```
; ****************************************************************************
; UNIX386.ASM (RETRO UNIX 386 Kernel) - v0.2.0.16
; ----------------------------------------------------------------------------
; NASM version 2.11 (unix386.s)
;
; RETRO UNIX 386 (Retro Unix == Turkish Rational Unix)
; Operating System Project (v0.2) by ERDOGAN TAN (Beginning: 24/12/2013)
;
; Derived from 'Retro UNIX 8086 v1' source code by Erdogan Tan
; (v0.1 - Beginning: 11/07/2012)
;
; [ Last Modification: 09/12/2015 ]
;
; Derived 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>
;
; Derived from 'UNIX v7/x86' source code by Robert Nordier (1999)
; UNIX V7/x86 source code: see www.nordier.com/v7x86 for details.
;
; ****************************************************************************

; 24/12/2013

; Entering protected mode:
; Derived from 'simple_asm.txt' source code file and
; 'The world of Protected mode' tutorial/article by Gregor Brunmar (2003)
; (gregor.brunmar@home.se)
; http://www.osdever.net/tutorials/view/the-world-of-protected-mode
;
; "The Real, Protected, Long mode assembly tutorial for PCs"
; by Michael Chourdakis (2009)
; http://www.codeproject.com/Articles/45788/
; http://www.michaelchourdakis.com
;
; Global Descriptor Table:
; Derived from 'head.s" source code of Linux v1.0 kernel
; by Linus Torvalds (1991-1992)
;
KLOAD   equ 10000h ; Kernel loading address
        ; NOTE: Retro UNIX 8086 v1 /boot code loads kernel at 1000h:0000h
KCODE   equ 08h ; Code segment descriptor (ring 0)
KDATA   equ 10h ; Data segment descriptor (ring 0)
; 19/03/2015
UCODE   equ 1Bh ; 18h + 3h  (ring 3)
UDATA   equ 23h ; 20h + 3h  (ring 3)
; 24/03/2015
TSS     equ 28h ; Task state segment descriptor (ring 0)
; 19/03/2015
CORE    equ 400000h  ; Start of USER's virtual/linear address space
                     ; (at the end of the 1st 4MB)
ECORE   equ 0FFC00000h ; End of USER's virtual address space (4GB - 4MB)
                     ; ULIMIT = (ECORE/4096) - 1 = 0FFBFFh (in GDT)
; 27/12/2013
KEND     equ KLOAD + 65536 ; (28/12/2013) (end of kernel space)

; IBM PC/AT BIOS ----- 10/06/85 (postequ.inc)
;--------- CMOS TABLE LOCATION ADDRESS'S -------------------------------------
CMOS_SECONDS   EQU    00H           ; SECONDS (BCD)
CMOS_MINUTES   EQU    02H           ; MINUTES (BCD)
CMOS_HOURS     EQU    04H           ; HOURS (BCD)
CMOS_DAY_WEEK  EQU    06H           ; DAY OF THE WEEK  (BCD)
CMOS_DAY_MONTH EQU    07H           ; DAY OF THE MONTH (BCD)
CMOS_MONTH     EQU    08H           ; MONTH (BCD)
CMOS_YEAR      EQU    09H           ; YEAR (TWO DIGITS) (BCD)
CMOS_CENTURY   EQU    32H           ; DATE CENTURY BYTE (BCD)
CMOS_REG_A     EQU    0AH           ; STATUS REGISTER A
CMOS_REG_B     EQU    00BH          ; STATUS REGISTER B  ALARM
CMOS_REG_C     EQU    00CH          ; STATUS REGISTER C  FLAGS
CMOS_REG_D     EQU    0DH           ; STATUS REGISTER D  BATTERY
CMOS_SHUT_DOWN EQU    0FH           ; SHUTDOWN STATUS COMMAND BYTE
;----------------------------------------
;       CMOS EQUATES FOR THIS SYSTEM  ;
;-----------------------------------------------------------------------------
CMOS_PORT      EQU    070H          ; I/O ADDRESS OF CMOS ADDRESS PORT
CMOS_DATA      EQU    071H          ; I/O ADDRESS OF CMOS DATA PORT
NMI            EQU    10000000B     ; DISABLE NMI INTERRUPTS MASK -
                                    ; HIGH BIT OF CMOS LOCATION ADDRESS
```

```
; Memory Allocation Table Address
; 05/11/2014
; 31/10/2014
MEM_ALLOC_TBL  equ     100000h        ; Memory Allocation Table at the end of
                                      ; the 1st 1 MB memory space.
                                      ; (This address must be aligned
                                      ;  on 128 KB boundary, if it will be
                                      ;  changed later.)
                                      ; ((lower 17 bits of 32 bit M.A.T.
                                      ;   address must be ZERO)).
                                      ; ((((Reason: 32 bit allocation
                                      ;     instructions, dword steps)))
                                      ; (((byte >> 12 --> page >> 5)))
;04/11/2014
PDE_A_PRESENT  equ     1              ; Present flag for PDE
PDE_A_WRITE    equ     2              ; Writable (write permission) flag
PDE_A_USER     equ     4              ; User (non-system/kernel) page flag
;
PTE_A_PRESENT  equ     1              ; Present flag for PTE (bit 0)
PTE_A_WRITE    equ     2              ; Writable (write permission) flag (bit 1)
PTE_A_USER     equ     4              ; User (non-system/kernel) page flag (bit 2)
PTE_A_ACCESS    equ    32             ; Accessed flag (bit 5) ; 09/03/2015

; 17/02/2015 (unix386.s)
; 10/12/2014 - 30/12/2014 (0B000h -> 9000h) (dsectrm2.s)
DPT_SEGM equ 09000h  ; FDPT segment (EDD v1.1, EDD v3)
;
HD0_DPT  equ 0     ; Disk parameter table address for hd0
HD1_DPT  equ 32    ; Disk parameter table address for hd1
HD2_DPT  equ 64    ; Disk parameter table address for hd2
HD3_DPT  equ 96    ; Disk parameter table address for hd3


; FDPT (Phoenix, Enhanced Disk Drive Specification v1.1, v3.0)
;      (HDPT: Programmer's Guide to the AMIBIOS, 1993)
;
FDPT_CYLS      equ 0 ; 1 word, number of cylinders
FDPT_HDS       equ 2 ; 1 byte, number of heads
FDPT_TT        equ 3 ; 1 byte, A0h = translated FDPT with logical values
                     ; otherwise it is standard FDPT with physical values
FDPT_PCMP      equ 5 ; 1 word, starting write precompensation cylinder
                     ; (obsolete for IDE/ATA drives)
FDPT_CB        equ 8 ; 1 byte, drive control byte
                      ; Bits 7-6 : Enable or disable retries (00h = enable)
                      ; Bit 5 : 1 = Defect map is located at last cyl. + 1
                      ; Bit 4 : Reserved. Always 0
                      ; Bit 3 : Set to 1 if more than 8 heads
                      ; Bit 2-0 : Reserved. Alsways 0
FDPT_LZ        equ 12 ; 1 word, landing zone (obsolete for IDE/ATA drives)
FDPT_SPT       equ 14 ; 1 byte, sectors per track

; Floppy Drive Parameters Table (Programmer's Guide to the AMIBIOS, 1993)
; (11 bytes long) will be used by diskette handler/bios
; which is derived from IBM PC-AT BIOS (DISKETTE.ASM, 21/04/1986).

 [BITS 16]       ; We need 16-bit intructions for Real mode

[ORG 0]
       ; 12/11/2014
       ; Save boot drive number (that is default root drive)
       mov     [boot_drv], dl ; physical drv number

       ; Determine installed memory
       ; 31/10/2014
       ;
       mov     ax, 0E801h ; Get memory size
       int     15h       ; for large configurations
       jnc     short chk_ms
       mov     ah, 88h    ; Get extended memory size
       int     15h
       ;
       ;mov    al, 17h ; Extended memory (1K blocks) low byte
       ;out    70h, al ; select CMOS register
       ;in     al, 71h ; read data (1 byte)
       ;mov    cl, al
       ;mov    al, 18h ; Extended memory (1K blocks) high byte
       ;out    70h, al ; select CMOS register
       ;in     al, 71h ; read data (1 byte)
```

```
        ;mov    ch, al
        ;
        mov     cx, ax
        xor     dx, dx
chk_ms:
        mov     [mem_1m_1k], cx
        mov     [mem_16m_64k], dx
        ; 05/11/2014
        ;and    dx, dx
        ;jz     short L2
         cmp     cx, 1024
        jnb     short L0
                 ; insufficient memory_error
                 ; Minimum 2 MB memory is needed...
        ; 05/11/2014
        ; (real mode error printing)
        sti
        mov     si, msg_out_of_memory
        mov     bx, 7
        mov     ah, 0Eh ; write tty
oom_1:
        lodsb
        or      al, al
        jz      short oom_2
        int     10h
        jmp     short oom_1
oom_2:
         hlt
        jmp     short oom_2

L0:
%include 'diskinit.inc' ; 07/03/2015

        ; 10/11/2014
        cli     ; Disable interrupts (clear interrupt flag)
                ; Reset Interrupt MASK Registers (Master&Slave)
        ;mov    al, 0FFh        ; mask off all interrupts
        ;out    21h, al         ; on master PIC (8259)
        ;jmp    $+2  ; (delay)
        ;out    0A1h, al        ; on slave PIC (8259)
        ;
        ; Disable NMI
        mov     al, 80h
        out     70h, al         ; set bit 7 to 1 for disabling NMI
        ;23/02/2015
        nop                     ;
        ;in     al, 71h         ; read in 71h just after writing out to 70h
                                ; for preventing unknown state (!?)
        ;
        ; 20/08/2014
        ; Moving the kernel 64 KB back (to physical address 0)
        ; DS = CS = 1000h
        ; 05/11/2014
        xor     ax, ax
        mov     es, ax ; ES = 0
        ;
        mov     cx, (KEND - KLOAD)/4
        xor     si, si
        xor     di, di
        rep     movsd
        ;
        push    es ; 0
        push    L17
        retf
        ;
L17:
        ; Turn off the floppy drive motor
         mov     dx, 3F2h
         out     dx, al ; 0 ; 31/12/2013

        ; Enable access to memory above one megabyte
L18:
        in      al, 64h
        test    al, 2
         jnz     short L18
        mov     al, 0D1h        ; Write output port
        out     64h, al
```

```
L19:
        in      al, 64h
        test    al, 2
        jnz     short L19
        mov     al, 0DFh        ; Enable A20 line
        out     60h, al
;L20:
        ; Load global descriptor table register

        ;mov     ax, cs
        ;mov     ds, ax

        lgdt    [cs:gdtd]

        mov     eax, cr0
        ; or    eax, 1
        inc      ax
        mov      cr0, eax

        ; Jump to 32 bit code

        db 66h                  ; Prefix for 32-bit
        db 0EAh                 ; Opcode for far jump
        dd StartPM              ; Offset to start, 32-bit
                                ; (1000h:StartPM = StartPM + 10000h)
        dw KCODE                ; This is the selector for CODE32_DESCRIPTOR,
                                ; assuming that StartPM resides in code32

[BITS 32]

StartPM:
        ; Kernel Base Address = 0 ; 30/12/2013
        mov ax, KDATA             ; Save data segment identifier
        mov ds, ax                ; Move a valid data segment into DS register
        mov es, ax                ; Move data segment into ES register
        mov fs, ax                ; Move data segment into FS register
        mov gs, ax                ; Move data segment into GS register
        mov ss, ax                ; Move data segment into SS register
        mov esp, 90000h           ; Move the stack pointer to 090000h

clear_bss: ; Clear uninitialized data area
        ; 11/03/2015
        xor  eax, eax ; 0
        mov  ecx, (bss_end - bss_start)/4
        ;shr  ecx, 2 ; bss section is already aligned for double words
        mov  edi, bss_start
        rep  stosd

memory_init:
        ; Initialize memory allocation table and page tables
        ; 16/11/2014
        ; 15/11/2014
        ; 07/11/2014
        ; 06/11/2014
        ; 05/11/2014
        ; 04/11/2014
        ; 31/10/2014 (Retro UNIX 386 v1 - Beginning)
        ;
;       xor     eax, eax
;       xor     ecx, ecx
        mov     cl, 8
        mov     edi, MEM_ALLOC_TBL
        rep     stosd             ; clear Memory Allocation Table
                                  ; for the first 1 MB memory
        ;
        mov     cx, [mem_1m_1k]           ; Number of contiguous KB between
                                  ; 1 and 16 MB, max. 3C00h = 15 MB.
        shr     cx, 2             ; convert 1 KB count to 4 KB count
        mov     [free_pages], ecx
        mov     dx, [mem_16m_64k]  ; Number of contiguous 64 KB blocks
                                  ; between 16 MB and 4 GB.
        or      dx, dx
        jz      short mi_0
        ;
        mov     ax, dx
        shl     eax, 4            ; 64 KB -> 4 KB (page count)
        add     [free_pages], eax
        add     eax, 4096         ; 16 MB = 4096 pages
        jmp     short mi_1
```

```
mi_0:
        mov     ax, cx
        add     ax, 256           ; add 256 pages for the first 1 MB
mi_1:
        mov     [memory_size], eax ; Total available memory in pages
                                  ; 1 alloc. tbl. bit = 1 memory page
                                  ; 32 allocation bits = 32 mem. pages
        ;
        add     eax, 32767        ; 32768 memory pages per 1 M.A.T. page
        shr     eax, 15           ; ((32768 * x) + y) pages (y < 32768)
                                  ;  --> x + 1 M.A.T. pages, if y > 0
                                  ;  --> x M.A.T. pages, if y = 0
        mov     [mat_size], ax    ; Memory Alloc. Table Size in pages
        shl     eax, 12           ; 1 M.A.T. page = 4096 bytes
        ;                         ; Max. 32 M.A.T. pages (4 GB memory)
        mov     ebx, eax          ; M.A.T. size in bytes
        ; Set/Calculate Kernel's Page Directory Address
        add     ebx, MEM_ALLOC_TBL
        mov     [k_page_dir], ebx  ; Kernel's Page Directory address
                                  ; just after the last M.A.T. page
        ;
        sub     eax, 4            ; convert M.A.T. size to offset value
        mov     [last_page], eax   ; last page ofset in the M.A.T.
        ;                         ; (allocation status search must be
                                  ; stopped after here)
        xor     eax, eax
        dec     eax               ; FFFFFFFFh (set all bits to 1)
        push    cx
        shr     ecx, 5            ; convert 1 - 16 MB page count to
                                  ; count of 32 allocation bits
        rep     stosd
        pop     cx
        inc     eax               ; 0
        and     cl, 31            ; remain bits
        jz      short mi_4
        mov     [edi], eax        ; reset
mi_2:
        bts     [edi], eax        ; 06/11/2014
        dec     cl
        jz      short mi_3
        inc     al
        jmp     short mi_2
mi_3:
        sub     al, al            ; 0
        add     edi, 4            ; 15/11/2014
mi_4:
        or      dx, dx           ; check 16M to 4G memory space
        jz      short mi_6       ; max. 16 MB memory, no more...
        ;
        mov     ecx, MEM_ALLOC_TBL + 512 ; End of first 16 MB memory
        ;
        sub     ecx, edi         ; displacement (to end of 16 MB)
        jz      short mi_5       ; jump if EDI points to
                                 ;         end of first 16 MB
        shr     ecx, 1           ; convert to dword count
        shr     ecx, 1           ; (shift 2 bits right)
        rep     stosd            ; reset all bits for reserved pages
                                 ; (memory hole under 16 MB)
mi_5:
        mov     cx, dx           ; count of 64 KB memory blocks
        shr     ecx, 1           ; 1 alloc. dword per 128 KB memory
        pushf                    ; 16/11/2014
        dec     eax              ; FFFFFFFFh (set all bits to 1)
        rep     stosd
        inc     eax              ; 0
        popf                     ; 16/11/2014
        jnc     short mi_6
        dec     ax               ; eax = 0000FFFFh
        stosd
        inc     ax               ; 0
mi_6:
        cmp     edi, ebx         ; check if EDI points to
        jnb     short mi_7       ; end of memory allocation table
        ;                        ; (>= MEM_ALLOC_TBL + 4906)
        mov     ecx, ebx         ; end of memory allocation table
        sub     ecx, edi         ; convert displacement/offset
        shr     ecx, 1           ; to dword count
        shr     ecx, 1           ; (shift 2 bits right)
        rep     stosd            ; reset all remain M.A.T. bits
```

```
mi_7:
        ; Reset M.A.T. bits in M.A.T. (allocate M.A.T. pages)
        mov     edx, MEM_ALLOC_TBL
        ;sub    ebx, edx          ; Mem. Alloc. Tbl. size in bytes
        ;shr    ebx, 12           ; Mem. Alloc. Tbl. size in pages
        mov     cx, [mat_size]    ; Mem. Alloc. Tbl. size in pages
        mov     edi, edx
        shr     edi, 15           ; convert M.A.T. address to
                                  ; byte offset in M.A.T.
                                  ; (1 M.A.T. byte points to
                                  ;          32768 bytes)
                                  ; Note: MEM_ALLOC_TBL address
                                  ; must be aligned on 128 KB
                                  ; boundary!
        add     edi, edx          ; points to M.A.T.'s itself
        ; eax = 0
        sub     [free_pages], ecx ; 07/11/2014
mi_8:
        btr     [edi], eax        ; clear bit 0 to bit x (1 to 31)
        ;dec    bl
        dec     cl
        jz      short mi_9
        inc     al
        jmp     short mi_8
mi_9:
        ; Reset Kernel's Page Dir. and Page Table bits in M.A.T.
        ;               (allocate pages for system page tables)

        ; edx = MEM_ALLOC_TBL
        mov     ecx, [memory_size] ; memory size in pages (PTEs)
        add     ecx, 1023        ; round up (1024 PTEs per table)
        shr     ecx, 10          ; convert memory page count to
                                 ; page table count (PDE count)
        ;
        push    ecx              ; (**) PDE count (<= 1024)
        ;
        inc     ecx              ; +1 for kernel page directory
        ;
        sub     [free_pages], ecx ; 07/11/2014
        ;
        mov     esi, [k_page_dir] ; Kernel's Page Directory address
        shr     esi, 12          ; convert to page number
mi_10:
        mov     eax, esi         ; allocation bit offset
        mov     ebx, eax
        shr     ebx, 3           ; convert to alloc. byte offset
        and     bl,  0FCh        ; clear bit 0 and bit 1
                                 ;   to align on dword boundary
        and     eax, 31          ; set allocation bit position
                                 ;  (bit 0 to bit 31)
        ;
        add     ebx, edx         ; offset in M.A.T. + M.A.T. address
        ;
        btr     [ebx], eax       ; reset relevant bit (0 to 31)
        ;
        inc     esi              ; next page table
        loop    mi_10            ; allocate next kernel page table
                                 ; (ecx = page table count + 1)
        ;
        pop     ecx              ; (**) PDE count (= pg. tbl. count)
        ;
        ; Initialize Kernel Page Directory and Kernel Page Tables
        ;
        ; Initialize Kernel's Page Directory
        mov     edi, [k_page_dir]
        mov     eax, edi
        or      al, PDE_A_PRESENT + PDE_A_WRITE
                             ; supervisor + read&write + present
        mov     edx, ecx       ; (**) PDE count (= pg. tbl. count)
mi_11:
        add     eax, 4096      ; Add page size (PGSZ)
                                ; EAX points to next page table
        stosd
        loop    mi_11
        sub     eax, eax       ; Empty PDE
        mov     cx, 1024       ; Entry count (PGSZ/4)
        sub     ecx, edx
        jz      short mi_12
        rep     stosd          ; clear remain (empty) PDEs
```

```
        ;
        ; Initialization of Kernel's Page Directory is OK, here.
mi_12:
        ; Initialize Kernel's Page Tables
        ;
        ; (EDI points to address of page table 0)
        ; eax = 0
        mov     ecx, [memory_size] ; memory size in pages
        mov     edx, ecx       ; (***)
        mov     al, PTE_A_PRESENT + PTE_A_WRITE
                             ; supervisor + read&write + present
mi_13:
        stosd
        add     eax, 4096
        loop    mi_13
        and     dx, 1023       ; (***)
        jz      short mi_14
        mov     cx, 1024
        sub     cx, dx         ; from dx (<= 1023) to 1024
        xor     eax, eax
        rep     stosd          ; clear remain (empty) PTEs
                               ; of the last page table
mi_14:
        ;  Initialization of Kernel's Page Tables is OK, here.
        ;
        mov     eax, edi       ; end of the last page table page
                                ; (beginging of user space pages)
        shr     eax, 15        ; convert to M.A.T. byte offset
        and     al, 0FCh       ; clear bit 0 and bit 1 for
                               ; aligning on dword boundary

        mov     [first_page], eax
        mov     [next_page], eax ; The first free page pointer
                                ; for user programs
                                ; (Offset in Mem. Alloc. Tbl.)
        ;
        ; Linear/FLAT (1 to 1) memory paging for the kernel is OK, here.
        ;

        ; Enable paging
        ;
         mov     eax, [k_page_dir]
        mov     cr3, eax
        mov     eax, cr0
        or      eax, 80000000h ; set paging bit (bit 31)
        mov     cr0, eax
         ;jmp    KCODE:StartPMP

        db 0EAh                ; Opcode for far jump
         dd StartPMP           ; 32 bit offset
        dw KCODE               ; kernel code segment descriptor


StartPMP:
        ; 06/11//2014
        ; Clear video page 0
        ;
        ; Temporary Code
        ;
        mov     ecx, 80*25/2
        mov     edi, 0B8000h
        xor     eax, eax       ; black background, black fore color
        rep     stosd

        ; 19/08/2014
        ; Kernel Base Address = 0
        ; It is mapped to (physically) 0 in the page table.
        ; So, here is exactly 'StartPMP' address.
        ;
        ;;mov   ah, 4Eh ; Red background, yellow forecolor
        ;;mov   esi, msgPM
        ;; 14/08/2015 (kernel version message will appear
        ;;             when protected mode and paging is enabled)
        mov     ah, 0Bh ; Black background, light cyan forecolor
        mov     esi, msgKVER
        mov     edi, 0B8000h ; 27/08/2014
        ; 20/08/2014
        call    printk
```

```
        ; 'UNIX v7/x86' source code by Robert Nordier (1999)
        ; // Set IRQ offsets
        ;
        ;  Linux (v0.12) source code by Linus Torvalds (1991)
        ;
                                        ;; ICW1
        mov     al, 11h                 ; Initialization sequence
        out     20h, al                 ;       8259A-1
        ; jmp   $+2
        out     0A0h, al                ;       8259A-2
                                        ;; ICW2
        mov     al, 20h                 ; Start of hardware ints (20h)
        out     21h, al                 ;       for 8259A-1
        ; jmp   $+2
        mov     al, 28h                 ; Start of hardware ints (28h)
        out     0A1h, al                ;       for 8259A-2
                                        ;
        mov     al, 04h                 ;; ICW3
        out     21h, al                 ;       IRQ2 of 8259A-1 (master)
        ; jmp   $+2
        mov     al, 02h                 ;       is 8259A-2 (slave)
        out     0A1h, al                ;
                                        ;; ICW4
        mov     al, 01h                 ;
        out     21h, al                 ;       8086 mode, normal EOI
        ; jmp   $+2
        out     0A1h, al                ;       for both chips.

