

```
##### LIBDEF.TXT
; Purpose.....: Epson PX-8 library modules
; Version.....: 2025.02.19
#####

;***** hardware
;+++++ definitions
;===== misc
hwCYMS:      EQU      3500      ; clock cycles per ms (CPU 3.5 MHz)

;===== special keys codes
hbKAUP:      EQU      01EH      ; arrow up
hbKADN:      EQU      01FH      ; arrow down
hbKALE:      EQU      01DH      ; arrow left
hbKARI:      EQU      01CH      ; arrow right

;===== system addresses
haWB:        EQU      0EC03H     ; MAPLE warm boot address
haRSCL:      EQU      haWB+03CH  ; close RS-232
haRSIO:      EQU      haWB+051H  ; RSIOX functiton

haCTL1:      EQU      0F0B0H     ; image of CTRLR1
hpCTL1:      EQU      00H        ; port address CTRLR1

haCTL2:      EQU      0F0B2H     ; image of CTRLR2
hpCTL2:      EQU      02H        ; port address CTRLR2
hiSWRS:      EQU      3         ; RS-232 power on/off bit
hiINHR:      EQU      4         ; inhibit RS bit
hiAUX:       EQU      5         ; connect 8251 to RS-232 connector

haIER:       EQU      0F0B3H     ; image of interrupt enable bits
hpIER:       EQU      04H        ; interrupt get/set port address
hiSIRX:      EQU      1         ; 8251 rx interrupt bit
hiSICD:      EQU      2         ; carrier detect interrupt bit

haRSMO:      EQU      0F6D0H     ; 8251 mode register image
haRSCM:      EQU      0F6D1H     ; 8251 command register image
haRSOP:      EQU      0F2C8H     ; RS232 open flag

;===== 8251 USART
;----- port addresses
hpSCS:       EQU      0DH        ; 8251 status register (read)
hpSCM:       EQU      0DH        ; 8251 command register (write)
hpSDA:       EQU      0CH        ; 8251 data register

;----- baudrates (clock divisor 1/16)
; The values are for use by the RSOPEN function of RSIOX BIOS function
;-----
hbSR11:      EQU      02H        ; 110 baud
hbSR15:      EQU      04H        ; 150 baud
hbSR30:      EQU      06H        ; 300 baud
hbSR60:      EQU      08H        ; 600 baud
hbSR1K:      EQU      0AH        ; 1200 baud
hbSR2K:      EQU      0CH        ; 2400 baud
hbSR4K:      EQU      0DH        ; 4800 baud
hbSR9K:      EQU      0EH        ; 9600 baud
hbSR9T:      EQU      0FH        ; 19200 baud

;----- data and stop bits, parity
; Mode instruction bit setup
; | | | | | | | |
; | | | | | | +-+--- Baudrate factor 00=sync,    01=1x,    10=16x,    11=64x
; | | | | +-+----- Data bits      00=5bit,     01=6bit,   10=7bit,   11=8bit
; | | +-+----- Parity              00=off,      01=odd,    10=off,    11=even
; +-+----- Stop bits                00=inhibit, 01=1bit,   10=1.5bit, 11=2bit
;
; If RSIOX is used, those bits must be rearranged in hSAC()
```

```

;-----
hbs8N1:      EQU      01001110B      ; 8 data, 1 stop bit, no parity
hbs8E1:      EQU      01111110B      ; 8 data, 1 stop bit, even parity
hbs8O1:      EQU      01011110B      ; 8 data, 1 stop bit, odd parity
hbs8N2:      EQU      11001110B      ; 8 data, 2 stop bit, no parity

hbs7N1:      EQU      01001010B      ; 7 data, 1 stop bit, no parity
hbs7E1:      EQU      01111010B      ; 7 data, 1 stop bit, even parity
hbs7O1:      EQU      01011010B      ; 7 data, 1 stop bit, odd parity
hbs7N2:      EQU      11001010B      ; 7 data, 2 stop bit, no parity
hbs7E2:      EQU      11111010B      ; 7 data, 2 stop bit, even parity
hbs7O2:      EQU      11011010B      ; 7 data, 2 stop bit, odd parity

;----- handshake lines
hbsHRT:      EQU      00000010B      ; use RTS/CTS handshake
