```
; 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:
        retn

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 bufaloc ; 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 dik 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 w1pr ; 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

       retn

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_flgs], 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_flgs], 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
```