

Listing CONDRV.ASM

```

;*****;
;*                                C O N D R V                                *;
;*-----*
;* Task      : This program represents a normal Console                    *;
;*            Driver (Keyboard and Display Monitor). It should             *;
;*            serve as a framework for a driver in the form of             *;
;*            an ANSI.SYS driver.                                          *;
;*-----*
;* Author    : MICHAEL TISCHER                                             *;
;* Developed on : 08/04/87                                                 *;
;* Last update : 04/07/95                                                 *;
;*-----*
;* Assembly  : MASM CONDRV;                                               *;
;*            LINK CONDRV;                                               *;
;*            EXE2BIN CONDRV CONDRV.SYS                                   *;
;*            or                                                         *;
;*            TASM CONDRV                                                *;
;*            LINK CONDRV;                                               *;
;*            EXE2BIN CONDRV CONDRV.SYS                                   *;
;*-----*
;* Call      : Copy into root directory, copy the command                *;
;*            DEVICE=CONDRV.SYS into the file CONFIG.SYS                 *;
;*            and then boot the system.                                    *;
;*****;

```

code segment

assume cs:code,ds:code

```

org 0 ;Program has no PSP therefore start
;at offset address 0

```

```

;== Constants =====

cmd_fld equ 2           ;Offset command field in data block
status equ 3           ;Offset status field in data block
end_adr equ 14          ;Offset driver end addr. in data block
num_db equ 18           ;Offset number in data block
b_adr equ 14            ;Offset buffer address in data block

KEY_SZ equ 20           ;Size of keyboard buffer
num_cmd equ 16          ;Subfunctions 0-16 are supported

;== Data =====

;-- Device driver header -----

dw -1,-1               ;Link to next driver
dw 1010100000000011b   ;Driver attribute
dw offset strat         ;Pointer to strategy routine
dw offset intr          ;Pointer to interrupt routine
db "CONDRV "           ;New console driver

;-- Jump table for functions -----

fct_tab dw offset init   ;Function 0: Initialization
dw offset dummy          ;Function 1: Media check
dw offset dummy          ;Function 2: Create BPB
dw offset no_sup         ;Function 3: I/O control read
dw offset read           ;Function 4: Read
dw offset read_b         ;Function 5: Non-destructive read
dw offset dummy          ;Function 6: Input status
dw offset del_in_b       ;Function 7: Delete input buffer

```

```

        dw offset write           ;Function 8: Write
        dw offset write           ;Function 9: Write & verify
        dw offset dummy           ;Function 10: Output status
        dw offset dummy           ;Function 11: Delete output buffer
        dw offset no_sup           ;Function 12: I/O control write
        dw offset dummy           ;Function 13: Open (Ver. 3.0 and up)
        dw offset dummy           ;Function 14: Close
        dw offset dummy           ;Function 15: Changeable medium
        dw offset write           ;Function 16: Output until busy

db_ptr  dw (?), (?)               ;Address of data block passed

key_a   dw 0                     ;Pointer to next character in KEY_SZ
key_e   dw 0                     ;Pointer to last character in KEY_SZ
key_bu  db KEY_SZ dup (?)        ;Internal keyboard buffer

;== Driver routines and functions =====

strat   proc far                 ;Strategy routine

        mov  cs:db_ptr,bx        ;Store address of data block in the
        mov  cs:db_ptr+2,es      ;Variable DB_PTR

        ret                      ;Return to caller

strat   endp

;-----

intr    proc far                 ;Interrupt routine

        push ax                  ;Push registers onto the stack

```

```

push bx
push cx
push dx
push di
push si
push bp
push ds
push es
pushf                                ;Push flag register onto the stack

push cs                             ;Set data segment register
pop  ds                             ;Code and data are identical

les  di,dword ptr db_ptr;Address of data block to ES:DI
mov  bl,es:[di+cmd_fld] ;Get command code
cmp  bl,num_cmd         ;is command code permitted?
jle  bc_ok              ;YES --> bc_ok

mov  ax,8003h           ;Code for "Unknown command"
jmp  short intr_end     ;Return to caller

;-- Command code was O.K. --> Execute command -----

bc_ok:  shl  bl,1         ;Calculate pointer in jump table
        xor  bh,bh       ;Clear BH
        call [fct_tab+bx] ;Call function
        les  di,dword ptr db_ptr;Data block address to ES:DI

;-- End execution of the function -----

intr_end label near
        or   ax,0100h     ;Set ready bit