hiSHRT:      EQU      1              ; bit number for BIT/SET/RES
hbsHDT:      EQU      00000001B      ; use DTR/DSR handshake
hiSHDT:      EQU      0              ; bit number for BIT/SET/RES
hbsHRD:      EQU      00000011B      ; both RTS & DTR

;----- RX error bits
hiSAEB:      EQU      2              ; receive buffer overflow
hiSAEP:      EQU      4              ; parity error
hiSAEO:      EQU      5              ; overrun error
hiSAEF:      EQU      6              ; framing error
hbSAER:      EQU      01110010B      ; errors mask

;----- rx buffer
haSAB:      EQU      0FB90H          ; rx buffer address (system buffer)
hwSABZ:      EQU      0FD16H-0FB90H ; rx buffer length (system buffer)
hwSABE:      EQU      hwsABZ/10*1    ; buffer empty level (handshake on)
hwSABF:      EQU      hwsABZ/10*7    ; buffer full level (handshake off)

```

```

;+++++ routines
;===== hSAC()
; Purpose.....: Configure 8251 USART
; Input.....: B: Baudrate, one of hbSRxx
;             C: data/stop/parity, one of hbSdps (e.g. hbS8N1)
; Registers...: AF destroyed
; Labels.....: h)ardware S)IO-A) C)onfigure
;=====

;===== hSACD()
; Purpose.....: Set RS232 to default configuration
; Registers...: None saved
; Labels.....: h)ardware S)IO-A) C)onfigure D)efault
;=====

;===== hSACB()
; Purpose.....: Set USART baud rate
; Input.....: A: Baud rate to set, one of hbSRxx
; Registers...: AF destroyed
; Labels.....: h)ardware S)io A) C)onfigure B)audrate
;=====

;===== hSARE()
; Purpose.....: Read USART RX error bits from last operation,
;               cleared by BIOS after reading
; Output.....: A: Error bits: rx buffer overflow (hiSAEB)
;               parity error (hiSAEP)
;               framing error (hiSAEF)
;               overrun error (hiSAEO)
;               Z: 0= some error(s)
;                 1= no error
; Registers...: AF destroyed
; Labels.....: h)ardware S)io A) R)ead E)rror
;=====

;===== hSARD()
; Purpose.....: Read a byte from USART after waiting for it to be ready
; Input.....: CY=0: Wait for byte to arrive or cancelled
;             CY=1: Just check for RX byte, keyboard is not checked
; Output.....: A: received byte
;             CY=0: not cancelled
;             CY=1: cancelled while waiting for byte to arrive
;             Z=1: No byte received
;             Z=0: Byte received
; Registers...: AF destroyed
; Labels.....: h)ardware S)io A) R)eaD)
;=====

;===== hSARC()
; Purpose.....: Clear USART RX buffer
; Registers...: All saved
; Labels.....: h)ardware S)IO channel A) R)x C)lear
;=====

;===== hSAWW()
; Purpose.....: Wait until TX FIFO is empty
; Input.....: CY: 1= check for keyboard cancel request
;             0= no cancel check
; Output.....: CY: 0= FIFO is empty
;             1= cancelled while waiting for TX empty
; Registers...: AF destroyed
; Labels.....: h)ardware S)IO channel A) W)rite W)ait
;=====

```

```

;===== hSAWR()
; Purpose.....: Write a byte to USART after waiting for it to be ready. CTS
;               must be asserted, otherwise the transmitter is disabled by
;               hardware.
; Input.....: A: byte to write
; Output.....: CY=0: data sent
;             CY=1: cancelled while waiting for DSR and/or TX empty
; Registers...: F destroyed
; Labels.....: h)ardware S)io A) W)R)ite
;=====

;===== hSAHC()
; Purpose.....: Configure handshake. If a line is not configured, it will
;               not be changed.
