     ($F5),y
        iny
        cpy     #$0C
        bne     L0575
        beq     L0539           ; Always jump to



L0581:                          ; *** Check for ERROR, Read FAT back ***
        lda     LE027           ; Check  Error number
        beq     L058E           ; Return, if no ERROR
        pha
        jsr     L03C7           ; On Error READ FAT back (7)
        pla
        sta     LE027           ; Recover Write Error
```

```
        pla
        pla                     ; Remove caller address
L058E:
        rts


;                               ; ********** DREN Rename **************
CHANGE: jsr     L037E           ; Get 1st String parameters to Stack
        jsr     $B117           ; Keep string in memory because of 2nd string
        jsr     $AC01           ; ROM BASIC, Evaluate String
        JSR     L037E           ; Get 2nd String parameters to Stack
        jsr     L034E           ; EVALUATE Byte/Word Parameters
        jsr     L03A4           ; Process Parameter: Drive
        beq     L0599           ; Jump, if NO ERROR

        pla                     ; Clear 2nd String discriptor Stack
        pla
        pla
        jmp     L04D1           ; Goto clear 1st String discriptor Stack and return
;
L0599:                          ; Continue on NO ERROR
        pla
        sta     $9E             ; Get 2nd String
        pla
        sta     $9F
        pla
        sta     $BF             ; Lenghth to $BF
        bne     L05AE

                                ; 2nd string was empty , delete file now
        lda     #FINAME>>8      ; Empty file name vector
        sta     $9F
        lda     #FINAME&255     ; *** POINTER TO "00" NAME (EMPTY)? ***
        sta     $9E
        lda     #$01            ; Lenth of 1 is enough
L05AE:
        sta     $C0
        pla                     ; Get 1st String
        tay
        pla
        sta     $A3
        pla
        jsr     L07FA           ; FAT name Search (1 sector)
        beq     L05BE
        jmp     L0504           ; Return wit ERROR 9 File not found
L05BE:
        lda     $BF             ; if BF length = 0, delete file
        bne     L05C2X          ; Jump, if not delete

        jsr     L05D2           ; DELETE CURRENT SELECTED FILE
L05C2X:
        jsr     L05C2           ; COPY NAME and TYPE to FAT
        jmp     L03CD           ; Write FAT (6) after file save and RETURN


L05C2:                          ; **** COPY NAME and TYPE to FAT *******
        lda     $C0             ; Transfer String vector to A2/A2 and 9F
```

```
        sta     $94
        lda     $9E
        sta     $A2
        lda     $9F
        sta     $A3
        jmp     L048B           ; Copy FILE NAME to FAT and RETURN

;
L05D2:                          ; ******** DELETE CURRENT SELECTED FILE ****
        lda     #$FF
        sta     LE021           ; Read or Delete flag set to DELETE (FF)
        jsr     L0618           ; ROM: DOS READ OR DELETE (5)
        ldy     #$00
        sty     LE021           ; Read or Delete flag set to READ (00)
        jmp     L0581           ; Check for ERROR, Read FAT back

;                               ; ***************** PTR ********************

DEZ:    jsr     $ABFE           ; BASIC: Check for "("
        beq     L0615           ; FC ERROR

        jsr     $AAC1           ; BASIC: Evaluate
        bit     $5F
        bpl     DEZ_1           ; Jump if Number
        jsr     $B2B3           ; ROM BASIC, Release string
        beq     L0615           ; FC ERROR, String is empty

DEZ_1:
        jsr     $ABFB           ; BASIC: Check for ")"

        ldy     $71
        lda     $72
        jmp     BASIC_16_FLOAT  ; Convert Fixed Point to Floating Point
;
L0615:
        JMP     $AE88           ; BASIC: FC ERROR
;
SEL:    beq     L0615           ; ***************** SEL ***************
        ldy     #$00            ; SEL will always reload FAT (diskchange)
        jsr     L0351           ; Evaluate single Integer parameter
        lda     LE020
        cmp     PAR_STOR        ; First Paramneter is DRIVE number the same ?
        beq     L03C7           ; Reload in case of diskchanges
        jsr     L03A4           ; Process Parameter: Drive
        bne     L03C7           ; on ERROR Second try to reload FAT and exit
        rts


