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 TO KERNEL EPROM1_V54 in 2024 ***
*******************************************
```

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

- replace the OSI Boot ROM by EPROM1_V54.ROM ($F800..$FFFF)

- add 5,5k RAM memory to

  $E000-EFFF = (4k)

  $F200-F7FF = (1,5k)

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

- main memory requirements are min. 8k up to 40k with Hires Mode

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

- Older YE-OSI DOS 3.54 versions required an inverted Write Enable (WE) to prevent data corruption for drives without Head Load mechanism.

Instead, with version 3.54, just 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! ⚠

```
**********************************************************************
** Minimal Disk Basic extensions (KERNEL EPROM1_V54.ROM YE-OSI DOS) **
**********************************************************************
```

With EPROM1_V54.ROM you will get additional BASIC commands as:

PAGE, SET, CALL, SUB, OUT for general purpose

DLOD, ERR, DISK, DOS, ASS for rudimental/kernel disk management

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

Boot sequence selecting "DOS":

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

From here, you have to select

A) DOS COLD START (will clear all memory)

B) DOS WARM START or C) or D)

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 after DOSSUP boot. Loading DOSSUP during boot is not mandatory.

DOS Memory Map:

	Address	Content
ROM EPROM1_V54.ROM	0xFFFF 0xF800	Modified SYSTEM ROM
RAM for DOS	0xF7FF 0xF200	FAT at 0xF400 DOS Memory
OSI I/O	0xF000	Serial ACIA 0xF000 ACIA 1 Option 0xF400 ACIA 2
RAM for DOS	0xEFFF 0xE000	DOSSUP code area 0xE900 YE-OSI Disk Operating System
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	BASIC Code start at 0x0300 up to 32K(40k) User RAM
STACK ZEROPAGE	0x01FF 0x0000	

********** RAM for DOS Extension MAP (5,5k) **********

OPERATIONG SYSTEM RAM:

$E000-E8FF = 2.25k used by YE-OSI DOS 3.54 (Boot Sector Code)

$E8C0-E8FF = DOS TEMP Memory for STACK and Zero page (2x32 Bytes)

$E900-EFFF = 1.75k 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 RAM REQUIRED:

$F200-F2FF = (256 Bytes Free RAM, (BASIC DISK OR FILE COPY or NMI Routine)

$F300-F3FF = TEMPORARY memory for building Track/Sector list

$F400-F7FF = DOS FAT memory location (RAM)

- $F400 SECTOR USAGE INFORMATION	(Address Vector in E02D)
- $F450 DISK TITLE 16 bytes	(Address Vector in E02B)
- $F560 START OF FILE DIRECTORY	(Address Vector in E02F)

ROM:

$F800-FFFF = EPROM1_V54.ROM (includes simple Disk Basic commands)

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.

ROM (EPROM1 V54.ROM) new BASIC commands summary :

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

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

**************** 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.

**************** [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

**************** COMMAND: SUB**

Sub will jump to the Direct Vector Address $0229 pointing (jump) to $000A

Pointer $000A is per default set to $AE88 in Basic (Function Error)

This basically replaces the X=USR(0) BASIC construct.

**************** COMMAND: OUT [Type, Parameters, …]**

OUT comes in three versions:

OUT 0, Address, Blocks and OUT 1, Data/String,... or only OUT

It provides Interrupt driven Serial output buffer, that frees up the program during execution to wait for sending out the last serial byte to a printer.

Reading is not changed and can be done in parallel to the output operation.

IMPORTANT! Interrupts have to be disabled, before using any DOS command

***************** COMMAND: OUT**

OUT without parameter will initialize the Serial port to 8N2 @ 600 baud.

The Receive interrupt flag of the ACIA is set, the CPU Interrupts will be disabled.

***************** COMMAND: OUT 0, Address, Pages**

This will reserve a serial output buffer stating at "Address". I must be a staring address at the start of a 256 Byte block.

Pages are blocks of 256 bytes to be reserved for the serial buffer.

Memory location $FA indicates the buffer status. 0=empty, >128=busy

The serial buffer is cleared and the CPU interrupt is enabled.

***************** COMMAND: OUT 1, Data, ...**

This command will transfer "Data" like strings or variables to the serial output buffer. If the buffer is full, the command will wait for the next transfer opportunity. Otherwise OUT 1, Data, ... will return and BASIC can continue while the Interrupt driven Serial output is working.

Check Memory location $FA for buffer status.

***************** 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 "EDI will load the file "EDITOR".

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.

************** **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

************** **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.

**************** COMMAND: ASS (identical to DLOD "EDITOR")**

Loads program called "EDITOR" 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.

REMARKS:

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

When using only the minimal Disk Basic extensions (KERNEL EPROM1_V54.ROM)

you have to poke and peek some memory location to get additional functions.

For example:

- Drive selected by $E020 (from 0 to 3)

- Single/Double by $E01E

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

- and so on

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

Will load and run Boot sector of Disk 0 to load YE-OSI DOS routines to $E000

Like OSI Boot ROM, you have to select afterwards

A) DOS COLD START (clears all memory)

B) DOS WARM START or C) or D)

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. Otherwise, you can also enter “SEL {drive number] and DOS will read the disk directory of the chosen drive.

*************** **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:

$E020 Drive to be selected

$E027 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 only supports Disk 0 and Disk 2 (two single sided disks)

******* General:**

Usage of SS or DS 3.5 and 5.25 inch disk drives with 40 or 80 Tracks.

(40 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)

***** 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:**

E000: JUMP SEARCH FILE (0)

E002: JUMP READ FILE OR DELETE (1)

E004: JUMP WRITE FILE (2)

E006: JUMP FORMAT OR WRITE BOOT SECTOR (3)

E008: JUMP CHECK DRIVES ATTACHED AND LOADS FAT (4)

E00A: JUMP READ SELECTED FILE (5)

E00C: JUMP WRITE DISK FAT (6)

E00E: JUMP LOAD DISK FAT (7)

DOS INITIAL DISK PARAMETER TABLE

E010: COPY OF START/END ADRESS OF BASIC 2x2

E014: DRIVE FLAGS 4x

FF= Drive not available

00= Drive OK

E018: Last Drive Index

E019: Step delay in ms (24)

E01A: $C002 PIA Port Mirror (FE)

E01B: PIA PORT B MASK (FE)

E01C: ACTUAL TRACK ON READING / SECTOR COUNTER FOR WRITING

E01D: Used space sector counter High

E01E: Drive Double sided (FF), default single sided (00)

E01F: FAT has changed if >00

E020: Selected Drive (0=A side 0, 2=B side 0)

E021: Read or Delete flag (00 = READ)

E022: Low FAT File Name Pointer / Free sector count LOW

E023: High FAT File Name Pointer

E024: USER Define: Search free (FF) or take next (00) sector

E025: USER defined: FAT Single Sector flag LE025, 00(default) or single with zero or FF with E022/32

E026: READ ($FF) Bit or VERIFY / FULL FORMAT ($00)

E027: Error Code ($00)

E028: DOS BOOT Start entry

E02B: DISK ID Vector Address

E02D: DISK FAT Vector Address

E02F: 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 $E026: Verify (00) or default Read Data (FF)

Val $E01C: 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   : $E022/23 TRK/SEC will show next Sector in chain

******* DISK 2**

WRITE FILE

File length is max. 128 sectors or 32kB

Num $E01C: Number of sectors (1...128)

Num $E020: Selected Drive (0=1 side A, 2=2 side A)

Flag $E024: 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 $E025: FILE FAT LIST default (00) will end with (00 00) or (FF) by $E022/23 TRK/SEC

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

******* DISK 3**

FORMAT OR WRITE BOOT SECTOR

Flag $E026: "00" will clear and format entire disk

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

Flag $E020: Selected Drive (0=1 side A, 2=2 side A)

Flag $E01E: Drive Double sided (FF), default single sided (00)

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

Set $E026=0 , $E020=2 , $E01E=0, "DISK 3" , $E026=255

***** 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 $E026: 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 $F400.. to currently active drive, if FAT data has been changed

$E01F indicates FAT Changes if >00

***** DISK 7

Will load FAT data from currently active drive to memory $F400..

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

SECTOR Table ($50 bytes), Starts at $F400, BIT 0=Sector 0, BIT 1=Sector 1, ...

DOS VERSION INFO ($10 bytes), Starts at $F450. Should end with "00"

MAX 71 File Entries in FAT, Starts at $F460 (.. $F8FB), 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   (E000)

zz   Cluster number of 256 bytes (09)

DOS Start code E000-E8FF (2.25 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 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 – 6..12 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.

PART 1 SYSTEM ROM

Listing

