

```

; *****
;
; UNIXFDFS.ASM
; -----
;
; RETRO UNIX 8086 (Retro Unix == Turkish Rational Unix)
; Operating System Project (v0.1) by ERDOGAN TAN (Beginning: 11/07/2012)
; 1.44 MB Floppy Disk
; Bootable Unix (RUFFS) File System Installation/Formatting Code
;
; UNIXFDFS.ASM -> Last Modification: 21/04/2014
;
; Derivation from UNIX Operating System (v1.0 for PDP-11)
; (Original) Source Code by Ken Thompson (1971-1972)
; <Bell Laboratories (17/3/1972)>
; <Preliminary Release of UNIX Implementation Document>
;
; *****

; 21/04/2014 (tty8=COM1, tty9=COM2)
; 22/12/2013
; 09/07/2013

RUFFS_INSTL      SEGMENT PUBLIC 'CODE'
                  assume cs:RUFFS_INSTL,ds:RUFFS_INSTL,es:RUFFS_INSTL,ss:RUFFS_INSTL

ruffs_fd_format  proc near
                  ; 28/10/2012
                  ; 19/9/2012
                  ; 14/8/2012
                  ; 13/8/2012
                  ; 12/8/2012

                  org 100h

INSTALL:

; - - - - -
; see if drive specified
; - - - - -

                mov si, offset 80h                ; PSP command tail
                mov cl, byte ptr [SI]
                or cl, cl
                jz short ruffs_fd_format_7         ; jump if zero

ruffs_fd_format_1:
                inc si
                mov al, byte ptr [SI]
                cmp al, ' '                        ; is it SPACE ?
                jne short ruffs_fd_format_2

                dec cl
                jne short ruffs_fd_format_1
                jmp short ruffs_fd_format_7

ruffs_fd_format_2:
                cmp al, "f"
                jne short ruffs_fd_format_3
                inc si
                mov al, byte ptr [SI]
                cmp al, "d"
                jne short ruffs_fd_format_7
                inc si
                mov ax, word ptr [SI]
                cmp al, '0'
                jnb short ruffs_fd_format_7
                cmp al, '1'
                ja short ruffs_fd_format_7
                cmp ah, 20h
                ja short ruffs_fd_format_7
                mov byte ptr [RUFFS_DRIVE], al
                sub al, '0'
                jmp short ruffs_fd_format_5

```

```

rufs_fd_format_3:
    cmp al, 'A'
    jb short rufs_fd_format_7
    cmp al, 'B'                                ; A - Z
    jna short rufs_fd_format_4
    cmp al, 'a'                                ; a - z
    jb short rufs_fd_format_7
    cmp al, 'b'
    ja short rufs_fd_format_7

    sub al, 'a'-'A'                            ; to upper case

; - - - - -
; Write message
; - - - - -

rufs_fd_format_4:
    mov byte ptr [RUFSDRIVE], al
    sub al, 'A'                                ; make it zero based

rufs_fd_format_5:
    mov dl, al
    mov byte ptr [bsDriveNumber], dl
    mov ah, 08h
    int 13h                                    ; return disk parameters
    push cs
    pop es                                    ; restore es
    jc rufs_fd_format_17

    cmp bl, 04h                                ; Drive Type
    jb rufs_fd_format_17

    mov si, offset Msg_DoYouWantToFormat
    call PRINT_MSG

rufs_fd_format_6:
    xor ax, ax
    int 16h                                    ; wait for keyboard command
    cmp al, 'C'-40h
    je short rufs_fd_format_8
    cmp al, 27
    je short rufs_fd_format_8
    and al, 0DFh
    cmp al, 'Y'                                ; Yes?
    je short rufs_fd_format_10                 ; write
    cmp al, 'N'                                ; No?
    je short rufs_fd_format_9                 ; no write (exit)

rufs_fd_format_7:
    mov si, offset UNIX_Welcome
    call PRINT_MSG

rufs_fd_format_8:
    mov si, offset UNIX_CRLF
    call PRINT_MSG

    int 20h

infinive_loop: jmp short infinive_loop

rufs_fd_format_9:
    mov si, offset msg_NO
    call PRINT_MSG

    jmp short rufs_fd_format_8

```

```

;- - - - -
; get drive parameters
;- - - - -

rufs_fd_format_10:
    mov si, offset msg_YES
    call PRINT_MSG

rufs_fd_format_11:
    xor ax, ax
    int 1Ah                                ; get time of day
    mov word ptr [bsVolumeSerial], dx
    mov word ptr [bsVolumeSerial]+2, cx    ; set unique volume ID

rufs_fd_format_12:
    mov si, offset Msg_installing_file_system
    call PRINT_MSG

    mov dl, byte ptr [bsDriveNumber] ; 14/8/2012

    call unix_fs_install
    jnc short rufs_fd_format_14

    mov ah, byte ptr [Error]

rufs_fd_format_13: ; loc_rw_error
    mov al, ah
    push ax
    mov si, offset Msg_Disk_RW_Error
    call PRINT_MSG
    pop ax
    call proc_hex
    mov word ptr [Str_Err], ax
    mov si, Offset Msg_Error_Number
    call PRINT_MSG

    int 20h

rufs_fd_format_14:
    mov si, offset Msg_OK
    call PRINT_MSG

rufs_fd_format_15:
    mov si, offset Msg_writing_boot_sector
    call PRINT_MSG

    mov byte ptr [RetryCount], 4

rufs_fd_format_16:
    mov ax, 0301h                                ; write to disk

    mov bx, offset Start                        ; location of boot code

    mov cx, 1                                ; cylinder = 0
                                           ; sector = 1
    mov dh, 0                                ; head = 0
    mov dl, byte ptr [bsDriveNumber]

    int 13h
    jnc short rufs_fd_format_17
    dec byte ptr [RetryCount]
    jnz short rufs_fd_format_16

    jmp short rufs_fd_format_13

rufs_fd_format_17:
    mov si, offset Msg_OK
    call PRINT_MSG

    ;int 20h
    jmp rufs_fd_format_8

rufs_fd_format endp

```

```

PRINT_MSG      proc near
                mov     BX,07h
                mov     AH,0Eh

PRINT_MSG_LOOP:
                lodsb                    ; Load byte at DS:SI to AL
                and     AL,AL
                jz      short PRINT_MSG_OK

                int     10h                ; BIOS Service func ( ah ) = 0Eh
                                           ; Write char as TTY
                                           ; ↑AL-char BH-page BL-color
                jmp     short PRINT_MSG_LOOP

PRINT_MSG_OK:
                retn

PRINT_MSG      endp

proc_hex        proc     near

                db 0D4h,10h                ; Undocumented inst. AAM
                                           ; AH = AL / 10h
                                           ; AL = AL MOD 10h
                or  AX,'00'                ; Make it ZERO (ASCII) based

                xchg  AH,AL

; 1999
                cmp  AL,'9'
                jna  pass_cc_al
                add  AL,7

pass_cc_al:
                cmp  AH,'9'
                jna  pass_cc_ah
                add  AH,7

pass_cc_ah:

; 1998
                retn

proc_hex        endp

;;;;
include         uninstall.asm
include         unixproc.asm
;;;;

; - - - - -
;  messages
; - - - - -

UNIX_Welcome:
                db 0Dh, 0Ah
                db 'RETRO UNIX 1.44 MB Floppy Disk (RUFFS) Format Utility'
                db 0Dh, 0Ah
                db 'by Erdogan TAN [21/04/2014] '
                db 0Dh,0Ah
                db 0Dh,0Ah
                db 'Usage: unixfdfs [Drive] '
                db 0Dh,0Ah
                db 0Dh,0Ah
                db "Drive names:"
                db 0Dh,0Ah
                db 0Dh,0Ah
                db "fd0      (Floppy Disk 1)", 0Dh, 0Ah
                db "fd1      (Floppy Disk 2)", 0Dh, 0Ah
                db "...", 0Dh, 0Ah
                db "A:      (Floppy Disk 1)", 0Dh, 0Ah
                db "B:      (Floppy Disk 2)", 0Dh, 0Ah
                db 0Dh, 0Ah
                db 0

```

```

Msg_DoYouWantToFormat:
    db 07h
    db 0Dh, 0Ah
    db 'WARNING!'
    db 0Dh, 0Ah
    db 'All data on the drive will be erased.'
    db 0Dh, 0Ah
    db 0Dh, 0Ah
    db 'Do you want to format drive '

RUFFS_DRIVE:
    db 'A: (Yes/No)? ', 0

Msg_Installing_File_System:
    db 0Dh, 0Ah
    db "Installing UNIX v1 File Sytem...", 0

Msg_Writing_Boot_Sector:
    db 0Dh, 0Ah
    db "Writing UNIX boot sector...", 0

Cursor_Pos:    dw 0

Msg_Volume_Name:
    db 0Dh, 0Ah
    db "Volume Name: ", 0

Msg_OK:
    db ' OK.', 0

msg_YES:
    db ' YES'
    db 0
msg_NO:
    db ' NO'
    db 0

; 12/8/2012
msg_disk_rw_error:
    db 0Dh, 0Ah
    db 'Disk r/w error!'
    db 0

msg_error_Number:
    db 0Dh, 0Ah
    db 'Error No:'
str_err:
    dw 3030h
    db 'h'
UNIX_CRLF:
    db 0Dh, 0Ah, 0

Error_Code:    db 0

RetryCount:    dw 0

str_volume_name: db 15 dup (0)

    db 'Turkish Rational UNIX', 0
    db 'RETRO UNIX 8086 by Erdogan TAN', 0
    db '11/07/2012', 0, '21/04/2014', 0

    db 1 dup (?)