        ;mov    al, 0FFh        ; mask off all interrupts for now
        ;out    21h, al
        ;; jmp $+2
        ;out    0A1h, al

        ; 02/04/2015
        ; 26/03/2015 System call (INT 30h) modification
        ;  DPL = 3 (Interrupt service routine can be called from user mode)

        ;
        ;; Linux (v0.12) source code by Linus Torvalds (1991)
        ;  setup_idt:
        ;
         ;; 16/02/2015
        ;;mov      dword [DISKETTE_INT], fdc_int ; IRQ 6 handler
        ; 21/08/2014 (timer_int)
        mov     esi, ilist
        lea     edi, [idt]
        ; 26/03/2015
        mov     ecx, 48         ; 48 hardware interrupts (INT 0 to INT 2Fh)
        ; 02/04/2015
        mov     ebx,  80000h
rp_sidt1:
        lodsd
        mov     edx, eax
        mov     dx, 8E00h
        mov     bx, ax
        mov     eax, ebx        ; /* selector = 0x0008 = cs */
                                 ; /* interrupt gate - dpl=0, present */
        stosd   ; selector & offset bits 0-15
        mov     eax, edx
        stosd   ; attributes & offset bits 16-23
        loop    rp_sidt1
        mov     cl, 16         ; 16 software interrupts (INT 30h to INT 3Fh)
rp_sidt2:
        lodsd
        and     eax, eax
        jz      short rp_sidt3
        mov     edx, eax
        mov     dx, 0EE00h      ; P=1b/DPL=11b/01110b
        mov     bx, ax
        mov     eax, ebx        ; selector & offset bits 0-15
        stosd
        mov     eax, edx
        stosd
        loop    rp_sidt2
        jmp     short sidt_OK
rp_sidt3:
        mov     eax, ignore_int
        mov     edx, eax
        mov     dx, 0EE00h      ; P=1b/DPL=11b/01110b
```

```
        mov     bx, ax
        mov     eax, ebx        ; selector & offset bits 0-15
rp_sidt4:
        stosd
        xchg    eax, edx
        stosd
        xchg    edx, eax
        loop    rp_sidt4
sidt_OK:
        lidt    [idtd]
        ;
        ; TSS descriptor setup ; 24/03/2015
        mov     eax, task_state_segment
        mov     [gdt_tss0], ax
        rol     eax, 16
        mov     [gdt_tss1], al
        mov     [gdt_tss2], ah
        mov     word [tss.IOPB], tss_end - task_state_segment
                ;
                ; IO Map Base address (When this address points
                ; to end of the TSS, CPU does not use IO port
                ; permission bit map for RING 3 IO permissions,
                ; access to any IO ports in ring 3 will be forbidden.)
                ;
        ;mov    [tss.esp0], esp ; TSS offset 4
        ;mov    word [tss.ss0], KDATA ; TSS offset 8 (SS)
        mov     ax, TSS  ; It is needed when an interrupt
                        ; occurs (or a system call -software INT- is requested)
                        ; while cpu running in ring 3 (in user mode).

                        ; (Kernel stack pointer and segment will be loaded
                        ; from offset 4 and 8 of the TSS, by the CPU.)
        ltr     ax  ; Load task register
        ;
esp0_set0:
        ; 30/07/2015
        mov     ecx, [memory_size] ; memory size in pages
        shl     ecx, 12 ; convert page count to byte count
        cmp     ecx, CORE ; beginning of user's memory space (400000h)
                         ; (kernel mode virtual address)
        jna     short esp0_set1
        ;
        ; If available memory > CORE (end of the 1st 4 MB)
        ; set stack pointer to CORE
        ;(Because, PDE 0 is reserved for kernel space in user's page directory)
        ;(PDE 0 points to page table of the 1st 4 MB virtual address space)
        mov     ecx, CORE
esp0_set1:
        mov     esp, ecx ; top of kernel stack (**tss.esp0**)
esp0_set_ok:
        ; 30/07/2015 (**tss.esp0**)
        mov     [tss.esp0], esp
         mov     word [tss.ss0], KDATA
        ; 14/08/2015
        ; 10/11/2014 (Retro UNIX 386 v1 - Erdogan Tan)
        ;
        ;cli    ; Disable interrupts (for CPU)
        ;    (CPU will not handle hardware interrupts, except NMI!)
        ;
        xor     al, al          ; Enable all hardware interrupts!
        out     21h, al         ; (IBM PC-AT compatibility)
        jmp     $+2             ; (All conventional PC-AT hardware
        out     0A1h, al        ;  interrupts will be in use.)
                                ; (Even if related hardware component
                                ;  does not exist!)
        ; Enable NMI
        mov     al, 7Fh         ; Clear bit 7 to enable NMI (again)
        out     70h, al
        ; 23/02/2015
        nop
        in      al, 71h         ; read in 71h just after writing out to 70h
                                ; for preventing unknown state (!?)
        ;
        ; Only a NMI can occur here... (Before a 'STI' instruction)
        ;
        ; 02/09/2014
        xor     bx, bx
        mov     dx, 0200h       ; Row 2, column 0  ; 07/03/2015
        call    set_cpos
```

```
        ;
        ; 06/11/2014
        ; Temporary Code
        ;
        call    memory_info
        ; 14/08/2015
        ;call getch ; 28/02/2015
drv_init:
        sti     ; Enable Interrupts
        ; 06/02/2015
        mov     edx, [hd0_type] ; hd0, hd1, hd2, hd3
        mov     bx, [fd0_type] ; fd0, fd1
        ; 22/02/2015
        and     bx, bx
        jnz     short di1
        ;
        or      edx, edx
        jnz     short di2
        ;
setup_error:
        mov     esi, setup_error_msg
psem:
        lodsb
        or      al, al
        ;jz     short haltx ; 22/02/2015
        jz      short di3
        push    esi
        xor     ebx, ebx ; 0
                        ; Video page 0 (bl=0)
        mov     ah, 07h ; Black background,
                        ; light gray forecolor
        call    write_tty
        pop     esi
        jmp     short psem

di1:
        ; supress 'jmp short T6'
        ;  (activate fdc motor control code)
        mov     word [T5], 9090h ; nop
        ;
        ;mov    ax, int_0Eh     ; IRQ 6 handler
        ;mov    di, 0Eh*4       ; IRQ 6 vector
        ;stosw
        ;mov    ax, cs
        ;stosw
        ;; 16/02/2015
        ;;mov     dword [DISKETTE_INT], fdc_int ; IRQ 6 handler
        ;
        CALL    DSKETTE_SETUP  ; Initialize Floppy Disks
        ;
        or      edx, edx
        jz       short di3
di2:
        call    DISK_SETUP     ; Initialize Fixed Disks
        jc       short setup_error
di3:
        call    setup_rtc_int  ; 22/05/2015 (dsectrpm.s)
        ;
        call    display_disks ; 07/03/2015  (Temporary)
;haltx:
        ; 14/08/2015
        ;call   getch ; 22/02/2015
        sti     ; Enable interrupts (for CPU)
        ; 14/08/2015
        mov     ecx, 0FFFFFFFFh
md_info_msg_wait:
        push    ecx
        mov     al, 1
        mov     ah, [ptty] ; active (current) video page
        call    getc_n
        pop     ecx
        jnz     short md_info_msg_ok
        loop    md_info_msg_wait
md_info_msg_ok:
        ; 30/06/2015
        call    sys_init
        ;
        ;jmp    cpu_reset ; 22/02/2015
```

```
hang:
        ; 23/02/2015
        ;sti                    ; Enable interrupts
        hlt
        ;
        ;nop
        ;; 03/12/2014
        ;; 28/08/2014
        ;mov    ah, 11h
        ;call   getc
        ;jz      _c8
        ;
        ; 23/02/2015
        ; 06/02/2015
        ; 07/09/2014
        xor     ebx, ebx
        mov     bl, [ptty]      ; active_page
        mov     esi, ebx
        shl     si, 1
        add     esi, ttychr
        mov     ax, [esi]
        and     ax, ax
        ;jz     short _c8
        jz      short hang
        mov     word [esi], 0
        cmp     bl, 3           ; Video page 3
        ;jb     short _c8
        jb      short hang
        ;
        ; 02/09/2014
        mov     ah, 0Eh         ; Yellow character
                                ; on black background
        ; 07/09/2014
nxtl:
        push    bx
        ;
        ;xor    bx, bx          ; bl = 0 (video page 0)
                                ; bh = 0 (video mode)
                                ; Retro UNIX 386 v1 - Video Mode 0
                                ; (PC/AT Video Mode 3 - 80x25 Alpha.)
        push    ax
        call    write_tty
        pop     ax
        pop     bx ; 07/09/2014
        cmp     al, 0Dh         ; carriage return (enter)
        ;jne    short _c8
        jne     short hang
        mov     al, 0Ah         ; next line
        jmp     short nxtl

;_c8:
;       ; 25/08/2014
;       cli                             ; Disable interrupts
;       mov     al, [scounter + 1]
;       and     al, al
;       jnz     hang
;       call    rtc_p
;       jmp      hang


        ; 27/08/2014
        ; 20/08/2014
printk:
         ;mov    edi, [scr_row]
pkl:
        lodsb
        or      al, al
        jz      short pkr
        stosw
        jmp     short pkl
pkr:
        retn
```

```
; 25/07/2015
; 14/05/2015 (multi tasking -time sharing- 'clock', x_timer)
; 17/02/2015
; 06/02/2015 (unix386.s)
; 11/12/2014 - 22/12/2014 (dsectrm2.s)
;
; IBM PC-XT Model 286 Source Code - BIOS2.ASM (06/10/85)
;
;-- HARDWARE INT  08 H - ( IRQ LEVEL 0 ) --------------------------------------
;       THIS ROUTINE HANDLES THE TIMER INTERRUPT FROM FROM CHANNEL 0 OF        :
;       THE 8254 TIMER.  INPUT FREQUENCY IS 1.19318 MHZ AND THE DIVISOR        :
;       IS 65536, RESULTING IN APPROXIMATELY 18.2 INTERRUPTS EVERY SECOND.     :
;                                                                               :
;       THE INTERRUPT HANDLER MAINTAINS A COUNT (40:6C) OF INTERRUPTS SINCE    :
;       POWER ON TIME, WHICH MAY BE USED TO ESTABLISH TIME OF DAY.             :
;       THE INTERRUPT HANDLER ALSO DECREMENTS THE MOTOR CONTROL COUNT (40:40)  :
;       OF THE DISKETTE, AND WHEN IT EXPIRES, WILL TURN OFF THE                :
;       DISKETTE MOTOR(s), AND RESET THE MOTOR RUNNING FLAGS.                   :
;       THE INTERRUPT HANDLER WILL ALSO INVOKE A USER ROUTINE THROUGH           :
;       INTERRUPT 1CH AT EVERY TIME TICK.  THE USER MUST CODE A                :
;       ROUTINE AND PLACE THE CORRECT ADDRESS IN THE VECTOR TABLE.             :
;-------------------------------------------------------------------------------
;

timer_int:     ; IRQ 0
;int_08h:      ; Timer
       ; 14/10/2015
       ; Here, we are simulating system call entry (for task switch)
       ; (If multitasking is enabled,
       ; 'clock' procedure may jump to 'sysrelease')
       push    ds
       push    es
       push    fs
       push    gs
       pushad  ; eax, ecx, edx, ebx, esp -before pushad-, ebp, esi, edi
       mov      cx, KDATA
        mov      ds, cx
        mov      es, cx
        mov      fs, cx
        mov      gs, cx
       ;
       mov     ecx, cr3
       mov     [cr3reg], ecx ; save current cr3 register value/content
       ;
       cmp     ecx, [k_page_dir]
       je      short T3
       ;
       ; timer interrupt has been occurred while OS is in user mode
       mov     [u.r0], eax
       mov     ecx, esp
       add     ecx, ESPACE ; 4 * 12 (stack frame)
       mov     [u.sp], ecx ; kernel stack pointer at the start of interrupt
       mov     [u.usp], esp ; kernel stack points to user's registers
       ;
       mov     ecx, [k_page_dir]
       mov     cr3, ecx
T3:
       sti                          ; INTERRUPTS BACK ON
       INC     word [TIMER_LOW]      ; INCREMENT TIME
       JNZ     short T4              ; GO TO TEST_DAY
       INC     word [TIMER_HIGH]     ; INCREMENT HIGH WORD OF TIME
T4:                                 ; TEST_DAY
       CMP     word [TIMER_HIGH],018H; TEST FOR COUNT EQUALING 24 HOURS
       JNZ     short T5              ; GO TO DISKETTE_CTL
       CMP     word [TIMER_LOW],0B0H
       JNZ     short T5              ; GO TO DISKETTE_CTL

;-----  TIMER HAS GONE 24 HOURS
       ;;SUB   AX,AX
       ;MOV    [TIMER_HIGH],AX
       ;MOV    [TIMER_LOW],AX
       sub     eax, eax
       mov     [TIMER_LH], eax
       ;
       MOV     byte [TIMER_OFL],1
```

```
;-----  TEST FOR DISKETTE TIME OUT

T5:
        ; 23/12/2014
        jmp     short T6                ; will be replaced with nop, nop
                                        ; (9090h) if a floppy disk
                                        ; is detected.
        ;mov    al,[CS:MOTOR_COUNT]
        mov     al, [MOTOR_COUNT]
        dec     al
        ;mov    [CS:MOTOR_COUNT], al    ; DECREMENT DISKETTE MOTOR CONTROL
        mov     [MOTOR_COUNT], al
        ;mov    [ORG_MOTOR_COUNT], al
        JNZ     short T6                ; RETURN IF COUNT NOT OUT
        mov     al,0F0h
        ;AND    [CS:MOTOR_STATUS],al    ; TURN OFF MOTOR RUNNING BITS
        and     [MOTOR_STATUS], al
        ;and    [ORG_MOTOR_STATUS], al
        MOV     AL,0CH                  ; bit 3 = enable IRQ & DMA,
                                        ; bit 2 = enable controller
                                        ;       1 = normal operation
                                        ;       0 = reset
                                        ; bit 0, 1 = drive select
                                        ; bit 4-7 = motor running bits
        MOV     DX,03F2H                ; FDC CTL PORT
        OUT     DX,AL                   ; TURN OFF THE MOTOR
T6:
        ;inc    word [CS:wait_count]    ; 22/12/2014 (byte -> word)
                                        ; TIMER TICK INTERRUPT
        ;;inc   word [wait_count] ;;27/02/2015
        ;INT    1CH                     ; TRANSFER CONTROL TO A USER ROUTINE
        ;;;;cli
        ;call   u_timer                 ; TRANSFER CONTROL TO A USER ROUTINE
        call    [x_timer] ; 14/05/2015
T7:
        ; 14/10/2015
        MOV     AL,EOI                  ; GET END OF INTERRUPT MASK
        CLI                             ; DISABLE INTERRUPTS TILL STACK CLEARED
        OUT     INTA00,AL               ; END OF INTERRUPT TO 8259 - 1
        ;
        mov     eax, [cr3reg]           ; previous value/content of cr3 register
        mov     cr3, eax  ; restore cr3 register content
        ;
        popad ; edi, esi, ebp, temp (icrement esp by 4), ebx, edx, ecx, eax
        ;
        pop     gs
        pop     fs
        pop     es
        pop     ds
        iretd   ; return from interrupt

; 14/05/2015 - Multi tasking 'clock' procedure (sys emt)
x_timer:
        dd      u_timer                 ; 14/05/2015
        ;dd     clock

; 14/10/2015
cr3reg: dd 0

        ; 06/02/2015
        ; 07/09/2014
        ; 21/08/2014
u_timer:
;timer_int:     ; IRQ 0
        ; 06/02/2015
        ;push   eax
        ;push   edx
        ;push   ecx
        ;push   ebx
        ;push   ds
        ;push   es
        ;mov    eax, KDATA
        ;mov    ds, ax
        ;mov    es, ax
        inc     dword [tcount]
        mov     ebx, tcountstr + 4
        mov     ax, [tcount]
        mov     ecx, 10
```

```
rp_divtcnt:
        xor     edx, edx
        div     ecx
        add     dl, 30h
        mov     [ebx], dl
        or      ax, ax
        jz      short print_lzero
        dec     ebx
        jmp     short rp_divtcnt
print_lzero:
        cmp     ebx, tcountstr
        jna     short print_tcount
        dec     ebx
        mov     byte [ebx], 30h
        jmp     short print_lzero
print_tcount:
        push    esi
        push    edi
        mov     esi, timer_msg ; Timer interrupt message
        ; 07/09/2014
        mov     bx, 1          ; Video page 1
ptmsg:
        lodsb
        or      al, al
        jz      short ptmsg_ok
        push    esi
        push    bx
         mov     ah,  2Fh ; Green background, white forecolor
        call    write_tty
        pop     bx
        pop     esi
        jmp     short ptmsg
        ;; 27/08/2014
        ;mov      edi, 0B8000h + 0A0h ; Row 1
        ;call   printk
        ;
ptmsg_ok:
        ; 07/09/2014
        xor     dx, dx         ; column 0, row 0
        call    set_cpos       ; set cursor position to 0,0
        ; 23/02/2015
        ; 25/08/2014
        ;mov    ebx, scounter          ; (seconds counter)
        ;dec    byte [ebx+1]           ; (for reading real time clock)
;       dec     byte [scounter+1]
;;      jns     short timer_eoi               ; 0 -> 0FFh ?
;       jns     short u_timer_retn
        ; 26/02/2015
;       call    rtc_p
;       mov     ebx, scounter          ; (seconds counter)
;       mov     byte [ebx+1], 18       ; (18.2 timer ticks per second)
;       dec     byte [ebx]             ; 19+18+18+18+18 (5)
;       jnz     short timer_eoi               ; (109 timer ticks in 5 seconds)
;       jnz     short u_timer_retn ; 06/02/2015
;       mov     byte [ebx], 5
;       inc     byte [ebx+1] ; 19
;;timer_eoi:
;;      mov     al, 20h ; END OF INTERRUPT COMMAND TO 8259
;;      out     20h, al ; 8259 PORT
        ;
;u_timer_retn:  ; 06/02/2015
        pop     edi
        pop     esi
        ;pop    es
        ;pop    ds
        ;pop    ebx
        ;pop    ecx
        ;pop    edx
        ;pop    eax
        ;iret
        retn    ; 06/02/2015
```

```
        ; 28/08/2014
irq0:
         push   dword 0
        jmp     short which_irq
irq1:
         push   dword 1
        jmp     short which_irq
irq2:
         push   dword 2
        jmp     short which_irq
irq3:
        ; 20/11/2015
        ; 24/10/2015
        call    dword [cs:com2_irq3]
        push    dword 3
        jmp     short which_irq
irq4:
        ; 20/11/2015
        ; 24/10/2015
        call    dword [cs:com1_irq4]
         push   dword 4
        jmp     short which_irq
irq5:
         push   dword 5
        jmp     short which_irq
irq6:
         push   dword 6
        jmp     short which_irq
irq7:
         push   dword 7
        jmp     short which_irq
irq8:
         push   dword 8
        jmp     short which_irq
irq9:
         push   dword 9
        jmp     short which_irq
irq10:
         push   dword 10
        jmp     short which_irq
irq11:
         push   dword 11
        jmp     short which_irq
irq12:
         push   dword 12
        jmp     short which_irq
irq13:
         push   dword 13
        jmp     short which_irq
irq14:
         push   dword 14
        jmp     short which_irq
irq15:
         push   dword 15
        ;jmp    short which_irq

        ; 19/10/2015
        ; 29/08/2014
        ; 21/08/2014
which_irq:
        xchg    eax, [esp]  ; 28/08/2014
        push    ebx
        push    esi
        push    edi
        push    ds
        push    es
        ;
        mov     bl, al
        ;
        mov     eax, KDATA
        mov     ds, ax
        mov     es, ax
        ; 19/10/2015
        cld
         ; 27/08/2014
         add      dword [scr_row], 0A0h
        ;
        mov     ah, 17h ; blue (1) background,
                        ; light gray (7) forecolor
```

```
        mov     edi, [scr_row]
        mov     al, 'I'
        stosw
        mov     al, 'R'
        stosw
        mov     al, 'Q'
        stosw
        mov     al, ' '
        stosw
        mov     al, bl
        cmp     al, 10
        jb      short iix
        mov     al, '1'
        stosw
        mov     al, bl
        sub     al, 10
iix:
        add     al, '0'
        stosw
        mov     al, ' '
        stosw
        mov     al, '!'