; Input.....: A: hiSHRT (RTS/CTS) and/or hiSHDT (DTR/DSR)
; Registers...: AF destroyed
; Labels.....: h)ardware S)IO-A) H)andshake C)onfigure
;=====

;===== hSAHS()
; Purpose.....: Set/Reset configured handshake line(s)
; Input.....: CY=1: Set line(s) (active)
;             CY=0: Reset lines(s) (inactive)
; Registers...: All saved
; Labels.....: h)ardware S)IO-A) H)andshake S)et
;=====

;===== hSAHF()
; Purpose.....: Force set/reset handshake line(s), can be used to preset
;               unused auto-line(s) to a default state
; Input.....: A: hiSHRT (RTS) and/or hiSHDT (DTR)
;             CY=1: Force line(s) active, untouched by auto-reset
;             CY=0: Force line(s) inactive, untouched by auto-set
; Registers...: All saved
; Labels.....: h)ardware S)IO-A) H)andshake F)orce
;=====

;===== hSAHO()
; Purpose.....: Get configured output (RTS/DTR) handshake status
; Output.....: CY=1: handshake is active
;             CY=0: handshake is inactive
; Registers...: AF destroyed
; Labels.....: h)ardware S)io A) H)andshake get O)utput
;=====

;===== hSAHI()
; Purpose.....: Get input (DSR) handshake status (CTS cannot be read)
; Output.....: CY=1: handshake enabled, CY=0: it is not
; Registers...: A destroyed
; Labels.....: h)ardware S)IO-A) H)andshake read I)nput
;=====

;===== hSAIO()
; Purpose.....: Call RSIOX BIOS functions
; Input.....: A: Function ID, see OsReferenceManual, page 129ff
; Output.....: As defined by function
; Registers...: AF,BC destroyed
; Labels.....: h)ardware S)IO-A) RSI)O)X
;=====

```

```

;===== nLPWR()
; Purpose.....: Check printer status, write byte to printer
; Input.....: A: byte to send
; Output.....: CY=0: data sent
;             CY=1: cancelled while printing
; Registers...: AF destroyed
; Labels.....: h)ardware L)ine P)rinter W)R)ite
;=====

;===== hKCS()
; Purpose.....: Configure cancel key
; Input.....: A: cancel key
; Registers...: AF destroyed
; Labels.....: h)ardware K)ey C)ancel S)et
;=====

;===== hKC()
; Purpose.....: Check for cancel request
; Output.....: Z: 1= cancel request
;             0= no request
; Registers...: AF destroyed
; Labels.....: h)ardware K)ey C)ancel
;=====

;===== hKR()
; Purpose.....: Check if a key is ready from the console. No direct reading,
;             uses BIOS functions
; Output.....: A=presseD key, 0 if no key pressed
;             Z=0: a key was pressed, code in A
;             Z=1: no key pressed
; Registers...: AF destroyed
; Labels.....: h)ardware K)eyboard R)ead
;=====

;===== hKL()
; Purpose.....: Get and clear the last pressed key
; Output.....: A=last pressed key, 0 if no key pressed
;             Z=0: a key was pressed, code in A
;             Z=1: no key pressed
; Registers...: AF destroyed
; Labels.....: h)ardware K)eyboard L)ast
;=====

;+++++ data
;===== text constants
htMNM:                ; the machine name

;===== video control sequences
hbVCSL:                ; cursor left
hbVCSR:                ; non-detructive cursor right
hvVCL:                ; CR/LF sequence
hvVHOM:                ; home cursor
hvVCLS:                ; clear screen and home cursor
hvVCES:                ; clear to end of screen
hvVCEL:                ; clear to end of line
hvVRCL:                ; CR, then clear to end of line
hvVRU:                ; CR and cuRsor Up
hvVRCU:                ; CR, clear to EOL, cursor up
hvVCON:                ; cursor on
hvVCOF:                ; cursor off
hvVCP:                ; cursor positioning lead-in
hvVCPC:                ; cursor positioning, col offset
hvVCPR:                ; row offset
hvVCPO:                ; rOw before column (=1)

```

```

;***** common
;++++++ definitions
;===== ASCII characters
ccCTLC:      EQU      003H          ; CTRL-C character
ccBEL:       EQU      007H          ; ring the bell
ccCTLG:      EQU      007H          ; CTRL-G character (same as BEL)
ccBS:        EQU      008H          ; backspace
ccCTLH:      EQU      008H          ; CTRL-H character (same as BS)
ccTAB:       EQU      009H          ; tabulator