```
;
;                       (c) Copyright TB 2022
;
;           File:                   D_F800_FFFF.bin                 EPROM 54 SYSTEM ROM
;
;           Date:                   Dec 2023
;
;           CPU:                    MOS Technology 6502 (MCS6500 family)
;
;    Version:       DOS ROM called EPROM1_V54
;           Diff:                   Difference to Original OSI ROM (MONDE) starting $FC00
;           NMI & IRQ Routines removed from 01xx Stack area to $0223 and $0226
;           TUNE Value command removed
;           BACKSPACE and Clean Cursor Movement works now
;           DISK only added (will start boot sector)
;           WE active is back to "0".
;           To prevent disk erase during Power cycle, pull up resistors have to be removed from 610 board
;           Display may be configured by SCREEN_TYPE,SCREEN_RES and FULL23X32
;           $FFE1 <32 or >32 will indicate horizontal screen resolution (32 or 64)
;           $FFE2 0 or 1 will indicate Screen RAM size of 2K or 4k


ORG_POS                     = $F800          ; Kernel ROM
VER             = 54                ; ROM Version
SCREEN_TYPE     = 0                 ; Screen Type: 1kB (0) or 2kB (1) video memory (usage up to D3FF or $D7FF)
SCREEN_RES      = 0                 ; 0=32x32 or 1=64x16 and 64x32
FULL32X32       = 0                 ; 1=max out to 32x32 or 64/32 characters
WE_TYPE                     = 0              ; 0=active low or 1=active high

                .IF SCREEN_TYPE==1
D_OFFSET        = $0700
                .ELSE
D_OFFSET        = $0300
                .ENDIF

                .IF SCREEN_RES==1

                .IF FULL32X32==1
C_OFFSET        = $C0               ; Bottom Screen Text Input offset to last Display segment
MON_OFFSET      = $1D0              ; Start of Monitor Adress field
SCREEN_CHARS    = 64                ; Displayed characters per line
LINE_C          = 64                ; Line memory bytes
SCREEN_OFFSET   = 0                 ; Left Screen start of characters (must be uneven)
                .ELSE
C_OFFSET        = $80               ; Bottom Screen Text Input offset to last Display segment
MON_OFFSET      = $1D0              ; Start of Monitor Adress field
SCREEN_CHARS    = 48                ; Displayed characters per line
LINE_C          = 64                ; Line memory bytes
SCREEN_OFFSET   = 13                ; Left Screen start of characters (must be uneven)
                .ENDIF

                .ELSE
                .IF FULL32X32==1
C_OFFSET        = $C0           ; Bottom Screen Text Input offset to last Display segment
MON_OFFSET      = $0C6          ; Start of Monitor Adress field
SCREEN_CHARS    = 32            ; Displayed characters per line
LINE_C          = 32            ; Line memory bytes
SCREEN_OFFSET   = 0             ; Left Screen start of characters (must be uneven)
                .ELSE
C_OFFSET        = $60           ; Bottom Screen Text Input offset to last Display segment
MON_OFFSET      = $0C6          ; Start of Monitor Adress field
SCREEN_CHARS    = 24            ; Displayed characters per line
LINE_C          = 32            ; Line memory bytes
SCREEN_OFFSET   = 5             ; Left Screen start of characters (must be uneven)
                .ENDIF

                .ENDIF




VIDEO_RAM       = $D000
VMON_ADR        = VIDEO_RAM+MON_OFFSET

SCREEN_START    = C_OFFSET+SCREEN_OFFSET
SCREEN_LENGTH   = SCREEN_CHARS-1
LOWER           = VIDEO_RAM+D_OFFSET+SCREEN_START        ; CURSOR BOTTOM last Display Byte
UPPER           = VIDEO_RAM+LINE_C*4+SCREEN_OFFSET       ; CURSOR TOP first Display Byte
NEW_LINE        = LOWER>>8      ; NEW line Screen position for BASIC
SCREEN_EDIT     = D_OFFSET      ; Edit position
SCREEN_PAR      = $FFE0         ; Screen parameter table


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

ACIA_DC         = $C010         ; ACIA DISK Control Port
ACIA_DD         = $C011         ; ACIA DISK Data Port

ACIA_S          = $F000         ; SERIAL ACIA Control Port
ACIA_D          = $F001         ; SERIAL ACIA Data Port

DOS_COLD        = $E000         ; DOS Vector(0) only before first call !
DOS_READ_DEL    = DOS_COLD+$0A  ; DOS Vector(5)
DOS_WRITE_FAT   = DOS_COLD+$0C  ; DOS Vector(6)
DOS_PARAM       = DOS_COLD+$10 ; DOS Parameter table for BASIC Start and End

DOS_E022        = DOS_COLD+$22  ; Low FAT File Name Pointer or Free sector count
DOS_E023        = DOS_COLD+$23  ; High FAT File Name Pointer
DOS_E025        = DOS_COLD+$25  ; USER DEF:FAT Single Sector flag LE025, 00(default)  or single with zero or FF with
E022/32
DOS_E027        = DOS_COLD+$27  ; Error Code ($00)

KBD_PORT        = $DF00

ENTRY_CNT       = $0E    ; Entry key counter (Starts with $00)
```

```
ENTRY_BUF       = $13    ; Entry Buffer for Keybord entries

BASIC_LPT       = $BC    ; Basic Line Processing Vector in Zero-Page
BASIC_OUTVAR    = $97
BASIC_WORKS     = $79
CASS_FLAG       = $FB
String_Var      = $5F
USR_ADR                  = $0B     ; User Jump Adress (.dw)
Cursor          = $E7    ; Cursor temp pointer (.dw)


BASIC_WARM      = $0000  ; BASIC WARMSTART
BASIC_16_FLOAT  = $AFC1  ; Convert Fixed Point to Floating Point
BAS_GET_CR      = $00C2  ; BASIC GET CURREMT CHAR FROM BASIC LINE
BASIC_FINDL     = $A432  ; BASIC SEARCH BASIC LINE NUMBER, ADR in AA-AB
BASIC_A477      = $A477  ; INITIALIZE, KIND OF BASIC CLEAR
BASIC_CLEAR     = $A47A  ; BASIC CLEAR
BASIC_RUN       = $A5FC ; BASIC RUN
BASIC_RSTOR     = $A621  ; BASIC FINALIZE RESTORE
BASIC_CTRLC     = $A636  ; BASIC CONTRL C CHECK
BASIC_DATA      = $A70F  ; BASIC DATA COMMAND
BASIC_EVAL      = $AAC1  ; BASIC EVALUATE EXPRESSION
BASIC_G16B      = $AAAD  ; BASIC GET 16BIT ARG FROM BASIC LINE
BASIC_ALPHA     = $AD81  ; BASIC CHECK CHARACTER FOR ALPHA
BASIC_FCERR     = $AE88  ; BASIC FC ERROR
BASIC_PMSG      = $B0AE  ; BASIC PRINT MESSAGE
BASIC_B2B6      = $B2B6  ; BASIC Free Temp String
BASIC_POKE_PARM = $B3FC            ; BASIC Evaluate like POKE (adress, value)
BASIC_B408      = $B408  ; BASIC Convert FLOAT to INT, Result in 11-12
BASIC_B96E      = $B96E  ; BASIC Build ASCII number in 100 form AC-AF
BASIC_INI       = $BDF6  ; INI BASIC with Start Vector in X,Y
BASIC_COLD      = $BD11  ; BASIC COLD START asdress
BASIC_OUT       = $BF2D  ; BASIC Old OUTPUT Vector by BASIC
BASIC_CHECK     = $AC01  ; BAIC CHECK SYBBOLS IN BASIC CODE

CLS_SCR         = $F9EE  ; !! Fixed Clear Screen
COM_DOS         = $FA72  ; !! Fixed DOS Routine INITIALZE DOS COMMAND EXTENSION
COM_ASS         = $FA78  ; !! Fixed DOS Routine LOAD EXTENDED ASSSEMBLER to $0700
DOS_CLD         = $FAFA  ; !! Fixed DOS COLD START
DOS_R_D                  = $FAFD  ; !! Fixed DOS READ OR DELETE
DOS_WFAT        = $FBA3  ; !! Fixed DOS WRITE FAT