L03E0:
        jmp     (LE000)         ; DOS: SEARCH FILE(0)
L03CA:
        jmp     (LE004)         ; DOS: WRITE FILE(2)
FMT0:
        jmp     (LE006)         ; DOS: FORMAT DISK(6)
L0618:
        jmp     (LE00A)         ; DOS: WRITE OR DELETE FILE (5)
L03CD:
```

```
        jmp     (LE00C)         ; DOS: WRITE FAT(6)
L03C7:
        jmp     (LE00E)         ; DOS: LOAD DISK FAT (7)

INVEC:
        jmp     ($0218)         ; INFO: indirect jump INVEC
OUTVEC:
        jmp     ($021A)         ; INFO: indirect jump OUTVEC



;
SCR:    beq     L0659           ; ********** SCR for 32x32 and 64x15/31 **********
        jsr     $B3AE
        txa
        pha
        jsr     $B3AB
        lda     HORZ_SIZE       ; Create flag for 32 or 64 in $BF
        cmp     #33
        ror     $BF             ; Bit 7 is set for 64 mode
        pla
        bit     $BF             ; Check horizontal chars
        bmi     SCR64
        and     #$1F            ; mask off all above 32
        bpl     SCR32           ; Branch always
SCR64:
        and     #$3F            ; mask off all above 64
SCR32:
        sta     $11
        txa                     ; Y-value
        ldx     #$00
        clc
        eor     #$FF
        bit     $BF
        bmi     SCR642
        ror
SCR642:
        ror
        ror
        ror
        tay
        rol
        cpx     VERT_SIZE       ; Check for 64x32
        bne     SCR32x32        ; jump on 2k video RAM
        and     #$03
SCR32x32:
        and     #$07
        ora     #$D0
        sta     $12
        tya
        and     #$E0
SCR643:
        ora     $11
        sta     $11
        jsr     $AC01
        bne     L0665
L0659:
```

```
        rts
;
L065A:
        jsr     L0675
        jsr     $00C2
        beq     L0659
        jsr     $AC01
L0665:
        jsr     $AAC1
        bit     $5F
        bmi     L065A
        jsr     $B96E
        jsr     $B0AE
        clc
        BCC     L065A

L0675:                          ;***** Sub Get Parameter
        jsr     $B2B6           ; as chars to ($11)
        ldy     #$00
        tax
        bne     L0682
        inx
        lda     #$20
        bne     L0688
L0682:
        inx
L0683:
        dex
        beq     L068D
        lda     ($71),y
L0688:
        sta     ($11),y
        iny
        bne     L0683
L068D:
        tya
        clc
        adc     $11
        sta     $11
        bcc     L0697
        inc     $12
L0697:
        rts

CLG:    beq     L06A1           ;************* CLG **************
                                ; CLG : same as CLG 0
                                ; CLG 3: Clear TOP with "00"
                                ; CLG 2: Clear BOT with "20"
                                ; CLG 1: ENABLE LOW RES MODE
                                ; CLG 0: DISABLE LOW RES MODE
L069D:
        cmp     #$30
        bne     CLG0
L06A1:
        lda     #$11
CLGX:
        sta     ACIA_S          ; Turn off RTS at standard Baudrate /16,8N2
```

```
        bne     CLGRET                  ; and return
CLG0:
        cmp     #$31
        bne     CLG1
        lda     #$51                    ; Turn on RTS at standard Baudrate /16,8N2
        bne     CLGX


CLG1:
        ldy     #$00
        ldx     #$00
        sty     $9E
CLGC2:
        cmp     #$32
        bne     CLGC3                   ; Jump on CLG 3 , clear with 00 starting at D0
        ldx     #$20                    ; clear with 20
        lda     #$D2                    ; clear from D2
        cpy     VERT_SIZE               ; 0=1kB >0=2kB
        beq     CLGC4
        lda     #$D4                    ; clear from D4
        bne     CLGC4
CLGC3:
        lda     #$D0
CLGC4:
        sta     $9F
        txa
        ldx     #$02                    ; 2 blocks
        cpy     VERT_SIZE               ; 0=1kB >0=2kB
        beq     CLGCL
        ldx     #$04                    ; 4 blocks
CLGCL:                                  ; Clear screen section with A, D0 or D2/D4
        sta     ($9E),y
        iny
        bne     CLGCL
        inc     $9F
        dex
        bne     CLGCL
CLGRET:
        jmp     $00BC                   ; Return to BASIC