ccLF:        EQU      00AH          ; end-of-line character
ccCTLL:      EQU      00CH          ; CTRL-L character
ccCR:        EQU      00DH          ; CR character
ccCTLN:      EQU      00EH          ; CTRL-N character
ccCTLP:      EQU      010H          ; CTRL-P character
ccCTLR:      EQU      012H          ; CTRL-R character
ccCTLS:      EQU      013H          ; CTRL-S character
ccCTLT:      EQU      014H          ; CTRL-T character
ccCTLV:      EQU      016H          ; ctrl-V character
ccEOF:       EQU      01AH          ; end-of-file character
ccESC:       EQU      01BH          ; escape character
ccSPC:       EQU      020H          ; space character

;===== CP/M
caWBOT:      EQU      0000H         ; boot system
caBDOS:      EQU      0005H         ; BDOS entry point
cbDIRZ:      EQU      64            ; max number of directory entries
caDMA:       EQU      0080H         ; default DMA address
caFCB:       EQU      005CH         ; default FCB address
cbFCBZ:      EQU      36            ; size of FCB
cbFCBN:      EQU      1             ; offset to filename
caFCBN:      EQU      caFCB+cbFCBN  ; start of filename
cbFCBT:      EQU      9             ; offset to filetype
caFCBT:      EQU      caFCB+cbFCBT  ; start of filetype
cbFCBX:      EQU      12            ; offset to extent number
caFCBX:      EQU      caFCB+cbFCBX  ; current extent number
cbFCBR:      EQU      33            ; offset to record number
caFCBR:      EQU      caFCB+cbFCBR  ; current record number, random access
cbFCBO:      EQU      35            ; offset to record number overflow
caFCBO:      EQU      caFCB+cbFCBO  ; current record number overflow

caCMBF:      EQU      0080H         ; default command buffer address
caTPA:       EQU      0100H         ; start of TPA
caTPAE:      EQU      0006H         ; end of TPA address

;===== error codes
ceDFRN:      EQU      0FFH          ; file not found error
ceDFRE:      EQU      0FEH          ; end of file error
ceDFWD:      EQU      0FDH          ; out of directory space
ceDFWF:      EQU      0FCH          ; disk full/io error

;----- errors returned by BDOS random write (oWR())
ceDOK:       EQU      00H           ; no error
ceDFUL:      EQU      02H           ; disk full
ceDEXT:      EQU      03H           ; cannot close extent
ceDDIR:      EQU      05H           ; directory full
ceDROR:      EQU      06H           ; record number out of range
ceDFCB:      EQU      09H           ; invalid FCB

;----- dIRW() errors
ceIRWD:      EQU      0FFH          ; drive select error
ceIRWI:      EQU      0FEH          ; i/o error

```

```

;+++++ routines
;===== uCBD()
; Purpose.....: Convert binary to decimal (0..65535)
; Input.....: DE: Value to convert
;             HL: Where to store result
;             C : Number of digits to convert (1..5)
;             CY=1: leading zeros
;             CY=0: leading blanks
; Output.....: (HL): Converted ASCII string
; Registers...: All saved
; Labels.....: u)tility C)onvert B)inary to D)ecimal
;=====

;===== uCDB()
; Purpose.....: Convert decimal ASCII to binary (0..65535)
;             Conversion stops at the 1st non-numeric or C gets zero
; Input.....: DE: start of ASCII digits
;             C : Number of digits to convert (1..5)
; Output.....: HL: Binary value
;             CY=1 if overflow or number of digits =0
; Registers...: AF, HL destroyed
; Labels.....: u)tility C)onvert D)ecimal to B)inary
;=====

;===== uCBH()
; Purpose.....: Convert binary to hex (0..FFFF), leading zeros always shown
; Input.....: DE: Value to convert
;             HL: Where to store result
;             C : Number of digits to convert (1..4)
; Output.....: (HL): Converted ASCII string
; Registers...: All saved
; Labels.....: u)tility C)onvert B)inary to H)ex
;=====

;===== uCHB()
; Purpose.....: Convert HEX ASCII to binary (0..65535)
;             Conversion stops at the 1st non-HEX or C gets zero
; Input.....: DE: start of ASCII digits
;             C : Number of digits to convert (1..4)
; Output.....: HL: Binary value