                                ; trick for assembler
                                ; to keep 'start'
                                ; at 7C00h

```

```

BF_BUFFER equ 700h
BF_INODE equ 600h
inode_flg equ 600h
inode_nlks equ 602h
inode_uid equ 603h
inode_size equ 604h
inode_dskp equ 606h
inode_ctim equ 616h
inode_mtim equ 61Ah
inode_reserved equ 61Eh

boot_file_load_address equ 7E00h
boot_file_segment equ 7E0h

        org 7C00h

;+++++
;±
;±          PROCEDURE unixbootsector
;±
;+++++

unixbootsector proc    near

Start:
        jmp     short @f

; RETRO UNIX 8086 FS v0.1 BootSector Identification (Data) Block
; 29-10-2012 RUFS 1.44MB FD Boot Sector

bsFSystemID:    db 'RUFS'
bsVolumeSerial: dd 0
                db 'fd'
bsDriveNumber:  db 0
bsReserved:     db 0 ; 512 bytes per sector
bsSecPerTrack:  db 18
bsHeads:        db 2
bsTracks:       dw 80
bs_BF_I_number: dw 0
                db '@'

@@:
        mov ax, cs
        mov ds, ax
        mov es, ax

        cli
        mov ss, ax
        mov sp, 0FFFEh
        sti

        mov ax, word ptr [bs_BF_I_number]

        or ax, ax
        jz loc_no_bootable_disk

        mov byte ptr [bsDriveNumber], DL ; from INT 19h

        ;;call load_boot_file
        ;;jc short loc_unix_bl_error
load_boot_file:
        ;; 22/12/2013
        ;; 28/10/2012
        ;; 20/10/2012
        ;
        ; RETRO UNIX v1 FS
        ; Boot sector version
        ;
        ; loads boot file
        ;
        ; ax = i-number

```

```

load_bf_1:
i_get:
    ;; 22/12/2013
    ; 20/10/2010 (i_i)
    ; 14/10/2012
    ; boot sector version of "iget" procedure
    ; Derived from (original) UNIX v1 source code
    ; PRELIMINARY release of Unix Implementation Document,
    ; 20/6/1972
    ; input -> AX = inode number
    ; RETRO UNIX v1 FS
    ; boot sector version
    ; return => if cf=1 error number in [Error]

    ;;cmp ax, word ptr [i_i] ; AX (R1) = i-number of current file
    ;;je short i_get_3

    ;; mov di, ax ; i-number
    add ax, 47 ; add 47 to inode number
    push ax ;
    shr ax, 1 ; divide by 16
    shr ax, 1
    shr ax, 1
    shr ax, 1
    ; ax contains block number of block in which
    ; inode exists
    call dsk_rd
    pop dx ;
    ;;jc short i_get_3 ; Error code in AH
    jc loc_unix_bl_error

    ;;mov word ptr [i_i], di

i_get_1:
    and dx, 0Fh ; (i+47) mod 16
    shl dx, 1
    shl dx, 1
    shl dx, 1
    shl dx, 1
    shl dx, 1
    ; DX = 32 * ((i+47) mod 16)
    ; DX points to first word in i-node i.
    mov di, BF_INODE
    ; inode is address of first word of current inode
    mov cx, 16 ;

    mov si, bx ; offset Buffer

    add si, dx

i_get_2:
    ; copy new i-node into inode area of (core) memory
    rep movsw

;;i_get_3:
    ;;retn

lbf_2:
    ;; 22/12/2013

    mov bx, inode_flg

    test word ptr [bx], 10h ; executable file attribute bit
    ;;jz short load_bf_stc
    jz loc_unix_bl_error

    mov bx, inode_size ; offset

    ;; 22/12/2013
    ;;cmp word ptr [bx], 0
    ;;jna short load_bf_stc
    ;;jna short loc_unix_bl_error
    mov ax, word ptr [bx]
    and ax, ax
    jz loc_unix_bl_error

    mov word ptr [b_base], boot_file_load_address

    ;;xor ax, ax
    ;;mov word ptr [b_off], ax ; u_off is file offset

```

```

;; 22/12/2013
xor dx, dx
mov word ptr [b_off], dx ; u_off is file offset

;mov bx, inode_size
;;mov ax, word ptr [bx]
mov word ptr [b_count], ax

;;mov ax, word ptr [i_i]
;;call read_i
;;jc short load_bf_retn

read_i:
;; 22/12/2013
; 28/10/2012
; 14/10/2012
; Boot sector version of "readi" procedure
; Derived from (original) UNIX v1 source code
; PRELIMINARY release of Unix Implementation Document,
; 20/6/1972
;;AX (R1) = i-number
; RETRO UNIX v1 FS
; Boot sector version
;
; read from an i-node
;
;;xor dx, dx ; 0
mov word ptr [b_nread], dx ; accumulated number of bytes transmitted
;;cmp word ptr [b_count], dx ; is number of byte to read greater than 0
;;jna short read_i_retn

read_i_1:
; AX = I-Number
;;push ax
;;call i_get ; get i-node into i-node section of core
mov bx, inode_size
mov dx, word ptr [bx] ; file size in bytes in r2 (DX)
sub dx, word ptr [b_off] ; subtract file offset
;;jna short read_i_3
; jna read_i_retn ;; 22/12/2013
cmp dx, word ptr [b_count]
; are enough bytes left in file to carry out read
jnb short read_i_2
mov word ptr [b_count], dx

read_i_2:
;;call m_get ; returns physical block number of block in file
;; ; where offset points

m_get:
;; 22/12/2013
; 05/03/2013
; 03/03/2013
; 28/10/2012
; 20/10/2012
; Boot sector version of "mget" procedure
; Derived from (original) UNIX v1 source code
; PRELIMINARY release of Unix Implementation Document,
; 20/6/1972
;

m_get_0:
mov bl, byte ptr [b_off]+1
xor bh, bh
mov si, inode_flg
test word ptr [si], 4096 ; 1000h
; is this a large or small file
jnz short m_get_1 ; large file

test bl, 0F0h ; !0Fh ; error if BX (R2) >= 16
jnz short m_get_5

and bl, 0Eh ; clear all bits but bits 1,2,3
mov ax, word ptr inode_dskp[bx] ; AX = R1, physical block number

jmp short m_get_3

```



```

m_get_1:      ; large file
               ; 05/03/2013
               ; 03/03/2013
               ;mov ax, bx
               ;mov cx, 256
               ;xor dx, dx
               ;div cx
               ;and bx, 1FEh ; zero all bit but 1,2,3,4,5,6,7,8
               ; gives offset in indirect block
               ;push bx
               ;mov bx, ax ; calculate offset in i-node for pointer
               ; to proper indirect block
               ;and bx, 0Eh
               ;mov ax, word ptr inode_dskp[bx]
               and bl, 0FEh
               ;push bx
               ; mov di, bx
               mov si, bx ; 22/12/2013
               mov bx, inode_dskp
               mov ax, word ptr [BX]
               or ax, ax
               ;jz short m_get_4
               jz short loc_unix_bl_error ; 22/12/2013

m_get_2:
               call dsk_rd ; read indirect block
               ;jc short m_get_5
               jc short loc_unix_bl_error ; 22/12/2013
               ;pop ax
               ;add bx, ax ; R5, first word of indirect block
               ;add bx, di
               add bx, si ; 22/12/2013