GDIS:   jsr     $B3AE                   ; ********** GDIS x 0..127 , y 0...31/63 **************
        ldy     #$FF
L06BB:
        iny
        sty     $C0
        beq     L06C3
        jsr     $B3AB                   ; Arg from Basic line
L06C3:
        ldy     $C0
        stx     $F0,y                   ; F0=X F1=Y F2=DOT(1) or LINE(2)
        cmp     #$2C
        BEQ     L06BB                   ; Store parameters in F0....

        ldy     $F1                     ; Y
```

```
        lda     $F2                     ; Type
        and     #$03                    ; Mask Type    0  1  2  3

        cmp     #$02
        bcc     L06_P
        eor     #$83                    ; 00 01 81 80 (clear dot, dot, line, clear line)
L06_P:
        ldx     $F0
        stx     $30
        sty     $31
        sta     $34

        lda     HORZ_SIZE               ; Create flag for 32 or 64 in $BF
        cmp     #33
        ror     $BF                     ; Bit 7 is set for 64 mode
        lda     $34
        bpl     L06DC                   ; Bit 8 of Mask set? (Line Mode)
        jmp     L072E
L06DC:

        lda     $31
        sta     $37
        lda     $30
        sta     $36

;*******************************************************************************
;
L06E9:                                  ; *** Plot pixel (30,31) x,y preserverd
        stx     $3E
        lda     $30
        cmp     #$80
        bcs     L072B                   ; Jump and quit if X >= 128
        sta     $32

        bit     $BF
        bmi     PP64                    ; jump to 64 cgar section
        asl     $32                     ; Only for 32 char mode
        and     #$03
        tax

        lda     $31
        clc
        adc     #$C0
        bcs     L072B                   ; Jump and quit if Y >= 32 or Y>=64
        eor     #$FF

        lsr                             ; 32 char section
        php
        lsr
        ror     $32
        lsr
        ror     $32
        lsr
        ror     $32
        ora     #$D0
        sta     $33
        txa
```

```
        plp
        bcc     L0716
        adc     #$03
L0716:
        tax

PPexit:

        lda     $34                     ; Check delete or write
        lsr
        lda     GDPIX,x
        ldx     #$00
        bcc     L0725
        ora     ($32,x)
        bne     L0729
L0725:
        eor     #$FF
        and     ($32,x)
L0729:
        sta     ($32,x)
L072B:
        ldx     $3E
        rts

PP64:                                   ; ********** 64 char section
        lda     $31
        clc
        adc     #$C0
        bcs     L072B                   ; Jump and quit if Y >= 64
        eor     #$FF
        sta     $33                     ; Save converted Y
        and     #$03
        asl                             ; Y0,1 for mask offset
        tax
        lda     $33
        asl     $32
        lsr
        lsr                             ; move Y2 down
        lsr
        ror     $32
        lsr
        ror     $32                     ; now x0 is in carry , 32 ready

        bcc     L06E9x
        inx                             ; Mask X0 set, X ready
L06E9x:
        lda     $33                     ; Correction for double size display
        lsr
        lsr
        lsr
        lsr
        clc
        adc     #$D0


L06E9z:
        sta     $33
```

```
        jmp     PPexit

;*******************************************************************************
;
L072E:                          ;*** LINE MODE
        ldx     #$02
L0730:
        lda     $2F,x
        sec
        sbc     $35,x
        bcs     L073D
        eor     #$FF
        adc     #$01
        clc
L073D:
        sta     $37,x
        ror     $35
        dex
        bne     L0730
        lda     $38
        bne     L074C
        cmp     $39
        beq     L0757
L074C:
        asl
        bcs     L0756
        asl     $39
        bcc     L074C
        ror     $39
        clc
L0756:
        ror
L0757:
        sta     $38
        ldy     #$00
        ldx     #$00
L075D:
        txa
        clc
        adc     $38
        tax
        bcc     L076E
        bit     $35             ;Check Bit 7
        bmi     L076C
        inc     $30             ;increment x coordinate
        bcs     L076E
L076C:
        dec     $30             ;decrement x coordinate
L076E:
        tya                     ;Delta Y in Y
        clc
        adc     $39
        tay
        bcc     L077F
        bit     $35             ;Check Bit 6
        bvs     L077D
        inc     $31             ;increment y coordinate
```