```

```

        mov     es:[di+status],ax    ;Store everything in the status field

        popf                     ;Restore flag register
        pop     es                 ;Restore other registers
        pop     ds
        pop     bp
        pop     si
        pop     di
        pop     dx
        pop     cx
        pop     bx
        pop     ax

        ret                       ;Return to caller

intr     endp

;-----

dummy    proc near                ;This routine does nothing

        xor     ax,ax             ;Clear busy bit
        ret                       ;Return to caller

dummy    endp

;-----

no_sup   proc near                ;This routine called for all functions
                                       ;which should really not be called
        mov     ax,8003h          ;Error: Command not recognized
        ret                       ;Return to caller

```

no_sup endp

;-----

store_c proc near ;Stores a character in the internal
 ;keyboard buffer
 ;Input: AL = Character
 ; BX = Character position

 mov [bx+key_bu],al ;Store character in internal buffer
 inc bl ;Increment pointer to end
 cmp bl,KEY_SZ ;End of buffer reached?
 jne store_e ;NO --> STORE_E

 xor bl,bl ;New end is the beginning of buffer

store_e: ret ;Return to caller

store_c endp

;-----

read proc near ;Read a certain number of characters
 ;from the keyboard to a buffer

 mov cx,es:[di+num_db] ;Read number of characters
 jcxz read_e ;Test for equality to 0
 les di,es:[di+b_adr] ;Address of character buffer to ES:DI
 cld ;Increment on STOSB
 mov si,key_a ;Pointer to next character in KEY_SZ
 mov bx,key_e ;Pointer to last character in KEY_SZ

```

read_1: cmp  si,bx                ;Other characters in keyboard buffer?
        jne  read_3              ;Yes --> READ_3

read_2: xor   ah,ah               ;Function number for reading is 0
        int  16h                ;Call BIOS keyboard interrupt
        call store_c             ;Store characters in internal buffer
        cmp  al,0                ;Test if extended code
        jne  read_3              ;No --> READ_3

        mov  al,ah               ;Extended code is in AH
        call store_c             ;Store

read_3: mov  al,[si+key_bu]       ;Read character from keyboard buffer
        stosb                    ;Transmit to buffer of calling funct.
        inc  si                  ;Increment pointer to next character
        cmp  si,KEY_SZ           ;End of buffer reached?
        jne  read_4              ;No --> READ_4

        xor  si,si               ;Next character is the first character
                                   ;in the keyboard buffer

read_4: loop read_1              ;Repeat until all characters read
        mov  key_a,si            ;Store position of the next character
                                   ;in the keyboard buffer
        mov  byte ptr key_e,bl   ;Store position of the last character
                                   ;in the keyboard buffer

read_e: xor  ax,ax               ;Everything O.K.
        ret                     ;Return to caller

read    endp

```

;-----

```
read_b proc near                                ;Read the next character from the
                                                ;keyboard but leave in the buffer

    mov  ah,1                                  ;Function number for BIOS interrupt
    int  16h                                  ;Call BIOS keyboard interrupt
    je   read_p1                              ;No character present --> READ_P1

    mov  es:[di+13],al                         ;Store character in data block
    xor  ax,ax                                 ;Everything O.K.
    ret                                       ;Return to caller

read_p1 label near

    mov  ax,0100h                             ;Set busy bit (no character)
    ret                                       ;Return to caller

read_b endp
```

;-----

```
del_in_b proc near                             ;Clear input buffer

    mov  ah,1                                  ;Still characters in the buffer?
    int  16h                                  ;Call BIOS keyboard interrupt
    je   del_e                                ;No character in the buffer --> END

    xor  ah,ah                                 ;Remove character from buffer
    int  16h                                  ;Call BIOS keyboard interrupt
    jmp  short del_in_b                       ;Test for additional characters
```



```

del_e:  xor ax,ax                ;Everything O.K.
        ret                    ;Return to caller

del_in_b endp

;-----

write proc near                  ;Write a specified number of
                                ;characters on the display screen

        mov  cx,es:[di+num_db]  ;Number of characters read
        jcxz write_e           ;Test for equality to 0
        lds  si,es:[di+b_adr]  ;Address of character buffer to DS:SI
        cld                    ;Increment on LODSB

        mov  ah,3              ;Read current display page
        int  16h              ;Call BIOS video interrupt

        mov  ah,14             ;Function number for BIOS interrupt

write_1:  lodsb                ;Read character to be output to AL
        int  10h              ;Call BIOS video interrupt
        loop write_1          ;Repeat until all characters output

write_e:  xor ax,ax            ;Everything O.K.
        ret                    ;Return to caller

write endp

;-----

init      proc near            ;Initialization routine

```

```
mov word ptr es:[di+end_adr],offset init ;Set end address of
mov es:[di+end_adr+2],cs ;the driver
```

```
xor ax,ax ;Everything O.K.
ret ;Return to caller
```

```
init endp
```

```
;=====
```

```
code ends
end
```