               mov ax, word ptr [BX] ; put physical block no of block
               ; in file sought in R1 (AX)

m_get_3: ; 2
               ; ax = R1, block number of new block
               ;cmp ax, 1
               ;retn
               or ax, ax
               jz short loc_unix_bl_error ; 22/12/2013

m_get_4:
               ;stc

m_get_5:
               ;pop bx
               ;retn
               ;jc short loc_unix_bl_error ; 22/12/2013

               ; AX = Physical block number
               call dsk_rd ; read in block, BX points to 1st word of data in
               ; buffer
               ;jc short read_i_3
               ;jc short_read_i_retn
               jc short loc_unix_bl_error ; 22/12/2013

readi_sioreg:
               mov si, word ptr [b_off] ; R2
               mov cx, si ; cx = R3, si = R2
               or cx, 0FE00h ; set bits 9...15 of file offset in R3
               and si, 1FFh ; calculate file offset mod 512
               add si, bx ; offset Buffer ; si now points to 1st byte in buffer
               ; where data is to be placed
               mov di, word ptr [b_base] ; R1
               neg cx ; 512 - file offset(mod512) in R3 (cx)
               cmp cx, word ptr [b_count]
               jna short @f ; 2f

               mov cx, word ptr [b_count]

@@:
               add word ptr [b_nread], cx ; r3 + number of bytes
               ; xmitted during write is put into
               ; u_nread
               sub word ptr [b_count], cx
               add word ptr [b_base], cx ; points to 1st of remaining
               ; data bytes
               add word ptr [b_off], cx ; new file offset = number
               ; of bytes done + old file offset

```

```

; end of readi_sioreg
    ; DI = file (user data) offset
    ; SI = sector (I/O) buffer offset
    ; CX = byte count

    rep movsb
    ;pop ax

    cmp word ptr [b_count], 0
    ja read_i_1

read_i_retn: ;; 22/12/2013
    ;retn

;;read_i_3:
    ;; pop ax ; i-number

;;read_i_retn:
    ;; retn

    ;;; jc short load_bf_retn

    mov cx, word ptr [b_nread]
    mov bx, inode_size

    ;; cmp cx, word ptr [bx]
    ;; retn

;;load_bf_stc:
    ;; stc

;;load_bf_retn:
    ;; retn

    ;;; jc short loc_unix_bl_error

loc_launch_bootfile:
    mov si, offset msg_CRLF
    call print_string

    mov ax, boot_file_segment ; 7E0h
    mov ds, ax
    mov es, ax
    cli
    mov ss, ax
    ;mov sp, 0FFFEh
    sti

    mov dl, byte ptr [bsDriveNumber]

    ; MASM.EXE don't accept
    ; jmp 07E0h:0000h
    ; for OP Code: EA0000E007
    db 0EAh
    dw 0
    dw 07E0h

NeverComeHere: jmp short NeverComeHere

loc_no_bootable_disk:
    mov si, offset msg_press_any_key
    call print_string
    xor ax, ax
    int 16h
    int 19h

loc_unix_bl_error:
    mov si, offset unix_bfl_error_msg
    call print_string
    jmp short NeverComeHere

unixbootsector endp

```

```

dsk_rd  proc near
    ; 22/12/2013
    ; 28/10/2012 (bf_buff_s)
    ; 20/10/2012
    ; 14/10/2012
    ; fd boot sector version of "dskrd" procedure
    ; Derived from (original) UNIX v1 source code
    ; PRELIMINARY release of Unix Implementation Document,
    ; 20/6/1972
    ; RETRO UNIX v1 FS
    ; floppy disk boot sector version
    ; return => if cf=1 error number in [Error]

    ; ax = sector/block number

    ;cmp ax, word ptr [bf_buff_s] ; buffer sector
    ;je short dsk_rd_3

    ;mov si, ax

    mov bx, BF_BUFFER ; offset Buffer

    xor ch, ch
    mov cl, 4 ; Retry count

dsk_rd_1:
    push cx
    mov dx, 18          ; Sectors per track, 18
    div dl
    mov cl, ah          ; Sector (zero based)
    inc cl              ; To make it 1 based
    shr al, 1           ; Convert Track to Cylinder
    adc dh, 0           ; Heads (0 or 1)

    mov dl, byte ptr [bsDriveNumber] ; Physical drive number
    mov ch, al

    mov ah, 2           ; 2=read
    mov al, 01h
    int 13h             ; BIOS Service func ( ah ) = 2
                        ; Read disk sectors
                        ; BIOS Service func ( ah ) = 3
                        ; Write disk sectors
                        ; ↑AL-sec num CH-cyl CL-sec
                        ; DH-head DL-drive ES:BX-buffer
                        ; ↓CF-flag AH-stat AL-sec read

    pop cx
    jnc short dsk_rd_2
    loop dsk_rd_1

dsk_rd_2:
    ;mov word ptr [bf_buff_s], si

dsk_rd_3:
    retn

dsk_rd  endp

print_string  proc near

    mov     BX, 07
    mov     AH, 0Eh

loc_print:
    lodsb                    ; Load byte at DS:SI to AL
    and     AL,AL
    je      short loc_return  ; If AL = 00h then return

    int     10h              ; BIOS Service func ( ah ) = 0Eh
                        ; Write char as TTY
                        ; ↑AL-char BH-page BL-color

    jmp     short loc_print

loc_return:
    retn

print_string  endp

unix_bfl_error_msg:
    db 07h, "UNIX boot error!"

```

```
msg_CRLF:          db 0Dh, 0Ah, 0

msg_press_any_key:
    db 07h
    db "Not a bootable floppy disk!"
    db 0Dh,0Ah

b_base: dw 0
b_off: dw 0
b_count: dw 0
b_nread: dw 0

;bf_buff_s: dw 0

;;i_i:             db 2 dup (0)

                    org 7DFEh

bsBootSign:        dw 0AA55h

RUFS_INSTL          ends

                    end  INSTALL
```

```
; UINSTALL.ASM
; -----
; RETRO UNIX v0.1 'fd0' formatting procedures
; Last Update: 09/07/2013
; (new /dev directory format
; according to Retro UNIX 8086 v1 kernel)
; 21/04/2014 (tty8, tty9)
; 05/03/2013 (ALIGN)
; 31/10/2012, 16/12/2012 (unixproc.asm -> sioreg)
; ERDOGAN TAN [ 14-15-16-21-27/7/2012, 4-5-12-13-14-15-21/8/2012 ]
; These procedures will be located in UNIXFDFS.ASM file
; when they are completed.
; (NOTE: only for (R)UFS initialization of FD0 1.44MB floppy disk

SIZE_FREE_MAP equ 360
SIZE_INODE_MAP equ 32

DISK_SIZE equ 2880 ; in blocks
INODE_COUNT equ SIZE_INODE_MAP * 8
INODE_LIST_BLOCKS equ (INODE_COUNT / 16)

ROOT_DIR_INODE equ 41

SIZE_Reserved1 equ 512 - (2+SIZE_FREE_MAP+2+SIZE_INODE_MAP)

SuperBlock struc

sb_FreeMapSize      dw ?
sb_FreeMap          db SIZE_FREE_MAP dup(?)
sb_InodeMapSize     dw ?
sb_InodeMap         db SIZE_INODE_MAP dup(?)
sb_Reserved1       db SIZE_Reserved1 dup(?)
sb_Reserved2       db 512 dup(?)