DSK_BOOT        = $FC00  ; !! Fixed entry to disk (not active)
INP_CHAR        = $FD00  ; !! Fixed Input Char from Keyboard result in a and 0213
MON_ENTY        = $FE00  ; !! Fixed Entry to Monitor
MON_HBIN        = $FE93  ; !! Fixed Convert ASCII HEX to BIN
RESET           = $FF00  ; !! Fixed Reset entry point

INI_CASS        = $FCA6  ; !! Fixed Initialise ACIA
OUT_CASS        = $FCB1  ; !! Fixed Output char to cassette

BAS_LF          = $FE70  ; !! Fixed Set load flag
BAS_SF          = $FE7B  ; !! Fixed Set save flag

BAS_OLD         = $FF69  ; !! Fixed old OUT MAIN
BAS_SVEC        = $FF96  ; !! Fixed old Save Vector
BAS_LVEC        = $FF8B  ; !! Fixed old Load Vector
```

```
BAS_CTRC        = $FF9B  ;!! Control C Check
BAS_INP         = $FFBA  ;!! Fixed Basic input routine
ROM_PAGE        = $FFEB  ;!! Fixed Basic ROM vectors
ROM_VECT        = $FFFA  ;!! Fixed CPU Vectors

VERSION         = $FFDF ; Version Info Byte ADR

                         ; Input Rotine Workspace
SCR_BAS_PARAM   = $0200  ; Basic Screen Parameter Table/Vector address
CursorPOS       = $020F  ; Cursor Position (.dw)
BAS_CHAR        = $0213  ; Char Code
REPT_FLAG       = $0214  ; Repeat Flag
ORG_CHAR        = $0215  ; Char Code ignoring Shift, Ctrl
ORG_CHAR2       = $0216  ; Same as ORG_CHAR
CursorChr       = $0222  ; Cursor Character Adress
CursorDEF       = $BB    ; Square Cursor default definition
IRQ_VEC         = $0223
NMI_VEC         = $0226


BASIC_SUB       = $0229  ; Basic SUB Jump Direct Vector
USR_SUB                  = $000A ; BASIC USER() Vector

SCR_LINE_XPOS   = $0200
FAT_D           = $F400          ; FAT Memory area
Unused          = FAT_D-$0200    ; Unused 256 Bytes
NMI_ENTRY       = Unused         ; Temporary NMI Entry point in $0226

        .ORG    ORG_POS

        .DB     COM_DOS&255      ; New Vector to DOS Loader
        .DB     COM_DOS>>8
        .DB     COM_ASS&255      ; New Vector to ASS Loader
        .DB     COM_ASS>>8


;*************************************************************************************
;                               BASIC "OUT" COMMAND
;                               SYNTAX: OUT - OUT0,a,b - OUT1,a,b
;                                       a=start address (only Segments of 256 bytes)
;                                       b=number of 256 byte blocks
;                               OUT0,.. will initialize the serial buffer
;                               OUT1,.. Will place data in output buffer
;*************************************************************************************
LF800:
        beq     LF834
        tax
        jsr     $00BC
        jsr     BASIC_CHECK
        cpx     #$30             ; Parameter 1 = 0
        bne     LF846            ; Jump if <> OUT0 command
        jsr     BASIC_POKE_PARM          ; OUT0 - evaluate word value a and integer b
        lda     $12              ; Resullts a in $11-12, b in register X
        ldy     #$06
LF814:
        sta     $022F,y          ; write same (high) adr value of $12  to $0230...0235
        dey
        bne     LF814
        sty     $0230           ; Y=0  (like base adress in $0230) - 00 (start)
        sty     $0232           ; Y=0  (like base adress in $0232) - 00 (start)
        txa                     ; b to ACCU
        clc
        adc     $12
        sta     $0235           ; Add b to high base adress and save $0235 (end)
                                ; What is with $0234 ???? MISSING
        tya                     ; Y=0
LF828:
        sta     ($11),y         ; Clear 256 Bytes
        iny
        bne     LF828
        inc     $12
        dex                     ; b x 256 byte blocks , Clear Serial Output buffer
        bne     LF828
        sty     $FA             ; $FA Disable IRQ

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

LF834:                          ; OUT without PARAMETER
        lda     #$31            ; 8N2 DIV16, RTS LOW, IRQ Output Enabled
        sta     ACIA_S
        cli                     ; Enable IRQ
        rts
;
LF83B:                          ; Take care for String variable on OUT1,"test.."
        jsr     LF857           ; SUB string variable handling
        jsr     $00C2
        beq     LF856           ; Finish, when no new value found
        jsr     BASIC_CHECK

; *******************************************************************************
LF846:                          ; OUT1,x,y Command evaluation
        jsr     BASIC_EVAL
        bit     String_Var
        bmi     LF83B           ; Jump if String
        jsr     BASIC_B96E
        jsr     BASIC_PMSG
        clc
        bcc     LF83B           ; Go back to parameter out
LF856:
        rts
;
LF857:                          ; ************ SUB STRING Variable handling
        jsr     BASIC_B2B6      ; Free temp Strings
        tax
        ldy     #$00
        inx
LF85E:
        dex
        beq     LF869
        lda     ($71),y
        jsr     LF86A
        iny
        bne     LF85E
LF869:
        rts
;
LF86A:                          ; ************ SUB place (A) into Serial buffer chain
        sta     $BF
        tya
        pha
        lda     $0232
        sta     BASIC_OUTVAR
        lda     $0233
        sta     BASIC_OUTVAR+1
        ldy     #$00
        lda     $BF
        sei                             ; Disable CPU Interrupt for update buffer
        sta     (BASIC_OUTVAR),y
        lda     #$FF                    ; Allow IRQ Output
        sta     $FA
        cli                             ; Enable CPU Interrupt
        inc     BASIC_OUTVAR
        bne     LF88A
        inc     BASIC_OUTVAR+1
LF88A:
        lda     BASIC_OUTVAR+1          ; End of buffer reached ??
        cmp     $0235
        bne     LF894
        lda     $0234                   ; Get back Start segment
LF894:
        sta     $0233
        sta     BASIC_OUTVAR+1
        lda     BASIC_OUTVAR
        sta     $0232
LF89E:
        lda     (BASIC_OUTVAR),y
        bne     LF89E                   ; Loop on output buffer full (nice)
        pla
        tay
        rts
;
IRQ_ENTRY:                      ; ACIA IRQ ENTRY ADDRESS
        bit     ACIA_S
        bmi     LF8AD           ; Jump in ACIA IRQ request
        jmp     ($022E)         ; INFO: indirect jump
;
LF8AD:
        bit     $FA             ; IRQ request found
        bmi     LF8BA           ; Jump if Bit7 of $FA=1 to SUB PROCESS ACIA OUTPUT
        sta     $F9             ; Save (A)
        pla
        ora     #$04            ; Get Status bit from stack and do a SEI (disable IRQ)
        pha                     ; Back on stack
        lda     $F9             ; Reload (A)
        rti                     ; Return from Interrupt
;
LF8BA:
        pha                     ; ************* SUB PROCESS ACIA OUTPUT
        tya
        pha
        ldy     #$00
```

```
          lda       $0230
          sta       $F9
          lda       $0231
          sta       $FA
          lda       ($F9),y
          sta       ACIA_D
          lda       #$00
          sta       ($F9),y
          inc       $F9
          bne       LF8D8
          inc       $FA
LF8D8:
          lda       $FA
          cmp       $0235               ; End segment reached ?
          bne       LF8E2
          lda       $0234               ; Go back to Start Segment
LF8E2:
          sta       $0231
          sta       $FA
          lda       $F9
          sta       $0230
          lda       ($F9),y
          beq       LF8F2               ; Jump and stop IRQ Buffer output
          lda       #$FF                ; Keeps IRQ buffer Out alive
LF8F2:
          sta       $FA
          pla
          tay
          pla
          rti                           ; Return from Interrupt !!
;
;*************************************************************************************
;                                       BASIC LINE INPUT VECTOR
;*************************************************************************************