        stosw
        mov     al, ' '
        stosw
        ; 23/02/2015
        cmp     bl, 7 ; check for IRQ 8 to IRQ 15
        jna     iiret
        mov     al, 20h  ; END OF INTERRUPT COMMAND TO
        out     0A0h, al ; the 2nd 8259
        jmp      iiret
        ;
        ; 22/08/2014
        ;mov    al, 20h ; END OF INTERRUPT COMMAND TO 8259
        ;out    20h, al ; 8259 PORT
        ;
        ;pop    es
        ;pop    ds
        ;pop    edi
        ;pop    esi
        ;pop    ebx
        ;pop    eax
        ;iret

        ; 02/04/2015
        ; 25/08/2014
exc0:
         push   dword 0
         jmp     cpu_except
exc1:
         push   dword 1
         jmp     cpu_except
exc2:
         push   dword 2
         jmp     cpu_except
exc3:
         push   dword 3
         jmp     cpu_except
exc4:
         push   dword 4
         jmp     cpu_except
exc5:
         push   dword 5
         jmp     cpu_except
exc6:
         push   dword 6
         jmp     cpu_except
exc7:
         push   dword 7
         jmp     cpu_except
exc8:
        ; [esp] = Error code
         push   dword 8
         jmp     cpu_except_en
exc9:
         push   dword 9
         jmp     cpu_except
```

```
exc10:
        ; [esp] = Error code
         push   dword 10
         jmp     cpu_except_en
exc11:
        ; [esp] = Error code
         push   dword 11
         jmp     cpu_except_en
exc12:
        ; [esp] = Error code
         push   dword 12
         jmp     cpu_except_en
exc13:
        ; [esp] = Error code
         push   dword 13
         jmp     cpu_except_en
exc14:
        ; [esp] = Error code
         push   dword 14
        jmp     short cpu_except_en
exc15:
         push   dword 15
         jmp     cpu_except
exc16:
         push   dword 16
         jmp     cpu_except
exc17:
        ; [esp] = Error code
         push   dword 17
        jmp     short cpu_except_en
exc18:
         push   dword 18
        jmp     short cpu_except
exc19:
         push   dword 19
        jmp     short cpu_except
exc20:
         push   dword 20
        jmp     short cpu_except
exc21:
         push   dword 21
        jmp     short cpu_except
exc22:
         push   dword 22
        jmp     short cpu_except
exc23:
         push   dword 23
        jmp     short cpu_except
exc24:
         push   dword 24
        jmp     short cpu_except
exc25:
         push   dword 25
        jmp     short cpu_except
exc26:
         push   dword 26
        jmp     short cpu_except
exc27:
         push   dword 27
        jmp     short cpu_except
exc28:
         push   dword 28
        jmp     short cpu_except
exc29:
         push   dword 29
        jmp     short cpu_except
exc30:
         push   dword 30
        jmp     short cpu_except_en
exc31:
         push   dword 31
         jmp     short cpu_except
```

```
        ; 19/10/2015
        ; 19/09/2015
        ; 01/09/2015
        ; 28/08/2015
        ; 28/08/2014
cpu_except_en:
        xchg    eax, [esp+4] ; Error code
        mov     [ss:error_code], eax
        pop     eax  ; Exception number
        xchg    eax, [esp]
                ; eax = eax before exception
                ; [esp] -> exception number
                ; [esp+4] -> EIP to return
        ; 19/10/2015
        ; 19/09/2015
        ; 01/09/2015
        ; 28/08/2015
        ; 29/08/2014
        ; 28/08/2014
        ; 25/08/2014
        ; 21/08/2014
cpu_except:     ; CPU Exceptions
        cld
        xchg    eax, [esp]
                ; eax = Exception number
                ; [esp] = eax (before exception)
        push    ebx
        push    esi
        push    edi
        push    ds
        push    es
        ; 28/08/2015
        mov     bx, KDATA
        mov     ds, bx
        mov     es, bx
        mov     ebx, cr3
        push    ebx ; (*) page directory
        ; 19/10/2015
        cld
        ; 25/03/2015
        mov     ebx, [k_page_dir]
        mov     cr3, ebx
        ; 28/08/2015
        cmp     eax, 0Eh ; 14, PAGE FAULT
        jne     short cpu_except_nfp
        call    page_fault_handler
        and     eax, eax
         jz     iiretp ; 01/09/2015
        mov     eax, 0Eh ; 14
cpu_except_nfp:
        ; 02/04/2015
        mov     ebx, hang
        xchg    ebx, [esp+28]
                ; EIP (points to instruction which faults)
                ; New EIP (hang)
        mov     [FaultOffset], ebx
        mov     dword [esp+32], KCODE ; kernel's code segment
        or      dword [esp+36], 200h ; enable interrupts (set IF)
        ;
        mov     ah, al
        and     al, 0Fh
        cmp     al, 9
        jna     short h1ok
        add     al, 'A'-':'
h1ok:
        shr     ah, 1
        shr     ah, 1
        shr     ah, 1
        shr     ah, 1
        cmp     ah, 9
        jna     short h2ok
        add     ah, 'A'-':'
h2ok:
        xchg    ah, al
        add     ax, '00'
        mov     [excnstr], ax
        ;
        ; 29/08/2014
        mov     eax, [FaultOffset]
```

```
        push    ecx
        push    edx
        mov     ebx, esp
        ; 28/08/2015
        mov     ecx, 16   ; divisor value to convert binary number
                          ; to hexadecimal string
        ;mov    ecx, 10     ; divisor to convert
                            ; binary number to decimal string
b2d1:
        xor     edx, edx
        div     ecx
        push    dx
        cmp     eax, ecx
        jnb     short b2d1
        mov     edi, EIPstr ; EIP value
                            ; points to instruction which faults
        ; 28/08/2015
        mov     edx, eax
b2d2:
        ;add    al, '0'
        mov     al, [edx+hexchrs]
        stosb               ; write hexadecimal digit to its place
        cmp     ebx, esp
        jna     short b2d3
        pop     ax
        mov     dl, al
        jmp     short b2d2
b2d3:
        mov     al, 'h' ; 28/08/2015
        stosb
        mov     al, 20h     ; space
        stosb
        xor     al, al      ; to do it an ASCIIZ string
        stosb
        ;
        pop     edx
        pop     ecx
        ;
        mov     ah, 4Fh ; red (4) background,
                        ; white (F) forecolor
        mov     esi, exc_msg ; message offset
        ;
        jmp     short piemsg
        ;
         ;add    dword [scr_row], 0A0h
         ;mov    edi, [scr_row]
         ;
        ;call   printk
        ;
        ;mov    al, 20h ; END OF INTERRUPT COMMAND TO 8259
        ;out    20h, al ; 8259 PORT
        ;
        ;pop    es
        ;pop    ds
        ;pop    edi
        ;pop    esi
        ;pop    eax
        ;iret

        ; 28/08/2015
        ; 23/02/2015
        ; 20/08/2014
ignore_int:
        push    eax
        push    ebx ; 23/02/2015
        push    esi
        push    edi
        push    ds
        push    es
        ; 28/08/2015
        mov     eax, cr3
        push    eax ; (*) page directory
        ;
        mov     ah, 67h ; brown (6) background,
                        ; light gray (7) forecolor
        mov     esi, int_msg ; message offset
```

```
piemsg:
         ; 27/08/2014
         add     dword [scr_row], 0A0h
         mov     edi, [scr_row]
         ;
        call    printk
        ;
        ; 23/02/2015
        mov     al, 20h  ; END OF INTERRUPT COMMAND TO
        out     0A0h, al ; the 2nd 8259
iiretp: ; 01/09/2015
        ; 28/08/2015
        pop     eax ; (*) page directory
        mov     cr3, eax
        ;
iiret:
        ; 22/08/2014
        mov     al, 20h ; END OF INTERRUPT COMMAND TO 8259
        out     20h, al ; 8259 PORT
        ;
        pop     es
        pop     ds
        pop     edi
        pop     esi
        pop     ebx ; 29/08/2014
        pop     eax
        iretd

        ; 26/02/2015
        ; 07/09/2014
        ; 25/08/2014
rtc_int:        ; Real Time Clock Interrupt (IRQ 8)
        ; 22/08/2014
        push    eax
        push    ebx ; 29/08/2014
        push    esi
        push    edi
        push    ds
        push    es
        ;
        mov     eax, KDATA
        mov     ds, ax
        mov     es, ax
        ;
        ; 25/08/2014
        call    rtc_p
        ;
        ; 22/02/2015 - dsectpm.s
        ; [ source: http://wiki.osdev.org/RTC ]
        ; read status register C to complete procedure
        ;(it is needed to get a next IRQ 8)
        mov     al, 0Ch ;
        out     70h, al ; select register C
        nop
        in      al, 71h ; just throw away contents
        ; 22/02/2015
        MOV     AL,EOI          ; END OF INTERRUPT
        OUT     INTB00,AL       ; FOR CONTROLLER #2
        ;
        jmp     short iiret

        ; 22/08/2014
        ; IBM PC/AT BIOS source code ----- 10/06/85 (bios.asm)
        ; (INT 1Ah)
        ;; Linux (v0.12) source code (main.c) by Linus Torvalds (1991)
time_of_day:
        call    UPD_IPR                 ; WAIT TILL UPDATE NOT IN PROGRESS
         jc      short rtc_retn
        mov     al, CMOS_SECONDS
        call    CMOS_READ
        mov     [time_seconds], al
        mov     al, CMOS_MINUTES
        call    CMOS_READ
        mov     [time_minutes], al
        mov     al, CMOS_HOURS
        call    CMOS_READ
         mov     [time_hours], al
        mov     al, CMOS_DAY_WEEK
        call    CMOS_READ
```

```
        mov     [date_wday], al
        mov     al, CMOS_DAY_MONTH
        call    CMOS_READ
        mov     [date_day], al
        mov     al, CMOS_MONTH
        call    CMOS_READ
        mov     [date_month], al
        mov     al, CMOS_YEAR
        call    CMOS_READ
        mov     [date_year], al
        mov     al, CMOS_CENTURY
        call    CMOS_READ
        mov     [date_century], al
        ;
        mov     al, CMOS_SECONDS
        call    CMOS_READ
        cmp     al, [time_seconds]
        jne     short time_of_day

rtc_retn:
        retn

rtc_p:
        ; 07/09/2014
        ; 29/08/2014
        ; 27/08/2014
        ; 25/08/2014
        ; Print Real Time Clock content
        ;
        ;
        call    time_of_day
        jc      short rtc_retn
        ;
        cmp     al, [ptime_seconds]
         je      short rtc_retn ; 29/08/2014
        ;
        mov     [ptime_seconds], al
        ;
        mov     al, [date_century]
        call    bcd_to_ascii
        mov     [datestr+6], ax
        mov     al, [date_year]
        call    bcd_to_ascii
        mov     [datestr+8], ax
        mov     al, [date_month]
        call    bcd_to_ascii
        mov     [datestr+3], ax
        mov     al, [date_day]
        call    bcd_to_ascii
        mov     [datestr], ax
        ;
        movzx   ebx, byte [date_wday]
        shl     bl, 2
        add     ebx, daytmp
        mov     eax, [ebx]
        mov     [daystr], eax
        ;
        mov     al, [time_hours]
        call    bcd_to_ascii
        mov     [timestr], ax
        mov     al, [time_minutes]
        call    bcd_to_ascii
        mov     [timestr+3], ax
        mov     al, [time_seconds]
        call    bcd_to_ascii
        mov     [timestr+6], ax
        ;
        mov     esi, rtc_msg ; message offset
        ; 23/02/2015
        push    edx
        push    ecx
        ; 07/09/2014
        mov     bx, 2           ; Video page 2
prtmsg:
        lodsb
        or      al, al
        jz      short prtmsg_ok
        push    esi
        push    bx
```

```
         mov    ah, 3Fh ; cyan (6) background,
                        ; white (F) forecolor
        call   write_tty
        pop    bx
        pop    esi
        jmp    short prtmsg
        ;
        ;mov   edi, 0B8000h+0A0h+0A0h ; Row 2
        ;call  printk
prtmsg_ok:
        ; 07/09/2014
        xor    dx, dx          ; column 0, row 0
        call   set_cpos        ; set curspor position to 0,0
        ; 23/02/2015
        pop    ecx
        pop    edx
        retn

; Default IRQ 7 handler against spurious IRQs (from master PIC)
; 25/02/2015 (source: http://wiki.osdev.org/8259_PIC)
default_irq7:
        push   ax
        mov    al, 0Bh  ; In-Service register
        out    20h, al
         jmp short $+2
        jmp short $+2
        in     al, 20h
        and    al, 80h ; bit 7 (is it real IRQ 7 or fake?)
         jz      short irq7_iret ; Fake (spurious) IRQ, do not send EOI
         mov     al, 20h ; EOI
        out    20h, al
irq7_iret:
        pop    ax
        iretd

        ; 22/08/2014
        ; IBM PC/AT BIOS source code ----- 10/06/85 (test4.asm)
CMOS_READ:
        pushf           ; SAVE INTERRUPT ENABLE STATUS AND FLAGS
        rol    al, 1    ; MOVE NMI BIT TO LOW POSITION
        stc             ; FORCE NMI BIT ON IN CARRY FLAG
        rcr    al, 1    ; HIGH BIT ON TO DISABLE NMI - OLD IN CY
        cli             ; DISABLE INTERRUPTS
        out    CMOS_PORT, al  ; ADDRESS LOCATION AND DISABLE NMI
        nop             ; I/O DELAY
        in     al, CMOS_DATA  ; READ THE REQUESTED CMOS LOCATION
        push   ax       ; SAVE (AH) REGISTER VALUE AND CMOS BYTE
        ; 15/03/2015 ; IBM PC/XT Model 286 BIOS source code
                    ; ----- 10/06/85 (test4.asm)
        mov    al, CMOS_SHUT_DOWN*2 ; GET ADDRESS OF DEFAULT LOCATION
        ;mov   al, CMOS_REG_D*2 ; GET ADDRESS OF DEFAULT LOCATION
        rcr    al, 1    ; PUT ORIGINAL NMI MASK BIT INTO ADDRESS
        out    CMOS_PORT, al  ; SET DEFAULT TO READ ONLY REGISTER
        pop    ax       ; RESTORE (AH) AND (AL), CMOS BYTE
        popf
        retn            ; RETURN WITH FLAGS RESTORED

        ; 22/08/2014
        ; IBM PC/AT BIOS source code ----- 10/06/85 (bios2.asm)
UPD_IPR:                               ; WAIT TILL UPDATE NOT IN PROGRESS
        push   ecx
        mov    ecx, 65535              ; SET TIMEOUT LOOP COUNT (= 800)
               ; mov cx, 800
UPD_10:
        mov    al, CMOS_REG_A          ; ADDRESS STATUS REGISTER A
        cli                            ; NO TIMER INTERRUPTS DURING UPDATES
        call   CMOS_READ               ; READ UPDATE IN PROCESS FLAG
        test   al, 80h                 ; IF UIP BIT IS ON ( CANNOT READ TIME )
        jz     short UPD_90            ; EXIT WITH CY= 0 IF CAN READ CLOCK NOW
        sti                            ; ALLOW INTERRUPTS WHILE WAITING
        loop   UPD_10                  ; LOOP TILL READY OR TIMEOUT
        xor    eax, eax                ; CLEAR RESULTS IF ERROR
               ; xor ax, ax
        stc                            ; SET CARRY FOR ERROR
UPD_90:
        pop    ecx                     ; RESTORE CALLERS REGISTER
        cli                            ; INTERRUPTS OFF DURING SET
        retn                           ; RETURN WITH CY FLAG SET
```

```
bcd_to_ascii:
        ; 25/08/2014
        ; INPUT ->
        ;       al = Packed BCD number
        ; OUTPUT ->
        ;       ax  = ASCII word/number
        ;
        ; Erdogan Tan - 1998 (proc_hex) - TRDOS.ASM (2004-2011)
        ;
        db 0D4h,10h                     ; Undocumented inst. AAM
                                     ; AH = AL / 10h
                                     ; AL = AL MOD 10h
        or ax,'00'                      ; Make it ASCII based

         xchg ah, al

        retn


%include 'keyboard.inc' ; 07/03/2015

%include 'video.inc' ; 07/03/2015

setup_rtc_int:
; source: http://wiki.osdev.org/RTC
        cli             ; disable interrupts
        ; default int frequency is 1024 Hz (Lower 4 bits of register A is 0110b or 6)
        ; in order to change this ...
        ; frequency  = 32768 >> (rate-1) --> 32768 >> 5 = 1024
        ; (rate must be above 2 and not over 15)
        ; new rate = 15 --> 32768 >> (15-1) = 2 Hz
        mov     al, 8Ah
        out     70h, al ; set index to register A, disable NMI
        nop
        in      al, 71h ; get initial value of register A
        mov     ah, al
        and     ah, 0F0h
        mov     al, 8Ah
        out     70h, al ; reset index to register A
        mov     al, ah
        or      al, 0Fh ; new rate (0Fh -> 15)
        out     71h, al ; write only our rate to A. Note, rate is the bottom 4 bits.
        ; enable RTC interrupt
        mov     al, 8Bh ;
        out     70h, al ; select register B and disable NMI
        nop
        in      al, 71h ; read the current value of register B
        mov     ah, al  ;
        mov     al, 8Bh ;
        out     70h, al ; set the index again (a read will reset the index to register B)
        mov     al, ah  ;
        or      al, 40h ;
        out     71h, al ; write the previous value ORed with 0x40. This turns on bit 6 of
register B
        sti
        retn

; Write memory information
; Temporary Code
; 06/11/2014
; 14/08/2015
memory_info:
        mov     eax, [memory_size] ; in pages
        push    eax
        shl     eax, 12         ; in bytes
        mov     ebx, 10
        mov     ecx, ebx        ; 10
        mov     esi, mem_total_b_str
        call    bintdstr
        pop     eax
        mov     cl, 7
        mov     esi, mem_total_p_str
        call    bintdstr
        ; 14/08/2015
        call    calc_free_mem
        ; edx = calculated free pages
        ; ecx = 0
        mov     eax, [free_pages]
```

```
        cmp     eax, edx ; calculated free mem value
                ; and initial free mem value are same or not?
        jne     short pmim ; print mem info with '?' if not
        push    edx ; free memory in pages
        ;mov    eax, edx
        shl     eax, 12 ; convert page count
                        ; to byte count
        mov     cl, 10
        mov     esi, free_mem_b_str
        call    bintdstr
        pop     eax
        mov     cl, 7
        mov     esi, free_mem_p_str
        call    bintdstr
pmim:
        mov     esi, msg_memory_info
pmim_nb:
        lodsb
        or      al, al
        jz      short pmim_ok
        push    esi
        xor     ebx, ebx ; 0
                        ; Video page 0 (bl=0)
        mov     ah, 07h ; Black background,
                        ; light gray forecolor
        call    write_tty
        pop     esi
        jmp     short pmim_nb
pmim_ok:
        retn

; Convert binary number to hexadecimal string
; 10/05/2015
; dsectpm.s (28/02/2015)
; Retro UNIX 386 v1 - Kernel v0.2.0.6
; 01/12/2014
; 25/11/2014
;
bytetohex:
        ; INPUT ->
        ;       AL = byte (binary number)
        ; OUTPUT ->
        ;       AX = hexadecimal string
        ;
        push    ebx
        xor     ebx, ebx
        mov     bl, al
        shr     bl, 4
        mov     bl, [ebx+hexchrs]
        xchg    bl, al
        and     bl, 0Fh
        mov     ah, [ebx+hexchrs]
        pop     ebx
        retn

wordtohex:
        ; INPUT ->
        ;       AX = word (binary number)
        ; OUTPUT ->
        ;       EAX = hexadecimal string
        ;
        push    ebx
        xor     ebx, ebx
        xchg    ah, al
        push    ax
        mov     bl, ah
        shr     bl, 4
        mov     al, [ebx+hexchrs]
        mov     bl, ah
        and     bl, 0Fh
        mov     ah, [ebx+hexchrs]
        shl     eax, 16
        pop     ax
        pop     ebx
        jmp     short bytetohex
        ;mov    bl, al
        ;shr    bl, 4
        ;mov    bl, [ebx+hexchrs]
        ;xchg   bl, al
```

```
        ;and    bl, 0Fh
        ;mov    ah, [ebx+hexchrs]
        ;pop    ebx
        ;retn

dwordtohex:
        ; INPUT ->
        ;       EAX = dword (binary number)
        ; OUTPUT ->
        ;       EDX:EAX = hexadecimal string
        ;
        push    eax
        shr     eax, 16
        call    wordtohex
        mov     edx, eax
        pop     eax
        call    wordtohex
        retn

; 10/05/2015
hex_digits:
hexchrs:
        db '0123456789ABCDEF'

; Convert binary number to decimal/numeric string
; 06/11/2014
; Temporary Code
;

bintdstr:
        ; EAX = binary number
        ; ESI = decimal/numeric string address
        ; EBX = divisor (10)
        ; ECX = string length (<=10)
        add     esi, ecx
btdstr0:
        dec     esi
        xor     edx, edx
        div     ebx
        add     dl, 30h
        mov     [esi], dl
        dec     cl
        jz      btdstr2
        or      eax, eax
        jnz     short btdstr0
btdstr1:
        dec     esi
         mov     byte [esi], 20h ; blank space
        dec     cl
        jnz     short btdstr1
btdstr2:
        retn

; Calculate free memory pages on M.A.T.
; 06/11/2014
; Temporary Code
;

calc_free_mem:
        xor     edx, edx
        ;xor    ecx, ecx
        mov     cx, [mat_size] ; in pages
        shl     ecx, 10 ; 1024 dwords per page
        mov     esi, MEM_ALLOC_TBL
cfm0:
        lodsd
        push    ecx
        mov     ecx, 32
cfm1:
        shr     eax, 1
        jnc     short cfm2
        inc     edx
cfm2:
        loop    cfm1
        pop     ecx
        loop    cfm0
        retn
```

```
%include 'diskio.inc'  ; 07/03/2015
%include 'memory.inc'  ; 09/03/2015
%include 'sysdefs.inc' ; 09/03/2015
%include 'u0.s'        ; 15/03/2015
%include 'u1.s'        ; 10/05/2015
%include 'u2.s'        ; 11/05/2015
%include 'u3.s'        ; 10/05/2015
%include 'u4.s'        ; 15/04/2015
%include 'u5.s'        ; 03/06/2015
%include 'u6.s'        ; 31/05/2015
%include 'u7.s'        ; 18/04/2015
%include 'u8.s'        ; 11/06/2015
%include 'u9.s'        ; 29/06/2015

; 07/03/2015
; Temporary Code
display_disks:
        cmp     byte [fd0_type], 0
        jna     short ddsks1
        call    pdskm
ddsks1:
        cmp     byte [fd1_type], 0
        jna     short ddsks2
        mov     byte [dskx], '1'
        call    pdskm
ddsks2:
        cmp     byte [hd0_type], 0
        jna     short ddsk6
        mov     word [dsktype], 'hd'
        mov     byte [dskx], '0'
        call    pdskm
ddsks3:
        cmp     byte [hd1_type], 0
        jna     short ddsk6
        mov     byte [dskx], '1'
        call    pdskm
ddsks4:
        cmp     byte [hd2_type], 0
        jna     short ddsk6
        mov     byte [dskx], '2'
        call    pdskm
ddsks5:
        cmp     byte [hd3_type], 0
        jna     short ddsk6
        mov     byte [dskx], '3'
        call    pdskm
ddsk6:
        mov     esi, nextline
        call    pdskml
pdskm_ok:
        retn
pdskm:
        mov     esi, dsk_ready_msg
pdskml:
        lodsb
        or      al, al
        jz      short pdskm_ok
        push    esi
        xor     ebx, ebx ; 0
                        ; Video page 0 (bl=0)
        mov     ah, 07h ; Black background,
                        ; light gray forecolor
        call    write_tty
        pop     esi
        jmp     short pdskml
```

```
align 16

gdt:    ; Global Descriptor Table
        ; (30/07/2015, conforming cs)
        ; (26/03/2015)
        ; (24/03/2015, tss)
        ; (19/03/2015)
        ; (29/12/2013)
        ;
        dw 0, 0, 0, 0         ; NULL descriptor
        ; 18/08/2014
                        ; 8h kernel code segment, base = 00000000h
        dw 0FFFFh, 0, 9A00h, 00CFh    ; KCODE
                        ; 10h kernel data segment, base = 00000000h
        dw 0FFFFh, 0, 9200h, 00CFh    ; KDATA
                        ; 1Bh user code segment, base address = 400000h ; CORE
        dw 0FBFFh, 0, 0FA40h, 00CFh   ; UCODE
                        ; 23h user data segment, base address = 400000h ; CORE
        dw 0FBFFh, 0, 0F240h, 00CFh   ; UDATA
                        ; Task State Segment
        dw 0067h ; Limit = 103 ; (104-1, tss size = 104 byte,
                               ;  no IO permission in ring 3)
gdt_tss0:
        dw 0  ; TSS base address, bits 0-15
gdt_tss1:
        db 0  ; TSS base address, bits 16-23
                        ; 49h
        db 11101001b ; E9h => P=1/DPL=11/0/1/0/B/1 --> B = Task is busy (1)
        db 0 ; G/0/0/AVL/LIMIT=0000 ; (Limit bits 16-19 = 0000) (G=0, 1 byte)
gdt_tss2:
        db 0  ; TSS base address, bits 24-31

gdt_end:
        ;; 9Ah = 1001 1010b (GDT byte 5) P=1/DPL=00/1/TYPE=1010,
                                      ;; Type= 1 (code)/C=0/R=1/A=0
                ; P= Present, DPL=0=ring 0,  1= user (0= system)
                ; 1= Code C= non-Conforming, R= Readable, A = Accessed

        ;; 92h = 1001 0010b (GDT byte 5) P=1/DPL=00/1/TYPE=1010,
                                      ;; Type= 0 (data)/E=0/W=1/A=0
                ; P= Present, DPL=0=ring 0,  1= user (0= system)
                ; 0= Data E= Expansion direction (1= down, 0= up)
                ; W= Writeable, A= Accessed

        ;; FAh = 1111 1010b (GDT byte 5) P=1/DPL=11/1/TYPE=1010,
                                      ;; Type= 1 (code)/C=0/R=1/A=0
                ; P= Present, DPL=3=ring 3,  1= user (0= system)
                ; 1= Code C= non-Conforming, R= Readable, A = Accessed

        ;; F2h = 1111 0010b (GDT byte 5) P=1/DPL=11/1/TYPE=0010,
                                      ;; Type= 0 (data)/E=0/W=1/A=0
                ; P= Present, DPL=3=ring 3,  1= user (0= system)
                ; 0= Data E= Expansion direction (1= down, 0= up)

        ;; CFh = 1100 1111b (GDT byte 6) G=1/B=1/0/AVL=0, Limit=1111b (3)

                ;; Limit = FFFFFh (=> FFFFFh+1= 100000h) // bits 0-15, 48-51 //
                ;        = 100000h * 1000h (G=1) = 4GB
                ;; Limit = FFBFFh (=> FFBFFh+1= FFC00h) // bits 0-15, 48-51 //
                ;        = FFC00h * 1000h (G=1) = 4GB - 4MB
                ; G= Granularity (1= 4KB), B= Big (32 bit),
                ; AVL= Available to programmers

gdtd:
         dw gdt_end - gdt - 1    ; Limit (size)
         dd gdt                 ; Address of the GDT

        ; 20/08/2014
idtd:
         dw idt_end - idt - 1    ; Limit (size)
         dd idt                 ; Address of the IDT
```

```
Align 4

        ; 21/08/2014
ilist:
        ;times 32 dd cpu_except ; INT 0 to INT 1Fh
        ;
        ; Exception list
        ; 25/08/2014
        dd      exc0    ; 0h,  Divide-by-zero Error
        dd      exc1
        dd      exc2
        dd      exc3
        dd      exc4
        dd      exc5
        dd      exc6    ; 06h,  Invalid Opcode
        dd      exc7
        dd      exc8
        dd      exc9
        dd      exc10
        dd      exc11
        dd      exc12
        dd      exc13   ; 0Dh, General Protection Fault
        dd      exc14   ; 0Eh, Page Fault
        dd      exc15
        dd      exc16
        dd      exc17
        dd      exc18
        dd      exc19
        dd      exc20
        dd      exc21
        dd      exc22
        dd      exc23
        dd      exc24
        dd      exc25
        dd      exc26
        dd      exc27
        dd      exc28
        dd      exc29
        dd      exc30
        dd      exc31
        ; Interrupt list
        dd      timer_int       ; INT 20h
                ;dd     irq0
        dd      keyb_int        ; 27/08/2014
                ;dd     irq1
        dd      irq2
                ; COM2 int
        dd      irq3
                ; COM1 int
        dd      irq4
        dd      irq5
;DISKETTE_INT: ;06/02/2015
        dd      fdc_int         ; 16/02/2015, IRQ 6 handler
                ;dd     irq6
; Default IRQ 7 handler against spurious IRQs (from master PIC)
; 25/02/2015 (source: http://wiki.osdev.org/8259_PIC)
        dd      default_irq7    ; 25/02/2015
                ;dd     irq7
; Real Time Clock Interrupt
        dd      rtc_int         ; 23/02/2015, IRQ 8 handler
                ;dd     irq8    ; INT 28h
        dd      irq9
        dd      irq10
        dd      irq11
        dd      irq12
        dd      irq13
;HDISK_INT1:  ;06/02/2015
        dd      hdc1_int        ; 21/02/2015, IRQ 14 handler
                ;dd     irq14
;HDISK_INT2:  ;06/02/2015
        dd      hdc2_int        ; 21/02/2015, IRQ 15 handler
                ;dd     irq15   ; INT 2Fh
                ; 14/08/2015
        dd      sysent          ; INT 30h (system calls)

        ;dd     ignore_int
        dd      0

;;;
```

```
;;; 11/03/2015
%include 'kybdata.inc'; KEYBOARD (BIOS) DATA
%include 'vidata.inc' ; VIDEO (BIOS) DATA
%include 'diskdata.inc'       ; DISK (BIOS) DATA (initialized)
;;;

; 27/08/2014
scr_row:
       dd 0B8000h + 0A0h + 0A0h + 0A0h ; Row 3
scr_col:
       dd 0

;; 14/08/2015
;;msgPM:
;;      db "Protected mode and paging are ENABLED ... ", 0
msgKVER:
       db "Retro UNIX 386 v1 - Kernel v0.2.0.16 [09/12/2015]", 0

Align 2

; 20/08/2014
  ; /* This is the default interrupt "handler" :-) */
  ; Linux v0.12 (head.s)
int_msg:
       db "Unknown interrupt ! ", 0

Align 2

; 21/08/2014
timer_msg:
       db "IRQ 0 (INT 20h) ! Timer Interrupt : "
tcountstr:
       db "00000 "
       db 0

Align 2
       ; 21/08/2014
exc_msg:
       db "CPU exception ! "
excnstr:              ; 25/08/2014
       db "??h", "  EIP : "
EIPstr: ; 29/08/2014
       times 12 db 0
rtc_msg:
       db "Real Time Clock - "
datestr:
       db "00/00/0000"
       db " "
daystr:
       db "DAY "
timestr:
        db "00:00:00"
       db " "
       db 0

daytmp:
       ; 28/02/2015
       db "??? SUN MON TUE WED THU FRI SAT "

ptime_seconds: db 0FFh

       ; 23/02/2015
       ; 25/08/2014
;scounter:
;      db 5
;      db 19

; 05/11/2014
msg_out_of_memory:
       db     07h, 0Dh, 0Ah
        db     'Insufficient memory ! (Minimum 2 MB memory is needed.)'
