

Listing RAMDISK.ASM

```
;*****;
;*               R A M D I S K               *;
;*-----*
;* Task          : This program is a driver for a 160K RAM disk. *;
;*-----*
;* Author        : MICHAEL TISCHER               *;
;* Developed on   : 08/04/87                       *;
;* Last update    : 04/07/95                       *;
;*-----*
;* Assembly      : MASM RAMDISK;                   *;
;*               LINK RAMDISK;                     *;
;*               EXE2BIN RAMDISK RAMDISK.SYS        *;
;*               or                                   *;
;*               TASM RAMDISK                       *;
;*               LINK RAMDISK;                     *;
;*               EXE2BIN RAMDISK RAMDISK.SYS        *;
;*-----*
;* Call          : Copy into root directory, add the command *;
;*               DEVICE=RAMDISK.SYS to the CONFIG.SYS file and *;
;*               reboot the system.                 *;
;*****;
```

code segment

assume cs:code,ds:code,es:code,ss:code

org 0 ;Program has no PSP so begin
;at offset address 0

== Constants =====

```

cmd_fld    equ 2                ;Offset command field in data block
status     equ 3                ;Offset status field in data block
num_dev    equ 13               ;Offset number of supported devices
changed    equ 14               ;Offset medium changed?
end_addr   equ 14               ;Offset driver end addr. in data block
b_addr     equ 14               ;Offset buffer address in data block
num_cmd    equ 16               ;Functions 0-16 are supported
num_db     equ 18               ;Offset number in data block
bpb_addr   equ 18               ;Offset Address of BPB of the media
sector     equ 20               ;Offset first sector number
dev_des    equ 22               ;Offset device description of RAM disk

```

```

;== Data =====

```

```

first_b    equ this byte        ;First byte of the driver

```

```

;-- Device driver header -----

```

```

    dw -1,-1                    ;Link to next driver
    dw 01001000000000000b      ;Driver attribute
    dw offset strat             ;Pointer to strategy routine
    dw offset intr              ;Pointer to interrupt routine
    db 1                        ;Device supported
    db 7 dup (0)                ;These bytes give the name

```

```

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

```

```

fct_tab    dw offset init       ;Function 0: Initialization
            dw offset med_test   ;Function 1: Media test
            dw offset get_bpb    ;Function 2: Created BPB
            dw offset read       ;Function 3: Direct read
            dw offset read       ;Function 4: Read

```

```

        dw offset dummy          ;Function 5: Read, remain in buffer
        dw offset dummy          ;Function 6: Input status
        dw offset dummy          ;Function 7: Erase 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: Clear output buffer
        dw offset write          ;Function 12: Direct write
        dw offset dummy          ;Function 13: Open (Ver. 3.0 and up)
        dw offset dummy          ;Function 14: Close
        dw offset no_rem         ;Function 15: Changeable medium?
        dw offset write          ;Function 16: Output until busy

db_ptr   dw (?), (?)            ;Pass data block address
rd_seg   dw (?)                 ;RD_SEG:0000 = start of RAM disk

bpb_ptr  dw offset bpb, (?)     ;Accept BPB address

boot_sek db 3 dup (0)           ;Jump to the boot routine is
                                   ;normally stored here
        db "MITI 1.0"           ;Name of creator & version number
bpb      dw 512                  ;512 bytes per sector
        db 1                    ;1 sector per cluster
        dw 1                    ;1 reserved sector (boot sector)
        db 1                    ;1 File Allocation Table (FAT)
        dw 64                   ;64 entry maximum in root directory
        dw 320                  ;320 sectors total = 160 K
        db 0FEh                 ;Media descriptor (1 side, 40 tracks
                                   ;of 8 sectors each)
        dw 1                    ;FAT occupies one sector

;-- Boot routine omitted (systems cannot -----

```

```

;-- be booted from a RAM disk)

vol_name db "RAMDISK      "      ;The actual volume name
        db 8                      ;Attribute, defines volume name

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

strat    proc far                ;Strategy routine

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

        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 the 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

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

intr_end label near
push    cs                 ;Set data segment register
pop     ds                 ;(Code and data are identical)

les     di,dword ptr db_ptr;Address of the data block to ES:DI
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

;-----

init     proc near                ;Initialization routine

        ;-- the following code is overwritten -----
        ;-- after the RAM disk is installed

        ;-- Determine device designation of the RAM disk -----

        mov    ah,30h            ;Check DOS Version with function 30(h)
        int    21h              ;of DOS interrupt 21(h)
        cmp    al,3             ;is it Version 3 or higher?
        jb     prinm            ;Yes --> PRINM

        mov     al,es:[di+dev_des] ;Get device designation
        add     al,"A"          ;Convert to letters
        mov     im_ger,al       ;Store in installation message

prinm:    mov    dx,offset initm   ;Address of installation message
        mov     ah,9             ;Output function number for string
        int     21h             ;Call DOS interrupt

        ;-- Calculate address of first byte following the RAM disk ---

```

;-- and set as the driver's end address

```
mov word ptr es:[di+end_adr],offset ramdisk+8000h
mov ax,cs ;RAM disk size =
add ax,2000h ;32K + (2 * 64K) = 160K
mov es:[di+end_adr+2],ax
mov byte ptr es:[di+num_dev],1 ;1 device supported
mov word ptr es:[di+bpb_adr],offset bpb_ptr ;BPB pointer
mov es:[di+bpb_adr+2],ds ;address
```

```
mov ax,cs ;Segment address: start of RAM disk
mov bpb_ptr+2,ds ;Segment address: BPB in BPB pointer
mov dx,offset ramdisk ;Calculate to offset address 0
mov cl,4 ;Divide offset address by 16 and
shr dx,cl ;convert into segment address
add ax,dx ;Add the two segment addresses
mov rd_seg,ax ;and store
```