```
        bne     L077F
L077D:
        dec     $31             ; decrement y coordinate
L077F:
        JSR     L06E9           ; Plot pixel (30,31) x,y preserverd

        lda     $30             ; compare that end point is reached
        cmp     $36
        bne     L075D
        lda     $31
        cmp     $37
        bne     L075D
        rts
;
SAFI:   jsr     $AAAD           ; ********** SEQW **************
        jsr     $B408           ; BASIC GET 16BIT ARG FROM BASIC LINE
        jsr     $00C2           ; BASIC GET CURREMT CHAR FROM BASIC LINE
        beq     L07A2
        jsr     $AC01           ; BASIC CHECK SYBBOLS IN BASIC CODE
        bne     L07C8
L079F:
        jsr     L0675           ; Parameter Loop
L07A2:
        lda     #$2C
        ldy     #$00
        sta     ($11),y
        inc     $11
        bne     L07AE
        inc     $12
L07AE:
        jsr     $00C2           ; BASIC GET CURREMT CHAR FROM BASIC LINE
        bne     L07C5
        ldy     $11
        lda     $12
        ldx     #$00
        stx     $5F
        ldx     #$90
        sta     $AD
        sty     $AE
        sec
        jmp     $B7E8           ; BASIC Return new 16bit address
;
L07C5:
        jsr     $AC01           ; BAIC CHECK SYBBOLS IN BASIC CODE
L07C8:
        jsr     $AAC1           ; BASIC EVALUATE EXPRESSION
        bit     $5F
        bmi     L079F           ; Loop next parameter
        jsr     $B96E           ; BASIC Build ASCII number in 100 form AC-AF
        jsr     $B0AE           ; BASIC PRINT MESSAGE
        jmp     L079F           ; Loop next parameter

;       ********************************************************************
;                               **** SEQS Set Read poiter to memory ******

SEFI:   jsr     $AAAD           ; BASIC GET 16BIT ARG FROM BASIC LINE
        jsr     $B408           ; BASIC Convert FLOAT to INT, Result in 11-12
```

```
        lda     $11
        ldy     $12
        jmp     $A624           ; within BASIC FINALIZE RESTORE

;       ********************************************************************
;
EOF:    ldy     #$01            ; ************** EOF ***********
        lda     ($8F),y
        cmp     #$2C
        bne     L07EE           ; EOF = 1 (more data)
        dey                     ; EOF = 0 (end of file)
L07EE:
        jmp     $AFD0           ; BASIC Returns value in Y ??


;       ********************************************************************

;                               ; ********** DCHK Check/Verify **************

CHECK:  JSR     L037E           ; Get String parameters to Stack
        pla                     ;
        tay                     ; Low address
        pla                     ;
        sta     $A3             ; High address
        pla                     ; String length
        jsr     L07FA           ; FAT name Search
        beq     CHEC1           ; Check if found, jump if name exist in FAT
        sta     LE027           ; ERROR 9 File not found returned
        rts

CHEC1:
        sta     LE026
        jsr     L0618           ; DOS READ/VERIFY FILE(5)
        lda     #$FF
        sta     LE026           ; Verify bit set to 1
        rts
;


;                               ; ************** FAT name Search ********
                                ; (A) length, (Y) low
L07FA:
        bne     L07FF           ; String lenght >0
        jmp     $B268           ; BASIC: Sting length ERROR

L07FF:
        tax

L0800:                          ; MOVED FROM ROM TO DOSSUP
        sty     $A2             ; Store DOS FILNAME VECTOR ADDRESS
        stx     $94             ; String length in X and $94
        jsr     L03E0           ; Call DOS SEARCH FILE (0)
        lda     LE022           ; Load DOS Vector for FILE POINTER
        sta     $F5
        lda     LE023
        sta     $F6
```