INP_VECT:                                         ; LINE Input Vector subroutine ($F8F8)
          jsr       INP_KBD
          sta       $0217
          tya
          pha
          lda       CursorPOS           ; $020F
          sta       $E7
          lda       CursorPOS+1
          sta       $E8
          ldy       #$00

          jsr       LFA4E               ; Check for Screen change to come and hide Cursor on CR
          cmp       #$EC                ; Check for Cursor "LEFT"
          bne       LF932
          lda       #-1                 ; Substract 1 from Cursor Position
          bne       LF918
LF916:
          lda       #-LINE_C            ; Subtract LINE_C form Cursor Position
LF918:
          jsr       LFA28               ; Call Cursor Off and Pop up Old Character
          clc
          adc       $E7
```

```
        sta     $E7
        bcs     LF924
        dec     $E8
LF924:
        sec                     ; CHANGED !
        sbc     #UPPER&255      ; Check Lower Limit Upper Display Limit
        lda     $E8
        sbc     #UPPER>>8
        bcc     LF956
LF92F:
        jsr     LFA37           ; Call Cursor Placement and update Cursor POS
LF932:                          ; Compare for Cursor "UP"
        cmp     #$1A
        beq     LF916
        cmp     #$EE            ; Compare for Cursor "RIGHT"
        bne     LF960
LF93A:
        lda     #1              ; Add 1 to Cursor POS
        bne     LF93F
LF93D:                          ; Add LINE_C to Cursor POS
        lda     #LINE_C
LF93F:
        jsr     LFA28           ; Call Cursor Off and Pop up Old Character
        clc
        adc     $E7
        sta     $E7
        bcc     LF94B
        inc     $E8
LF94B:
        sec
        lda     #LOWER&255      ; Check LOWER Display Limit
        sbc     $E7
        lda     #LOWER>>8
        sbc     $E8
        bcs     LF92F
LF956:                          ; Reset Cursor Default location  UPPER
        lda     #UPPER&255
        sta     $E7
        lda     #UPPER>>8
        sta     $E8
        bne     LF92F           ; Go always to Cursor Placement
LF960:
        cmp     #$0A            ; Compare for Cursor "DOWN"
        beq     LF93D
        cmp     #$1B            ; CHANGED ! Compare to "ESC"
        bne     LF96D
        jsr     LF9EE           ; Clear Screen
        sty     ENTRY_CNT       ; UPDATE: Also Clear Input Buffer Pointer
        sty     ENTRY_BUF
        lda     SCREEN_PAR      ; Get Default Entry start position
        sta     SCR_LINE_XPOS
        bne     LF956
LF96D:
        cmp     #$1D            ; Compare to "SHIFT RETURN"
        bne     LF978
        lda     $0211
        pha
        clc
        bcc     LF93A
LF978:
        CMP     #$7F            ; PATCHED to Compare to "BACKSPACE"
        BNE     LF978x
        dec     ENTRY_CNT       ; Correct Entry Key counter
        dec     ENTRY_CNT
        lda     #$5F            ; Feed in Shift "O" Character
        sta     $0217
LF978x:
        cmp     #$00            ; Compare to "CURSOR Updated"
        bne     LF98A
        lda     $0217           ; Load last Keyboard key and continue
        cmp     #$1D
        beq     LF986
        lda     #$00
;
        .db     $24             ; Mask out PLA
LF986:
        pla
        sta     $0217
LF98A:
        pla
        tay
        jsr     LFA1D           ; PATCHED to Modify Extended Vector also at COLD start
        lda     $0217
        rts                     ; Finally Return (Y and ACCU preserved)
;
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
        rts

;

;**************************************************************************************
;                               BASIC OUTPUT VECTOR (NEW)
;**************************************************************************************

OUT_VECT:                               ; Output Vector table subroutine ($F99C)
        cmp     #$5F
        bne     LF9AB                   ; Check for "_" (BACKSPACE)
        sty     $E7
        ldy     SCR_LINE_XPOS
        cpy     #SCREEN_START+1                 ; Begin of Edit Line Text entry !! Changed to SCREEN_START+1
        bcs     LF9AE
        ldy     $E7
LF9AB:
        jmp     LFF69
;
LF9AE:
        dec     SCR_LINE_XPOS
        lda     $0201                   ; Char under old Cursor
        sta     VIDEO_RAM+SCREEN_EDIT,y
        lda     #$20
        sta     $0201
        lda     #$5F
        sta     VIDEO_RAM+SCREEN_EDIT-1,y       ; New Execute of Backspace (Shift O)
        ldy     $E7
        jmp     LFF6C

        nop
        nop
        nop
        nop
        nop
        nop
        nop
        nop
        nop
        nop
        nop

; 11 bytes free

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

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

LF9EE:
        jmp     CLS_SUB

LF9CE:                                  ; ***** Control-C correction
        tya
        pha
        lda     #$BF
        sta     KBD_PORT
        lda     KBD_PORT
        and     #$04
        bne     LF9DF
        jsr     LF9EE
LF9DF:
        pla
        tay
        lda     #$FE
        sta     KBD_PORT
LF9E6:
        bit     KBD_PORT
        bpl     LF9E6
        jmp     LFFA5
```

```
; BASIC "TUNE" COMNAND removed
; ********************************************************
LFA1D:
        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
        rts
; ********************************************************
LFA28:                                  ; Support routine
        pha                             ; Cursor Clear and Pop up Old Character
        lda     $0222
        cmp     ($E7),y                 ; Position equals Cursor Character ?
        bne     LFA35
        lda     $0211                   ; Pop up old Char under Cursor
        sta     ($E7),y
LFA35:
        pla
        rts
; ********************************************************
LFA37:                                  ; Support routine
        lda     ($E7),y                 ; Cursor Placement
        sta     $0211                   ; Store Char at new Cursor position
        lda     $0222
        sta     ($E7),y                 ; Show Cursor at new Position

LFA41:                                  ; Update new CursorPOS
        lda     $E7
        sta     CursorPOS               ; $020F
        lda     $E8
        sta     CursorPOS+1             ; Store temp Cursor Pos to Cursor Position Pointer
;       lda     #$00                    ; New Input Char = $00 to exit higher routine
        tya                             ; Y should be always zero
LFA4D:
        rts
; ********************************************************
LFA4E:                                  ; Support Routine CHANGED !!
        lda     $0217                   ; HIDE Cursor on CR
        cmp     #$0D                    ; CR ?
        bne     LFA60
        jmp     LFA28                   ; Hide Cursor

LFA60:
        pha
        LDA     $0222                   ; Check if Screen moved cursor away (scroll)
        cmp     ($E7),y                 ; Position equals Cursor Character ?
        beq     LFA35                   ; All OK

        lda     #-LINE_C                ; Subtract LINE_C form Cursor Position
        clc
        adc     $E7
        sta     $E7
        bcs     LFA35
        dec     $E8
        bne     LFA35
```

```
LFA66:
          .DB       "EDITOR",$00              ; Filename max 6 Char (MOVED)
LFA6C:
          .DB       "DOSSUP", $00             ; Filename max 6 Char (MOVED)


;
;*************************************************************************************
;         NEW BASIC "DOS" COMNAND
;                         Syntax:
;*************************************************************************************
HERE_POS            .SET *
                    .ORG  COM_DOS
DELTA               .SET HERE_POS - *
                    .IF DELTA > 0
                    .ERROR "*** A D D R E S S Conflict !! ***"
                    .ENDIF

LFA72:
          LDA       #LFA6C >> 8               ; DOSSUB COMMAND STRING
          ldy       #LFA6C & 255
          bne       LFA7C                     ; Always Jump
;
;*************************************************************************************
;         NEW BASIC "ASS" COMNAND
;                         Syntax:
;*************************************************************************************

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

LFA78:
          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

          jsr       LF990                     ; Transfer $79.. to DOS VECTOR $E010..
          jsr       LFAFA                     ; Call DOS SEARCH FILE (0) OK
          lda       DOS_E022                  ; Load DOS Vector for FILE POINTER
          sta       $F5
          lda       DOS_E023
          STA       $F6

          cmp       #$F8                      ; Pointing outside FAT (Means Name not found)
          bne       LFA9E
          lda       #$09                          ; Load ERR=9 (File not found)
LFA9A;                                            ;
          sta       DOS_E027
LFA9D:
          rts
;
LFA9E:                                            ; File Name found
          jsr       LFAFD                         ; 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
;
;HERE_POS       .SET *
;               .ORG DOS_CLD
;DELTA          .SET HERE_POS - *
;               .IF DELTA > 0
;               .ERROR "*** A D D R E S S Conflict !! ***"
;               .ENDIF
LFAFA:
        jmp     (DOS_COLD)              ;INFO: indirect jump ($E000) wich is DOS Search File