SuperBlock ends

; UNIX v1 I-node Flags:
; 1000000000000000b i-node is allocated (8000h)
; 0100000000000000b directory (4000h)
; 0010000000000000b file has been modified (2000h)
; 0001000000000000b large file (1000h)
; 0000000000010000b set user id on execution (20h)
; 0000000000001000b executable (10h)
; 0000000000000100b read, owner (8)
; 0000000000000010b write, owner (4)
; 0000000000000001b read, non-owner (2)
; 0000000000000000b write, non-owner (1)

unix_fs_install proc near
; 8086 code by Erdogan Tan
; 31/10/2012
; 21/08/2012
; 15/08/2012
; 14/08/2012
; 13/08/2012
; 05/08/2012
; 04/08/2012
; Derived from (original) UNIX v1 source code
; PRELIMINARY release of Unix Implementation Document,
; 20/6/1972
; RETRO UNIX v1 FS
; initialization/format version
; NOTE:
; The "cold" unix (u0, PDP-11) code is modified for fd0
; -> 1.44 MB floppy disk (Retro UNIX v1, 8086) fs

mov byte ptr [buff_d], dl ; 14/8/2012, drive number

mov word ptr [systm.sb_FreemapSize], SIZE_FREE_MAP ; 360
mov word ptr [systm.sb_InodeMapSize], SIZE_INODE_MAP ; 32
mov ax, DISK_SIZE ; 2880 blocks/sectors

uinstall_1:
; set bit AX/R1 in free storage map in core/memory
dec ax ; R1
call free

cmp ax, INODE_LIST_BLOCKS + 4 ; 15/8/2012
ja short uinstall_1
```

```
uinstall_2:
; zero i-list
dec ax
; AX (R1) = Block number
call clear
jc short uinstall_10 ; rw_error

and ax, ax
jnz short uinstall_2

uinstall_3:
; initialize inodes for special files (1 to 40)
mov bx, 40 ; BX = R1, 41 = root directory i-number
uinstall_4:
call iget
jc short uinstall_10 ; rw_error

mov word ptr [i_flg], 800Fh ; 100000000000111b
mov byte ptr [i_nlks], 1
mov byte ptr [i_uid], 0
call setimod
dec bx
jnz short uinstall_4

uinstall_5:
;push di
;push si
mov si, offset idata ; base address of assembled dirs
mov di, offset dirs ; directory data for assembled dirs
mov bx, 41
uinstall_6:
call imap
xchg bx,dx ; 13/8/2012
; 21/8/2012 (AX -> AL, word ptr [BX] -> byte ptr [BX])
or byte ptr [BX], al ; BX/DX = R2, ax = mq
; set the bit to indicate the i-node
; is not available/free
xchg bx, dx ; 13/8/2012
call iget
;jnc short uinstall_7
jc short uinstall_10 ; rw_error

@@:
;pop si
;pop di
;jmp short uinstall_10 ; rw_error

uinstall_7:
; SI, DI registers are not modified
; in imap, iget, setimod and writei procedures
lodsw
mov word ptr [i_flg], ax
lodsb
mov byte ptr [i_nlks], al
lodsb
mov byte ptr [i_uid], al
call setimod
lodsw
mov word ptr [u_count], ax

add si, 26 ; now, si points 1st word of next inode

mov word ptr [u_base], di
add di, ax

mov word ptr [u_fofp], offset u_off ; 31/10/2012

mov word ptr [u_off], 0

call writei
;jc short @b
jc short uinstall_10 ; rw_error

cmp bx, 46
jnb short uinstall_8

inc bx
jmp short uinstall_6
```

```
uinstall_8:
    ;pop si
    ;pop di

uinstall_9:
    call sync ; write modified super block and buffer to disk
    ;jc short rw_error

uinstall_10:
    retn

unix_fs_install endp

sync    proc near
    ; 12/8/2012
    ; updates super block and the last i-node on disk
    ; if modified
    ; e.g. smod = 1, imod = 1, buffer_m = 1
    ;
    ; RETRO UNIX v1 FS
    ; initialization/format version

    xor bx, bx ; mov bx, 0
    call iget
    jc short sync_2

    xor ax, ax
    cmp byte ptr [smod], al ; 0
    jna short sync_3

sync_1:
    mov byte ptr [smod], al ; 0

    mov cx, 256
    mov si, offset System
    mov di, offset Buffer
    rep movsw

    inc al

    mov word ptr [buff_s], ax ; 1 ; superblock sector number
    mov byte ptr [buff_w], al

    call poke

sync_2:
    mov ax, word ptr [Error]

sync_3:
    retn

sync    endp

align 2

buff_d: db 0
buff_s: dw 0FFFFh ; Buffer sector
buff_m: db 0 ; buffer daha changed/modified (dirty) flag
buff_w: db 0 ; read/write flag (write=1, read=0)

align 16

system: ; superblock
db 512 dup(0)
```

```
; 5/8/2012
; 14/7/2012
dirs:
root_dir: ; root directory
    dw 41
    db "...", 0,0,0,0,0,0
    dw 41
    db ".", 0,0,0,0,0,0,0
    dw 42
    db "dev",0,0,0,0,0
    dw 43
    db "bin",0,0,0,0,0
    dw 44
    db "etc",0,0,0,0,0
    dw 45
    db "usr",0,0,0,0,0
    dw 46
    db "tmp",0,0,0,0,0

size_root_dir equ $ - offset root_dir

dev_dir: ; device directory
    dw 41
    db "...", 0,0,0,0,0,0
    dw 42
    db ".", 0,0,0,0,0,0,0
    dw 1
    db "tty",0,0,0,0,0
    dw 2
    db "mem",0,0,0,0,0
    dw 3
    db "fd0",0,0,0,0,0
    dw 4
    db "fd1",0,0,0,0,0
    dw 5
    db "hd0",0,0,0,0,0
    dw 6
    db "hd1",0,0,0,0,0
    dw 7
    db "hd2",0,0,0,0,0
    dw 8
    db "hd3",0,0,0,0,0
    dw 9
    db "lpr",0,0,0,0,0
    dw 10
    db "tty0",0,0,0,0
    dw 11
    db "tty1",0,0,0,0
    dw 12
    db "tty2",0,0,0,0
    dw 13
    db "tty3",0,0,0,0
    dw 14
    db "tty4",0,0,0,0
    dw 15
    db "tty5",0,0,0,0
    dw 16
    db "tty6",0,0,0,0
    dw 17
    db "tty7",0,0,0,0
    dw 18
    db "COM1",0,0,0,0 ; 09/07/2013
    dw 19
    db "COM2",0,0,0,0 ; 09/07/2013
    dw 18
    db "tty8",0,0,0,0 ; 21/04/2014
    dw 19
    db "tty9",0,0,0,0 ; 21/04/2014

size_dev_dir equ $ - offset dev_dir

bin_dir: ; binary directory
    dw 41
    db "...", 0,0,0,0,0,0
    dw 43
    db ".", 0,0,0,0,0,0,0
```



```
size_bin_dir equ $ - offset bin_dir

etc_dir: ; etcetra directory
        dw 41
        db "..", 0,0,0,0,0,0
        dw 44
        db ".", 0,0,0,0,0,0,0

size_etc_dir equ $ - offset etc_dir

usr_dir: ; user directory
        dw 41
        db "..", 0,0,0,0,0,0
        dw 45
        db ".", 0,0,0,0,0,0,0

size_usr_dir equ $ - offset usr_dir

tmp_dir: ; temporary directory
        dw 41
        db "..", 0,0,0,0,0,0
        dw 46
        db ".", 0,0,0,0,0,0,0

size_tmp_dir equ $ - offset tmp_dir

align 2

;dw 0

; 31/10/2012
u_off: dw 0

; 12/08/2012
u_count: dw 0
u_base: dw 0
u_fofp: dw 0
u_nread: dw 0

; 17/08/2012
; 05/08/2012
; 14/07/2012
inode:
i_flg: dw 800Fh ; special (device) files flags
i_nlks: db 1 ; Number of links
i_uid: db 0 ; user id
i_size: dw 0 ; file size
i_dskp: dw 8 dup(0) ; direct or indirect blocks
i_ctim: dd 0 ; creation time
i_mtim: dd 0 ; last modification time
i_reserved: dw 0 ; reserved (not in use)

; 05/08/2012
; 14/07/2012
idata:
inodes:

root_inode: ; 41
        dw 0C00Eh ; Flags (1100000000001110b)
        db 7 ; number of links
        db 0 ; user ID (0 = root)
        dw size_root_dir ; initial size = 70 bytes
        dw 8 dup (0) ; indirect or contents blocks
        dd 0 ; creation date & time
        dd 0 ; modification date & time
        dw 0 ; unused

dev_inode: ; 42
        dw 0C00Eh ; Flags (1100000000001110b)
        db 2 ; number of links
        db 0 ; user ID (0 = root)
        dw size_dev_dir ; 200
        dw 8 dup (0) ; indirect or contents blocks
        dd 0 ; creation date & time
        dd 0 ; modification date & time
        dw 0 ; unused
```

```
bin_inode: ; 43
    dw 0C00Eh ; Flags (1100000000001110b)
    db 2      ; number of links
    db 0      ; user ID (0 = root)
    dw size_bin_dir ; 20
    dw 8 dup (0) ; indirect or contents blocks
    dd 0      ; creation date & time
    dd 0      ; modification date & time
    dw 0      ; unused

etc_inode: ; 44
    dw 0C00Eh ; Flags (1100000000001110b)
    db 2      ; number of links
    db 0      ; user ID (0 = root)
    dw size_etc_dir ; 20
    dw 8 dup (0) ; indirect or contents blocks
    dd 0      ; creation date & time
    dd 0      ; modification date & time
    dw 0      ; unused

usr_inode: ; 45
    dw 0C00Eh ; Flags (1100000000001110b)
    db 2      ; number of links
    db 0      ; user ID (0 = root)
    dw size_usr_dir ; 20
    dw 8 dup (0) ; indirect or contents blocks
    dd 0      ; creation date & time
    dd 0      ; modification date & time
    dw 0      ; unused

tmp_inode: ; 46
    dw 0C00Fh ; Flags (1100000000001111b)
    db 2      ; number of links
    db 0      ; user ID (0 = root)
    dw size_tmp_dir ; 20
    dw 8 dup (0) ; indirect or contents blocks
    dd 0      ; creation date & time
    dd 0      ; modification date & time
    dw 0      ; unused

align 16

Buffer:
sector_buffer:
db 512 dup (0)
```

```
; UNIXPROC.ASM
;-----
; RETRO UNIX v0.1 'fd0' formatting procedures
; Last Update: 09/07/2013
; ERDOGAN TAN
; 01/03/2013, 03/03/2013, 05/03/2013
; 16/12/2012 -> sioreg (bugfix)
; [ 14-27/7/2012, 4-21/8/2012, 16/9/2012, 20/10/2012, 31/10/2012 ]
; These procedures will be located in UNIXFDFS.ASM file
; when they are completed.