```
        cmp     #FAT_E          ; Pointing outside FAT (Means Name not found)
        bne     L0822
        lda     #$09            ; Load ERR=9 (File not found)
        rts
L0822:
        lda     #$00            ; File found
L0824:
        rts
;
;*********************************************************************

                                ; **** VERSION ****
SVER:
        lda     #FATVER&255
        ldy     #FATVER>>8
        jmp     $A8C3           ; BASIC PRNT string

;
;*********************************************************************


FORMAT:                         ; **** FORMAT ****
        ldy     #$00
        sty     PAR_STOR+1      ; Second Paramneter is Full(0) or BootSec(>0) only
        sty     PAR_STOR+1      ; Third Parameter is Single (0) or Double(>0) sided
        JSR     L0351           ; Evaluare integer parameters
        lda     PAR_STOR        ; First Paramneter is DRIVE number
        cmp     #$04            ; Check for range 0...3
        bcc     FMT1            ; Jump, if <4
        jmp     $AE88           ; FC-Error
FMT1:
        sta     LE020           ; Store target drive number
        lda     PAR_STOR+1
        beq     FMT2
        lda     #$FF
FMT2:
        sta     LE026           ; Store 00/FF for format option
        lda     PAR_STOR+2
        beq     FMT3
        lda     #$FF
FMT3:
        sta     LE01E           ; Store 00/FF for single/double sided

        jsr     FMT0            ; DOS: FORMAT DISK (6)
        ldy     #$00
        sty     LE01E           ; Default Single (0) sided
        dey
        sty     LE026           ; Back to read mode

        lda     LE027           ; Check Error
        bne     L0824           ; Jump on Error
        jsr     L03C7           ; DOS_READ_FAT(7), from new drive
VERSION:                ; Updates VERSION Info
        ldy     #$0F
VERLP:
        lda     VER,y
        sta     FATVER,y
```

```
dey
bpl     VERLP
inc     FATCHANGE       ; Mark changed FAT
jmp     L03CD           ; DOS: WRITE DISK FAT (7)


.db $00                 ; End of Table

.dw ERROR-1
.db 'R'+$80                                     ; ERR COMNAND
.db 'R'
.db 'E'

.dw DISK -1
.db 'K'+$80                                     ; DISK COMNAND
.db 'S'
.db 'I'
.db 'D'

.dw PAGE-1
.db 'E'+$80                                     ; PAGE COMNAND
.db 'G'
.db 'A'
.db 'P'

.dw DLOAD-1
.db 'D'+$80                                     ; DLOD COMNAND
.db 'O'
.db 'L'
.db 'D'

.dw DOS_C-1
.db 'S'+$80                                     ; DOS COMNAND
.db 'O'
.db 'D'

.dw ASS_C-1
.db 'S'+$80                                     ; ASS COMNAND
.db 'S'
.db 'A'

.dw B_SET-1
.db 'T'+$80                                     ; SET COMNAND
.db 'E'
.db 'S'

.dw B_CALL-1
.db 'L'+$80         ;                           ; CALL COMNAND
.db 'L'
.db 'A'
.db 'C'

.DW SVER-1
.db $D2, $45, $56                       ; VER Commad

.DW FORMAT-1
.db $D4, $4D, $46, $44                          ; DFMT Command
```

```
    .DW DIC-1
    .DB $D2, $49, $44                       ; DIR Command

    .DW CLG-1
    .DB $C7, $4C, $43                       ; CLG Command

    .dw SEL-1
    .DB $CC, $45, $53                       ; SEL Command

        .DW CHECK-1
    .DB $CB, $48, $43, $44                  ; DCHK Verify Command

    .DW CHANGE-1
    .DB $CE, $45, $52, $44                  ; DREN Command

    .dw STR-1
    .DB $D6, $41, $53, $44                  ; DSAV Command

    .dw EOF-1
    .DB $C6, $4F, $45                       ; EOF Command

    .DW DEZ-1
    .DB $D2, $54, $50                       ; PTR Command

    .DW SAFI-1
    .DB $D7, $51, $45, $53                  ; SEQW Command

    .dw SEFI-1
    .DB $D3, $51, $45, $53                  ; SEQS Command

    .DW SCR-1
    .DB $D2, $43, $53                       ; SCR Command

    .dw GDIS-1
    .DB $D3, $49, $44, $47                  ; GDIS Command



TOKEN:

;*****************************************************************************
;       NEW BASIC "PAGE" COMMAND
;*****************************************************************************
PAGE:
        ldy     #$00
        lda     VERT_SIZE
        bne     CLS4K
        lda     #$20
CLS_LOOP:
        sta     VIDEO_RAM,y
        sta     VIDEO_RAM+$100,y
        sta     VIDEO_RAM+$200,y
        sta     VIDEO_RAM+$300,y
        iny
        bne     CLS_LOOP
        rts
```

```
CLS4K:
        lda     #$20
CLS_LOOP4K:
        sta     VIDEO_RAM,y
        sta     VIDEO_RAM+$100,y
        sta     VIDEO_RAM+$200,y
        sta     VIDEO_RAM+$300,y
        sta     VIDEO_RAM+$400,y
        sta     VIDEO_RAM+$500,y
        sta     VIDEO_RAM+$600,y
        sta     VIDEO_RAM+$700,y
        iny
        bne     CLS_LOOP4K
        rts

;*******************************************************************************
;       NEW BASIC "DISK" COMNAND
;                       Syntax: Disk 1...7
;*******************************************************************************

DISK:
        bne     LF9CB
        jmp     DISK_BOOT       ; Changed: boot DOS extension

LF9CB:
        and     #$07
        asl
        tax
        lda     DOS_POS,x
        sta     $EE
        lda     DOS_POS+1,x
        sta     $EF
        jsr     BASIC_LPT       ; Process Basic Line
        JMP     ($EE)           ; INFO: indirect jump

;*******************************************************************************
;       NEW BASIC "ERR" COMNAND
;                       Syntax:
;*******************************************************************************

ERROR:
        ldy     LE027           ; 16 Bit value in A/Y
        lda     #$00
        jmp     BASIC_16_FLOAT  ; Indirect jump to BASIC routine


;*******************************************************************************
;       NEW BASIC "DLOAD" COMNAND
;                       Syntax:
;*******************************************************************************


DLOAD:

        beq     LFBA6
        jsr     BASIC_EVAL              ; BASIC $AAC1 Evaluate string or numeric
```

```
                                        ; Direct:String placed at top of memory
                                        ; VEC:69/6A L:68 - VEC:81/82
                                        ; Indirect: Vector in String variable
        ldy     #$02
        lda     ($AE),y                 ; AE contains pointer to parameter for String
        sta     $A3
        dey
        lda     ($AE),y
        sta     $A2
        dey
        lda     ($AE),y
        beq     LFBA6                   ; Check for String length = 0 ?
        cmp     #$06
        bcc     LFB9F                   ; jump, if lentgh of String >6
        lda     #$06                    ; Set to max 6
LFB9F:
        pha
        jsr     BASIC_B2B6              ; Free Temp String
        pla
        tax                             ; Lenght to X
        jmp     LFA82                   ; GOTO ASS Command section to Load file name

LFBA6:
        jmp     BASIC_FCERR

;*********************************************************************************
;       NEW BASIC "DOS" COMNAND
;                       Syntax:
;*********************************************************************************


DOS_C:
        LDA     #LFA6C >> 8             ; DOSSUB COMMAND STRING
        ldy     #LFA6C & 255
        bne     LFA7C                   ; Always Jump

;*********************************************************************************
;       NEW BASIC "ASS" COMNAND
;                       Syntax:
;*********************************************************************************


ASS_C:
        lda     #LFA66 >> 8             ; EDITOR COMMAND STRING
        ldy     #LFA66 & 255
LFA7C:                                  ; Execute DOS Load "Name" routine
        ldx     #$06                    ; Set String length to 6
        sta     $A3
LFA80:                                  ; ENTRY from DOSSUP ?!?!?!
        STY     $A2                     ; Store DOS FILNAME VECTOR ADDRESS

LFA82:                                  ; LOAD File Name by pointer A2/A3
        stx     $94                     ; String length in X and $94

LF990:                                  ; Transfer Data from ($79) to DOS_VECTOR
        ldx     #$03                    ; 4 Bytes to $E010...
LF992:
```