;HERE_POS       .SET *
;               .ORG DOS_R_D
;DELTA          .SET HERE_POS - *
;               .IF DELTA > 0
;               .ERROR "*** A D D R E S S Conflict !! ***"
;               .ENDIF
LFAFD:
        jmp     (DOS_READ_DEL)          ;INFO: indirect jump ($E00A)
;
;
; ********************************* Routine Extension of Basic ($FB00)
;                               Basic Extension Main routine
EX_BASIC:
        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
LFB28:
        iny
        dex
```

```
LFB2A:
        lda     BAS_EXT_COM-255,x
        beq     LFB56                   ; End of EXT BASIC token reached
        sec
        sbc     ($C3),y
        beq     LFB28
        cmp     #$80
        beq     LFB44
        ldy     #$00
LFB3A:
        dex
        lda     BAS_EXT_COM-254,x
        bpl     LFB3A
        dex
        dex
        bne     LFB2A
LFB44:
        jsr     BASIC_DATA
        pla
        pla
        pla
        pla
        lda     BAS_EXT_COM-256,x
        pha
        lda     BAS_EXT_COM-257,x
        pha
        jmp     LFB13

LFB56:                          ; Code not found
        jmp     ($022C)         ; indirect jump  (more BASIC extensions !!)
;
CODE_NF:                        ; Jump Vector in $022C set to $FB59
        ldy     #$01            ; Basically Code not found
        jsr     BASIC_DATA
        jmp     LFB11
LFB61:
        JMP     ($11)           ; indirect jump to USER routine address

;
;************************************************************************************
;       NEW BASIC "CALL" COMNAND
;                       Syntax: Var=CALL Adress, Value (Value will be in Accu)
;                               Var is integer return value in Accu (optional)
;************************************************************************************

LFB64:
        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
;
;************************************************************************************
;       NEW BASIC "SET" COMNAND
;                       Syntax: SET LineNumver, Sets DATA Read Pointer to LineNumber
;           SET 0 resets Data pointer
;                                   If line not found, Overflow error is returned
;*********************************************************************************

LFB71:
        jsr     BASIC_G16B              ; Get 16 Bit Parameter from BASIC statement
        jsr     BASIC_B408              ; ??
        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 "OLD" COMNAND
;                       Syntax:
;*********************************************************************************

LFB84:
        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


;!! MOVED TO HERE
;*********************************************************************************
;       NEW BASIC "DISK" COMNAND
;                       Syntax: Disk 1...7
;*********************************************************************************

LF9C6:
        bne     LF9CB
        jmp     DISK_BOOT               ; Changed: boot DOS extension

LF9CB:
        jmp     LFBB1
        nop
        nop

; 2 bytes left
;
; *********************************** Fixed Subroutine in ROM <<<DISK BOOT>>>
;HERE_POS      .SET *
;              .ORG  DOS_WFAT
;DELTA         .SET HERE_POS - *
;              .IF DELTA > 0
;              .ERROR "*** A D D R E S S Conflict !! ***"
;              .ENDIF
LFBA3:
        jmp     (DOS_WRITE_FAT)             ;INFO: indirect jump DOS WRITE FAT Vector(6)

LFBA6:
        jmp     BASIC_FCERR
;
; ********************************************************************************
;       NEW BASIC "ERR" COMNAND
;                       Syntax:
; ********************************************************************************

LFBA9:
        ldy     DOS_E027                    ; 16 Bit value in A/Y
        lda     #$00
        jmp     BASIC_16_FLOAT              ; Indirect jump to BASIC routine
;
;
; ********************************************************************************
;
;              NEW DISK Routines (moved to here)
; ********************************************************************************

LFBB1:
        and     #$07
        asl
        tax
        lda     DOS_COLD,x
        sta     $EE
        lda     DOS_COLD+1,x
        sta     $EF
        jsr     BASIC_LPT       ; Process Basic Line
        JMP     ($EE)           ; INFO: indirect jump

;(we need a $00 at the end !! wich is given by JUMP address, 1byte saved.. yeeea!)

        .dw     LF9C6 -1
        .db     'K'+$80   ; BASIC DISK COMNAND
        .db     'S'
        .db     'I'
        .db     'D'

        .dw     LFBA9-1
        .db     'R'+$80   ; BASIC ERR COMNAND
        .db     'R'
        .db     'E'
        .dw     LFB84-1
        .db     'D'+$80   ; BASIC DLOD COMNAND
        .db     'O'
        .db     'L'
        .db     'D'

        .dw     LFA72-1
        .db     'S'+$80   ; BASIC DOS COMNAND
        .db     'O'
        .db     'D'

        .dw     LFA78-1
        .db     'S'+$80   ; BASIC ASS COMNAND
        .db     'S'
        .db     'A'

        .dw     LF800-1
        .db     'T'+$80   ; BASIC OUT COMNAND
        .db     'U'
        .db     'O'

        .dw     LF9EE-1
        .db     'E'+$80   ; BASIC PAGE COMNAND
        .db     'G'
        .db     'A'
        .db     'P'

        .dw     LFB71-1
        .db     'T'+$80   ; BASIC SET COMNAND
        .db     'E'
        .db     'S'

        .dw     LFB64-1
        .db     'L'+$80   ; BASIC CALL COMNAND
        .db     'L'
        .db     'A'
        .db     'C'

        .dw     BASIC_SUB-1
        .db     'B'+$80   ; BASIC SUB COMNAND
        .db     'U'

BAS_EXT_COM:
        .DB     'S'

RESET_FDC:
        lda     #$03                ; *********** RESET FD ACIA to 8N1 (needed for YE-DOS)
        sta     ACIA_DC
        lda     #$54
        sta     ACIA_DC
        rts

; ************************************ Fixed Subroutine in ROM <<<DISK BOOT>>>
HERE_POS        .SET *
                .ORG DSK_BOOT
DELTA           .SET HERE_POS - *
```

```
                        .IF DELTA > 0
                        .ERROR "*** A D D R E S S Conflict !! ***"
                        .ENDIF

DISK_BOOT:
            jsr         LFC0C                   ; Init FDC Ports
            jsr         LFC26
            jmp         ($FD)                   ; INFO: indirect jump

LFC0C:

            ldy         #$00                    ; New FDC Port Initialization
            sty         PIA_DA                  ; Select DDR_A
            sty         PIA_PA                  ; DDRA=0=All Input
            ldx         #$04                    ;
            stx         PIA_DA                  ; Select DATA_A (for later access)
            sty         PIA_DB                  ; Select DDR_B
            dey                                 ; now y=FF
            sty         PIA_PB                  ; DDRB=1=All Output
            stx         PIA_DB                  ; Select DATA_B (for later access)
LFC21:
            .IF WE_TYPE==1
            dey                                 ; Set PB0=WE=0
            .ELSE
            nop                                 ; Set PB0=WE=1
            .ENDIF
            sty         PIA_PB                  ; Set PB
            jmp         RESET_FDC
;           LDX         #$04                    ; ** Prevents "0" glitch on reset **
;           LDA         #$00                    ; But not enough ROM space
;           LDY         #$FF
;           STA         PIA_DA                  ; DDR_A
;           STX         PIA_DB                  ; DATA_B
;           STA         PIA_PA                  ; PA=All Input
;           STY         PIA_PB                  ; PB=Pre-Set FF
;           STX         PIA_DA                  ; DATA_A
;           STA         PIA_DB                  ; DDR_B
;           STY         PIA_PB                  ; PB=All Output
;           STX         PIA_DB                  ; DATA_B


LFC26:
            .IF WE_TYPE==1
            lda         #$BA                    ; Diff: was $FB in OSI ROM (MOTOR ON IS ADDED, Changed to WE=0)
            .ELSE
            lda         #$BB                    ; Diff: was $FB in OSI ROM (MOTOR ON IS ADDED, Changed to WE=1)
            .ENDIF
            bne         LFC33
LFC2A:
            lda         #$02                    ; BOOT LOOP WAIT FOR TRK00
            bit         PIA_PA
            beq         LFC4D
            .IF WE_TYPE==1
            lda         #$BE                    ; Diff: was $FF in OSI ROM (WITH MOTOR ON, Changed to WE=0)
            .ELSE
            lda         #$BF                    ; Diff: was $FF in OSI ROM (WITH MOTOR ON, Changed to WE=1)
            .ENDIF
```