;             CY=1 if overflow or number of digits=0
; Registers...: AF, HL destroyed
; Labels.....: u)tiliuty C)onvert H)ex to B)inary
;=====

;===== uCUC()
; Purpose.....: Convert letter to uppercase
; Input.....: A: Letter to convert
; Output.....: A: Converted letter
; Registers...: A destroyed
; Labels.....: u)tility C)onvert to U)pper C)ase
;=====

;===== uCF2T()
; Purpose.....: Convert FCB filename to 8.3 format
; Input.....: HL: Filename in FCB format
;             DE: Where to store result
; Registers...: HL,DE destroyed
; Labels.....: u)tility C)onvert F)CB 2)to T)ext
;=====

```

```

;===== uDLY()
; Purpose.....: Delay for given number of milliseconds
;               Uses xxx machine cycles for one loop (BC=1)
; Input.....: BC=Number of milliseconds
; Registers...: All saved
; Labels.....: u)tility D)eL)aY)
;=====

;===== uMF()
; Purpose.....: Fill memory area with constant
; Input.....: A : Fill byte
;               HL: Starting address
;               BC: Number of bytes to write
; Registers...: All saved
; Labels.....: u)tility M)emory F)ill
;=====

;===== uHLDE()
; Purpose.....: Compare HL and DE
; Input.....: HL, DE to check
; Output.....: SBC HL,DE flags
; Registers...: F destroyed
; Labels.....: u)tility HL) DE)
;=====

;===== uDIV()
; Purpose.....: Divide and get modulo
; Input.....: HL: Dividend
;               DE: Divisor
; Output.....: HL: HL % DE, -1 if divisor = zero
;               DE: HL / DE
; Registers...: HL,DE Destroyed
; Labels.....: u)titivity DIV)ide
;=====

;===== uMOD()
; Purpose.....: Calculate modulo
; Input.....: A= Value to take modulo of
;               B= Modulo
; Output.....: B= Resulting modulo
; Registers...: B destroyed
;=====

;===== uSTER()
; Purpose.....: Change string terminator
; Input.....: A= New terminator
; Registers...: All saved
; Labels.....: u)tility S)tring TER)minator
;=====

;===== uSLEN()
; Purpose.....: Calculate length of string with configured terminator
;               If length of string is >255, it is reported as 255
; Input.....: HL: address of string
; Output.....: A: length of string
;               Z: 1 if string length is NULL
; Registers...: AF destroyed
; Labels.....: u)tility S)tring LEN)gth
;=====

;===== uSTRM()
; Purpose.....: Remove trailing spaces in a string
; Input.....: DE: String address
; Output.....: A: Remaining string length
; Registers...: AF destroyed
; Labels.....: u)tility S)tring TR)iM)
;=====

```



```

;===== nMNU()
; Purpose.....: Display a menu and process selection
;               Border length is taken from the longest item. Uses var space
;               at 0080 because the number of items is dynamic.
; Input.....: DE= address of menu definition:
;             Character to use for borders
;             Menu title (z-string)
;             Selection prompt (z-string)
;             Menuitems: Menu item text (z-string)
;                       Address to jump to or call on selection
;                       00= don't clear the menu and CALL the function
;                       01= clear the menu and JP to the function
;             NULL z-string: termination indicator
; Registers....: None saved
; Labels.....: con)sole M)enu)
;=====

;===== nIS()
; Purpose.....: Read string from console
; Input.....: DE: String buffer address
;             BC: Cursor coordinates, B=row, C=column
;             L: Maximal input length
;             A:  '!': upper case convert
;                 '#': numeric (0..9) only
;                 '$': hex (0..9, A..F) only. Implies uppercase convert
;                 anything else: no checks/conversions
;                 bit7: draw '|' borders
; Output.....: (DE): Edited string, terminated with (gvSTER)
;             A: last input character
;             CY: 1= terminated with ESC, 0= terminated with CR
; Registers....: AF destroyed
; Labels.....: con)sole I)np)ut S)tring)
;=====

;===== nIC()
; Purpose.....: Read keyboard character
; Input.....: A:Input mask
;             '!' : upper case convert
;             '#' : numeric (0..9) only