; (NOTE: only for (R)UFS initialization of FD0 1.44MB floppy disk

err_INVALIDDATA equ 100h
err_NOFREEBLOCK equ 200h

iget    proc near
; 16/9/2012
; 14/7/2012
; Derived from (original) UNIX v1 source code
; PRELIMINARY release of Unix Implementation Document,
; 20/6/1972
;; AX=R0, BX=R1
; RETRO UNIX v1 FS
; initialization/format version
; (cdev, idev,mnt, mntd are excluded)
; ; return => if cf=1 error number in [Error]

    cmp bx, word ptr [ii] ; BX (R1) = i-number of current file
    je short iget_5

iget_1:
    push ax
    xor ah, ah ; mov ah, 0
    mov al, byte ptr [imod]
    and al, al ; has i-node of current file been modified ?
    jz short iget_2
    xor al, al ; mov al, 0
    mov byte ptr [imod], al
    push bx
    mov bx, word ptr [ii]
    inc al ; mov al, 1
    ; ax = 1 = write
    call icalc
    pop bx
    jc short iget_4
; 16/9/2012
    xor al, al ; xor ax, ax

iget_2:
    and bx, bx
    jz short iget_3
    mov word ptr [ii], bx
    ; ax = 0 = read
    call icalc

iget_3:
    mov bx, word ptr [ii]

iget_4:
    pop ax

iget_5:
    ret

iget    endp

icalc   proc near
; 17/8/2012
; 16/8/2012
; 15/8/2012
; 14/8/2012
; 13/8/2012
; 15/7/2012
; 14/7/2012
; Derived from (original) UNIX v1 source code
; PRELIMINARY release of Unix Implementation Document,
; 20/6/1972
;; AX=R0, BX=R1, CX=R3, DX=R5
; 0 = read, 1 = write
; RETRO UNIX v1 FS
; initialization/format version
;
; i-node is located in block (i+47)/16 and
```

```
; begins 32*(i+47) mod 16 bytes from its start
;; return => if cf=1 error number in [Error]

; input -> ax = 0 -> read, 1 = Write

add bx, 47 ; add 47 to inode number, 15/8/2012
push bx ; R1 -> -(SP)
shr bx, 1 ; divide by 16
shr bx, 1
shr bx, 1
shr bx, 1
; bx contains block number of block in which
; inode exists
call dskrd
pop dx ; 14/8/2012
jc short icalc_5

icalc_1:
and dx, 0Fh ; (i+47) mod 16
shl dx, 1
shl dx, 1
shl dx, 1
shl dx, 1
shl dx, 1
; DX = 32 * ((i+47) mod 16)
; DX (R5) points to first word in i-node i.

; 14/8/2012
push di
push si

mov si, offset inode ; 14/8/2012
; inode is address of first word of current inode
mov cx, 16 ; CX = R3

push ax

mov di, offset Buffer ; 16/8/2012

add di, dx ; 13/8/2012

and ax, ax
jz short icalc_3 ; 0 = read (and copy i-node to memory)

icalc_2:
; 14/8/2012
; over write old i-node (in buffer to be written)
rep movsw

; 31/10/2012
call dskwr
jmp short icalc_4

icalc_3:
xchg si, di ; 14/8/2012
; copy new i-node into inode area of (core) memory
rep movsw

icalc_4:
pop ax
; 14/8/2012
pop si
pop di

; OUTPUTS ->
; inode
; DX/R5 (internal), BX/R1 (internal), CX/R3 (internal)

icalc_5:
retn

icalc endp

dskrd proc near
; 31/10/2012
; 19/08/2012
; 15/07/2012
; 14/07/2012
```

```
; Derived from (original) UNIX v1 source code
; PRELIMINARY release of Unix Implementation Document,
; 20/6/1972
;; AX=R0, BX=R1, CX=R3, DX=R5
; RETRO UNIX v1 FS
; initialization/format version
;
; BX = R1 = block/sector number
;
; call bufalloc ; get a free I/O buffer
; R5 = pointer to buffer
;; return => if cf=1 error number in [Error]

cmp bx, word ptr [buff_s] ; buffer sector
je short dskrd_4

dskrd_1:
  cmp byte ptr [buff_m], 0 ; is buffer data changed ?
  jna short dskrd_3

  mov byte ptr [buff_w], 1 ; r/w flag = write
  call poke
  jc short dskrd_4
dskrd_3:
  mov word ptr [buff_s], bx
  mov byte ptr [buff_w], 0 ; r/w flag = read
  call poke
dskrd_4:
  ; 19/8/2012
  retn

dskrd  endp

dskwr  proc near
; 31/10/2012
; 15/07/2012
; 14/07/2012
; Derived from (original) UNIX v1 source code
; PRELIMINARY release of Unix Implementation Document,
; 20/6/1972
;; AX=R0, BX=R1, CX=R3, DX=R5
; RETRO UNIX v1 FS
; initialization/format version
;
;; return => if cf=1 error number in [Error]
;; cf = 1 => dx = 0
; input:
; BX = Block/Sector number

dskwr_1:
  mov byte ptr [buff_w], 1 ; r/w flag = write
  call poke
  ; cf = 1 -> Error code in [Error]
  ; cf = 0 -> Successful
  retn

dskwr  endp

poke  proc near
; 15/7/2012
; Basic I/O functions for block structured devices
;
; Derived from (original) UNIX v1 source code
; PRELIMINARY release of Unix Implementation Document,
; 20/6/1972
;; AX=R0, BX=R1, CX=R3, DX=R5
; [SP] = Argument 1, 0 = read, 1 = write
; RETRO UNIX v1 FS
; initialization/format version
;
; [buff_s] = block/sector number
; [buff_w] = read/write flag (1=write, 0=read)

;; return => if cf=1 error number in [Error]

mov word ptr [Error], 0 ; Error code reset
```

```
    cmp byte ptr [buff_w], 1
    jna short poke_1

    inc byte ptr [Error]+1 ; mov byte ptr [Error]+1, 1
    ; high byte 1 -> invalid data/parameter

    stc
    retn
poke_1:
    ; Physical disk read/write for 8086 PC (via ROMBIOS)
    call fd_rw_sector
    jc short poke_2

    mov byte ptr [buff_m], 0
poke_2:
    retn

poke    endp

fd_rw_sector proc near
    ; 14/8/2012
    ; 15/7/2012
    ; Only for 1.44 MB Floppy Disks (18 sector/track)

    ; buff_s = sector number, buffer = r/w buffer offset
    ; buff_d = phy drv number, buff_w = 0/1 -> r/w

    ;push es
    push bx
    push dx
    push cx
    push ax

    ;push ds
    ;pop es
    mov bx, offset Buffer

    xor ch, ch
    mov cl, byte ptr [RetryCount] ; 4
fd_rw_sector_1:
    push cx
    mov ax, word ptr [buff_s] ; LOGICAL SECTOR NUMBER
    mov dx, 18 ; Sectors per track
    div dl
    mov cl, ah ; Sector (zero based)
    inc cl ; To make it 1 based
    shr al, 1 ; Convert Track to Cylinder
    adc dh, 0 ; Heads (0 or 1)

    mov dl, byte ptr [buff_d] ; Physical drive number
    mov ch, al

    mov ah, byte ptr [buff_w] ; 0=read, 1=write (unix)
    add ah, 2 ; 2=read, 3=write (bios)
    mov al, 01h
    int 13h
    ; BIOS Service func ( ah ) = 2
    ; Read disk sectors
    ; BIOS Service func ( ah ) = 3
    ; Write disk sectors
    ;AL-sec num CH-cyl CL-sec
    ; DH-head DL-drive ES:BX-buffer
    ;CF-flag AH-stat AL-sec read

    mov byte ptr [Error], ah
    pop cx
    jnc short fd_rw_sector_2
    loop fd_rw_sector_1
fd_rw_sector_2:
    pop ax
    pop cx
    pop dx