```
LFC33:
            sta         PIA_PB
            jsr         LFCA5                   ; Short delay
            and         #$F7
            sta         PIA_PB
            jsr         LFCA5                   ; Short delay
            ora         #$08
            sta         PIA_PB
            ldx         #$18
            jsr         LFC91                   ; STEP DELAY during Booting
            beq         LFC2A
LFC4D:
            .IF WE_TYPE==1
            ldx         #$3E                    ; Diff: was $7F in OSI ROM (WITH MOTOR ON, Changed to WE=0)
            .ELSE
            ldx         #$3F                    ; Diff: was $7F in OSI ROM (WITH MOTOR ON, Changed to WE=1)
            .ENDIF
            stx         PIA_PB
            jsr         LFC91                   ; More delay
LFC55:
            lda         PIA_PA
            bmi         LFC55                   ; WAIT FOR INDEX LOW
LFC5A:
            lda         PIA_PA
            bpl         LFC5A                   ; WAIT FOR INDEX HIGH

            jsr         RESET_FDC               ; RESET FD ACIA to 8N1 (needed for YE-DOS)
            jsr         LFC9C                   ; Get byte from ACIA
            sta         $FE
            tax                                 ; remember Start high byte
            jsr         LFC9C
            sta         $FD
            jsr         LFC9C
            sta         $FF
            ldy         #$00
LFC7B:
            jsr         LFC9C
            sta         ($FD),y                 ; Store Boot Sector
            iny
            bne         LFC7B
            inc         $FE
            dec         $FF
            bne         LFC7B
            stx         $FE
            dey                                 ; y=FF
            bne         LFC21                   ; CHANGED: ALL OFF (AND MOTOR, WE OFF))

;
LFC91:                                          ; ********** DELAY LOOP **********
            ldy         #$F8
LFC93:
            dey
            bne         LFC93
            nop                                 ; Diff: was EXOR FF(X) in OSI ROM
            nop                                 ; Diff: $55, $FF
            dex
```

```
        bne     LFC91
        rts
;
LFC9C:                          ; LOAD FD ACIA BYTE
        lda     ACIA_DC
        lsr
        bcc     LFC9C
        lda     ACIA_DD
LFCA5:
        rts
;
LFCA6:                          ; Reset support subroutine
        lda     #$03            ; Initialise ACIA & FCD PORT
        sta     ACIA_S
        lda     #$11
        sta     ACIA_S          ; Set 8N2 DIV16, RTS low, no IRQ
        jmp     LFC0C           ; Init FDC Ports
        nop
        nop
        nop

;
LFCB1:                          ; Fixed Output Char to Cassette
        pha
LFCB2:
        lda     ACIA_S
        lsr
        lsr
        bcc     LFCB2
        pla
        sta     ACIA_D
        rts
;
LFCBE:
        eor     #$FF
        sta     KBD_PORT
        eor     #$FF
        rts
;
LFCC6:
        pha
        jsr     LFCCF
        tax
        pla
        dex
        inx
        rts
;
LFCCF:
        lda     KBD_PORT
        eor     #$FF
        rts


;
;********************************************************************************
;                               BASIC IO PARAMETER TRANSFER TABLE
```

```
;********************************************************************************
;
SCR_PARAMETER:                                          ; Data to be moved to $0200 to $022B
                                                ; Diff: was all $FF in OSI ROM
        .db     NEW_LINE                        ; Screen cursor initial ($0200)
        .db     $20                             ; Save character to be printed
        .db     $20                             ; Temp char for screen
        .db     $00, $00, $00, $00              ; LOAD/EDITOR/SAVE/DELAY flags ($0203)

                                                ; Screen Memory scroll support routine
        lda     VIDEO_RAM,y
        sta     VIDEO_RAM,y
        iny
        rts

;********************************************************************************
;                       BASIC IO VECTOR an FLAG TABLE
;               $020F ............................ $0217
;               $0213..$0216  P.K Input Routine Workspace
;               $0200..$0212  UNUSED Area
;********************************************************************************
        .dw     UPPER                           ; New Cursor Start Screen Position ($020F..$0210)
        .db     LINE_C                          ; New Cursor Start Char under Cursor ($0211)
        .db     $00                             ; ??? ($0212)
        .db     $00                             ; Basic Keyboard Char value  ($0213)
        .db     $00                             ; Basic Keyboard Repeat Flag  ($0214)
        .db     $00                             ; Basic Keyboard Original Key ($0215)
        .db     $00                             ; Same as before when valid  ($0216)
        .db     $00                             ; Last Keyboard Character ($0217)

        .DW     INP_VECT                        ;  $0218 Input Vector
        .dw     OUT_VECT                        ;  $021A Output Vector
        .dw     BAS_CTRC                        ;  $021C Ctrl C Check
        .dw     BAS_LVEC                        ;  $021E Load Vector
        .dw     BAS_SVEC                        ;  $0220 Save Vector
        .db     CursorDEF                       ; ($BB) to $0222 (Cursor Character)

        .db   $4C
        .dw     IRQ_ENTRY                       ; ($0223) IRQ ENTRY ADDRESS

        .db     $4C
        .dw   NMI_ENTRY                 ; ($0226) NMI ENTRY ADDRESS to DOS MEM

        .db   $4C
        .dw     USR_SUB                                 ; ($0229) USR SUB ENTRY to $000A

        .db     $FF

;
;*********************************** Fixed Subroutine in ROM <<<KEYBOARD>>>
HERE_POS        .SET *
                .ORG  INP_CHAR
DELTA           .SET HERE_POS - *
                .IF DELTA > 0
                .ERROR "*** A D D R E S S Conflict !! ***"
                .ENDIF
```

```
INPUT_CHR:
        txa
        pha
        tya
        pha
LFD04:
        lda     #$01
LFD06:
        jsr     LFCBE
        jsr     LFCC6
        bne     LFD13
LFD0E:
        asl
        bne     LFD06
        beq     LFD66
LFD13:
        lsr
        bcc     LFD1F
        rol
        cpx     #$21
        bne     LFD0E
        lda     #$1B
        bne     LFD40
LFD1F:
        jsr     LFDC8
        tya
        sta     $0213
        asl
        asl
        asl
        sec
        sbc     $0213
        sta     $0213
        txa
        lsr
        jsr     LFDC8
        bne     LFD66
        clc
        tya
        adc     $0213
        tay
        lda     LFDCF,y
LFD40:
        cmp     $0215
        bne     LFD6B
        dec     $0214
        beq     LFD75
        ldy     #$05
LFD4C:
        ldx     #$C8
LFD4E:
        dex
        bne     LFD4E
        dey
        bne     LFD4C
        beq     LFD04
LFD56:
        cmp     #$01
        beq     LFD8F
        ldy     #$00
        cmp     #$02
        beq     LFDA7
        ldy     #$C0
        cmp     #$20
        beq     LFDA7
LFD66:
        lda     #$00
        sta     $0216
LFD6B:
        sta     $0215
        lda     #$02
        sta     $0214
        bne     LFD04
LFD75:
        ldx     #$96
        cmp     $0216
        bne     LFD7E
        ldx     #$14
LFD7E:
        stx     $0214
        sta     $0216
        lda     #$01
        jsr     LFCBE
        jsr     LFCCF
LFD8C:
        lsr
        bcc     LFDC2
LFD8F:
        tax
        and     #$03
        beq     LFD9F
        ldy     #$10
        lda     $0215
        bpl     LFDA7
        ldy     #$F0
        bne     LFDA7
LFD9F:
        ldy     #$00
        cpx     #$20
        bne     LFDA7
        ldy     #$C0
LFDA7:
        lda     $0215
        and     #$7F
        cmp     #$20
        beq     LFDB7
        sty     $0213
        clc
        adc     $0213
LFDB7:
        sta     $0213
        pla
        tay
        pla
        tax
        lda     $0213
        rts
;
LFDC2:
        bne     LFD56
        ldy     #$20
        bne     LFDA7
LFDC8:
        ldy     #$08
LFDCA:
        dey
        asl
        bcc     LFDCA
        rts
;
LFDCF:
        .db     $D0
LFDD0:
        .db     $BB, $2F, $20, $5A, $41, $51, $2C, $4D
        .db     $4E, $42, $56, $43, $58, $4B, $4A, $48
        .db     $47, $46, $44, $53, $49, $55, $59, $54
        .db     $52, $45, $57, $00, $00, $0D, $0A, $4F
        .db     $4C, $2E, $00, $FF, $2D, $BA, $30, $B9
        .db     $B8, $B7, $B6, $B5, $B4, $B3, $B2, $B1