       db     0Dh, 0Ah, 0
       ;
setup_error_msg:
       db 0Dh, 0Ah
       db 'Disk Setup Error!'
       db 0Dh, 0Ah,0
```

```
; 02/09/2014 (Retro UNIX 386 v1)
;crt_ulc : db 0 ; upper left column (for scroll)
;         db 0 ; upper left row (for scroll)

;crt_lrc : db 79 ; lower right column (for scroll)
;         db 24 ; lower right row (for scroll)


; 06/11/2014 (Temporary Data)
; Memory Information message
; 14/08/2015
msg_memory_info:
        db      07h
        db      0Dh, 0Ah
        ;db     "MEMORY ALLOCATION INFO", 0Dh, 0Ah, 0Dh, 0Ah
        db      "Total memory : "
mem_total_b_str: ; 10 digits
        db      "0000000000 bytes", 0Dh, 0Ah
        db      "                ", 20h, 20h, 20h
mem_total_p_str: ; 7 digits
        db      "0000000 pages", 0Dh, 0Ah
        db      0Dh, 0Ah
        db      "Free memory  : "
free_mem_b_str:  ; 10 digits
        db      "?????????? bytes", 0Dh, 0Ah
        db      "                ", 20h, 20h, 20h
free_mem_p_str:  ; 7 digits
        db      "??????? pages", 0Dh, 0Ah
        db      0Dh, 0Ah, 0

dsk_ready_msg:
        db      0Dh, 0Ah
dsktype:
        db      'fd'
dskx:
        db      '0'
        db      20h
        db      'is READY ...'
        db      0
nextline:
        db      0Dh, 0Ah, 0

; KERNEL - SYSINIT Messages
; 24/08/2015
; 13/04/2015 - (Retro UNIX 386 v1 Beginning)
; 14/07/2013
;kernel_init_err_msg:
;       db 0Dh, 0Ah
;       db 07h
;       db 'Kernel initialization ERROR !'
;       db 0Dh, 0Ah, 0
; 24/08/2015
;;; (temporary kernel init message has been removed
;;;  from 'sys_init' code)
;kernel_init_ok_msg:
;       db 0Dh, 0Ah
;       db 07h
;       db 'Welcome to Retro UNIX 386 v1 Operating System !'
;       db 0Dh, 0Ah
;        db 'by Erdogan Tan - 09/12/2015 (v0.2.0.16)'
;       db 0Dh, 0Ah, 0
panic_msg:
        db 0Dh, 0Ah, 07h
        db 'ERROR: Kernel Panic !'
        db 0Dh, 0Ah, 0
etc_init_err_msg:
        db 0Dh, 0Ah
        db 07h
        db 'ERROR: /etc/init !?'
        db 0Dh, 0Ah, 0

; 10/05/2015
badsys_msg:
        db 0Dh, 0Ah
        db 07h
        db 'Invalid System Call !'
        db 0Dh, 0Ah
        db 'EAX: '
```

```
bsys_msg_eax:
        db '00000000h'
        db 0Dh, 0Ah
        db 'EIP: '
bsys_msg_eip:
        db '00000000h'
        db 0Dh, 0Ah, 0

BSYS_M_SIZE equ $ - badsys_msg


align 2

; EPOCH Variables
; 13/04/2015 - Retro UNIX 386 v1 Beginning
; 09/04/2013 epoch variables
; Retro UNIX 8086 v1 Prototype: UNIXCOPY.ASM, 10/03/2013
;
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

; 04/11/2014 (Retro UNIX 386 v1)
mem_1m_1k:    dw 0  ; Number of contiguous KB between
                     ; 1 and 16 MB, max. 3C00h = 15 MB.
mem_16m_64k: dw 0  ; Number of contiguous 64 KB blocks
                  ;   between 16 MB and 4 GB.

; 12/11/2014 (Retro UNIX 386 v1)
boot_drv:     db 0 ; boot drive number (physical)
; 24/11/2014
drv:         db 0
last_drv:     db 0 ; last hdd
hdc:          db 0  ; number of hard disk drives
                    ; (present/detected)
;
; 24/11/2014 (Retro UNIX 386 v1)
; Physical drive type & flags
fd0_type:     db 0  ; floppy drive type
fd1_type:     db 0  ; 4 = 1.44 Mb, 80 track, 3.5" (18 spt)
                    ; 6 = 2.88 Mb, 80 track, 3.5" (36 spt)
                    ; 3 = 720 Kb, 80 track, 3.5" (9 spt)
                    ; 2 = 1.2 Mb, 80 track, 5.25" (15 spt)
                    ; 1 = 360 Kb, 40 track, 5.25" (9 spt)
hd0_type:     db 0  ; EDD status for hd0 (bit 7 = present flag)
hd1_type:     db 0  ; EDD status for hd1 (bit 7 = present flag)
hd2_type:     db 0  ; EDD status for hd2 (bit 7 = present flag)
hd3_type:     db 0  ; EDD status for hd3 (bit 7 = present flag)
                    ; bit 0 - Fixed disk access subset supported
                    ; bit 1 - Drive locking and ejecting
                    ; bit 2 - Enhanced disk drive support
                    ; bit 3 = Reserved (64 bit EDD support)
                    ; (If bit 0 is '1' Retro UNIX 386 v1
                    ; will interpret it as 'LBA ready'!)

; 11/03/2015 - 10/07/2015
drv.cylinders: dw 0,0,0,0,0,0,0
drv.heads:     dw 0,0,0,0,0,0,0
drv.spt:       dw 0,0,0,0,0,0,0
drv.size:      dd 0,0,0,0,0,0,0
drv.status:    db 0,0,0,0,0,0,0
drv.error:     db 0,0,0,0,0,0,0
```

```
;

align 16

bss_start:

ABSOLUTE bss_start

       ; 11/03/2015
       ; Interrupt Descriptor Table (20/08/2014)
idt:
       resb   64*8 ; INT 0 to INT 3Fh
idt_end:

;alignb 4

task_state_segment:
       ; 24/03/2015
tss.link:   resw 1
            resw 1
; tss offset 4
tss.esp0:   resd 1
tss.ss0:    resw 1
            resw 1
tss.esp1:   resd 1
tss.ss1:    resw 1
            resw 1
tss.esp2:   resd 1
tss.ss2:    resw 1
            resw 1
; tss offset 28
tss.CR3:    resd 1
tss.eip:    resd 1
tss.eflags: resd 1
; tss offset 40
tss.eax:    resd 1
tss.ecx:    resd 1
tss.edx:    resd 1
tss.ebx:    resd 1
tss.esp:    resd 1
tss.ebp:    resd 1
tss.esi:    resd 1
tss.edi:    resd 1
; tss offset 72
tss.ES:     resw 1
            resw 1
tss.CS:     resw 1
            resw 1
tss.SS:     resw 1
            resw 1
tss.DS:     resw 1
            resw 1
tss.FS:     resw 1
            resw 1
tss.GS:     resw 1
            resw 1
tss.LDTR:   resw 1
            resw 1
; tss offset 100
            resw 1
tss.IOPB:   resw 1
; tss offset 104
tss_end:

k_page_dir:  resd 1 ; Kernel's (System) Page Directory address
                 ;   (Physical address = Virtual address)
memory_size: resd 1 ; memory size in pages
free_pages:  resd 1 ; number of free pages
next_page:   resd 1 ; offset value in M.A.T. for
                 ;   first free page search
last_page:   resd 1 ; offset value in M.A.T. which
                 ;   next free page search will be
                 ; stopped after it. (end of M.A.T.)
first_page:  resd 1 ;   offset value in M.A.T. which
                 ; first free page search
                 ;   will be started on it. (for user)
mat_size:    resd 1 ; Memory Allocation Table size in pages

;;;
```

```
; 02/09/2014 (Retro UNIX 386 v1)
; 04/12/2013 (Retro UNIX 8086 v1)
CRT_START:   resw 1     ; starting address in regen buffer
                        ; NOTE: active page only
cursor_posn: resw 8     ; cursor positions for video pages
active_page:
ptty:       resb 1      ; current tty
; 01/07/2015
ccolor:     resb 1      ; current color attributes ('sysmsg')
; 26/10/2015
; 07/09/2014
ttychr:      resw ntty+2  ; Character buffer (multiscreen)

; 21/08/2014
tcount:     resd 1

; 18/05/2015 (03/06/2013 - Retro UNIX 8086 v1 feature only!)
p_time:      resd 1     ; present time (for systime & sysmdate)

; 18/05/2015 (16/08/2013 - Retro UNIX 8086 v1 feature only !)
; (open mode locks for pseudo TTYs)
; [ major tty locks (return error in any conflicts) ]
ttyl:        resw ntty+2 ; opening locks for TTYs.

; 15/04/2015 (Retro UNIX 386 v1)
; 22/09/2013 (Retro UNIX 8086 v1)
wlist:       resb ntty+2 ; wait channel list (0 to 9 for TTYs)
; 15/04/2015 (Retro UNIX 386 v1)
;; 12/07/2014 -> sp_init set comm. parameters as 0E3h
;; 0 means serial port is not available
;;comprm: ; 25/06/2014
com1p:       resb 1  ;;0E3h
com2p:       resb 1  ;;0E3h

; 17/11/2015
; request for response (from the terminal)
req_resp:     resw 1
; 07/11/2015
ccomport:    resb 1 ; current COM (serial) port
                  ; (0= COM1, 1= COM2)
; 09/11/2015
comqr:      resb 1 ; 'query or response' sign (u9.s, 'sndc')
; 07/11/2015
rchar:      resw 1 ; last received char for COM 1 and COM 2
schar:      resw 1 ; last sent char for COM 1 and COM 2

; 23/10/2015
; SERIAL PORTS - COMMUNICATION MODES
; (Retro UNIX 386 v1 feature only!)
; 0 - command mode (default/initial mode)
; 1 - terminal mode (Retro UNIX 386 v1 terminal, ascii chars)
;;; communication modes for futre versions:
; // 2 - keyboard mode (ascii+scancode input)
; // 3 - mouse mode
; // 4 - device control (output) mode
; VALID COMMANDS for current version:
;       'LOGIN'
;  Login request: db 0FFh, 'LOGIN', 0
;        ("Retro UNIX 386 v1 terminal requests login")
;  Login response: db 0FFh, 'login', 0
;        ("login request accepted, wait for login prompt")
; When a login requests is received and acknowledged (by
; serial port interrupt handler (communication procedure),
; Retro UNIX 386 v1 operating system will start terminal mode
; (login procedure) by changing comm. mode to 1 (terminal mode)
; and then running 'etc/getty' for tty8 (COM1) or tty9 (COM2)
;
; 'sys connect' system call is used to change communication mode
; except 'LOGIN' command which is used to start terminal mode
; by using (COM port) terminal.

;com1own:     resb 1 ; COM1 owner (u.uno)
;com2own:     resb 1 ; COM2 owner (u.uno)
;com1mode:    resb 1 ; communication mode for COM1
;com1com:     resb 1 ; communication command for COM1
;com2mode:    resb 1 ; communication mode for COM1
;com2com      resb 1 ; communication command for COM1
;com1cbufp:   resb 8 ; COM1 command buffer char pointer
;com2cbufp:   resb 8 ; COM2 command buffer char pointer
```

```
;com1cbuf:    resb 8 ; COM2 command buffer
;com2cbuf:    resb 8 ; COM2 command buffer

; 22/08/2014 (RTC)
; (Packed BCD)
time_seconds: resb 1
time_minutes: resb 1
time_hours:   resb 1
date_wday:    resb 1
date_day:     resb 1
date_month:   resb 1
date_year:    resb 1
date_century: resb 1

%include 'diskbss.inc'; UNINITIALIZED DISK (BIOS) DATA

;;; Real Mode Data (10/07/2015 - BSS)

;alignb 2

%include 'ux.s' ; 12/04/2015 (unix system/user/process data)

;; Memory (swap) Data (11/03/2015)
; 09/03/2015
swpq_count: resw 1 ; count of pages on the swap que
swp_drv:    resd 1 ; logical drive description table address of the swap drive/disk
swpd_size:  resd 1 ; size of swap drive/disk (volume) in sectors (512 bytes).

swpd_free:  resd 1 ; free page blocks (4096 bytes) on swap disk/drive (logical)
swpd_next:  resd 1 ; next free page block
swpd_last:  resd 1 ; last swap page block

alignb 4

; 10/07/2015
; 28/08/2014
error_code:    resd 1
; 29/08/2014
FaultOffset:   resd 1
; 21/09/2015
PF_Count:      resd 1  ; total page fault count
                       ; (for debugging - page fault analyze)
                       ; 'page _fault_handler' (memory.inc)
                       ; 'sysgeterr' (u9.s)
;; 21/08/2015
;;buffer: resb (nbuf*520) ;; sysdefs.inc, ux.s
;; ((NOTE: nbuf = 6, buffer r/w problem/bug here !? when nbuf > 4))

bss_end:

; 27/12/2013
_end:  ; end of kernel code (and read only data, just before bss)
```

```
; Retro UNIX 386 v1 Kernel - DISKINIT.INC
; Last Modification: 10/07/2015

; DISK I/O SYSTEM INITIALIZATION - Erdogan Tan (Retro UNIX 386 v1 project)

; ///////// DISK I/O SYSTEM STRUCTURE INITIALIZATION ///////////////

       ; 10/12/2014 - 02/02/2015 - dsectrm2.s
;L0:
       ; 12/11/2014 (Retro UNIX 386 v1 - beginning)
       ; Detecting disk drives... (by help of ROM-BIOS)
       mov     dx, 7Fh
L1:
       inc     dl
       mov     ah, 41h ; Check extensions present
                      ; Phoenix EDD v1.1 - EDD v3
       mov     bx, 55AAh
       int     13h
       jc      short L2
       cmp     bx, 0AA55h
       jne     short L2
       inc     byte [hdc]     ; count of hard disks (EDD present)
        mov     [last_drv], dl  ; last hard disk number
       mov     bx, hd0_type - 80h
       add     bx, dx
       mov     [bx], cl ; Interface support bit map in CX
                       ; Bit 0 - 1, Fixed disk access subset ready
                       ; Bit 1 - 1, Drv locking and ejecting ready
                       ; Bit 2 - 1, Enhanced Disk Drive Support
                        ;            (EDD) ready (DPTE ready)
                       ; Bit 3 - 1, 64bit extensions are present
                        ;            (EDD-3)
                       ; Bit 4 to 15 - 0, Reserved
       cmp     dl, 83h ; drive number < 83h
       jb      short L1
L2:
       ; 23/11/2014
       ; 19/11/2014
       xor     dl, dl  ; 0
       mov     esi, fd0_type
L3:
       ; 14/01/2015
       mov     [drv], dl
       ;
       mov     ah, 08h ; Return drive parameters
       int     13h
       jc      short L4
               ; BL = drive type (for floppy drives)
               ; DL = number of floppy drives
               ;
               ; ES:DI = Address of DPT from BIOS
               ;
       mov     [esi], bl ;  Drive type
                      ; 4 = 1.44 MB, 80 track, 3 1/2"
       ; 14/01/2015
       call    set_disk_parms
       ; 10/12/2014
       cmp     esi, fd0_type
       ja      short L4
       inc     esi ; fd1_type
       mov     dl, 1
       jmp     short L3
L4:
       ; Older BIOS (INT 13h, AH = 48h is not available)
       mov     dl, 7Fh
       ; 24/12/2014 (Temporary)
       cmp     byte [hdc], 0 ; EDD present or not ?
        ja      L10       ; yes, all fixed disk operations
                        ; will be performed according to
                        ; present EDD specification
L6:
       inc     dl
        mov     [drv], dl
        mov     [last_drv], dl ; 14/01/2015
       mov     ah, 08h ; Return drive parameters
       int     13h     ; (conventional function)
        jc      L13    ; fixed disk drive not ready
        mov     [hdc], dl ; number of drives
       ;; 14/01/2013
```

```
        ;;push  cx
        call    set_disk_parms
        ;;pop   cx
        ;
        ;;and   cl, 3Fh  ; sectors per track (bits 0-6)
        mov     dl, [drv]
        mov     bx, 65*4 ; hd0 parameters table (INT 41h)
        cmp     dl, 80h
        jna     short L7
        add     bx, 5*4  ; hd1 parameters table (INT 46h)
L7:
        xor     ax, ax
        mov     ds, ax
        mov     si, [bx]
        mov     ax, [bx+2]
        mov     ds, ax
        cmp     cl, [si+FDPT_SPT] ; sectors per track
        jne     L12 ; invalid FDPT
        mov     di, HD0_DPT
        cmp     dl, 80h
        jna     short L8
        mov     di, HD1_DPT
L8:
        ; 30/12/2014
        mov     ax, DPT_SEGM
        mov     es, ax
        ; 24/12/2014
        mov     cx, 8
        rep     movsw  ; copy 16 bytes to the kernel's DPT location
        mov     ax, cs
        mov     ds, ax
        ; 02/02/2015
        mov     cl, [drv]
        mov     bl, cl
        mov     ax, 1F0h
        and     bl, 1
        jz      short L9
        shl     bl, 4
        sub     ax, 1F0h-170h
L9:
        stosw   ; I/O PORT Base Address (1F0h, 170h)
        add     ax, 206h
        stosw   ; CONTROL PORT Address (3F6h, 376h)
        mov     al, bl
        add     al, 0A0h
        stosb   ; Device/Head Register upper nibble
        ;
        inc     byte [drv]
        mov     bx, hd0_type - 80h
        add     bx, cx
        or      byte [bx], 80h  ; present sign (when lower nibble is 0)
        mov     al, [hdc]
        dec     al
        jz      L13
        cmp     dl, 80h
        jna     L6
        jmp     L13
L10:
        inc     dl
        ; 25/12/2014
        mov     [drv], dl
        mov     ah, 08h ; Return drive parameters
        int     13h     ; (conventional function)
        jc      L13
        ; 14/01/2015
        mov     dl, [drv]
        push    dx
        push    cx
        call    set_disk_parms
        pop     cx
        pop     dx
        ;
        mov     esi, _end ; 30 byte temporary buffer address
                        ; at the '_end' of kernel.
        mov     word [esi], 30
        mov     ah, 48h  ; Get drive parameters (EDD function)
        int     13h
        jc      L13
        ; 14/01/2015
```

```
        sub     ebx, ebx
        mov     bl, dl
        sub     bl, 80h
        add     ebx, hd0_type
        mov     al, [ebx]
        or      al, 80h
        mov     [ebx], al
        sub     ebx, hd0_type - 2 ; 15/01/2015
        add     ebx, drv.status
        mov     [ebx], al
        mov     eax, [esi+16]
        ; 28/02/2015
        and     eax, eax
        jz      short L10_A0h
                       ; 'CHS only' disks on EDD system
                       ;  are reported with ZERO disk size
        sub     ebx, drv.status
        shl     ebx, 2
        add     ebx, drv.size ; disk size (in sectors)
        mov     [ebx], eax
L10_A0h: ; Jump here to fix a ZERO (LBA) disk size problem
         ; for CHS disks (28/02/2015)
        ; 30/12/2014
        mov     di, HD0_DPT
        mov     al, dl
        and     ax, 3
        shl     al, 5 ; *32
        add     di, ax
        mov     ax, DPT_SEGM
        mov     es, ax
        ;
        mov     al, ch  ; max. cylinder number (bits 0-7)
        mov     ah, cl
        shr     ah, 6   ; max. cylinder number (bits 8-9)
        inc     ax      ; logical cylinders (limit 1024)
        stosw
        mov     al, dh  ; max. head number
        inc     al
        stosb           ; logical heads (limits 256)
        mov     al, 0A0h ; Indicates translated table
        stosb
        mov     al, [si+12]
        stosb            ; physical sectors per track
        xor     ax, ax
        ;dec    ax       ; 02/01/2015
        stosw            ; precompensation (obsolete)
        ;xor    al, al   ; 02/01/2015
        stosb            ; reserved
        mov     al, 8    ; drive control byte
                          ; (do not disable retries,
                         ; more than 8 heads)
        stosb
        mov     ax, [si+4]
        stosw            ; physical number of cylinders
        ;push   ax       ; 02/01/2015
        mov     al, [si+8]
        stosb            ; physical num. of heads (limit 16)
        sub     ax, ax
        ;pop    ax       ; 02/01/2015
        stosw            ; landing zone (obsolete)
        mov     al, cl   ; logical sectors per track (limit 63)
        and     al, 3Fh
        stosb
        ;sub    al, al   ; checksum
        ;stosb
        ;
        add     si, 26    ; (BIOS) DPTE address pointer
        lodsw
        push    ax       ; (BIOS) DPTE offset
        lodsw
        push    ax       ; (BIOS) DPTE segment
        ;
        ; checksum calculation
        mov     si, di
        push    es
        pop     ds
        ;mov    cx, 16
        mov     cx, 15
        sub     si, cx
```

```
        xor     ah, ah
        ;del    cl
L11:
        lodsb
        add     ah, al
        loop    L11
        ;
        mov     al, ah
        neg     al      ; -x+x = 0
        stosb           ; put checksum in byte 15 of the tbl
        ;
        pop     ds      ; (BIOS) DPTE segment
        pop     si      ; (BIOS) DPTE offset
        ;
        ; 23/02/2015
        push    di
        ; ES:DI points to DPTE (FDPTE) location
        ;mov    cx, 8
        mov     cl, 8
        rep     movsw
        ;
        ; 23/02/2015
        ; (P)ATA drive and LBA validation
        ; (invalidating SATA drives and setting
        ; CHS type I/O for old type fixed disks)
        pop     bx
        mov     ax, cs
        mov     ds, ax
        mov     ax, [es:bx]
        cmp     ax, 1F0h
        je      short L11a
        cmp     ax, 170h
        je      short L11a
        ; invalidation
        ; (because base port address is not 1F0h or 170h)
        xor     bh, bh
        mov     bl, dl
        sub     bl, 80h
        mov     byte [bx+hd0_type], 0 ; not a valid disk drive !
         or      byte [bx+drv.status+2], 0F0h ; (failure sign)
        jmp     short L11b
L11a:
        ; LBA validation
        mov     al, [es:bx+4] ; Head register upper nibble
        test    al, 40h ; LBA bit (bit 6)
        jnz     short L11b ; LBA type I/O is OK! (E0h or F0h)
        ; force CHS type I/O for this drive (A0h or B0h)
        sub     bh, bh
        mov     bl, dl
        sub     bl, 80h ; 26/02/2015
         and     byte [bx+drv.status+2], 0FEh ; clear bit 0
                              ; bit 0 = LBA ready bit
        ; 'diskio' procedure will check this bit !