    pop bx
    ;pop es
    retn

fd_rw_sector endp

setimod proc near
    ; 13/8/2012
```

```
; 21/7/2012
; 14/7/2012
; Derived from (original) UNIX v1 source code
; PRELIMINARY release of Unix Implementation Document,
; 20/6/1972
;; AX=R0, BX=R1, CX=R3, DX=R5
; [SP] = Argument 1, 0 = read, 1 = write
; RETRO UNIX v1 FS
; initialization/format version
;

; 21/7/2012
push dx
push ax

mov byte ptr [imod], 1

; Erdogan Tan 14-7-2012
call epoch

mov word ptr [i_mtim], ax
mov word ptr [i_mtim]+2, dx

; 21/7/2012
cmp word ptr [i_ctim], 0
ja short @f
cmp word ptr [i_ctim]+2, 0
ja short @f

mov word ptr [i_ctim], ax
mov word ptr [i_ctim]+2, dx
@@:

; 21/7/2012
pop ax
pop dx

retn

setimod endp

imap proc near
; 21/8/2012
; 5/8/2012
; 16/7/2012
; Derived from (original) UNIX v1 source code
; PRELIMINARY release of Unix Implementation Document,
; 20/6/1972
; RETRO UNIX v1 FS
; initialization/format version
;
; get the byte that the allocation bit
; for the i-number contained in R1

mov dx, bx ; DX = R2, BX = R1 (input, i-number)
sub dx, 41 ; DX has i-41
mov cl, dl ; CX = R3
mov ax, 1 ;
and cl, 7 ; CX has (i-41) mod 8 to get the bit position
jz short @f ; 21/8/2012
shl ax, cl ; AX has 1 in the calculated bit position
@@:

shr dx, 1
shr dx, 1
shr dx, 1 ; DX has (i-41) base 8 of byte number
; from the start of the (inode) map
; 5/8/2012
add dx, word ptr [systm] ; superblock free map size + 4
; 21/8/2012
add dx, offset systm+4 ; is inode map offset in superblock
; AX (MQ) has a 1 in the calculated bit position
; CX (R3) used internally
; DX (R2) has byte address of the byte with allocation bit
retn

imap endp

writei proc near
```

```
; 31/10/2012
; 18/08/2012
; 17/07/2012
; BX = R1, i-number
; Derived from (original) UNIX v1 source code
; PRELIMINARY release of Unix Implementation Document,
; 20/6/1972
;; AX=R0, BX=R1, i-number
; RETRO UNIX v1 FS
; initialization/format version
;
; writei: write file
;
; 8086 CPU & IBM PC architecture modifications by Erdogan Tan
;; return => if cf=1 error number in [Error]

; input:
; BX = R1 = I-Number
; u.count = byte count
; u.base = user buffer (offset)
; u.fofp = (pointer to) current file offset

xor ax, ax ; 0 ; clr u.nread
mov word ptr [u_nread], ax ; clear the number of bytes transmitted during
; read or write calls
; tst u.count

cmp word ptr [u_count], ax ; test the byte count specified by the user
;ja short write_1 ; 1f ; bgt 1f / any bytes to output; yes, branch
;retn ; rts 0 / no, return - no writing to do
jna short @f

write_1:
  cmp bx, 40 ;cmp r1,$40.
; does the i-node number indicate a special file?
ja short dskw_0 ; bgt dskw / no, branch to standard file output
@@:
  retn

; shl bx, 1 ; asl r1
; yes, calculate the index into the special file

; cmp bx, offset write_3 - offset writei_2 + 2
; ja short writei_error

; jmp word ptr [write_2][BX]-2 ; *1f-2(r1)
; jump table and jump to the appropriate routine
;write_2: ;1
; dw offset wtty ; tty
; dw offset wmem ; mem
; dw offset wfd ; fd0
; dw offset wfd ; fd1
; dw offset whd ; hd0
; dw offset whd ; hd1
; dw offset whd ; hd2
; dw offset whd ; hd3
; dw offset xmtt ; tty0
; dw offset xmtt ; tty1
; dw offset xmtt ; tty2
; dw offset xmtt ; tty3
; dw offset xmtt ; tty4
; dw offset xmtt ; tty5
; dw offset xmtt ; tty6
; dw offset xmtt ; tty7
; dw offset wlpr ; lpr
; writei_3:
; dw offset writei_error

;wtty: ; write to concole tty
; retn
;wmem: ; transfer characters from a user area of core to memory
; retn

;wfd: ; write to floppy disk (drive)
; retn

;whd: ; write to hard/fixed disk (drive)
; retn
;wlpr ; write to printer
```



```
;      retn

;xmtt:
;      retn

writei endp

dskw  proc near
; 01/03/2013
; 31/10/2012
; 19/8/2012
; 30/7/2012
; 17/7/2012
; Derived from (original) UNIX v1 source code
; PRELIMINARY release of Unix Implementation Document,
; 20/6/1972
; dskw: write routine for non-special files
;
; RETRO UNIX v1 FS
; initialization/format version
;
; write data to a file
;
; BX (R1) = I-node number
;

dskw_0:
push di
push si

push bx; save i-number on stack

call iget      ; jsr r0,iget
                ; write i-node out (if modified), read i-node 'r1'
                ; into i-node area of core
jc short dskw_5 ; 01/03/2013
mov si, word ptr [u_fofp]
mov dx, word ptr [SI]
                ; mov *u.fofp,r2
                ; put the file offset [(u.off) or the offset in
                ; the fsp entry for this file] in r2
add dx, word ptr [u_count]
                ; add u.count,r2
                ; no. of bytes to be written + file offset is
                ; put in r2

cmp dx, word ptr [i_size] ; cmp r2,i.size
                ; is this greater than the present size of
                ; the file?
jna short dskw_1 ; blos      1f / no, branch

mov word ptr [i_size], dx ; mov      r2,i.size
                ; yes, increase the file size to file offset +
                ; no. of data bytes
call setimod   ; jsr r0,setimod
                ; set imod=1 (i.e., core inode has been
                ; modified), stuff time of modification into
                ; core image of i-node

dskw_1: ; 1
call mget      ; jsr r0,mget
                ; get the block no. in which to write the next data
                ; byte
                ; AX = R1 = Block Number
jc short dskw_5 ; 01/03/2013
mov si, word ptr [u_fofp]
mov bx, word ptr [SI]
and bx, 1FFh   ; bit *u.fofp,$777
                ; test the lower 9 bits of the file offset
jnz short dskw_2 ; bne 2f
                ; if its non-zero, branch; if zero, file offset = 0,
                ; 512, 1024,...(i.e., start of new block)
cmp word ptr [u_count], 512 ; cmp u.count,$512.
                ; if zero, is there enough data to fill an
                ; entire block? (i.e., no. of
jnb short dskw_6 ; bhis      3f / bytes to be written greater than 512.?
                ; Yes, branch. / Don't have to read block
```

```
dskw_2: ; 2
; in as no past info. is to be saved (the entire block will be
; overwritten).