;
; ************************************ Fixed Subroutine in ROM <<<MONITOR ENTRY>>>
HERE_POS        .SET *
                .ORG MON_ENTY
DELTA           .SET HERE_POS - *
                .IF DELTA > 0
                .ERROR "*** A D D R E S S Conflict !! ***"
                .ENDIF

LFE00:
        ldx     #$28
        txs
        cld

        jsr     LF9EE                   ; Subroutine to Clear Text Screen

        sty     $FF
        sty     $FE
        sty     CASS_FLAG               ; Clear Cass Flag
        beq     LFE43
LFE2A:
        jsr     LFEE9
        cmp     #'/'
        beq     LFE4F
        cmp     #'G'
        beq     LFE4C
        cmp     #'L'
        beq     LFE7C
        jsr     LFE93
        bmi     LFE2A
        ldx     #$02
        jsr     LFEDA
LFE43:
        lda     ($FE),y
        sta     $FC
        jsr     LFEAC
        bne     LFE2A
LFE4C:
        jmp     ($FE)           ;INFO: indirect jump
;
LFE4F:                          ; Input Loop Monitor
        jsr     LFEE9
        cmp     #$2E
        beq     LFE2A
        cmp     #$0D
        bne     LFE69
        inc     $FE
        bne     LFE60
        inc     $FF
LFE60:
        ldy     #$00
        lda     ($FE),y
        sta     $FC
        jmp     LFE77
;
LFE69:
        jsr     LFE93
        bmi     LFE4F
        ldx     #$00
        jsr     LFEDA
        lda     $FC
        sta     ($FE),y
LFE77:
        jsr     LFEAC           ; Display ADR+Data
        bne     LFE4F           ; Always loop back
LFE7C:
        sta     CASS_FLAG       ; Set Cass flag <>0 for load from ACIA
        beq     LFE4F           ; Always loop back
LFE80:
        lda     ACIA_S
        lsr
        bcc     LFE80
        lda     ACIA_D
        and     #$7F
        rts

CLS_SUB:
        lda     #$20
        ldy     #$00
CLS_LOOP:

        sta     VIDEO_RAM,y
        sta     VIDEO_RAM+$100,y
        sta     VIDEO_RAM+$200,y
        sta     VIDEO_RAM+$300,y

        .IF     SCREEN_TYPE==1
        sta     VIDEO_RAM+$400,y
        sta     VIDEO_RAM+$500,y
        sta     VIDEO_RAM+$600,y
        sta     VIDEO_RAM+$700,y
        .ENDIF

        iny
        bne     CLS_LOOP
        rts


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

LFE93:                          ; Fixed entry Convert ASCII HEX to BIN
        cmp     #'0'            ; Result in (A)
        bmi     LFEA9
        cmp     #':'
        bmi     LFEA6
        cmp     #'A'
        bmi     LFEA9
        cmp     #'G'
        bpl     LFEA9
        sec
        sbc     #$07
LFEA6:
        and     #$0F
        rts
;
LFEA9:
        lda     #$80
        rts
;
LFEAC:                          ; *** Out monitor display Address ****
        ldx     #$03
        ldy     #$00
LFEB0:
        lda     $FC,x
        lsr
        lsr
        lsr
        lsr
        jsr     LFECA
        lda     $FC,x
        jsr     LFECA
        dex
        bpl     LFEB0
        lda     #$20
        sta     VMON_ADR+4
        sta     VMON_ADR+5
        rts
;
LFECA:                          ; *** HEX OUT ***
```

```
        and     #$0F
        ora     #$30
        cmp     #$3A
        bmi     LFED5
        clc
        adc     #$07
LFED5:
        sta     VMON_ADR,y
        iny
        rts
;
LFEDA:                          ; ** shift/roll adress ***
        ldy     #$04
        asl
        asl
        asl
        asl
LFEE0:
        rol
        rol     $FC,x
        rol     $FD,x
        dey
        bne     LFEE0
        rts

;                               ; Moved to here
LFEE9:
        lda     CASS_FLAG
        bne     LFEF1           ; Get Input from ACIA
        jmp     INPUT_CHR       ; Get Input form Keyboard
LFEF1:  jmp     LFE80

;
; ******************************************************************************************
;           Boot Message Text
; ******************************************************************************************
XFEF0:
        .db     $20, $0D, "YE-OSI : ", $00      ; Diff: was table in OSI ROM

;
; ************************************ Fixed Subroutine in ROM <<<RESET ENTRY>>>
HERE_POS        .SET *
                .ORG  RESET
DELTA           .SET HERE_POS - *
                .IF DELTA > 0
                .ERROR "*** A D D R E S S Conflict !! ***"
                .ENDIF

RST_VEC:
        cld
        ldx     #$28
        txs
        ldy     #$2B                    ; Diff: was $0A in OSI ROM
LFF06:
        lda     SCR_PARAMETER,y                 ; Diff: was $FF, $FE in OSI ROM
        sta     SCR_BAS_PARAM,y                 ; Diff: was $17, $02 in OSI ROM
        dey                             ; Diff: Range $0218..0221
        bpl     LFF06                   ; versus now: $0200..022A

        jsr     LFCA6                   ; Sub added to Initialise ACIA & FDC PORT

        ldy     #$00                    ; Diff: Code change, was sty $0212
        lda     #CODE_NF & 255          ; Basic Code NOT FOUND Anaysis LOW
        sta     $022C                   ; Diff: Code change, was sty $0203
        lda     #CODE_NF >> 8           ; Basic Code NOT FOUND Anaysis HIGH
        sta     $022D                   ; Diff: Code change, was sty $0205
                                        ; Sets last Address byte of SUB to $000A

        jsr     LF9EE                   ; Subroutine to Clear Text Screen

LFF24:
        lda     XFEF0,y                 ; Display Initial Boot Message
        beq     LFF2F

        jsr     BASIC_OUT               ; $BF2D   (Old OUTPUT Vector by BASIC)
        iny
        bne     LFF24
LFF2F:
        ldy     #$00
LFF35:
        lda     XFF5F,y                 ; Display Command String D/C/W/M
        beq     LFF40
        jsr     BASIC_OUT               ; $BF2D   (Old OUTPUT Vector by BASIC)
        iny
        bne     LFF35
LFF40:
        jsr     INP_KBD                 ; Get KEY
        cmp     #'M'
        bne     LFF4A
        jmp     LFE00                   ; JUMP "MONITOR"

LFF4A:
        cmp     #'W'
        bne     LFF51
        jmp     BASIC_WARM              ; JUMP "WARM START"

LFF51:
        cmp     #'C'
        bne     LFF58
        jmp     BASIC_COLD              ; JUMP "COLD START"

LFF58:
        cmp     #'D'
        bne     LFF40
        jmp     DISK_BOOT               ; JUMP "DISK BOOT"
;
XFF5F:                                  ; Command String
        .DB     "DOS/M/C/W ?" , $00

; ******************************** Fixed Subroutine in ROM <<<OLD MAIN OUTPUT>>>
HERE_POS        .SET *
                .ORG BAS_OLD
DELTA           .SET HERE_POS - *
                .IF DELTA > 0
                .ERROR "*** A D D R E S S Conflict !! ***"
                .ENDIF

LFF69:
        jsr     BASIC_OUT               ; $BF2D  (Old OUTPUT Vector by BASIC)
LFF6C:
        pha
        lda     $0205
        beq     LFF94
        pla
        jsr     LFCB1
        cmp     #$0D
        bne     LFF95
        pha
        txa
        pha
        ldx     #$0A
        lda     #$00
LFF81:
        jsr     LFCB1
        dex
        bne     LFF81
        pla
        tax
        pla
        rts
;
; ****************************** Fixed Vectors in ROM <<<BASIC Load Vector>>>
HERE_POS        .SET *
                .ORG BAS_LVEC
DELTA           .SET HERE_POS - *
                .IF DELTA > 0
                .ERROR "*** A D D R E S S Conflict !! ***"
                .ENDIF

        pha
        dec     $0203
        lda     #$00
LFF91:
        sta     $0205
LFF94:
        pla
LFF95:
        rts
;
; ****************************** Fixed Vectors in ROM <<<BASIC Save Vector>>>
HERE_POS        .SET *
                .ORG BAS_SVEC
DELTA           .SET HERE_POS - *
                .IF DELTA > 0
                .ERROR "*** A D D R E S S Conflict !! ***"
                .ENDIF

        pha
        lda     #$01
        BNE     LFF91
```

```
;************************************ Fixed Subroutine in ROM <<<CONTROL C>>>
HERE_POS            .SET *
                    .ORG BAS_CTRC
DELTA               .SET HERE_POS - *
                    .IF DELTA > 0
                    .ERROR "*** A D D R E S S Conflict !! ***"
                    .ENDIF