L11b:
        cmp     dl, [last_drv] ; 25/12/2014
         jnb     short L13
         jmp     L10
L12:
        ; Restore data registers
        mov     ax, cs
        mov     ds, ax
L13:
        ; 13/12/2014
        push    cs
        pop     es
L14:
        mov     ah, 11h
        int     16h
        jz      short L15 ; no keys in keyboard buffer
        mov     al, 10h
        int     16h
        jmp     short L14
L15:
; //////
        ; 24/11/2014
        ; 19/11/2014
        ; 14/11/2014
        ; Temporary code for disk searching code check
        ;
```

```
        ; This code will show existing (usable) drives and also
        ; will show EDD interface support status for hard disks
        ; (If status bit 7 is 1, Identify Device info is ready,
        ; no need to get it again in protected mode...)
        ;
        ; 13/11/2014
        mov     bx, 7
        mov     ah, 0Eh
        mov     al, [fd0_type]
        and     al, al
        jz      short L15a
        mov     dl, al
        mov     al, 'F'
        int     10h
        mov     al, 'D'
        int     10h
        mov     al, '0'
        int     10h
        mov     al, ' '
        int     10h
        call    L15c
        mov     al, ' '
        int     10h
        ;
        mov     al, [fd1_type]
        and     al, al
        jz      short L15a
        mov     dl, al
        mov     al, 'F'
        int     10h
        mov     al, 'D'
        int     10h
        mov     al, '1'
        int     10h
        mov     al, ' '
        int     10h
        call    L15c
        mov     al, ' '
        int     10h
        mov     al, ' '
        int     10h
L15a:
        mov     al, [hd0_type]
        and     al, al
        jz      short L15b
        mov     dl, al
        mov     al, 'H'
        int     10h
        mov     al, 'D'
        int     10h
        mov     al, '0'
        int     10h
        mov     al, ' '
        int     10h
        call    L15c
        mov     al, ' '
        int     10h
        ;
        mov     al, [hd1_type]
        and     al, al
        jz      short L15b
        mov     dl, al
        mov     al, 'H'
        int     10h
        mov     al, 'D'
        int     10h
        mov     al, '1'
        int     10h
        mov     al, ' '
        int     10h
        call    L15c
        mov     al, ' '
        int     10h
        ;
        mov     al, [hd2_type]
        and     al, al
        jz      short L15b
        mov     dl, al
        mov     al, 'H'
```

```
        int     10h
        mov     al, 'D'
        int     10h
        mov     al, '2'
        int     10h
        mov     al, ' '
        int     10h
        call    L15c
        mov     al, ' '
        int     10h
        ;
        mov     al, [hd3_type]
        and     al, al
        jz      short L15b
        mov     dl, al
        mov     al, 'H'
        int     10h
        mov     al, 'D'
        int     10h
        mov     al, '3'
        int     10h
        mov     al, ' '
        int     10h
        call    L15c
        mov     al, ' '
        int     10h
        ;
L15b:
        mov     al, 0Dh
        int     10h
        mov     al, 0Ah
        int     10h
        ;;xor   ah, ah
        ;;int   16h
        ;
         jmp     L16  ; jmp short L16
         ;
L15c:
        mov     dh, dl
        shr     dh, 4
        add     dh, 30h
        and     dl, 15
        add     dl, 30h
        mov     al, dh
        int     10h
        mov     al, dl
        int     10h
        retn
        ;
        ; end of temporary code for disk searching code check

; //////

set_disk_parms:
        ; 10/07/2015
        ; 14/01/2015
        ;push   ebx
        sub     ebx, ebx
        mov     bl, [drv]
        cmp     bl, 80h
        jb      short sdp0
        sub     bl, 7Eh
sdp0:
        add     ebx, drv.status
        mov     byte [ebx], 80h ; 'Present' flag
        ;
        mov     al, ch ; last cylinder (bits 0-7)
        mov     ah, cl ;
        shr     ah, 6  ; last cylinder (bits 8-9)
        sub     ebx, drv.status
        shl     bl, 1
        add     ebx, drv.cylinders
        inc     ax  ; convert max. cyl number to cyl count
        mov     [ebx], ax
        push    ax ; ** cylinders
        sub     ebx, drv.cylinders
        add     ebx, drv.heads
        xor     ah, ah
        mov     al, dh ; heads
```

```
        inc     ax
        mov     [ebx], ax
         sub     ebx, drv.heads
         add     ebx, drv.spt
        xor     ch, ch
        and     cl, 3Fh ; sectors (bits 0-6)
        mov     [ebx], cx
         sub     ebx, drv.spt
        shl     ebx, 1
        add     ebx, drv.size ; disk size (in sectors)
        ; LBA size = cylinders * heads * secpertrack
        mul     cx
        mov     dx, ax  ; heads*spt
        pop     ax ; ** cylinders
        dec     ax ; 1 cylinder reserved (!?)
        mul     dx ; cylinders * (heads*spt)
        mov     [ebx], ax
        mov     [ebx+2], dx
        ;
        ;pop    ebx
        retn

;align 2

;cylinders :  dw 0, 0, 0, 0, 0, 0
;heads    :  dw 0, 0, 0, 0, 0, 0
;spt      :  dw 0, 0, 0, 0, 0, 0
;disk_size :  dd 0, 0, 0, 0, 0, 0

;last_drv:
;       db  0
;drv_status:
;       db  0,0,0,0,0,0
;       db 0


; End Of DISK I/O SYSTEM STRUCTURE INITIALIZATION /// 06/02/2015

L16:
```

```
; Retro UNIX 386 v1 Kernel - KEYBOARD.INC
; Last Modification: 17/10/2015
;                   (Keyboard Data is in 'KYBDATA.INC')
;
; ///////// KEYBOARD FUNCTIONS (PROCEDURES) ///////////////

; 30/06/2015
; 11/03/2015
; 28/02/2015
; 25/02/2015
; 20/02/2015
; 18/02/2015
; 03/12/2014
; 07/09/2014
; KEYBOARD INTERRUPT HANDLER
; (kb_int - Retro UNIX 8086 v1 - U0.ASM, 30/06/2014)

;getch:
;       ; 18/02/2015
;       ; This routine will be replaced with Retro UNIX 386
;       ; version of Retro UNIX 8086 getch (tty input)
;       ; routine, later... (multi tasking ability)
;       ; 28/02/2015
;       sti     ; enable interrupts
;       ;
;       ;push   esi
;       ;push   ebx
;       ;xor    ebx, ebx
;       ;mov    bl, [ptty]  ; active_page
;       ;mov    esi, ebx
;       ;shl    si, 1
;       ;add    esi, ttychr
;getch_1:
;       ;mov    ax, [esi]
;       mov     ax, [ttychr] ; video page 0 (tty0)
;       and     ax, ax
;       jz      short getch_2
;       mov     word [ttychr], 0
;       ;mov    word [esi], 0
;       ;pop    ebx
;       ;pop    esi
;       retn
;getch_2:
;       hlt     ; not proper for multi tasking!
;               ; (temporary halt for now)
;               ; 'sleep' on tty
;               ; will (must) be located here
;       nop
;       jmp     short getch_1

keyb_int:
        ; 30/06/2015
        ; 25/02/2015
        ; 20/02/2015
        ; 03/12/2014 (getc_int - INT 16h modifications)
        ; 07/09/2014 - Retro UNIX 386 v1
        ; 30/06/2014
        ; 10/05/2013
        ; Retro Unix 8086 v1 feature only!
        ; 03/03/2014

        push    ds
        push    ebx
        push    eax
        ;
        mov     ax, KDATA
        mov     ds, ax
        ;
        pushfd
        push    cs
        call    kb_int  ; int_09h
        ;
        mov     ah, 11h  ; 03/12/2014
        ;call   getc
        call    int_16h ; 30/06/2015
        jz      short keyb_int4
        ;
        mov     ah, 10h  ; 03/12/2014
        ;call   getc
```

```
        call    int_16h ; 30/06/2015
        ;
        ; 20/02/2015
         movzx   ebx, byte [ptty]  ; active_page
        ;
        and     al, al
        jnz     short keyb_int1
        ;
        cmp     ah, 68h ; ALT + F1 key
        jb      short keyb_int1
        cmp     ah, 6Fh  ; ALT + F8 key
        ja      short keyb_int1
        ;
        mov     al, bl
        add     al, 68h
        cmp     al, ah
        je      short keyb_int0
        mov     al, ah
        sub     al, 68h
        call    tty_sw
        ;movzx ebx, [ptty]  ; active_page
keyb_int0: ; 30/06/2015
        xor     ax, ax
keyb_int1:
        shl     bl, 1
        add     ebx, ttychr
        ;
        or      ax, ax
        jz      short keyb_int2
        ;
        cmp     word [ebx], 0
         ja      short keyb_int3
keyb_int2:
         mov    [ebx], ax  ; Save ascii code
                             ; and scan code of the character
                             ; for current tty (or last tty
                             ; just before tty switch).
keyb_int3:
         mov     al, [ptty]
        call    wakeup
        ;
keyb_int4:
        pop     eax
        pop     ebx
        pop     ds
        iret

; 18/02/2015
; REMINDER: Only 'keyb_int' (IRQ 9) must call getc.
; 'keyb_int' always handles 'getc' at 1st and puts the
; scancode and ascii code of the character
; in the tty input (ttychr) buffer.
; Test procedures must call 'getch' for tty input
; otherwise, 'getc' will not be able to return to the caller
; due to infinite (key press) waiting loop.
;
; 03/12/2014
; 26/08/2014
; KEYBOARD I/O
; (INT_16h - Retro UNIX 8086 v1 - U9.ASM, 30/06/2014)

;NOTE: 'k0' to 'k7' are name of OPMASK registers.
;       (The reason of using '_k' labels!!!) (27/08/2014)
;NOTE: 'NOT' keyword is '~' unary operator in NASM.
;       ('NOT LC_HC' --> '~LC_HC') (bit reversing operator)

int_16h: ; 30/06/2015
;getc:
        pushfd  ; 28/08/2014
        push    cs
        call    getc_int
        retn

getc_int:
        ; 28/02/2015
        ; 03/12/2014 (derivation from pc-xt-286 bios source code -1986-,
        ;             instead of pc-at bios - 1985-)
        ; 28/08/2014 (_k1d)
        ; 30/06/2014
```

```
; 03/03/2014
; 28/02/2014
; Derived from "KEYBOARD_IO_1" procedure of IBM "pc-xt-286"
; rombios source code (21/04/1986), 'keybd.asm', INT 16H, KEYBOARD_IO
;
; KYBD --- 03/06/86  KEYBOARD BIOS
;
;--- INT 16 H -------------------------------------------------------------
; KEYBOARD I/O                                                            :
;      THESE ROUTINES PROVIDE READ KEYBOARD SUPPORT                       :
; INPUT                                                                   :
;      (AH)= 00H  READ THE NEXT ASCII CHARACTER ENTERED FROM THE KEYBOARD, :
;                 RETURN THE RESULT IN (AL), SCAN CODE IN (AH).           :
;                 THIS IS THE COMPATIBLE READ INTERFACE, EQUIVALENT TO THE :
;                  STANDARD PC OR PCAT KEYBOARD                           :
;-------------------------------------------------------------------------:
;      (AH)= 01H  SET THE ZERO FLAG TO INDICATE IF AN ASCII CHARACTER IS  :
;                 AVAILABLE TO BE READ FROM THE KEYBOARD BUFFER.          :
;                 (ZF)= 1 -- NO CODE AVAILABLE                            :
;                 (ZF)= 0 -- CODE IS AVAILABLE  (AX)= CHARACTER           :
;                 IF (ZF)= 0, THE NEXT CHARACTER IN THE BUFFER TO BE READ IS :
;                 IN (AX), AND THE ENTRY REMAINS IN THE BUFFER.           :
;                 THIS WILL RETURN ONLY PC/PCAT KEYBOARD COMPATIBLE CODES :
;-------------------------------------------------------------------------:
;      (AH)= 02H  RETURN THE CURRENT SHIFT STATUS IN AL REGISTER          :
;                 THE BIT SETTINGS FOR THIS CODE ARE INDICATED IN THE     :
;                 EQUATES FOR @KB_FLAG                                    :
;-------------------------------------------------------------------------:
;      (AH)= 03H  SET TYPAMATIC RATE AND DELAY                            :
;            (AL) = 05H                                                   :
;            (BL) = TYPAMATIC RATE (BITS 5 - 7 MUST BE RESET TO 0)        :
;                                                                         :
;                    REGISTER     RATE      REGISTER     RATE             :
;                     VALUE     SELECTED     VALUE     SELECTED           :
;                    --------------------------------------------         :
;                    00H        30.0        10H        7.5                :
;                    01H        26.7        11H        6.7                :
;                    02H        24.0        12H        6.0                :
;                    03H        21.8        13H        5.5                :
;                    04H        20.0        14H        5.0                :
;                    05H        18.5        15H        4.6                :
;                    06H        17.1        16H        4.3                :
;                    07H        16.0        17H        4.0                :
;                    08H        15.0        18H        3.7                :
;                    09H        13.3        19H        3.3                :
;                    0AH        12.0        1AH        3.0                :
;                    0BH        10.9        1BH        2.7                :
;                    0CH        10.0        1CH        2.5                :
;                    0DH         9.2        1DH        2.3                :
;                    0EH         8.6        1EH        2.1                :
;                    0FH         8.0        1FH        2.0                :
;                                                                         :
;            (BH) = TYPAMATIC DELAY  (BITS 2 - 7 MUST BE RESET TO 0)      :
;                                                                         :
;                    REGISTER     DELAY                                   :
;                     VALUE       VALUE                                   :
;                    ------------------                                   :
;                    00H        250 ms                                    :
;                    01H        500 ms                                    :
;                    02H        750 ms                                    :
;                    03H       1000 ms                                    :
;-------------------------------------------------------------------------:
;      (AH)= 05H  PLACE ASCII CHARACTER/SCAN CODE COMBINATION IN KEYBOARD :
;                 BUFFER AS IF STRUCK FROM KEYBOARD                       :
;                 ENTRY:  (CL) = ASCII CHARACTER                          :
;                         (CH) = SCAN CODE                                :
;                 EXIT:   (AH) = 00H = SUCCESSFUL OPERATION               :
;                         (AL) = 01H = UNSUCCESSFUL - BUFFER FULL         :
;                 FLAGS:  CARRY IF ERROR                                  :
;-------------------------------------------------------------------------:
;      (AH)= 10H  EXTENDED READ INTERFACE FOR THE ENHANCED KEYBOARD,      :
;                 OTHERWISE SAME AS FUNCTION AH=0                         :
;-------------------------------------------------------------------------:
;      (AH)= 11H  EXTENDED ASCII STATUS FOR THE ENHANCED KEYBOARD,        :
;                 OTHERWISE SAME AS FUNCTION AH=1                         :
;-------------------------------------------------------------------------:
;      (AH)= 12H  RETURN THE EXTENDED SHIFT STATUS IN AX REGISTER         :
;                 AL = BITS FROM KB_FLAG, AH = BITS FOR LEFT AND RIGHT    :
;                 CTL AND ALT KEYS FROM KB_FLAG_1 AND KB_FLAG_3           :
```

```
        ; OUTPUT                                                                 :
        ;       AS NOTED ABOVE, ONLY (AX) AND FLAGS CHANGED                      :
        ;       ALL REGISTERS RETAINED                                           :
        ;--------------------------------------------------------------------------------

        sti                             ; INTERRUPTS BACK ON
        push    ds                      ; SAVE CURRENT DS
        push    ebx                     ; SAVE BX TEMPORARILY
        ;push   ecx                     ; SAVE CX TEMPORARILY
         mov     bx, KDATA
        mov     ds, bx                  ; PUT SEGMENT VALUE OF DATA AREA INTO DS
        or      ah, ah                  ; CHECK FOR (AH)= 00H
        jz      short _K1               ; ASCII_READ
        dec     ah                        ; CHECK FOR (AH)= 01H
         jz      short _K2                 ; ASCII_STATUS
        dec     ah                      ; CHECK FOR (AH)= 02H
         jz      _K3                       ; SHIFT STATUS
        dec     ah                      ; CHECK FOR (AH)= 03H
         jz      _K300                     ; SET TYPAMATIC RATE/DELAY
        sub     ah, 2                   ; CHECK FOR (AH)= 05H
         jz      _K500                     ; KEYBOARD WRITE
_KIO1:
        sub     ah, 11                  ; AH =  10H
        jz      short _K1E              ; EXTENDED ASCII READ
        dec     ah                      ; CHECK FOR (AH)= 11H
        jz      short _K2E              ; EXTENDED_ASCII_STATUS
        dec     ah                      ; CHECK FOR (AH)= 12H
        jz      short _K3E              ; EXTENDED_SHIFT_STATUS
_KIO_EXIT:
        ;pop    ecx                     ; RECOVER REGISTER
        pop     ebx                     ; RECOVER REGISTER
        pop     ds                      ; RECOVER SEGMENT
        iretd                           ; INVALID COMMAND, EXIT

        ;-----  ASCII CHARACTER
_K1E:
        call    _K1S                    ; GET A CHARACTER FROM THE BUFFER (EXTENDED)
        call    _KIO_E_XLAT             ; ROUTINE TO XLATE FOR EXTENDED CALLS
        jmp     short _KIO_EXIT           ; GIVE IT TO THE CALLER
_K1:
        call    _K1S                    ; GET A CHARACTER FROM THE BUFFER
        call    _KIO_S_XLAT             ; ROUTINE TO XLATE FOR STANDARD CALLS
        jc      short _K1               ; CARRY SET MEANS TROW CODE AWAY
_K1A:
        jmp     short _KIO_EXIT           ; RETURN TO CALLER

        ;-----  ASCII STATUS
_K2E:
        call    _K2S                    ; TEST FOR CHARACTER IN BUFFER (EXTENDED)
        jz      short _K2B              ; RETURN IF BUFFER EMPTY
        pushf                           ; SAVE ZF FROM TEST
        call    _KIO_E_XLAT             ; ROUTINE TO XLATE FOR EXTENDED CALLS
        jmp     short _K2A               ; GIVE IT TO THE CALLER
_K2:
        call    _K2S                    ; TEST FOR CHARACTER IN BUFFER
        jz      short _K2B              ; RETURN IF BUFFER EMPTY
        pushf                           ; SAVE ZF FROM TEST
        call    _KIO_S_XLAT             ; ROUTINE TO XLATE FOR STANDARD CALLS
        jnc     short _K2A               ; CARRY CLEAR MEANS PASS VALID CODE
        popf                            ; INVALID CODE FOR THIS TYPE OF CALL
        call    _K1S                    ; THROW THE CHARACTER AWAY
        jmp     short _K2               ; GO LOOK FOR NEXT CHAR, IF ANY
_K2A:
        popf                            ; RESTORE ZF FROM TEST
_K2B:
        ;pop    ecx                     ; RECOVER REGISTER
        pop     ebx                     ; RECOVER REGISTER
        pop     ds                      ; RECOVER SEGMENT
        retf    4                       ; THROW AWAY (e)FLAGS

        ;-----  SHIFT STATUS
_K3E:                                      ; GET THE EXTENDED SHIFT STATUS FLAGS
        mov     ah, [KB_FLAG_1]                ; GET SYSTEM SHIFT KEY STATUS
        and     ah, SYS_SHIFT           ; MASK ALL BUT SYS KEY BIT
        ;mov    cl, 5                   ; SHIFT THEW SYSTEMKEY BIT OVER TO
        ;shl    ah, cl                  ; BIT 7 POSITION
         shl    ah, 5
        mov     al, [KB_FLAG_1]         ; GET SYSTEM SHIFT STATES BACK
        and     al, 01110011b           ; ELIMINATE SYS SHIFT, HOLD_STATE AND INS_SHIFT
```

```
        or      ah, al                  ; MERGE REMAINING BITS INTO AH
        mov     al, [KB_FLAG_3]         ; GET RIGHT CTL AND ALT
        and     al, 00001100b           ; ELIMINATE LC_E0 AND LC_E1
        or      ah, al                  ; OR THE SHIFT FLAGS TOGETHER
_K3:
        mov     al, [KB_FLAG]           ; GET THE SHIFT STATUS FLAGS
        jmp     short _KIO_EXIT                 ; RETURN TO CALLER

        ;-----  SET TYPAMATIC RATE AND DELAY
_K300:
        cmp     al, 5                   ; CORRECT FUNCTION CALL?
        jne     short _KIO_EXIT                 ; NO, RETURN
        test    bl, 0E0h                ; TEST FOR OUT-OF-RANGE RATE
        jnz     short _KIO_EXIT                 ; RETURN IF SO
        test    BH, 0FCh                ; TEST FOR OUT-OF-RANGE DELAY
        jnz     short _KIO_EXIT                 ; RETURN IF SO
        mov     al, KB_TYPA_RD          ; COMMAND FOR TYPAMATIC RATE/DELAY
        call    SND_DATA                ; SEND TO KEYBOARD
        ;mov    cx, 5                   ; SHIFT COUNT
        ;shl    bh, cl                  ; SHIFT DELAY OVER
        shl     bh, 5
        mov     al, bl                  ; PUT IN RATE
        or      al, bh                  ; AND DELAY
        call    SND_DATA                ; SEND TO KEYBOARD
         jmp      _KIO_EXIT                 ; RETURN TO CALLER

        ;-----  WRITE TO KEYBOARD BUFFER
_K500:
        push    esi                     ; SAVE SI (esi)
        cli                             ;
        mov     ebx, [BUFFER_TAIL]      ; GET THE 'IN TO' POINTER TO THE BUFFER
        mov     esi, ebx                ; SAVE A COPY IN CASE BUFFER NOT FULL
        call    _K4                     ; BUMP THE POINTER TO SEE IF BUFFER IS FULL
        cmp     ebx, [BUFFER_HEAD]      ; WILL THE BUFFER OVERRUN IF WE STORE THIS?
        je      short _K502             ; YES - INFORM CALLER OF ERROR
        mov     [esi], cx               ; NO - PUT ASCII/SCAN CODE INTO BUFFER
        mov     [BUFFER_TAIL], ebx      ; ADJUST 'IN TO' POINTER TO REFLECT CHANGE
        sub     al, al                  ; TELL CALLER THAT OPERATION WAS SUCCESSFUL
        jmp     short _K504             ; SUB INSTRUCTION ALSO RESETS CARRY FLAG
_K502:
        mov     al, 01h                 ; BUFFER FULL INDICATION
_K504:
        sti
        pop     esi                     ; RECOVER SI (esi)
         jmp      _KIO_EXIT                 ; RETURN TO CALLER WITH STATUS IN AL

        ;-----  READ THE KEY TO FIGURE OUT WHAT TO DO -----
_K1S:
        cli     ; 03/12/2014
         mov      ebx, [BUFFER_HEAD]    ; GET POINTER TO HEAD OF BUFFER
         cmp      ebx, [BUFFER_TAIL]    ; TEST END OF BUFFER
        ;jne    short _K1U              ; IF ANYTHING IN BUFFER SKIP INTERRUPT
        jne     short _k1x ; 03/12/2014
        ;
        ; 03/12/2014
        ; 28/08/2014
        ; PERFORM OTHER FUNCTION ?? here !