mov bx, ax      ; R1 (block number)
call dskrd      ; jsr r0,dskrd
; no, must retain old info.. Hence, read block 'r1'
; into an I/O buffer
jc short dskw_5 ; 01/03/2013
dskw_3: ; 3
;call wslot

call sioreg

; SI = user data offset (r1)
; DI = sector (I/O) buffer offset (r2)
; CX = byte count (r3)

dskw_4: ; 2
rep movsb

mov byte ptr [buff_m], 1

call dskwr ; jsr r0,dskwr / write the block and the i-node
jc short dskw_5

cmp word ptr [u_count], 0 ; any more data to write?
ja short dskw_1 ; 1b ; yes, branch

dskw_5:
pop bx

pop si
pop di

ret

dskw_6:
cmp byte ptr [buff_m], 1
jb short dskw_3
call dskwr
jc short dskw_5
mov word ptr [buff_s], ax ; block number from mget procedure
jmp short dskw_3

dskw endp

mget proc near
; 05/03/2013
; 01/03/2013
; 31/10/2012
; 20/10/2012
; 19/8/2012
; 13/8/2012
; 27/7/2012
; 21/7/2012
; Derived from (original) UNIX v1 source code
; PRELIMINARY release of Unix Implementation Document,
; 20/6/1972
;; return -> AX=R1
; RETRO UNIX v1 FS
; initialization/format version
; cf -> 1 = error (no free block)

;push bx
;push cx
;push dx
;; contents of bx, cx, dx will be destroyed
mget_0:
; 31/10/2012
mov bx, word ptr [u_fofp]
mov ax, word ptr [BX]
mov bl, ah ; div ax by 256
xor bh, bh

; BX = R2
test word ptr [i_flg], 4096 ; 1000h
```

```
                                ; is this a large or small file
jnz short mget_5 ; 4f ; large file
test bl, 0F0h ; !0Fh ; branch if BX (R2) >= 16
jnz short mget_2 ; 3f

and bl, 0Eh ; clear all bits but bits 1,2,3
mov ax, word ptr i_dskp[BX] ; AX = R1, physical block number
or ax, ax
jnz short mget_1 ; if physical block number is zero
                        ; then need a new block for file
call alloc             ; allocate a new block for this file
                        ; AX (R1) = Block number
jc short mget_8        ; cf -> 1 & ax = 0 -> no free block

mov word ptr i_dskp[BX], ax

call setimod

call clear

mget_1: ; 2
        ; AX (R1) = Physical block number

        ;pop dx
        ;pop cx
        ;pop bx

        retn

mget_2: ; 3
        ; adding on block which changes small file to large file
call alloc
jc short mget_8 ; 01/03/2013
        ; call wslot ; setup I/O buffer for write
        ;          ; R5 points to the first data word in buffer

        ; push ds
        ; pop es

mov word ptr [buff_s], ax ; Block/Sector number

push si
push di
push ax

mov cx, 8 ; R3, transfer old physical block pointers
        ; into new indirect block area for the new
        ; large file
mov di, offset Buffer ; BX = R5
mov si, offset i_dskp

xor ax, ax ; mov ax, 0

mget_3: ; 1
movsw
mov word ptr [SI]-2, ax
loop mget_3

mov cl, 256-8 ; clear rest of data buffer

mget_4: ; 1
rep stosw

pop ax
pop di
pop si

mov byte ptr [buff_m], 1 ; modified

call dskwr
jc short mget_7 ; 01/03/2013

mov word ptr [i_dskp], ax
or word ptr [i_flg], 4096 ; 1000h

call setimod

jmp short mget_0
```

```
mget_9: ; 01/03/2013
        pop ax
mget_8:
        mov word ptr [Error], err_NOFREEBLOCK

        ;pop dx
        ;pop cx
        ;pop bx

        retn

mget_5: ; 4 ; large file
        ; 05/03/2013
        ; 03/03/2013
        ; 27/7/2012
        ;mov ax, bx
        ;mov cx, 256
        ;xor dx, dx
        ;div cx
        ;and bx, 1FEh ; zero all bit but 1,2,3,4,5,6,7,8
                        ; gives offset in indirect block
        ;push bx
                        ; R2
        ;mov bx, ax ; calculate offset in i-node for pointer
                        ; to proper indirect block
        ;and bx, 0Eh
        ;mov ax, word ptr i_dskp[BX] ; R1
        and bl, 0FEh ; 05/03/2013
        push bx
        mov ax, word ptr [i_dskp] ; 03/03/2013
        or ax, ax ; 20/10/2012
        jnz short mget_6 ; 2f

        call alloc
        jc short mget_9 ; 01/03/2013

        ;mov word ptr i_dskp[BX], ax ; R1, block number
        mov word ptr [i_dskp], ax

        call setimod

        call clear

mget_6: ;2
        ; 27/7/2012
        mov bx, ax ; R1
        call dskrd ; read indirect block
        pop bx ; R2, get offset
        ; 19/8/2012
        jc short mget_7
        add bx, offset Buffer ; R5, first word of indirect block
        mov ax, word ptr [bx] ; put physical block no of block
                        ; in file sought in R1 (AX)
        or ax, ax
        jnz short mget_7 ; 2f

        call alloc
        jc short mget_8 ; 01/03/2013

        mov word ptr [bx], ax ; R1

        mov byte ptr [buff_m], 1 ; modified

        ;call wslot
        call dskwr
        jc short mget_7 ; 01/03/2013

        ; ax = R1, block number of new block

        call clear

mget_7: ; 2
        ; ax = R1, block number of new block
        ;pop dx
        ;pop cx
        ;pop bx

        retn
```

```
mget endp

alloc proc near
; 21/8/2012
; 18/8/2012
; 17/8/2012
; 5/8/2012
; 21/7/2012
; Derived from (original) UNIX v1 source code
; PRELIMINARY release of Unix Implementation Document,
; 20/6/1972
;; input -> AX=R1
;; output -> AX=R1
; RETRO UNIX v1 FS
; initialization/format version

push cx
push bx ; R2
push dx ; R3

mov bx, offset systm ; SuperBlock
; start of inode and free storage map for disk
alloc_1: ; 1
mov ax, word ptr [BX] ; first word contains # of bytes
; in free storage map
shl ax, 1 ; multiply AX (R1) by 8 gives # of blocks
shl ax, 1
shl ax, 1
mov cx, ax ; R1, bit count of free storage map
xor ax, ax ; 0
alloc_2: ; 1
inc bx ; 18/8/2012
inc bx ;
mov dx, word ptr [BX] ; mov (R2)+, R3
or dx, dx
jnz short alloc_3 ; 1f
; branch if any free blocks in this word
add ax, 16
cmp ax, cx
jb short alloc_2 ; 1b

; jmp short panic ; no free storage

xor ax, ax
stc ; cf=1 --> error: no free block

jmp short alloc_7

alloc_3: ; 1
shr dx, 1 ; R3 ; Branch when free block found,
; bit for block k is in byte k/8
; in bit k (mod 8)
jc short alloc_4 ; 1f
inc ax ; R1 ; increment bit count in bit k (mod 8)
jmp short alloc_3 ; 1b

alloc_4:
; 5/8/2012
call free_3

alloc_5: ; 1
; 21/8/2012
not dx ; masking bit is '0' and others are '1'
and word ptr [BX], dx ; bic r3, (r2)
; 0 -> allocated retn
alloc_6:
; inc byte ptr [smod] ; super block modified sign
mov byte ptr [smod], 1
alloc_7:
pop dx ; R3
pop bx ; R2
pop cx
; AX (R1) = Block number
retn

alloc endp
```

```

free    proc near
        ; 17/8/2012
        ; 14/8/2012
        ; 5/8/2012
        ; Derived from (original) UNIX v1 source code
        ; PRELIMINARY release of Unix Implementation Document,
        ; 20/6/1972
        ;; input -> AX=R1
        ;; output -> free map (superblock) will be updated
        ; RETRO UNIX v1 FS
        ; initialization/format version

        push cx
        push dx ; R3
        push bx ; R2

        call free_3
        ; 21/8/2012
        or word ptr [BX], dx ; set bit for this block (available)
                                ; bis r3, (r2)

free_1: ; 2
        ;inc byte ptr [smod] ; super block modified sign
        mov byte ptr [smod], 1

        pop bx ; R2
        pop dx ; R1
        pop cx

free_2: ; 1
        retn

;;free_3:
;;      mov cx, ax ; BX = R2, AX = R1
;;      ;and cx, 7 ; clear all bit but 0,1,2
;;      ; CX = (k) mod 8
;;; bit masking
;;      mov dx, 1
;;      dec cl
;;      jz short @f
;;      shl dx, cl ; mask bit at required bit position
;;@@:
;;      mov bx, ax ; mov R1, R2
;;      ; divide block number (R2/BX) by 16
;;      shr bx, 1
;;      shr bx, 1
;;      shr bx, 1
;;      shr bx, 1
;;      jnc short free_4 ; 1f, branch if bit 3 in Bx (R1) was 0
;;      ; i.e. bit for block is in lower half of word
;;      xchg dh, dl ; swap bytes in DX (R3),
;;      ; bit in upper half word in free storage map
;;
;;
;;free_4: ; 1
;;      shl bx, 1 , multiply block number by 2, BX (R2) = k/8
;;      add bx, offset systm+2 ; SuperBlock+2

free_3:
        mov dx, 1 ; 21/8/2012
        mov cx, ax
        and cx, 0Fh
        jz short @f
        shl dx, cl ; 21/8/2012

@@:
        mov bx, ax
        shr bx, 1
        shr bx, 1
        shr bx, 1
        shr bx, 1

free_4: ; 1
        shl bx, 1 ; 21/8/2012
        ; BX (R2) = k/8
        add bx, offset systm+2 ; SuperBlock+2

        retn

free    endp

```

```
clear proc near
; 5/8/2012
; 21/7/2012
; Derived from (original) UNIX v1 source code
; PRELIMINARY release of Unix Implementation Document,
; 20/6/1972
;; input -> AX=R1 (block number)
;; output -> AX=R1
; RETRO UNIX v1 FS
; initialization/format version

;call wslot ; setup I/O buffer for write
; ; R5 points to the first data word in buffer
; BX = R5

mov word ptr [buff_s], ax

;push ds
;pop es

push di
push cx
push ax
xor ax, ax
; mov di, bx
mov di, offset Buffer
mov cx, 256
rep stosw

mov byte ptr [buff_m], 1 ; modified

call dskwr ; 5/8/2012

pop ax
pop cx
pop di

retn

clear endp

sioreg proc near
; 16/12/2012
; 31/10/2012
; 19/08/2012
; 04/08/2012
; Erdogan Tan - RETRO UNIX v0.1
; input -> R5 (DX) = sector buffer (data) address
; ; *u.fofp = file offset, to start writing
; ; u.base = address of 1st byte of user data
; ; u.count = byte count to be transferred
; ; u.nread = number of bytes written out
; ; previously.