LFF9B:
          lda       $0212
          bne       LFFB9
          lda       #$FE
          jmp       LF9CE                           ; Diff: Code change, was sta $DF00
;
LFFA5:
          bit       KBD_PORT
          bvs       LFFB9
          lda       #$FB
          sta       KBD_PORT
          bit       KBD_PORT
          bvs       LFFB9
          lda       #$03
          jmp       BASIC_CTRLC
LFFB9:
          rts
;
;********************************** Fixed Subroutine in ROM <<<BASIC Input Routine>>>
HERE_POS            .SET *
                    .ORG BAS_INP
DELTA               .SET HERE_POS - *
                    .IF DELTA > 0
                    .ERROR "*** A D D R E S S Conflict !! ***"
                    .ENDIF

INP_KBD:
          bit       $0203
          bpl       LFFD8
LFFBF:
          lda       #$FD
          sta       KBD_PORT
          lda       #$10
          bit       KBD_PORT
          beq       LFFD5
          lda       ACIA_S
          lsr
          bcc       LFFBF
          lda       ACIA_D
          rts
;
LFFD5:
          inc       $0203
LFFD8:
          jmp       INPUT_CHR
          jmp       EX_BASIC                        ; Diff: Code change, was $FF, $FF
;

          .DB       $FF
```

```
; *********************************** Fixed Vectors in ROM <<<VERSION INFO>>>
HERE_POS            .SET *
                    .ORG VERSION
DELTA               .SET HERE_POS - *
                    .IF DELTA > 0
                    .ERROR "*** A D D R E S S Conflict !! ***"
                    .ENDIF

          .DB       VER                                     ; New ROM Version INFO

; *********************************** Fixed Subroutine in ROM <<<BASIC SCR PARAMETER>>>
HERE_POS            .SET *
                    .ORG SCREEN_PAR
DELTA               .SET HERE_POS - *
                    .IF DELTA > 0
                    .ERROR "*** A D D R E S S Conflict !! ***"
                    .ENDIF

LFFE0:    .db       SCREEN_START      ; SCREEN Cursor displacement
          .db       SCREEN_LENGTH  ; SCREEN Line Chars
LFFE2:    .db       SCREEN_TYPE       ; SCREEN PAGE INFO $D300 or $D700 (0=$D300)
          .db       $00, $03 ; BEGIN OF RAM
          .db       $FF, $9F ; END OF RAM
          .db       $00, $03 ; BEGIN OF RAM
          .db       $FF, $9F ; END OF RAM
;
; *********************************** Fixed Vectors in ROM <<<PAGE $FF VECTORS>>>
HERE_POS            .SET *
                    .ORG ROM_PAGE
DELTA               .SET HERE_POS - *
                    .IF DELTA > 0
                    .ERROR "*** A D D R E S S Conflict !! ***"
                    .ENDIF
BASIC_VECTOR:
LFFEB:    jmp       ($0218)           ;INFO: indirect jump INVEC
LFFEE:    jmp       ($021A)           ;INFO: indirect jump OUTVEC
LFFF1:    jmp       ($021C)           ;INFO: indirect jump CCVEC
LFFF4:    jmp       ($021E)           ;INFO: indirect jump LDVEC
LFFF7:    jmp       ($0220)           ;INFO: indirect jump SVVEC
;
; *********************************** Fixed CPU Vectord in ROM <<<ROM CPU VECTORS>>>
HERE_POS            .SET *
                    .ORG ROM_VECT
DELTA               .SET HERE_POS - *
                    .IF DELTA > 0
                    .ERROR "*** A D D R E S S Conflict !! ***"
                    .ENDIF

HARD_VECTORS:
          .dw       NMI_VEC           ; NMI VECTOR       ; Diff: ADR change, was $0130 (Stack issue)
          .dw       RST_VEC                    ; RESET VECTOR
          .dw       IRQ_VEC           ; IRQ VECTOR       ; Diff: ADR change, was $01C0 (Stack issue)
```

PART 2 BOOT DOS

Listing

```
;
;                    (c) Copyright TB 2022
;
;          File:               D_E000_E8FF.bin                 BOOTSECTOR
;
;          Date:               Dec  2023
;
;          CPU:                MOS Technology 6502 (MCS6500 family)
;
;    VERSION:                  Version 3.54_40/80 - Updated for 40/80 Track drives
;                              HEAD LOAD will be present with MOTOR ON,
;                              Pull-Up resistors on 610 board removed to prevent disk errors during POWER cycle!!!
;                              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
;                              few bytes free memory left
ORG_POS= $E000

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

FAT_D  = ORG_POS+$1400         ; FAT Memory area $F400
FAT_S  = FAT_D+$60             ; FAT start of name are
FAT_ID = FAT_D+$50             ; DISK Title 16 Bytes
STACKS = ORG_POS+$08C0         ; DOS TEMP Stack Area

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

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
ROMDOS  = $F800  ; Vector to LOAD DOS Extension
ROMASS  = $F802  ; Vector to LOAD ASS

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
;
;***************************************************************
;              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)
   .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 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, 6/12ms 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                      ; ******* FORMAT OR WRITE BOOT SECTOR *******
;                                      ; LE026: Flag for Format all tracks

PAR_T  = BOOT_S-ORG_POS                ; Length of DOS parameter table
B_OFF  = $0900+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)
          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 ($DF)
          sta      DATA_S+1
          lda      #BC_ST&255          ; ($AF)
          sta      DATA_S
```

```
        lda     #$09
        jsr     LE127           ; WRITE Length byte to FDC

        ldy     #B_OFF          ; WRITE STD DOS PARAETER BLOCK TO DISK ($51)
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) ($79)
        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
        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 ??????
;

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


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


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 Dymmy 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     #$0D
        clc
        adc     DATA_S          ; Add 13 to Pointer to next FAT entry
        sta     DATA_S
        bcc     LE712
        inc     DATA_S+1
LE712:
        lda     DATA_S+1
        cmp     #$F8            ; 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

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

LE74B:                          ; NEW DOS COLD START ROUTINE

        lda     DOS_CFG0        ; Restore DOS JUMP (0) vector
        sta     LE000
        lda     DOS_CFG0+1
        sta     LE000+1

        jsr     LE335           ; CHECK DRIVES ATTACHED AND LOADS FAT (4)
        jsr     LE78BX          ; Load DOS
        jsr     LE795           ; Framed Text Output
LE768:
        jsr     ROMINP_V
        cmp     #'C'            ; C) BASIC COLD START
        bne     LE776
LE768X:
        jmp     BASIC_COLD

;
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     #'M'
        bne     LE78B
        jmp     MON_ROM                 ; c) MONITOR
;
LE78B:
        cmp     #'A'
        bne     LE768
        jsr     LE78BY          ; d) ASS EDITOR
        jmp     LE74B

LE78BX:
        jmp     (ROMDOS)
LE78BY:
        jmp     (ROMASS)
; *************************************************************************************
;                               ; **** 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







; **************************** STANDARD BOOT UP DOS Parameter Set  **************************
;
DOS_CFG0:
        .db   LE6C6&255, LE6C6>>8       ; JUMP SEARCH FILE (0)
```

```
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:

; ************************************ 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    "<M> MONITOR ", $F9
   .db    $8B, $0D, $0A, $8C, $20, $FF
   .db    "<A> ASSEMBLER ", $FB
   .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_E: ; UNUSED AREA

;
; **************** FREE STACK AND ZEROPAGE ******************************************
; Saves Stack values to     E8C0-E8DF (16 levels)
; Saves Zero Page E0 to FF to         E8E0-E8FF (32 bytes zeropage)

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


LE8C0:   .ORG      ORG_POS+$08C0   ; Stack Storage
         .dcb      $20, $FF

LE8E0:   .ORG      ORG_POS+$08E0   ; Zero Page Storage
         .dcb      $20, $FF
```