;-- Create boot sector -----

```
mov es,ax ;Transfer segment address to ES
xor di,di ;Boot sector begins at
;byte 1 of the RAM disk
mov si,offset boot_sek ;Boot sector's address in memory
mov cx,15 ;Only the first 15 words are used
rep movsw ;Copy boot sector into RAM disk
```

;-- Create FAT -----

```
mov di,512 ;FAT begins at byte 512 of the RAM disk
mov al,0FEh ;Write media descriptor into the first
stosb ;byte of the FAT
```

```

mov ax,0FFFFH      ;Store code for bytes 2 and 3 of FAT
stosw              ;in FAT
mov cx,236          ;Remaining 236 words occupied by FAT
inc ax              ;Set AX to 0
rep stosw           ;Set all FAT entries to unoccupied

;-- Create root directory with volume name -----

mov di,1024          ;Root directory starts in sector 3
mov si,offset vol_name ;Volume name address in memory
mov cx,6             ;Volume name is 6 words long
rep movsw            ;Copy volume name into RAM disk

mov cx,1017          ;Fill the rest of the directories in
xor ax,ax            ;sectors 2, 3, 4 and 5 with zeros
rep stosw

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

```

```
init      endp
```

```
;-----
```

```
dummy      proc near                ;This routine does nothing

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

```

```
dummy      endp
```

```
;-----
```



```

med_test proc near                                ;RAM disk medium cannot be changed

    mov  byte ptr es:[di+changed],1
    xor  ax,ax                                    ;Clear busy bit
    ret                                       ;Return to caller

```

```
med_test endp
```

```
;-----
```

```

get_bpb  proc near                                ;Pass address of BPB to DOS

    mov  word ptr es:[di+bpb_adr],offset bpb
    mov  word ptr es:[di+bpb_adr+2],ds

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

```

```
get_bpb  endp
```

```
;-----
```

```

no_rem   proc near                                ;RAM disk medium cannot be changed
    mov  ax,20                                    ;Set busy bit
    ret                                       ;Return to caller

```

```
no_rem   endp
```

```
;-----
```

```
write proc near
```

```

        xor    bp,bp                ;Send DOS --> RAM disk
        jmp    short move           ;Copy data

write endp

;-----

read    proc near

        mov    bp,1                ;Send RAM disk --> DOS

read    endp

;-- MOVE: Move a certain number of sectors between RAM disk and DOS
;-- Input      : BP = 0 : Transmit from DOS to RAM disk (Write)
;--            BP = 1 : Transmit from RAM disk to DOS (Read)
;-- Output     : none
;-- Registers  : AX, BX, CX, DX, SI, DI, ES, DS and FLAGS are affected
;-- Info       : Information required (number, first sector)
;--            is taken from the data block passed by DOS

move    proc near

        mov    bx,es:[di+num_db]    ;Number of sectors read
        mov    dx,es:[di+Sector]    ;Number of first sector
        les    di,es:[di+b_adr]     ;Address of buffer to ES:DI

move_1:  or     bx,bx                ;More sectors to read?
        je     move_e               ;No more sectors --> END
        mov    ax,dx                ;Sector number to AX
        mov    cl,5                  ;Calculate number of paragraphs

```

shl ax,cl	;Multiply segment units by 32,
add ax,cs:rd_seg	;add to segment start of RAM disk
mov ds,ax	;Transmit to DS
xor si,si	;Offset address is 0
mov ax,bx	;Number of sectors to be read to AX
cmp ax,128	;More than 128 sectors to read?
jbe move_2	;No --> Read all sectors
mov ax,128	;Yes --> Read 128 sectors (64K)
move_2: sub bx,ax	;Subtract number of sectors read
add dx,ax	;Add to sectors to be read next
mov ch,al	;Number sect. to be read * 256 words
xor cl,cl	;Set word counter low byte to 0
or bp,bp	;Should more be read?
jne move_3	;No --> MOVE_3
mov ax,es	;Store ES in AX
push ds	;Store DS on the stack
pop es	;Read ES
mov ds,ax	;Reverse ES and DS
xchg si,di	;Reverse SI and DI
move_3: rep movsw	;Copy data into DOS buffer
or bp,bp	;Read?
jne move_1	;No --> Maybe other sectors to copy
mov ax,es	;Store ES in AX
push ds	;Store DS on the stack
pop es	;Read ES
mov ds,ax	;Reverse ES and DS
xchg si,di	;Reverse SI and DI
jmp short move_1	;Additional sectors to copy
move_e: xor ax,ax	;Everything O.K.
ret	;Return to caller

```

move      endp

;-- RAM disk starts here -----

    if ($-fst_b) mod 16          ;Must start on a memory address
        org ($-fst_b) + 16 - (($-fst_b) mod 16) ; divisible by 16
    endif
ramdisk   equ this byte

initm     db "**** 160K RAMDISK DEVICE "
im_ger     db "?"
          db ": installed (c) 1987 by MICHAEL TISCHER$",13,10,10

;-----

code      ends
          end

```