        ;; MOV  AX, 9002h               ; MOVE IN WAIT CODE & TYPE
        ;; INT  15H                     ; PERFORM OTHER FUNCTION
_K1T:                                       ; ASCII READ
        sti                             ; INTERRUPTS BACK ON DURING LOOP
        nop                             ; ALLOW AN INTERRUPT TO OCCUR
_K1U:
        cli                             ; INTERRUPTS BACK OFF
         mov            ebx, [BUFFER_HEAD]      ; GET POINTER TO HEAD OF BUFFER
         cmp      ebx, [BUFFER_TAIL]    ; TEST END OF BUFFER
_k1x:
        push    ebx                     ; SAVE ADDRESS
        pushf                           ; SAVE FLAGS
        call    MAKE_LED                ; GO GET MODE INDICATOR DATA BYTE
        mov     bl, [KB_FLAG_2]         ; GET PREVIOUS BITS
        xor     bl, al                  ; SEE IF ANY DIFFERENT
        and     bl, 07h ; KB_LEDS       ; ISOLATE INDICATOR BITS
        jz      short _K1V              ; IF NO CHANGE BYPASS UPDATE
        call    SND_LED1
        cli                             ; DISABLE INTERRUPTS
```

```
_K1V:
        popf                            ; RESTORE FLAGS
        pop     ebx                     ; RESTORE ADDRESS
         je      short _K1T                ; LOOP UNTIL SOMETHING IN BUFFER
        ;
        mov     ax, [ebx]               ; GET SCAN CODE AND ASCII CODE
         call    _K4                       ; MOVE POINTER TO NEXT POSITION
         mov     [BUFFER_HEAD], ebx        ; STORE VALUE IN VARIABLE
        retn                            ; RETURN

        ;----- READ THE KEY TO SEE IF ONE IS PRESENT -----
_K2S:
        cli                             ; INTERRUPTS OFF
         mov     ebx, [BUFFER_HEAD]        ; GET HEAD POINTER
         cmp     ebx, [BUFFER_TAIL]        ; IF EQUAL (Z=1) THEN NOTHING THERE
        mov     ax, [ebx]
        pushf                           ; SAVE FLAGS
        push    ax                      ; SAVE CODE
        call    MAKE_LED                ; GO GET MODE INDICATOR DATA BYTE
        mov     bl, [KB_FLAG_2]         ; GET PREVIOUS BITS
        xor     bl, al                  ; SEE IF ANY DIFFERENT
        and     bl, 07h ; KB_LEDS       ; ISOLATE INDICATOR BITS
        jz      short _K2T              ; IF NO CHANGE BYPASS UPDATE
        call    SND_LED                 ; GO TURN ON MODE INDICATORS
_K2T:
        pop     ax                      ; RESTORE CODE
        popf                            ; RESTORE FLAGS
        sti                             ; INTERRUPTS BACK ON
        retn                            ; RETURN

        ;----- ROUTINE TO TRANSLATE SCAN CODE PAIRS FOR EXTENDED CALLS -----
_KIO_E_XLAT:
        cmp     al, 0F0h                ; IS IT ONE OF THE FILL-INs?
        jne     short _KIO_E_RET        ; NO, PASS IT ON
         or     ah, ah                  ; AH = 0 IS SPECIAL CASE
         jz     short _KIO_E_RET          ; PASS THIS ON UNCHANGED
        xor     al, al                  ; OTHERWISE SET AL = 0
_KIO_E_RET:
        retn                            ; GO BACK

        ;----- ROUTINE TO TRANSLATE SCAN CODE PAIRS FOR STANDARD CALLS -----
_KIO_S_XLAT:
        cmp     ah, 0E0h                ; IS IT KEYPAD ENTER OR / ?
        jne     short _KIO_S2           ; NO, CONTINUE
        cmp     al, 0Dh                 ; KEYPAD ENTER CODE?
         je     short _KIO_S1           ; YES, MASSAGE A BIT
        cmp     al, 0Ah                 ; CTRL KEYPAD ENTER CODE?
         je     short _KIO_S1           ; YES, MASSAGE THE SAME
        mov     ah, 35h                 ; NO, MUST BE KEYPAD /
_kio_ret: ; 03/12/2014
        clc
        retn
        ;jmp    short _KIO_USE          ; GIVE TO CALLER
_KIO_S1:
        mov     ah, 1Ch                 ; CONVERT TO COMPATIBLE OUTPUT
        ;jmp    short _KIO_USE          ; GIVE TO CALLER
        retn
_KIO_S2:
        cmp     ah, 84h                 ; IS IT ONE OF EXTENDED ONES?
        ja      short _KIO_DIS          ; YES, THROW AWAY AND GET ANOTHER CHAR
        cmp     al, 0F0h                ; IS IT ONE OF THE FILL-INs?
         jne    short _KIO_S3           ; NO, TRY LAST TEST
        or      ah, ah                  ; AH = 0 IS SPECIAL CASE
         jz     short _KIO_USE          ; PASS THIS ON UNCHANGED
        jmp     short _KIO_DIS          ; THROW AWAY THE REST
_KIO_S3:
        cmp     al, 0E0h                ; IS IT AN EXTENSION OF A PREVIOUS ONE?
        ;jne    short _KIO_USE          ; NO, MUST BE A STANDARD CODE
        jne     short _kio_ret
        or      ah, ah                  ; AH = 0 IS SPECIAL CASE
         jz     short _KIO_USE          ; JUMP IF AH = 0
        xor     al, al                  ; CONVERT TO COMPATIBLE OUTPUT
        ;jmp    short _KIO_USE          ; PASS IT ON TO CALLER
_KIO_USE:
        ;clc                            ; CLEAR CARRY TO INDICATE GOOD CODE
        retn                            ; RETURN
_KIO_DIS:
        stc                             ; SET CARRY TO INDICATE DISCARD CODE
        retn                            ; RETURN
```

```
	;-----  INCREMENT BUFFER POINTER ROUTINE -----
_K4:
	inc	ebx
	inc	ebx			; MOVE TO NEXT WORD IN LIST
	 cmp	 ebx, [BUFFER_END]	; AT END OF BUFFER?
	 ;jne	 short _K5		   ; NO, CONTINUE
	jb	short _K5
	 mov	 ebx, [BUFFER_START]	   ; YES, RESET TO BUFFER BEGINNING
_K5:
	retn

; 20/02/2015
; 05/12/2014
; 26/08/2014
; KEYBOARD (HARDWARE) INTERRUPT -  IRQ LEVEL 1
; (INT_09h - Retro UNIX 8086 v1 - U9.ASM, 07/03/2014)
;
; Derived from "KB_INT_1" procedure of IBM "pc-at"
; rombios source code (06/10/1985)
; 'keybd.asm', HARDWARE INT 09h - (IRQ Level 1)

;--------- 8042 COMMANDS -------------------------------------------------
ENA_KBD		equ	0AEh	; ENABLE KEYBOARD COMMAND
DIS_KBD		equ	0ADh	; DISABLE KEYBOARD COMMAND
SHUT_CMD	equ	0FEh	; CAUSE A SHUTDOWN COMMAND
;--------- 8042 KEYBOARD INTERFACE AND DIAGNOSTIC CONTROL REGISTERS ------------
STATUS_PORT	equ	064h	; 8042 STATUS PORT
INPT_BUF_FULL	equ	00000010b ; 1 = +INPUT BUFFER FULL
PORT_A		equ	060h	; 8042 KEYBOARD SCAN CODE/CONTROL PORT
;---------- 8042 KEYBOARD RESPONSE ---------------------------------------
KB_ACK		equ	0FAh	; ACKNOWLEDGE PROM TRANSMISSION
KB_RESEND	equ	0FEh	; RESEND REQUEST
KB_OVER_RUN	equ	0FFh	; OVER RUN SCAN CODE
;---------- KEYBOARD/LED COMMANDS ----------------------------------------
KB_ENABLE	equ	0F4h		; KEYBOARD ENABLE
LED_CMD		equ	0EDh		; LED WRITE COMMAND
KB_TYPA_RD	equ	0F3h		; TYPAMATIC RATE/DELAY COMMAND
;---------- KEYBOARD SCAN CODES ------------------------------------------
NUM_KEY		equ	69		; SCAN CODE FOR		NUMBER LOCK KEY
SCROLL_KEY	equ	70		; SCAN CODE FOR		SCROLL LOCK KEY
ALT_KEY		equ	56		; SCAN CODE FOR		ALTERNATE SHIFT KEY
CTL_KEY		equ	29		; SCAN CODE FOR		CONTROL KEY
CAPS_KEY	equ	58		; SCAN CODE FOR		SHIFT LOCK KEY
DEL_KEY		equ	83		; SCAN CODE FOR		DELETE KEY
INS_KEY		equ	82		; SCAN CODE FOR		INSERT KEY
LEFT_KEY	equ	42		; SCAN CODE FOR		LEFT SHIFT
RIGHT_KEY	equ	54		; SCAN CODE FOR		RIGHT SHIFT
SYS_KEY		equ	84		; SCAN CODE FOR		SYSTEM KEY
;---------- ENHANCED KEYBOARD SCAN CODES ---------------------------------
ID_1		equ	0ABh		; 1ST ID CHARACTER FOR KBX
ID_2		equ	041h		; 2ND ID CHARACTER FOR KBX
ID_2A		equ	054h		; ALTERNATE 2ND ID CHARACTER FOR KBX
F11_M		equ	87		; F11 KEY MAKE
F12_M		equ	88		; F12 KEY MAKE
MC_E0		equ	224		; GENERAL MARKER CODE
MC_E1		equ	225		; PAUSE KEY MARKER CODE
;---------- FLAG EQUATES WITHIN @KB_FLAG---------------------------------
RIGHT_SHIFT	equ	00000001b	; RIGHT SHIFT KEY DEPRESSED
LEFT_SHIFT	equ	00000010b	; LEFT SHIFT KEY DEPRESSED
CTL_SHIFT	equ	00000100b	; CONTROL SHIFT KEY DEPRESSED
ALT_SHIFT	equ	00001000b	; ALTERNATE SHIFT KEY DEPRESSED
SCROLL_STATE	equ	00010000b	; SCROLL LOCK STATE IS ACTIVE
NUM_STATE	equ	00100000b	; NUM LOCK STATE IS ACTIVE
CAPS_STATE	equ	01000000b	; CAPS LOCK STATE IS ACTIVE
INS_STATE	equ	10000000b	; INSERT STATE IS ACTIVE
;---------- FLAG EQUATES WITHIN	@KB_FLAG_1 ------------------------------------
L_CTL_SHIFT	equ	00000001b	; LEFT CTL KEY DOWN
L_ALT_SHIFT	equ	00000010b	; LEFT ALT KEY DOWN
SYS_SHIFT	equ	00000100b	; SYSTEM KEY DEPRESSED AND HELD
HOLD_STATE	equ	00001000b	; SUSPEND KEY HAS BEEN TOGGLED
SCROLL_SHIFT	equ	00010000b	; SCROLL LOCK KEY IS DEPRESSED
NUM_SHIFT	equ	00100000b	; NUM LOCK KEY IS DEPRESSED
CAPS_SHIFT	equ	01000000b	; CAPS LOCK KEY IS DEPRE55ED
INS_SHIFT	equ	10000000b	; INSERT KEY IS DEPRESSED
;---------- FLAGS EQUATES WITHIN @KB_FLAG_2 ----------------------------------
KB_LEDS		equ	00000111b	; KEYBOARD LED STATE BITS
;		equ	00000001b	; SCROLL LOCK INDICATOR
;		equ	00000010b	; NUM LOCK INDICATOR
```

```
;               equu    00000100b       ; CAPS LOCK INDICATOR
;               equ     00001000b       ; RESERVED (MUST BE ZERO)
KB_FA           equ     00010000b       ; ACKNOWLEDGMENT RECEIVED
KB_FE           equ     00100000b       ; RESEND RECEIVED FLAG
KB_PR_LED       equ     01000000b       ; MODE INDICATOR UPDATE
KB_ERR          equ     10000000b       ; KEYBOARD TRANSMIT ERROR FLAG
;----------- FLAGS EQUATES WITHIN @KB_FLAG_3 ----------------------------------
LC_E1           equ     00000001b       ; LAST CODE WAS THE E1 HIDDEN CODE
LC_E0           equ     00000010b       ; LAST CODE WAS THE E0 HIDDEN CODE
R_CTL_SHIFT     equ     00000100b       ; RIGHT CTL KEY DOWN
R_ALT_SHIFT     equ     00001000b       ; RIGHT ALT KEY DOWN
GRAPH_ON        equ     00001000b       ; ALT GRAPHICS KEY DOWN (WT ONLY)
KBX             equ     00010000b       ; ENHANCED KEYBOARD INSTALLED
SET_NUM_LK      equ     00100000b       ; FORCE NUM LOCK IF READ ID AND KBX
LC_AB           equ     01000000b       ; LAST CHARACTER WAS FIRST ID CHARACTER
RD_ID           equ     10000000b       ; DOING A READ ID (MUST BE BIT0)
;
;----------- INTERRUPT EQUATES -------------------------------------------------
EOI             equ     020h            ; END OF INTERRUPT COMMAND TO 8259
INTA00          equ     020h            ; 8259 PORT


kb_int:

; 17/10/2015 ('ctrlbrk')
; 05/12/2014
; 04/12/2014 (derivation from pc-xt-286 bios source code -1986-,
        ;             instead of pc-at bios - 1985-)
; 26/08/2014
;
; 03/06/86  KEYBOARD BIOS
;
;--- HARDWARE INT 09H -- (IRQ LEVEL 1) ------------------------------------------
;                                                                            ;
;       KEYBOARD INTERRUPT ROUTINE                                           ;
;                                                                            ;
;--------------------------------------------------------------------------------

KB_INT_1:
        sti                             ; ENABLE INTERRUPTS
        ;push   ebp
        push    eax
        push    ebx
        push    ecx
        push    edx
        push    esi
        push    edi
        push    ds
        push    es
        cld                             ; FORWARD DIRECTION
        mov     ax, KDATA
        mov     ds, ax
        mov     es, ax
        ;
        ;-----  WAIT FOR KEYBOARD DISABLE COMMAND TO BE ACCEPTED
        mov     al, DIS_KBD             ; DISABLE THE KEYBOARD COMMAND
        call    SHIP_IT                 ; EXECUTE DISABLE
        cli                             ; DISABLE INTERRUPTS
        mov     ecx, 10000h             ; SET MAXIMUM TIMEOUT
KB_INT_01:
        in      al, STATUS_PORT                 ; READ ADAPTER STATUS
        test    al, INPT_BUF_FULL       ; CHECK INPUT BUFFER FULL STATUS BIT
        loopnz  KB_INT_01               ; WAIT FOR COMMAND TO BE ACCEPTED
        ;
        ;-----  READ CHARACTER FROM KEYBOARD INTERFACE
        in      al, PORT_A              ; READ IN THE CHARACTER
        ;
        ;-----  SYSTEM HOOK INT 15H - FUNCTION 4FH (ON HARDWARE INT LEVEL 9H)
        ;MOV    AH, 04FH                ; SYSTEM INTERCEPT - KEY CODE FUNCTION
        ;STC                            ; SET CY=1 (IN CASE OF IRET)
        ;INT    15H                     ; CASETTE CALL (AL)=KEY SCAN CODE
        ;                               ; RETURNS CY=1 FOR INVALID FUNCTION
        ;JC     KB_INT_02               ; CONTINUE IF CARRY FLAG SET ((AL)=CODE)
        ;JMP    K26                     ; EXIT IF SYSTEM HANDLES SCAN CODE
        ;                               ; EX¦T HANDLES HARDWARE EOI AND ENABLE
        ;
```

```
        ;----- CHECK FOR A RESEND COMMAND TO KEYBOARD
KB_INT_02:                          ;         (AL)= SCAN CODE
        sti                         ; ENABLE INTERRUPTS AGAIN
        cmp     al, KB_RESEND       ; IS THE INPUT A RESEND
         je      short KB_INT_4         ; GO IF RESEND
        ;
        ;----- CHECK FOR RESPONSE TO A COMMAND TO KEYBOARD
        cmp     al, KB_ACK          ; IS THE INPUT AN ACKNOWLEDGE
         jne     short KB_INT_2         ; GO IF NOT
        ;
        ;----- A COMMAND TO THE KEYBOARD WAS ISSUED
        cli                         ; DISABLE INTERRUPTS
        or      byte [KB_FLAG_2], KB_FA ; INDICATE ACK RECEIVED
         jmp     K26                    ; RETURN IF NOT (ACK RETURNED FOR DATA)
        ;
        ;----- RESEND THE LAST BYTE
KB_INT_4:
        cli                         ; DISABLE INTERRUPTS
        or      byte [KB_FLAG_2], KB_FE ; INDICATE RESEND RECEIVED
         jmp     K26                    ; RETURN IF NOT ACK RETURNED FOR DATA)
        ;
;----- UPDATE MODE INDICATORS IF CHANGE IN STATE
KB_INT_2:
        push    ax                  ; SAVE DATA IN
        call    MAKE_LED            ; GO GET MODE INDICATOR DATA BYTE
        mov     bl, [KB_FLAG_2]     ; GET PREVIOUS BITS
        xor     bl, al              ; SEE IF ANY DIFFERENT
        and     bl, KB_LEDS         ; ISOLATE INDICATOR BITS
        jz      short UP0           ; IF NO CHANGE BYPASS UPDATE
        call    SND_LED             ; GO TURN ON MODE INDICATORS
UP0:
        pop     ax                  ; RESTORE DATA IN
;---------------------------------------------------------------------
;       START OF KEY PROCESSING                                       ;
;---------------------------------------------------------------------
        mov     ah, al              ; SAVE SCAN CODE IN AH ALSO
        ;
        ;----- TEST FOR OVERRUN SCAN CODE FROM KEYBOARD
        cmp     al, KB_OVER_RUN             ; IS THIS AN OVERRUN CHAR
         je      K62                ; BUFFER_FULL_BEEP
        ;
K16:
        mov     bh, [KB_FLAG_3]             ; LOAD FLAGS FOR TESTING
        ;
        ;----- TEST TO SEE IF A READ_ID IS IN PROGRESS
        test    bh, RD_ID+LC_AB     ; ARE WE DOING A READ ID?
        jz      short NOT_ID        ; CONTINUE IF NOT
        jns     short TST_ID_2      ; IS THE RD_ID FLAG ON?
        cmp     al, ID_1            ; IS THIS THE 1ST ID CHARACTER?
        jne     short RST_RD_ID
        or      byte [KB_FLAG_3], LC_AB ; INDICATE 1ST ID WAS OK
RST_RD_ID:
        and     byte [KB_FLAG_3], ~RD_ID ; RESET THE READ ID FLAG
         ;jmp    short ID_EX        ; AND EXIT
        jmp     K26
        ;
TST_ID_2:
        and     byte [KB_FLAG_3], ~LC_AB ; RESET FLAG
        cmp     al, ID_2A           ; IS THIS THE 2ND ID CHARACTER?
         je     short KX_BIT        ; JUMP IF SO
        cmp     al, ID_2            ; IS THIS THE 2ND ID CHARACTER?
         ;jne   short ID_EX         ; LEAVE IF NOT
        jne     K26
        ;
        ;----- A READ ID SAID THAT IT WAS ENHANCED KEYBOARD
        test    bh, SET_NUM_LK              ; SHOULD WE SET NUM LOCK?
         jz      short KX_BIT       ; EXIT IF NOT
        or      byte [KB_FLAG], NUM_STATE ; FORCE NUM LOCK ON
        call    SND_LED             ; GO SET THE NUM LOCK INDICATOR
KX_BIT:
        or      byte [KB_FLAG_3], KBX ; INDICATE ENHANCED KEYBOARD WAS FOUND
ID_EX:  jmp      K26                ; EXIT
        ;
NOT_ID:
        cmp     al, MC_E0           ; IS THIS THE GENERAL MARKER CODE?
        jne     short TEST_E1
        or      byte [KB_FLAG_3], LC_E0+KBX ; SET FLAG BIT, SET KBX, AND
        ;jmp    short EXIT          ; THROW AWAY THIS CODE
        jmp     K26A
```

```
TEST_E1:
        cmp     al, MC_E1               ; IS THIS THE PAUSE KEY?
        jne     short NOT_HC
        or      byte [KB_FLAG_3], LC_E1+KBX ; SET FLAG BIT, SET KBX, AND
EXIT:   jmp     K26A                    ; THROW AWAY THIS CODE
        ;
NOT_HC:
        and     al, 07Fh                ; TURN OFF THE BREAK BIT
        test    bh, LC_E0               ; LAST CODE THE E0 MARKER CODE
        jz      short NOT_LC_E0                 ; JUMP IF NOT
        ;
        mov     edi, _K6+6              ; IS THIS A SHIFT KEY?
        scasb
         je      K26 ; K16B                ; YES, THROW AWAY & RESET FLAG
        scasb
        jne     short K16A              ; NO, CONTINUE KEY PROCESSING
        ;jmp    short K16B              ; YES, THROW AWAY & RESET FLAG
        jmp     K26
        ;
NOT_LC_E0:
        test    bh, LC_E1               ; LAST CODE THE E1 MARKER CODE?
        jz      short T_SYS_KEY                 ; JUMP IF NOT
        mov     ecx, 4                  ; LENGHT OF SEARCH
        mov     edi, _K6+4              ; IS THIS AN ALT, CTL, OR SHIFT?
        repne   scasb                   ; CHECK IT
        ;je     short EXIT              ; THROW AWAY IF SO
        je      K26A
        ;
        cmp     al, NUM_KEY             ; IS IT THE PAUSE KEY?
        ;jne    short K16B              ; NO, THROW AWAY & RESET FLAG
        jne     K26
        test    ah, 80h                 ; YES, IS IT THE BREAK OF THE KEY?
        ;jnz    short K16B              ; YES, THROW THIS AWAY, TOO
        jnz     K26
         ; 20/02/2015
        test    byte [KB_FLAG_1],HOLD_STATE ;  NO, ARE WE PAUSED ALREADY?