;             '$' : hex (0..9, A..F) only. Implies uppercase convert
;             'Y' : yes/no only. Implies uppercase convert
;             anything else: no checks/conversions
; Output.....: A: ASCII code of read key
;             CY: Set if control character was pressed, reset otherwise
; Registers....: AF,AF',DE' destroyed
; Labels.....: con)sole I)np)ut C)haracter)
;=====

;===== nIDR()
; Purpose.....: Read drive letter from keyboard, test if accepted by BIOS
;               If valid drive selected, this drive is active
; Output.....: A: inputted drive letter, even if not accepted
;             HL: disk parameter header address of selected drive
;             CY: 1= input aborted with <ESC>, 0= input valid (A..P)
;             Z: 1= input is valid but BIOS did not accept drive
; Registers....: HL, A destroyed
; Labels.....: con)sole I)np)ut DR)ive letter)
;=====

;===== nICR()
; Purpose.....: Check for console character ready, if so, read it
; Output.....: A: ASCII code of read key
;             Z: 1= no character ready
;             0= pressed key in A
; Registers....: AF,AF',DE' destroyed
; Labels.....: con)sole I)np)ut C)heck R)ead)
;=====

```

```

;===== NOP1()
; Purpose.....: Show a progress indicator on console
;               Uses backspace + '|/-' characters in turn
; Registers....: All saved
; Labels.....: con)sole O)utput P)rogress I)ndicator
;=====

;===== nOSCL()
; Purpose.....: Clear string at defined cursor position by writing spaces
; Input.....: B: row (0..23)
;            C: column (0..79)
;            A: length of string
;            CY: 1=do cursor positioning, 0=write at current position
; Registers....: AF destroyed
; Labels.....: con)sole O)utput S)tring C)L)ear
;=====

;===== nOSCN()
; Purpose.....: Print string on console without cursor positioning
;               The 1st word contains an address of a screen control sequence
;               If none is required, it is null
; Input.....: DE: address of message
; Registers....: AF destroyed
; Labels.....: con)sole O)utput S)tring with C)ode N)o cursor positioning
;=====

;===== nOSC()
; Purpose.....: Print string on console
;               The 1st word contains an address of a screen control sequence
;               If none is required, it is null
; Input.....: DE: address of message
;            BC: cursor coordinates, B=row, C=column
;            CY: 1=position cursor, 0=display at current position
; Registers....: AF destroyed
; Labels.....: con)sole O)utput S)tring with C)ode
;=====

;===== nOSN()
; Purpose.....: Call nOS() without cursor positioning
; Input.....: DE: address of string to write
; Registers....: AF destroyed
; Labels.....: con)sole O)utput S)tring N)o C)ursor positioning
;=====

;===== nOS()
; Purpose.....: Write string optional at defined cursor position
; Input.....: B: row (0..24)
;            C: column (0..79)
;            DE: address of string to write
;            CY: 1=do cursor positioning, 0=write at current position
; Registers....: All saved
; Labels.....: con)sole O)utput S)tring
;=====

;===== nOCP()
; Purpose.....: Position cursor
; Input.....: B: X coordinate (row)
;            C: Y coordinate (column)
; Registers....: All saved
; Labels.....: con)sole O)utput C)ursor P)osition
;=====

```

```

;===== nOCS()
; Purpose.....: Clear screen, the required sequence is defined in the
;               hardware module
; Registers...: All saved
; Labels.....: con)sole O)utput C)lear S)creen)
;=====

;===== nOCRE()
; Purpose.....: Write character to console, repeated B times
; Input.....: A: character to write
;           B: number of times to write
; Registers...: AF destroyed
; Labels.....: con)sole O)utput C)haracter R)E)peated
;=====

;===== nOC()
; Purpose.....: Write character to console
; Input.....: A: character to write
; Registers...: AF destroyed
; Labels.....: con)sole O)utput C)haracter
;=====

;===== nOCOD()
; Purpose.....: Send code sequence to console
;               The code sequence is built like <length><code 1>..