; output -> *u.fofp = last (written) byte + 1
; ; u.count = number of bytes of data left
; ; to be transferred.
; ; u.nread = updated to include the count
; ; of bytes to be transferred.
; ; R1 (SI) = address of 1st byte of data
; ; R2 (DI) = specifies the byte in IO
; ; sector (I/O) buffer. (Offset)
; ; R3 (CX) = number of bytes of data to be
; ; transferred to/from sector (I/O)
; ; buffer.

;mov dx, offset Buffer ; R5
; 31/10/2012
mov si, word ptr [u_fofp] ; mov *u.fofp,r2
mov di, word ptr [SI] ; file offset (in bytes) is moved to r2
mov cx, di ; mov r2,r3 / and also to r3

or cx, 0FE00h ; set bits 9...15 of file offset in R3
and di, 1FFh ; calculate file offset mod 512
; 19/08/2012
```

```

add di, offset Buffer ; DI/r2 now points to 1st byte in buffer
                        ; where data is to be placed
;mov si, word ptr [u_base] ; address of data is in r1
neg cx ; 512- file offset(mod512) in R3 (cx)
                        ; the number of free bytes in the file block
cmp cx, word ptr [u_count] ;compare this with the number of data bytes
                        ; to be written to the file
jna short @f ; 2f
                        ; if less than branch. Use the number of free bytes
                        ; in the file block as the number to be written
mov cx, word ptr [u_count]
                        ; if greater than, use the number of data bytes
                        ; as the number to be written
@@: ; 2
;sioreg_1:
add word ptr [u_nread], cx ; r3 + number of bytes
                        ; xmitted during write is put into
                        ; u.nread
sub word ptr [u_count], cx
                        ; u.count = no. of bytes that still must be
                        ; written or read
mov si, word ptr [u_fofp]
add word ptr [SI], cx ; new file offset = number
                        ; of bytes done + old file offset

; 16/12/2012 BugFix
mov si, word ptr [u_base] ; address of data is in SI/r1

add word ptr [u_base], cx ; u.base points to 1st of remaining
                        ; data bytes
retn

sioreg endp

epoch proc near
; 21/7/2012
; 15/7/2012
; 14/7/2012
; Erdogan Tan - RETRO UNIX v0.1
; compute current date and time as UNIX Epoch/Time
; UNIX Epoch: seconds since 1/1/1970 00:00:00

; 21/7/2012
push bx
push cx

mov ah, 02h ; Return Current Time
int 1Ah
xchg ch,cl
mov word ptr [hour], cx
xchg dh,dl
mov word ptr [second], dx

mov ah, 04h ; Return Current Date
int 1Ah
xchg ch,cl
mov word ptr [year], cx
xchg dh,dl
mov word ptr [month], dx

mov cx, 3030h

mov al, byte ptr [hour] ; Hour
; AL <= BCD number)
db 0D4h,10h ; Undocumented inst. AAM
; AH = AL / 10h
; AL = AL MOD 10h

aad ; AX= AH*10+AL

mov byte ptr [hour], al

mov al, byte ptr [hour]+1 ; Minute
; AL <= BCD number)
db 0D4h,10h ; Undocumented inst. AAM
; AH = AL / 10h
; AL = AL MOD 10h

aad ; AX= AH*10+AL

```



```
mov byte ptr [minute], al

mov al, byte ptr [second] ; Second
; AL <= BCD number)
db 0D4h,10h ; Undocumented inst. AAM
; AH = AL / 10h
; AL = AL MOD 10h

aad ; AX= AH*10+AL

mov byte ptr [second], al

mov ax, word ptr [year] ; Year (century)
push ax
; AL <= BCD number)
db 0D4h,10h ; Undocumented inst. AAM
; AH = AL / 10h
; AL = AL MOD 10h

aad ; AX= AH*10+AL

mov ah, 100
mul ah
mov word ptr [year], ax

pop ax
mov al, ah
; AL <= BCD number)
db 0D4h,10h ; Undocumented inst. AAM
; AH = AL / 10h
; AL = AL MOD 10h

aad ; AX= AH*10+AL

add word ptr [year], ax

mov al, byte ptr [month] ; Month
; AL <= BCD number)
db 0D4h,10h ; Undocumented inst. AAM
; AH = AL / 10h
; AL = AL MOD 10h

aad ; AX= AH*10+AL

mov byte ptr [month], al

mov al, byte ptr [month]+1 ; Day
; AL <= BCD number)
db 0D4h,10h ; Undocumented inst. AAM
; AH = AL / 10h
; AL = AL MOD 10h

aad ; AX= AH*10+AL

mov byte ptr [Day], al

convert_to_epoch:

mov dx, word ptr [year]
sub dx, 1970
mov ax, 365
mul dx
xor bh, bh
mov bl, byte ptr [month]
dec bl
shl bl, 1
mov cx, word ptr DMonth[BX]
mov bl, byte ptr [Day]
dec bl

add ax, cx
adc dx, 0
add ax, bx
adc dx, 0
; DX:AX = days since 1/1/1970

mov cx, word ptr [year]
sub cx, 1969
shr cx, 1
shr cx, 1
```

```

        ; (year-1969)/4
add ax, cx
adc dx, 0
        ; + leap days since 1/1/1970

cmp byte ptr [month], 2 ; if past february
jna short @f
mov cx, word ptr [year]
and cx, 3 ; year mod 4
jnz short @f
        ; and if leap year
add ax, 1 ; add this year's leap day (february 29)
adc dx, 0
@@:      ; compute seconds since 1/1/1970
mov bx, 24
call proc_mul32

mov bl, byte ptr [hour]
add ax, bx
adc dx, 0

mov bx, 60
call proc_mul32

mov bl, byte ptr [minute]
add ax, bx
adc dx, 0

mov bx, 60
call proc_mul32

mov bl, byte ptr [second]
add ax, bx
adc dx, 0

; DX:AX -> seconds since 1/1/1970 00:00:00

; 21/7/2012
pop cx
pop bx

retn

epoch endp

;.....;
; 32 bit Multiply ;
; - - - - - ;
; ;
; input -> DX_AX = 32 bit multiplier ;
; input -> BX = 16 bit number to be multiplied by DX_AX ;
; output -> BX_DX_AX = 48 bit (16+32 bit) result number ;
; ;
; (c) Erdogan TAN 1999 ;
;.....;

proc_mul32 proc near

    ; push cx

    mov cx, bx
    mov bx, dx

    mul cx

    xchg ax, bx

    push dx

    mul cx

    pop cx

    add ax, cx
    adc dx, 0

    xchg bx, ax
    xchg dx, bx

```

```
        ; pop cx

        retn

proc_mul32 endp

year: dw 1970
month: dw 1
day: dw 1
hour: dw 0
minute: dw 0
second: dw 0

DMonth:
dw 0
dw 31
dw 59
dw 90
dw 120
dw 151
dw 181
dw 212
dw 243
dw 273
dw 304
dw 334
; dw 365

db 0

Error: db 0 ; Hardware error
       db 0 ; Software error

smod: db 0
imod: db 0

ii: dw 0

dotodot:
dw 3030h
db "h"
db 0Dh, 0Ah, 0
```