        ;jnz    short K16B              ;  YES, THROW AWAY
        jnz     K26
        jmp      K39P                      ; NO, THIS IS THE REAL PAUSE STATE
        ;
        ;-----  TEST FOR SYSTEM KEY
T_SYS_KEY:
        cmp     al, SYS_KEY             ; IS IT THE SYSTEM KEY?
        jnz     short K16A              ; CONTINUE IF NOT
        ;
        test    ah, 80h                 ; CHECK IF THIS A BREAK CODE
        jnz     short K16C              ; DO NOT TOUCH SYSTEM INDICATOR IF TRUE
        ;
        test    byte [KB_FLAG_1], SYS_SHIFT ; SEE IF IN SYSTEM KEY HELD DOWN
         ;jnz   short K16B              ; IF YES, DO NOT PROCESS SYSTEM INDICATOR
        jnz      K26
        ;
        or      byte [KB_FLAG_1], SYS_SHIFT ; INDICATE SYSTEM KEY DEPRESSED
        mov     al, EOI                 ; END OF INTERRUPT COMMAND
        out     20h, al ;out INTA00, al       ; SEND COMMAND TO INTERRUPT CONTROL PORT
                                        ; INTERRUPT-RETURN-NO-EOI
        mov     al, ENA_KBD             ; INSURE KEYBOARD IS ENABLED
        call    SHIP_IT                 ; EXECUTE ENABLE
        ; !!! SYSREQ !!! function/system call (INTERRUPT) must be here !!!
        ;MOV    AL, 8500H               ; FUNCTION VALUE FOR MAKE OF SYSTEM KEY
        ;STI                            ; MAKE SURE INTERRUPTS ENABLED
        ;INT    15H                     ; USER INTERRUPT
         jmp     K27A                      ; END PROCESSING
        ;
;K16B:  jmp     K26                     ; IGNORE SYSTEM KEY
        ;
K16C:
        and     byte [KB_FLAG_1], ~SYS_SHIFT ; TURN OFF SHIFT KEY HELD DOWN
        mov     al, EOI                 ; END OF INTERRUPT COMMAND
        out     20h, al ;out INTA00, al ; SEND COMMAND TO INTERRUPT CONTROL PORT
                                        ; INTERRUPT-RETURN-NO-EOI
        ;MOV    AL, ENA_KBD             ; INSURE KEYBOARD IS ENABLED
        ;CALL   SHIP_IT                 ; EXECUTE ENABLE
        ;
        ;MOV    AX, 8501H               ; FUNCTION VALUE FOR BREAK OF SYSTEM KEY
        ;STI                            ; MAKE SURE INTERRUPTS ENABLED
        ;INT    15H                     ; USER INTERRUPT
        ;JMP    K27A                    ; INGONRE SYSTEM KEY
        ;
```

```
        jmp     K27                     ; IGNORE SYSTEM KEY
        ;
        ;----- TEST FOR SHIFT KEYS
K16A:
        mov     bl, [KB_FLAG]           ; PUT STATE FLAGS IN BL
        mov     edi, _K6                ; SHIFT KEY TABLE offset
        mov     ecx, _K6L               ; LENGTH
        repne   scasb                   ; LOOK THROUGH THE TABLE FOR A MATCH
        mov     al, ah                  ; RECOVER SCAN CODE
         jne     K25                       ; IF NO MATCH, THEN SHIFT NOT FOUND
        ;
        ;------SHIFT KEY FOUND
K17:
         sub     edi, _K6+1                ; ADJUST PTR TO SCAN CODE MATCH
        mov      ah, [edi+_K7]          ; GET MASK INTO AH
        mov     cl, 2                   ; SETUP COUNT FOR FLAG SHIFTS
        test    al, 80h                 ; TEST FOR BREAK KEY
         jnz     K23                       ; JUMP OF BREAK
        ;
        ;----- SHIFT MAKE FOUND, DETERMINE SET OR TOGGLE
K17C:
        cmp     ah, SCROLL_SHIFT
        jae     short K18               ; IF SCROLL SHIFT OR ABOVE, TOGGLE KEY
        ;
        ;----- PLAIN SHIFT KEY, SET SHIFT ON
        or      [KB_FLAG], ah           ; TURN ON SHIFT BIT
         test   al, CTL_SHIFT+ALT_SHIFT ; IS IT ALT OR CTRL?
        ;jnz    short K17D              ; YES, MORE FLAGS TO SET
        jz      K26                     ; NO, INTERRUPT RETURN
K17D:
        test    bh, LC_E0               ; IS THIS ONE OF NEW KEYS?
        jz      short K17E              ; NO, JUMP
        or      [KB_FLAG_3], ah                 ; SET BITS FOR RIGHT CTRL, ALT
        jmp     K26                     ; INTERRUPT RETURN
K17E:
        shr     ah, cl                  ; MOVE FLAG BITS TWO POSITIONS
        or      [KB_FLAG_1], ah                 ; SET BITS FOR LEFT CTRL, ALT
        jmp     K26
        ;
        ;----- TOGGLED SHIFT KEY, TEST FOR 1ST MAKE OR NOT
K18:                                    ; SHIFT-TOGGLE
        test    bl, CTL_SHIFT           ; CHECK CTL SHIFT STATE
         ;jz            short K18A              ; JUMP IF NOT CTL STATE
         jnz     K25                       ; JUMP IF CTL STATE
K18A:
        cmp     al, INS_KEY             ; CHECK FOR INSERT KEY
        jne     short K22               ; JUMP IF NOT INSERT KEY
        test    bl, ALT_SHIFT           ; CHECK FOR ALTERNATE SHIFT
        ;jz     short K18B              ; JUMP IF NOT ALTERNATE SHIFT
         jnz     K25                       ; JUMP IF ALTERNATE SHIFT
K18B:
        test    bh, LC_E0 ;20/02/2015 ; IS THIS NEW INSERT KEY?
        jnz     short K22               ; YES, THIS ONE'S NEVER A '0'
K19:
        test    bl, NUM_STATE           ; CHECK FOR BASE STATE
        jnz     short K21               ; JUMP IF NUM LOCK IS ON
        test    bl, LEFT_SHIFT+RIGHT_SHIFT ; TEST FOR SHIFT STATE
        jz      short K22               ; JUMP IF BASE STATE
K20:                                    ; NUMERIC ZERO, NOT INSERT KEY
        mov     ah, al                  ; PUT SCAN CODE BACK IN AH
         jmp     K25                       ; NUMERAL '0', STNDRD. PROCESSING
K21:                                    ; MIGHT BE NUMERIC
        test    bl, LEFT_SHIFT+RIGHT_SHIFT
        jz      short K20               ; IS NUMERIC, STD. PROC.
        ;
K22:                                    ; SHIFT TOGGLE KEY HIT; PROCESS IT
        test    ah, [KB_FLAG_1]         ; IS KEY ALREADY DEPRESSED
         jnz     K26                       ; JUMP IF KEY ALREADY DEPRESSED
K22A:
         or      [KB_FLAG_1], ah        ; INDICATE THAT THE KEY IS DEPRESSED
        xor     [KB_FLAG], ah           ; TOGGLE THE SHIFT STATE
        ;
        ;----- TOGGLE LED IF CAPS, NUM  OR SCROLL KEY DEPRESSED
        test    ah, CAPS_SHIFT+NUM_SHIFT+SCROLL_SHIFT ; SHIFT TOGGLE?
        jz      short K22B              ; GO IF NOT
        ;
        push    ax                      ; SAVE SCAN CODE AND SHIFT MASK
        call    SND_LED                 ; GO TURN MODE INDICATORS ON
        pop     ax                      ; RESTORE SCAN CODE
```

```
K22B:
        cmp     al, INS_KEY             ; TEST FOR 1ST MAKE OF INSERT KEY
         jne     K26                       ; JUMP IF NOT INSERT KEY
        mov     ah, al                   ; SCAN CODE IN BOTH HALVES OF AX
         jmp     K28                       ; FLAGS UPDATED, PROC. FOR BUFFER
        ;
        ;-----  BREAK SHIFT FOUND
K23:                                    ; BREAK-SHIFT-FOUND
        cmp     ah, SCROLL_SHIFT        ; IS THIS A TOGGLE KEY
        not     ah                      ; INVERT MASK
        jae     short K24               ; YES, HANDLE BREAK TOGGLE
        and     [KB_FLAG], ah           ; TURN OFF SHIFT BIT
        cmp     ah, ~CTL_SHIFT          ; IS THIS ALT OR CTL?
        ja      short K23D              ; NO, ALL DONE
        ;
        test    bh, LC_E0               ; 2ND ALT OR CTL?
        jz      short K23A              ; NO, HANSLE NORMALLY
        and     [KB_FLAG_3], ah                 ; RESET BIT FOR RIGHT ALT OR CTL
        jmp     short K23B              ; CONTINUE
K23A:
        sar     ah, cl                  ; MOVE THE MASK BIT TWO POSITIONS
        and     [KB_FLAG_1], ah                 ; RESET BIT FOR LEFT ALT AND CTL
K23B:
        mov     ah, al                  ; SAVE SCAN CODE
        mov     al, [KB_FLAG_3]                 ; GET RIGHT ALT & CTRL FLAGS
        shr     al, cl                  ; MOVE TO BITS 1 & 0
        or      al, [KB_FLAG_1]                 ; PUT IN LEFT ALÌT & CTL FLAGS
        shl     al, cl                  ; MOVE BACK TO BITS 3 & 2
        and     al, ALT_SHIFT+CTL_SHIFT ; FILTER OUT OTHER GARBAGE
        or      [KB_FLAG], al           ; PUT RESULT IN THE REAL FLAGS
        mov     al, ah
K23D:
        cmp     al, ALT_KEY+80h                 ; IS THIS ALTERNATE SHIFT RELEASE
        jne     short K26               ; INTERRUPT RETURN
        ;
        ;-----  ALTERNATE SHIFT KEY RELEASED, GET THE VALUE INTO BUFFER
        mov     al, [ALT_INPUT]
        mov     ah, 0                   ; SCAN CODE OF 0
        mov     [ALT_INPUT], ah         ; ZERO OUT THE FIELD
        cmp     al, 0                   ; WAS THE INPUT = 0?
        je      short K26               ; INTERRUPT_RETURN
         jmp     K61                       ; IT WASN'T, SO PUT IN BUFFER
        ;
K24:                                    ; BREAK-TOGGLE
        and     [KB_FLAG_1], ah         ; INDICATE NO LONGER DEPRESSED
        jmp     short K26               ; INTERRUPT_RETURN
        ;
        ;-----  TEST FOR HOLD STATE
                                        ; AL, AH = SCAN CODE
K25:                                    ; NO-SHIFT-FOUND
        cmp     al, 80h                 ; TEST FOR BREAK KEY
        jae     short K26               ; NOTHING FOR BREAK CHARS FROM HERE ON
        test    byte [KB_FLAG_1], HOLD_STATE ; ARE WE IN HOLD STATE
        jz      short K28               ; BRANCH AROUND TEST IF NOT
        cmp     al, NUM_KEY
        je      short K26               ; CAN'T END HOLD ON NUM_LOCK
        and     byte [KB_FLAG_1], ~HOLD_STATE ; TURN OFF THE HOLD STATE BIT
        ;
K26:
        and     byte [KB_FLAG_3], ~(LC_E0+LC_E1) ; RESET LAST CHAR H.C. FLAG
K26A:                                   ; INTERRUPT-RETURN
        cli                             ; TURN OFF INTERRUPTS
        mov     al, EOI                 ; END OF INTERRUPT COMMAND
        out     20h, al ;out INTA00, al         ; SEND COMMAND TO INTERRUPT CONTROL PORT
K27:                                    ; INTERRUPT-RETURN-NO-EOI
        mov     al, ENA_KBD             ; INSURE KEYBOARD IS ENABLED
        call    SHIP_IT                 ; EXECUTE ENABLE
K27A:
        cli                             ; DISABLE INTERRUPTS
        pop     es                      ; RESTORE REGISTERS
        pop     ds
        pop     edi
        pop     esi
        pop     edx
        pop     ecx
        pop     ebx
        pop     eax
        ;pop    ebp
        iret                            ; RETURN
```

```
        ;-----  NOT IN  HOLD STATE
K28:                                    ; NO-HOLD-STATE
        cmp     al, 88                  ; TEST FOR OUT-OF-RANGE SCAN CODES
        ja      short K26               ; IGNORE IF OUT-OF-RANGE
        ;
        test    bl, ALT_SHIFT           ; ARE WE IN ALTERNATE SHIFT
         ;jz    short K28A              ; IF NOT ALTERNATE
         jz      K38
        ;
        test    bh, KBX                 ; IS THIS THE ENCHANCED KEYBOARD?
        jz      short K29               ; NO, ALT STATE IS REAL
         ;28/02/2015
        test    byte [KB_FLAG_1], SYS_SHIFT ; YES, IS SYSREQ KEY DOWN?
        ;jz     short K29               ;  NO, ALT STATE IS REAL
        jnz     K38                     ; YES, THIS IS PHONY ALT STATE
         ;                              ; DUE TO PRESSING SYSREQ
;K28A:  jmp     short K38
        ;
        ;-----  TEST FOR RESET KEY SEQUENCE (CTL ALT DEL)
K29:                                    ; TEST-RESET
        test    bl, CTL_SHIFT           ; ARE WE IN CONTROL SHIFT ALSO?
        jz      short K31               ; NO_RESET
        cmp     al, DEL_KEY             ; CTL-ALT STATE, TEST FOR DELETE KEY
        jne     short K31               ; NO_RESET, IGNORE
        ;
        ;-----  CTL-ALT-DEL HAS BEEN FOUND
        ; 26/08/2014
cpu_reset:
        ; IBM PC/AT ROM BIOS source code - 10/06/85 (TEST4.ASM - PROC_SHUTDOWN)
        ; Send FEh (system reset command) to the keyboard controller.
        mov     al, SHUT_CMD            ; SHUTDOWN COMMAND
        out     STATUS_PORT, al                 ; SEND TO KEYBOARD CONTROL PORT
khere:
        hlt                             ; WAIT FOR 80286 RESET
        jmp     short khere             ; INSURE HALT

        ;
        ;-----  IN ALTERNATE SHIFT, RESET NOT FOUND
K31:                                    ; NO-RESET
        cmp     al, 57                  ; TEST FOR SPACE KEY
        jne     short K311              ; NOT THERE
        mov     al, ' '                 ; SET SPACE CHAR
         jmp     K57                       ; BUFFER_FILL
K311:
        cmp     al, 15                  ; TEST FOR TAB KEY
        jne     short K312              ; NOT THERE
        mov     ax, 0A500h              ; SET SPECIAL CODE FOR ALT-TAB
         jmp     K57                       ; BUFFER_FILL
K312:
        cmp     al, 74                  ; TEST FOR KEY PAD -
         je      K37B                      ; GO PROCESS
        cmp     al, 78                  ; TEST FOR KEY PAD +
         je      K37B                      ; GO PROCESS
        ;
        ;-----  LOOK FOR KEY PAD ENTRY
K32:                                    ; ALT-KEY-PAD
        mov     edi, K30                ; ALT-INPUT-TABLE offset
        mov     ecx, 10                 ; LOOK FOR ENTRY USING KEYPAD
        repne   scasb                   ; LOOK FOR MATCH
        jne     short K33               ; NO_ALT_KEYPAD
        test    bh, LC_E0               ; IS THIS ONE OF THE NEW KEYS?
         jnz     K37C                      ; YES, JUMP, NOT NUMPAD KEY
        sub     edi, K30+1              ; DI NOW HAS ENTRY VALUE
        mov     al, [ALT_INPUT]         ; GET THE CURRENT BYTE
        mov     ah, 10                  ; MULTIPLY BY 10
        mul     ah
        add     ax, di                  ; ADD IN THE LATEST ENTRY
        mov     [ALT_INPUT], al         ; STORE IT AWAY
;K32A:
         jmp     K26                       ; THROW AWAY THAT KEYSTROKE
        ;
        ;-----  LOOK FOR SUPERSHIFT ENTRY
K33:                                    ; NO-ALT-KEYPAD
         mov     byte [ALT_INPUT], 0       ; ZERO ANY PREVIOUS ENTRY INTO INPUT
        mov     ecx, 26                 ; (DI),(ES) ALREADY POINTING
        repne   scasb                   ; LOOK FOR MATCH IN ALPHABET
        je      short K37A              ; MATCH FOUND, GO FILLL THE BUFFER
        ;
```

```
        ;-----  LOOK FOR TOP ROW OF ALTERNATE SHIFT
K34:                                    ; ALT-TOP-ROW
        cmp     al, 2                   ; KEY WITH '1' ON IT
        jb      short K37B              ; MUST BE ESCAPE
        cmp     al, 13                  ; IS IT IN THE REGION
        ja      short K35               ; NO, ALT SOMETHING ELSE
        add     ah, 118                 ; CONVERT PSEUDO SCAN CODE TO RANGE
        jmp     short K37A              ; GO FILL THE BUFFER
        ;
        ;-----  TRANSLATE ALTERNATE SHIFT PSEUDO SCAN CODES
K35:                                    ; ALT-FUNCTION
        cmp     al, F11_M               ; IS IT F11?
        jb      short K35A ; 20/02/2015        ; NO, BRANCH
        cmp     al, F12_M               ; IS IT F12?
        ja      short K35A ; 20/02/2015        ; NO, BRANCH
        add     ah, 52                  ; CONVERT TO PSEUDO SCAN CODE
        jmp     short K37A              ; GO FILL THE BUFFER
K35A:
        test    bh, LC_E0               ; DO WE HAVE ONE OF THE NEW KEYS?
        jz      short K37               ; NO, JUMP
        cmp     al, 28                  ; TEST FOR KEYPAD ENTER
         jne     short K35B                ; NOT THERE
        mov     ax, 0A600h              ; SPECIAL CODE
        jmp     K57                     ; BUFFER FILL
K35B:
        cmp     al, 83                  ; TEST FOR DELETE KEY
        je      short K37C              ; HANDLE WITH OTHER EDIT KEYS
        cmp     al, 53                  ; TEST FOR KEYPAD /
        ;jne    short K32A              ; NOT THERE, NO OTHER E0 SPECIALS
         jne     K26
        mov     ax, 0A400h              ; SPECIAL CODE
        jmp     K57                     ; BUFFER FILL
K37:
        cmp     al, 59                  ; TEST FOR FUNCTION KEYS (F1)
         jb      short K37B             ; NO FN, HANDLE W/OTHER EXTENDED
        cmp     al, 68                  ; IN KEYPAD REGION?
         ;ja    short K32A              ; IF SO, IGNORE
         ja      K26
        add     ah, 45                  ; CONVERT TO PSEUDO SCAN CODE
K37A:
        mov     al, 0                   ; ASCII CODE OF ZERO
         jmp     K57                       ; PUT IT IN THE BUFFER
K37B:
        mov     al, 0F0h                ; USE SPECIAL ASCII CODE
        jmp      K57                      ; PUT IT IN THE BUFFER
K37C:
        add     al, 80                  ; CONVERT SCAN CODE (EDIT KEYS)
        mov     ah, al                  ; (SCAN CODE NOT IN AH FOR INSERT)
        jmp      short K37A               ; PUT IT IN THE BUFFER
        ;
        ;-----  NOT IN ALTERNATE SHIFT
K38:                                    ; NOT-ALT-SHIFT
                                        ; BL STILL HAS SHIFT FLAGS
        test    bl, CTL_SHIFT           ; ARE WE IN CONTROL SHIFT?
        ;jnz    short K38A              ; YES, START PROCESSING
         jz      K44                       ; NOT-CTL-SHIFT
        ;
        ;-----  CONTROL SHIFT, TEST SPECIAL CHARACTERS
        ;-----  TEST FOR BREAK
K38A:
        cmp     al, SCROLL_KEY          ; TEST FOR BREAK
        jne     short K39               ; JUMP, NO-BREAK
        test    bh, KBX                 ; IS THIS THE ENHANCED KEYBOARD?
        jz      short K38B              ; NO, BREAK IS VALID
        test    bh, LC_E0               ; YES, WAS LAST CODE AN E0?
        jz      short K39               ; NO-BREAK, TEST FOR PAUSE
K38B:
        mov     ebx, [BUFFER_HEAD]      ; RESET BUFFER TO EMPTY
        mov     [BUFFER_TAIL], ebx
        mov     byte [BIOS_BREAK], 80h  ; TURN ON BIOS_BREAK BIT
        ;
        ;-----  ENABLE KEYBOARD
        mov     al, ENA_KBD             ; ENABLE KEYBOARD
        call    SHIP_IT                 ; EXECUTE ENABLE
        ;
        ; CTRL+BREAK code here !!!
        ;INT    1BH                     ; BREAK INTERRUPT VECTOR
        ; 17/10/2015
        call    ctrlbrk ; control+break subroutine
```

```
        ;
        sub     ax, ax                  ; PUT OUT DUMMY CHARACTER
         jmp     K57                       ; BUFFER_FILL
        ;
        ;----- TEST FOR PAUSE
K39:                                    ; NO_BREAK
        test    bh, KBX                 ; IS THIS THE ENHANCED KEYBOARD?
        jnz     short K41               ; YES, THEN THIS CAN'T BE PAUSE
        cmp     al, NUM_KEY             ; LOOK FOR PAUSE KEY
        jne     short K41               ; NO-PAUSE
K39P:
        or      byte [KB_FLAG_1], HOLD_STATE ; TURN ON THE HOLD FLAG
        ;
        ;----- ENABLE KEYBOARD
        mov     al, ENA_KBD             ; ENABLE KEYBOARD
        call    SHIP_IT                 ; EXECUTE ENABLE
K39A:
        mov     al, EOI                 ; END OF INTERRUPT TO CONTROL PORT
        out     20h, al ;out INTA00, al        ; ALLOW FURTHER KEYSTROKE INTERRUPTS
        ;
        ;----- DURING PAUSE INTERVAL, TURN COLOR CRT BACK ON
         cmp     byte [CRT_MODE], 7        ; IS THIS BLACK AND WHITE CARD
         je      short K40                     ; YES, NOTHING TO DO
        mov     dx, 03D8h               ; PORT FOR COLOR CARD
         mov     al, [CRT_MODE_SET]     ; GET THE VALUE OF THE CURRENT MODE
        out     dx, al                  ; SET THE CRT MODE, SO THAT CRT IS ON
        ;
K40:                                    ; PAUSE-LOOP
         test    byte [KB_FLAG_1], HOLD_STATE ; CHECK HOLD STATE FLAG
        jnz     short K40               ; LOOP UNTIL FLAG TURNED OFF
        ;
         jmp     K27                       ; INTERRUPT_RETURN_NO_EOI
         ;
        ;----- TEST SPECIAL CASE KEY 55
K41:                                    ; NO-PAUSE
        cmp     al, 55                  ; TEST FOR */PRTSC KEY
        jne     short K42               ; NOT-KEY-55
        test    bh, KBX                 ; IS THIS THE ENHANCED KEYBOARD?
        jz      short K41A              ; NO, CTL-PRTSC IS VALID
        test    bh, LC_E0               ; YES, WAS LAST CODE AN E0?
        jz      short K42B              ; NO, TRANSLATE TO A FUNCTION
K41A:
        mov     ax, 114*256             ; START/STOP PRINTING SWITCH
         jmp     K57                       ; BUFFER_FILL
        ;
        ;----- SET UP TO TRANSLATE CONTROL SHIFT
K42:                                    ; NOT-KEY-55
        cmp     al, 15                  ; IS IT THE TAB KEY?
        je      short K42B              ; YES, XLATE TO FUNCTION CODE
        cmp     al, 53                  ; IS IT THE / KEY?
        jne     short K42A              ; NO, NO MORE SPECIAL CASES
        test    bh, LC_E0               ; YES, IS IT FROM THE KEY PAD?
        jz      short K42A              ; NO, JUST TRANSLATE
        mov     ax, 9500h               ; YES, SPECIAL CODE FOR THIS ONE
        jmp     K57                     ; BUFFER FILL
K42A:
        ;;mov   ebx, _K8                ; SET UP TO TRANSLATE CTL
        cmp     al, 59                  ; IS IT IN CHARACTER TABLE?