```

```

;===== dFW()
; Purpose.....: Write a file to disk
; Input.....: HL: from address
;             BC: to address
;             DE: FCB address
;             CY: 0= overwrite
;               1= append
; Output.....: A: oWR() error code in case of error, 0 otherwise
;             CY: 0= success
;               1= some error
; Registers...: AF destroyed
; Labels.....: d)isk F)ile W)rite
;=====

;===== dFO()
; Purpose.....: Open or create file if it's not already open
; Input.....: DE: FCB address
;             A: 0= open in overwrite mode (file is deleted first)
;               1= open in append mode (filesize is computed)
;               this only applies if open is in write mode (CY=1)
;             CY: 0= do not create if it does not exist (read mode)
;               1= create it (write mode)
; Output.....: A: error code in case of error, 0 otherwise
;             CY: 0= file opened/created/already open
;               1= file does not exist/directory full
; Registers...: AF destroyed
; Labels.....: d)isk F)ile O)pen
;=====

;===== dFC()
; Purpose.....: Close a file if not already closed, clear FCB except for
;               filename
; Input.....: DE: FCB address
; Registers...: AF destroyed
; Labels.....: d)isk F)ile C)lose
;=====

;===== dFD()
; Purpose.....: Delete a file, clear FCB except for filename
; Input.....: DE: FCB address
; Output.....: A: error code, 0 if no error
;             CY: 1= file not found
; Registers...: AF destroyed
; Labels.....: d)isk F)ile D)delete
;=====

;===== dFE()
; Purpose.....: Check if a file exists. Uses it's own FCB
; Input.....: BC: Address of filename, 1st byte is drive (0=default,1=A,2=B)
; Output.....: CY: 0= file exists
;             1= it does not
; Registers...: AF destroyed
; Labels.....: d)isk F)ile E)xists
;=====

;===== dFCBC()
; Purpose.....: Clear FCB, leave filename in place
; Input.....: DE: FCB address
; Registers...: All saved
; Labels.....: d)iskIO F)ileC)ontrolB)lock C)lear
;=====

```

```

;===== dFCBS()
; Purpose.....: Clear and setup FCB for file operations
; Input.....: BC: address of filename in FCB format, 1st byte is drive
;             DE: address of FCB to use
; Registers...: All saved
; Labels.....: d)iskIO F)ileC)ontrolB)lock S)etup
;=====

;===== dIRW()
; Purpose.....: Read/Write sectors from/to disk via BIOS
;             Saves and restores the logged disk. This cannot be done
;             using BDOS (by caller) if it was changed via BIOS (here).
; Input.....: HL: Buffer address, if 0, use last used address
;             D: Starting track number
;             E: Starting sector number
;             B: Number of sectors to read/write
;             C: Drive number (A=0, B=1)
;             CY: 0= read, 1=write
; Output.....: CY: 0= ok, no error, 1= some error
;             A: if error, one of gcDRWD or gcDRWI
;             D: if I/O error, track number
;             E: if I/O error, sector number
;             HL: points to end of buffer, undefined if error
; Registers...: All destroyed
; Labels.....: d)isk bI)os R)ead W)rite
;=====

;===== dSPT()
; Purpose.....: Read number of sectors per track out of DiskParameterBlock
;             of the currently selected drive, select drive in C
; Input.....: C: drive to use (A=0, B=1, ...)
; Output.....: A: number of sectors per track
;             CY: 0= success
;             1= invalid drive in C on call
; Registers...: AF destroyed
; Labels.....: d)isk get S)ectors P)er T)rack
;=====

;===== oBDON()
; Purpose.....: BDOS routines which do not return a value
; Input.....: As requested by function
; Output.....: None
; Registers...: All main saved
;=====
oCO:           Console output           ; In: A= character to write
oPO:           Punch output             ; In: A= character to punch
oLO:           List Output              ; In: A= character to list
oDCO:          Direct console I/O       ; In: A= character to output
oSIOB:         Set I/O byte             ; In: A= I/O byte value to set
oRDS:          Reset disk system        ; no input
oSELD:         Select disk              ; In: A= disk to select, A=0, B=1...
oS0:           String output            ; In: DE= string address
oSDMA:         Set DMA address          ; In: DE= DMA address
oWPD:          Write protect disk       ; In: none
oSFA:          Set file attributes      ; In: DE= FCB address
oSUN:          Set user number          ; In: A= user number to set (0..15)
oCFS:          Compute file size        ; In: DE= FCB address
oSRR:          Set random record        ; In: DE= FCB address
;===== eofS oBDON()

```

```

;===== oBDOA()
; Purpose.....: BDOS routines which return a value in A
; Input.....: As requested by function
; Output.....: A: Function result
; Registers...: AF destroyed
;=====
oCI:           Console input           ; Out: A= character from console
oRI:           Reader input            ; Out: A= character from reader
oDCI:          Direct console input    ; Out: A= character from console
oGIOB:         Get I/O byte            ; Out: A= contents of I/O byte
oGCS:          Get console status      ; Out: A: FF= char ready, 00= not ready
oGCD:          Get current disk        ; Out: A= selected disk, A=0,B=1...