```
        lda     $79,x
        sta     DOS_PARAM,x
        dex
        bpl     LF992

        jsr     L03E0           ; Call DOS SEARCH FILE (0) OK
        lda     LE022           ; Load DOS Vector for FILE POINTER
        sta     $F5
        lda     LE023
        STA     $F6

        cmp     #$F8            ; Pointing outside FAT (Means Name not found)
        bne     LFA9E
        lda     #$09            ; Load ERR=9 (File not found)
LFA9A;                          ;
        sta     LE027
LFA9D:
        rts
;
LFA9E:                          ; File Name found
        jsr     L0618           ; Call DOS READ DELETE (5) (Not working after warm start)
        ldy     #$0C
        lda     ($F5),y         ; Get File descriptor +12
        tax                     ; X= File Type
LFAAB:
        dey
        lda     ($F5),y         ; Copy File disctriptor to $F0...$F3
        sta     $E8,y           ; $F2..F3= Length, $F0..F1= Start ADR
        cpy     #$08
        bne     LFAAB           ; Loop

        cpx     #$20            ; Check File Type ?
        bcc     LFABA           ; Jump if <32 (BAS OR MCODE)
LFAB9:
        rts                     ; Do nothing else on SEQ and VAR type
;
LFABA:                          ; DAT OR EXE FILE FOUND
        cpx     #$10
        bcc     LFAC5           ; Type equals <16 (BASIC File)

        txa                     ; ***MCODE***
        lsr                     ; Check for Bit 0 (autorun bit =1)
        bcc     LFAB9           ; Return if bit 0 equals zero

        jmp     ($F0)           ; Autorun indirect to Start ADR (EXE FILE)
                                ; Later, return will jump back to BASIC or Caller !

;
LFAC5:                          ; ***BASIC FILE***
        lda     $F0             ; Get File Start ADR
        clc
        adc     #$01
        sta     $79
        lda     $F1
        adc     #$00
        sta     $7A             ; Increment Start ADR+1 -> $79-7A
```

```
        lda     $F2                     ; File ending -> $7B..7C
        sta     $7B
        lda     $F3
        sta     $7C
        txa                             ; File type to (A)
        lsr                             ; Check Bit 0 of File Type (autorun)
        bcs     LFAF0                   ; Jump if one (autorun)
        jmp     BASIC_CLEAR             ; Basic now loaded , goto to BASIC ROM
;
FC_RET  =       BASIC_RUN-1

LFAF0:                                  ; Bit 0 of File Type set to autorun
        lda     #FC_RET>>8              ; On Run Fail, return to FC error
        pha
        lda     #FC_RET&255
        pha
        jmp     BASIC_A477              ; Return to Basic and finish with RUN


LFA66:
        .DB     "EXTMON",$00            ; Filename max 6 Char
LFA6C:
        .DB     "DOSSUP", $00           ; Filename max 6 Char



;*******************************************************************************
;       NEW BASIC "SET" COMNAND
;                       Syntax:
;*******************************************************************************


B_SET:
        jsr     BASIC_G16B              ; Get 16 Bit Parameter from BASIC statement
        jsr     BASIC_B408              ; BASIC Convert FLOAT to INT, Result in 11-12
        jsr     BASIC_FINDL             ; Execute from BASIC GOTO section
        sec
        lda     $AA                     ; Put result -1 into A/Y
        sbc     #$01
        ldy     $AB
        jmp     BASIC_RSTOR             ; Execute BASIC Restore

;*******************************************************************************
;       NEW BASIC "CALL" COMNAND
;                       Syntax:
;*******************************************************************************


B_CALL:
        jsr     BASIC_POKE_PARM                 ; Get Call Parameters from "Basic POKE" section
        txa
        jsr     LFB61
        tay
        lda     #$00                    ; 16 Bit value in Y/A
        jmp     BASIC_16_FLOAT          ; Indirect jump to BASIC routine

LFB61:  jmp     ($11)                   ; indirect jump to USER routine address
```

```
;
;**************** END $EFFF ***************************************

HERE_POS        .SET *
                .ORG ORG_POS+$1200
DELTA           .SET HERE_POS - *
                .IF DELTA > 0
                .ERROR "*** A D D R E S S Conflict !! ***"
                .ENDIF
```