oOPF:          Open file               ; In:  DE= FCB address
; Out: A= dir code, FF=file not found
oCLF:          Close file              ; In:  DE= FCB address
; Out: A= dir code, FF=file not found
oSF:           Search first            ; In:  DE= FCB address
; Out: A= dir code, FF=file not found
oSN:           Search next             ; In:  DE= FCB address
; Out: A= dir code, FF=file not found
oDLF:          Delete file             ; In:  DE= FCB address
; Out: A= dir code, FF=file not found
oRS:           Read sequential         ; In:  DE= FCB address
; Out: A= dir code, FF=file not found
oWS:           Write sequential        ; In:  DE= FCB address
; Out: A= dir code, FF=file not found
oMKF:          Make file               ; In:  DE= FCB address
; Out: A= dir code, FF=file not found
oRNF:          Rename file            ; In:  DE= FCB address
; Out: A= dir code, FF=file not found
oGUN:          Get user number         ; In:  DE= FCB address
; Out: A= active user number (0..15)
oRR:           Read random             ; In:  DE= FCB address
; Out: A= directory code
oWR:           Write random            ; In:  DE= FCB address
; Out: A= directory code
;===== eofS oBDOA()

;===== oBDODE()
; Purpose.....: BDOS routines which return a value in DE
; Input.....: As requested by function
; Output.....: DE: as defined by function
; Registers...: DE destroyed
;=====
oRCB:          Read console buffer     ; (DE): edited string, 2nd byte = len
;===== eofS oBDODE()

;===== oBDOHL()
; Purpose.....: BDOS routines which return a value in HL
; Input.....: As requested by function
; Output.....: HL: as defined by function
; Registers...: HL destroyed
;=====
oGLV:          Get login vector        ; Out: HL= online drives, L0=A, H7=P
oGAA:          Get Address (ALLOC)     ; Out: HL= base address of alloc vect
oGROV:         Get read only vector    ; Out: HL= readonly drives, L0=A, H7=P
oGDPA:         Get addr (DISK PARAMS) ; Out: HL= disk parameter block addr
oGVN:          Get version number      ; Out: A= version (e.g. A= 22 for V2.2)
;===== eofS oBDOHL()

```

```

;===== oBIOS()
; Purpose.....: Call BIOS function
; Input.....: A: Offset from BIOS entry point, one of the below obIxxx
;             DE: Input to requested BIOS routine
;             BC: Input to requested BIOS routine
; Result.....: As defined in CP/M BIOS
; Registers...: None saved
; Labels.....: BDO)S B)I)O)S)
;=====
obiWBT:      EQU      0           ; WBOOT:   warm boot, reload CCP
obICOS:      EQU      3           ; CONST:   console status
obICOI:      EQU      6           ; CONIN:   console input
obICOO:      EQU      9           ; CONOUT:  console output
obILST:      EQU      12          ; LIST:    printer output
obIPUN:      EQU      15          ; PUNCH:   punch output
obIRDR:      EQU      18          ; READER:  reader input
obIHME:      EQU      21          ; HOME:    move disk head to track 0
obISDR:      EQU      24          ; SELDSK:  select disk drive
obISTR:      EQU      27          ; SETTRK:  set track number
obISSE:      EQU      30          ; SETSEC:  set sector number
obISDM:      EQU      33          ; SETDMA:  set DMA address
obIRDS:      EQU      36          ; READ:    read a sector
obIWRS:      EQU      39          ; WRITE:   write a sector
obILSS:      EQU      42          ; LISTST:  get status of list device
obISCT:      EQU      45          ;:SECTRAN: sector translation table
;===== eoF oBIOS()
;+++++++ variables
oaCEND:      EQU      $           ; end of code
;##### EOF LIBDEF.TXT

```