         ;jb    short K45F                ; YES, GO TRANSLATE CHAR
        ;;jb    K56 ; 20/02/2015
        ;;jmp   K64 ; 20/02/2015
K42B:
        mov     ebx, _K8                ; SET UP TO TRANSLATE CTL
        jb      K56 ;; 20/02/2015
        jmp     K64
         ;
        ;----- NOT IN CONTROL SHIFT
K44:                                    ; NOT-CTL-SHIFT
        cmp     al, 55                  ; PRINT SCREEN KEY?
        jne     short K45               ; NOT PRINT SCREEN
        test    bh, KBX                 ; IS THIS ENHANCED KEYBOARD?
        jz      short K44A              ; NO, TEST FOR SHIFT STATE
        test    bh, LC_E0               ; YES, LAST CODE A MARKER?
        jnz     short K44B              ; YES, IS PRINT SCREEN
        jmp     short K45C              ; NO, TRANSLATE TO '*' CHARACTER
K44A:
        test    bl, LEFT_SHIFT+RIGHT_SHIFT ; NOT 101 KBD, SHIFT KEY DOWN?
        jz      short K45C              ; NO, TRANSLATE TO '*' CHARACTER
        ;
```

```
        ;----- ISSUE INTERRUPT TO INDICATE PRINT SCREEN FUNCTION
K44B:
        mov     al, ENA_KBD             ; INSURE KEYBOARD IS ENABLED
        call    SHIP_IT                 ; EXECUTE ENABLE
        mov     al, EOI                 ; END OF CURRENT INTERRUPT
        out     20h, al ;out INTA00, al       ; SO FURTHER THINGS CAN HAPPEN
        ; Print Screen !!!              ; ISSUE PRINT SCREEN INTERRUPT (INT 05h)
        ;PUSH   BP                      ; SAVE POINTER
        ;INT    5H                      ; ISSUE PRINT SCREEN INTERRUPT
        ;POP    BP                      ; RESTORE POINTER
         and     byte [KB_FLAG_3], ~(LC_E0+LC_E1) ; ZERO OUT THESE FLAGS
         jmp     K27                       ; GO BACK WITHOUT EOI OCCURRING
        ;
        ;----- HANDLE IN-CORE KEYS
K45:                                    ; NOT-PRINT-SCREEN
        cmp     al, 58                  ; TEST FOR IN-CORE AREA
        ja      short K46               ; JUMP IF NOT
        cmp     al, 53                  ; IS THIS THE '/' KEY?
        jne     short K45A              ; NO, JUMP
        test    bh, LC_E0               ; WAS THE LAST CODE THE MARKER?
        jnz     short K45C              ; YES, TRANSLATE TO CHARACTER
K45A:
        mov     ecx, 26                 ; LENGHT OF SEARCH
        mov     edi, K30+10             ; POINT TO TABLE OF A-Z CHARS
        repne   scasb                   ; IS THIS A LETTER KEY?
                ; 20/02/2015
        jne     short K45B                ; NO, SYMBOL KEY
        ;
        test    bl, CAPS_STATE          ; ARE WE IN CAPS_LOCK?
        jnz     short K45D              ; TEST FOR SURE
K45B:
        test    bl, LEFT_SHIFT+RIGHT_SHIFT ; ARE WE IN SHIFT STATE?
        jnz     short K45E              ; YES, UPPERCASE
                                        ; NO, LOWERCASE
K45C:
        mov     ebx, K10                ; TRANSLATE TO LOWERCASE LETTERS
        jmp     short K56
K45D:                                   ; ALMOST-CAPS-STATE
        test    bl, LEFT_SHIFT+RIGHT_SHIFT ; CL ON. IS SHIFT ON, TOO?
        jnz     short K45C              ; SHIFTED TEMP OUT OF CAPS STATE
K45E:
        mov     ebx, K11                ; TRANSLATE TO UPPER CASE LETTERS
K45F:   jmp     short K56
        ;
        ;----- TEST FOR KEYS F1 - F10
K46:                                    ; NOT IN-CORE AREA
        cmp     al, 68                  ; TEST FOR F1 - F10
        ;ja     short K47               ; JUMP IF NOT
        ;jmp    short K53               ; YES, GO DO FN KEY PROCESS
        jna     short K53
        ;
        ;----- HANDLE THE NUMERIC PAD KEYS
K47:                                    ; NOT F1 - F10
        cmp     al, 83                  ; TEST NUMPAD KEYS
        ja      short K52               ; JUMP IF NOT
        ;
        ;----- KEYPAD KEYS, MUST TEST NUM LOCK FOR DETERMINATION
K48:
        cmp     al , 74                 ; SPECIAL CASE FOR MINUS
        je      short K45E              ; GO TRANSLATE
        cmp     al , 78                 ; SPECIAL CASE FOR PLUS
        je      short K45E              ; GO TRANSLATE
        test    bh, LC_E0               ; IS THIS ONE OFTHE NEW KEYS?
        jnz     short K49               ; YES, TRANSLATE TO BASE STATE
        ;
        test    bl, NUM_STATE           ; ARE WE IN NUM LOCK
        jnz     short K50               ; TEST FOR SURE
        test    bl, LEFT_SHIFT+RIGHT_SHIFT ; ARE WE IN SHIFT STATE?
        ;jnz    short K51               ; IF SHIFTED, REALLY NUM STATE
        jnz     short K45E
        ;
        ;----- BASE CASE FOR KEYPAD
K49:
        cmp     al, 76                  ; SPECIAL CASE FOR BASE STATE 5
        jne     short K49A              ; CONTINUE IF NOT KEYPAD 5
        mov     al, 0F0h                ; SPECIAL ASCII CODE
        jmp     short K57               ; BUFFER FILL
```

```
K49A:
        mov     ebx, K10                ; BASE CASE TABLE
        jmp     short K64               ; CONVERT TO PSEUDO SCAN
        ;
        ;----- MIGHT BE NUM LOCK, TEST SHIFT STATUS
K50:                                    ; ALMOST-NUM-STATE
         test    bl, LEFT_SHIFT+RIGHT_SHIFT
        jnz     short K49               ; SHIFTED TEMP OUT OF NUM STATE
K51:    jmp     short K45E              ; REALLY NUM STATE
        ;
        ;----- TEST FOR THE NEW KEYS ON WT KEYBOARDS
K52:                                    ; NOT A NUMPAD KEY
        cmp     al, 86                  ; IS IT THE NEW WT KEY?
        ;jne    short K53               ; JUMP IF NOT
        ;jmp    short K45B              ; HANDLE WITH REST OF LETTER KEYS
        je      short K45B
        ;
        ;----- MUST BE F11 OR F12
K53:                                    ; F1 - F10 COME HERE, TOO
        test    bl, LEFT_SHIFT+RIGHT_SHIFT ; TEST SHIFT STATE
        jz      short K49               ; JUMP, LOWER CASE PSEUDO SC'S
                ; 20/02/2015
        mov     ebx, K11                ; UPPER CASE PSEUDO SCAN CODES
        jmp     short K64               ; TRANSLATE SCAN
        ;
        ;----- TRANSLATE THE CHARACTER
K56:                                    ; TRANSLATE-CHAR
        dec     al                      ; CONVERT ORIGIN
        xlat                            ; CONVERT THE SCAN CODE TO ASCII
        test    byte [KB_FLAG_3], LC_E0        ; IS THIS A NEW KEY?
        jz      short K57               ; NO, GO FILL BUFFER
        mov     ah, MC_E0               ; YES, PUT SPECIAL MARKER IN AH
        jmp     short K57               ; PUT IT INTO THE BUFFER
        ;
        ;----- TRANSLATE SCAN FOR PSEUDO SCAN CODES
K64:                                    ; TRANSLATE-SCAN-ORGD
        dec     al                      ; CONVERT ORIGIN
        xlat                             ; CTL TABLE SCAN
        mov     ah, al                  ; PUT VALUE INTO AH
        mov     al, 0                   ; ZERO ASCII CODE
        test    byte [KB_FLAG_3], LC_E0        ; IS THIS A NEW KEY?
        jz      short K57               ; NO, GO FILL BUFFER
        mov     al, MC_E0               ; YES, PUT SPECIAL MARKER IN AL
        ;
        ;----- PUT CHARACTER INTO BUFFER
K57:                                    ; BUFFER_FILL
        cmp     al, -1                  ; IS THIS AN IGNORE CHAR
         ;je    short K59               ; YES, DO NOTHING WITH IT
        je       K26                    ; YES, DO NOTHING WITH IT
        cmp     ah, -1                  ; LOOK FOR -1 PSEUDO SCAN
         ;jne   short K61               ; NEAR_INTERRUPT_RETURN
        je       K26                    ; INTERRUPT_RETURN
;K59:                                   ; NEAR_INTERRUPT_RETURN
;       jmp     K26                     ; INTERRUPT_RETURN
K61:                                    ; NOT-CAPS-STATE
        mov     ebx, [BUFFER_TAIL]      ; GET THE END POINTER TO THE BUFFER
        mov     esi, ebx                ; SAVE THE VALUE
        call    _K4                     ; ADVANCE THE TAIL
        cmp     ebx, [BUFFER_HEAD]      ; HAS THE BUFFER WRAPPED AROUND
        je      short K62               ; BUFFER_FULL_BEEP
        mov     [esi], ax               ; STORE THE VALUE
        mov     [BUFFER_TAIL], ebx      ; MOVE THE POINTER UP
        jmp     K26
        ;;cli                           ; TURN OFF INTERRUPTS
        ;;mov   al, EOI                 ; END OF INTERRUPT COMMAND
        ;;out   INTA00, al              ; SEND COMMAND TO INTERRUPT CONTROL PORT
        ;MOV    AL, ENA_KBD             ; INSURE KEYBOARD IS ENABLED
        ;CALL   SHIP_IT                 ; EXECUTE ENABLE
        ;MOV    AX, 9102H               ; MOVE IN POST CODE & TYPE
        ;INT    15H                     ; PERFORM OTHER FUNCTION
        ;;and   byte [KB_FLAG_3],~(LC_E0+LC_E1) ; RESET LAST CHAR H.C. FLAG
        ;JMP    K27A                    ; INTERRUPT_RETURN
        ;;jmp    K27
        ;
        ;----- BUFFER IS FULL SOUND THE BEEPER
K62:
        mov     al, EOI                 ; ENABLE INTERRUPT CONTROLLER CHIP
        out     INTA00, al
        mov     cx, 678                 ; DIVISOR FOR 1760 HZ
```

```
        mov     bl, 4                   ; SHORT BEEP COUNT (1/16 + 1/64 DELAY)
        call    beep                    ; GO TO COMMON BEEP HANDLER
        jmp      K27                    ; EXIT

SHIP_IT:
        ;-----------------------------------------------------------------------
-
        ; SHIP_IT
        ;       THIS ROUTINES HANDLES TRANSMISSION OF COMMAND AND DATA BYTES
        ;       TO THE KEYBOARD CONTROLLER.
        ;-----------------------------------------------------------------------
-
        ;
        push    ax                      ; SAVE DATA TO SEND

        ;-----  WAIT FOR COMMAND TO ACCEPTED
        cli                             ; DISABLE INTERRUPTS TILL DATA SENT
        ; xor   ecx, ecx                ; CLEAR TIMEOUT COUNTER
        mov     ecx, 10000h
S10:
        in      al, STATUS_PORT                 ; READ KEYBOARD CONTROLLER STATUS
        test    al, INPT_BUF_FULL       ; CHECK FOR ITS INPUT BUFFER BUSY
        loopnz  S10                     ; WAIT FOR COMMAND TO BE ACCEPTED

        pop     ax                      ; GET DATA TO SEND
        out     STATUS_PORT, al                 ; SEND TO KEYBOARD CONTROLLER
        sti                             ; ENABLE INTERRUPTS AGAIN
        retn                            ; RETURN TO CALLER

SND_DATA:
        ; ----------------------------------------------------------------------
--
        ; SND_DATA
        ;       THIS ROUTINES HANDLES TRANSMISSION OF COMMAND AND DATA BYTES
        ;       TO THE KEYBOARD AND RECEIPT OF ACKNOWLEDGEMENTS. IT ALSO
        ;       HANDLES ANY RETRIES IF REQUIRED
        ; ----------------------------------------------------------------------
--
        ;
        push    ax                      ; SAVE REGISTERS
        push    bx
        push    ecx
        mov     bh, al                  ; SAVE TRANSMITTED BYTE FOR RETRIES
        mov     bl, 3                   ; LOAD RETRY COUNT
SD0:
        cli                             ; DISABLE INTERRUPTS
        and     byte [KB_FLAG_2], ~(KB_FE+KB_FA) ; CLEAR ACK AND RESEND FLAGS
        ;
        ;-----  WAIT FOR COMMAND TO BE ACCEPTED
        mov     ecx, 10000h             ; MAXIMUM WAIT COUNT
SD5:
        in      al, STATUS_PORT                 ; READ KEYBOARD PROCESSOR STATUS PORT
        test    al, INPT_BUF_FULL       ; CHECK FOR ANY PENDING COMMAND
        loopnz  SD5                     ; WAIT FOR COMMAND TO BE ACCEPTED
        ;
        mov     al, bh                  ; REESTABLISH BYTE TO TRANSMIT
        out     PORT_A, al              ; SEND BYTE
        sti                             ; ENABLE INTERRUPTS
        ;mov    cx, 01A00h              ; LOAD COUNT FOR 10 ms+
        mov     ecx, 0FFFFh
SD1:
        test    byte [KB_FLAG_2], KB_FE+KB_FA ; SEE IF EITHER BIT SET
        jnz     short SD3               ; IF SET, SOMETHING RECEIVED GO PROCESS
        loop    SD1                     ; OTHERWISE WAIT
SD2:
        dec     bl                      ; DECREMENT RETRY COUNT
        jnz     short SD0               ; RETRY TRANSMISSION
        or      byte [KB_FLAG_2], KB_ERR ; TURN ON TRANSMIT ERROR FLAG
        jmp     short SD4               ; RETRIES EXHAUSTED FORGET TRANSMISSION
SD3:
        test    byte [KB_FLAG_2], KB_FA ; SEE IF THIS IS AN ACKNOWLEDGE
        jz      short SD2               ; IF NOT, GO RESEND
SD4:
        pop     ecx                     ; RESTORE REGISTERS
        pop     bx
        pop     ax
        retn                            ; RETURN, GOOD TRANSMISSION
```

```
SND_LED:
        ; ----------------------------------------------------------------------
--
        ; SND_LED
        ;       THIS ROUTINES TURNS ON THE MODE INDICATORS.
        ;
        ;-----------------------------------------------------------------------
--
        ;
        cli                         ; TURN OFF INTERRUPTS
        test    byte [KB_FLAG_2], KB_PR_LED ; CHECK FOR MODE INDICATOR UPDATE
        jnz     short SL1           ; DON'T UPDATE AGAIN IF UPDATE UNDERWAY
        ;
        or      byte [KB_FLAG_2], KB_PR_LED ; TURN ON UPDATE IN PROCESS
        mov     al, EOI             ; END OF INTERRUPT COMMAND
        out     20h, al ;out INTA00, al       ; SEND COMMAND TO INTERRUPT CONTROL PORT
        jmp     short SL0           ; GO SEND MODE INDICATOR COMMAND
SND_LED1:
        cli                         ; TURN OFF INTERRUPTS
        test    byte [KB_FLAG_2], KB_PR_LED ; CHECK FOR MODE INDICATOR UPDATE
        jnz     short SL1           ; DON'T UPDATE AGAIN IF UPDATE UNDERWAY
        ;
        or      byte [KB_FLAG_2], KB_PR_LED ; TURN ON UPDATE IN PROCESS
SL0:
        mov     al, LED_CMD         ; LED CMD BYTE
        call    SND_DATA            ; SEND DATA TO KEYBOARD
        cli
        call    MAKE_LED            ; GO FORM INDICATOR DATA BYTE
        and     byte [KB_FLAG_2], 0F8h; ~KB_LEDS ; CLEAR MODE INDICATOR BITS
        or      [KB_FLAG_2], al     ; SAVE PRESENT INDICATORS FOR NEXT TIME
        test    byte [KB_FLAG_2], KB_ERR ; TRANSMIT ERROR DETECTED
        jnz     short SL2           ; IF SO, BYPASS SECOND BYTE TRANSMISSION
        ;
        call    SND_DATA            ; SEND DATA TO KEYBOARD
        cli                         ; TURN OFF INTERRUPTS
        test    byte [KB_FLAG_2], KB_ERR ; TRANSMIT ERROR DETECTED
        jz      short SL3           ; IF NOT, DON'T SEND AN ENABLE COMMAND
SL2:
        mov     al, KB_ENABLE       ; GET KEYBOARD CSA ENABLE COMMAND
        call    SND_DATA            ; SEND DATA TO KEYBOARD
        cli                         ; TURN OFF INTERRUPTS
SL3:
        and     byte [KB_FLAG_2], ~(KB_PR_LED+KB_ERR) ; TURN OFF MODE INDICATOR
SL1:                                ; UPDATE AND TRANSMIT ERROR FLAG
        sti                         ; ENABLE INTERRUPTS
        retn                        ; RETURN TO CALLER

MAKE_LED:
        ;-----------------------------------------------------------------------
-
        ; MAKE_LED
        ;       THIS ROUTINES FORMS THE DATA BYTE NECESSARY TO TURN ON/OFF
        ;       THE MODE INDICATORS.
        ;-----------------------------------------------------------------------
-
        ;
        ;push   cx                  ; SAVE CX
        mov     al, [KB_FLAG]       ; GET CAPS & NUM LOCK INDICATORS
        and     al, CAPS_STATE+NUM_STATE+SCROLL_STATE ; ISOLATE INDICATORS
        ;mov    cl, 4               ; SHIFT COUNT
        ;rol    al, cl              ; SHIFT BITS OVER TO TURN ON INDICATORS
        rol     al, 4 ; 20/02/2015
        and     al, 07h             ; MAKE SURE ONLY MODE BITS ON
        ;pop    cx
        retn                        ; RETURN TO CALLER

; % include 'kybdata.inc'   ; KEYBOARD DATA ; 11/03/2015


; /// End Of KEYBOARD FUNCTIONS ///
```

```
; Retro UNIX 386 v1 Kernel - VIDEO.INC
; Last Modification: 13/08/2015
;                (Video Data is in 'VIDATA.INC')
;
; ///////// VIDEO (CGA) FUNCTIONS ///////////////

; 30/06/2015
; 27/06/2015
; 11/03/2015
; 02/09/2014
; 30/08/2014
; VIDEO FUNCTIONS
; (write_tty - Retro UNIX 8086 v1 - U9.ASM, 01/02/2014)

write_tty:
        ; 13/08/2015
        ; 02/09/2014
        ; 30/08/2014 (Retro UNIX 386 v1 - beginning)
        ; 01/02/2014 (Retro UNIX 8086 v1 - last update)
        ; 03/12/2013 (Retro UNIX 8086 v1 - beginning)
        ; (Modified registers: EAX, EBX, ECX, EDX, ESI, EDI)
        ;
        ; INPUT -> AH = Color (Forecolor, Backcolor)
        ;          AL = Character to be written
        ;          EBX = Video Page (0 to 7)
        ;          (BH = 0 --> Video Mode 3)

RVRT    equ     00001000b       ; VIDEO VERTICAL RETRACE BIT
RHRZ    equ     00000001b       ; VIDEO HORIZONTAL RETRACE BIT

; Derived from "WRITE_TTY" procedure of IBM "pc-at" rombios source code
; (06/10/1985), 'video.asm', INT 10H, VIDEO_IO
;
; 06/10/85  VIDEO DISPLAY BIOS
;
;--- WRITE_TTY ------------------------------------------------------------------
;                                                                      :
;   THIS INTERFACE PROVIDES A TELETYPE LIKE INTERFACE TO THE           :
;   VIDEO CARDS. THE INPUT CHARACTER IS WRITTEN TO THE CURRENT         :
;   CURSOR POSITION, AND THE CURSOR IS MOVED TO THE NEXT POSITION.     :
;   IF THE CURSOR LEAVES THE LAST COLUMN OF THE FIELD, THE COLUMN      :
;   IS SET TO ZERO, AND THE ROW VALUE IS INCREMENTED. IF THE ROW       :
;   ROW VALUE LEAVES THE FIELD, THE CURSOR IS PLACED ON THE LAST ROW,  :
;   FIRST COLUMN, AND THE ENTIRE SCREEN IS SCROLLED UP ONE LINE.       :
;   WHEN THE SCREEN IS SCROLLED UP, THE ATTRIBUTE FOR FILLING THE      :
;   NEWLY BLANKED LINE IS READ FROM THE CURSOR POSITION ON THE PREVIOUS :
;   LINE BEFORE THE SCROLL, IN CHARACTER MODE. IN GRAPHICS MODE,       :
;   THE 0 COLOR IS USED.                                               :
;   ENTRY --                                                           :
;     (AH) = CURRENT CRT MODE                                          :
;     (AL) = CHARACTER TO BE WRITTEN                                   :
;          NOTE THAT BACK SPACE, CARRIAGE RETURN, BELL AND LINE FEED ARE :
;          HANDLED AS COMMANDS RATHER THAN AS DISPLAY GRAPHICS CHARACTERS :
;     (BL) = FOREGROUND COLOR FOR CHAR WRITE IF CURRENTLY IN A GRAPHICS MODE :
;   EXIT --                                                            :
;     ALL REGISTERS SAVED                                              :
;--------------------------------------------------------------------------------

        cli
        ;
        ; READ CURSOR (04/12/2013)
        ; Retro UNIX 386 v1 Modifications: 30/08/2014
        or      bh, bh
        jnz     beeper
        ; 01/09/2014
        cmp     byte [CRT_MODE], 3
        je      short m3
        ;
        call    set_mode
m3:
        mov     esi, ebx ; 13/08/2015 (0 to 7)
        shl     si, 1
        add     esi, cursor_posn
        mov     dx, [esi]
        ;
        ; dx now has the current cursor position
        ;
        cmp     al, 0Dh         ; is it carriage return or control character
        jbe     short u8
```

```
        ; write the char to the screen
u0:
        ; ah = attribute/color
        ; al = character
        ; bl = video page number (0 to 7)
        ; bh = 0
        ;
        call    write_c_current
        ;
        ; position the cursor for next char
        inc     dl              ; next column
        ;cmp    dl, [CRT_COLS]
        cmp     dl, 80          ; test for column overflow
         jne     set_cpos
        mov     dl, 0           ; column = 0
u10:                            ; (line feed found)
        cmp     dh, 25-1        ; check for last row
        jb      short u6
        ;
        ; scroll required
u1:
        ; SET CURSOR POSITION (04/12/2013)
        call    set_cpos
        ;
        ; determine value to fill with during scroll
u2:
        ; READ_AC_CURRENT              :
        ;   THIS ROUTINE READS THE ATTRIBUTE AND CHARACTER
        ;    AT THE CURRENT CURSOR POSITION
        ;
        ; INPUT
        ;       (AH) = CURRENT CRT MODE
        ;       (BH) = DISPLAY PAGE ( ALPHA MODES ONLY )
        ;       (DS) = DATA SEGMENT
        ;       (ES) = REGEN SEGMENT
        ; OUTPUT
        ;       (AL) = CHARACTER READ
        ;       (AH) = ATTRIBUTE READ
        ;
        ; mov   ah, [CRT_MODE] ; move current mode into ah
        ;
        ; bl = video page number
        ;
        call    find_position  ; get regen location and port address
        ; dx = status port
        ; esi = cursor location/address
p11:
        sti                     ; enable interrupts
        nop                     ; allow for small interupts window
        cli                     ; blocks interrupts for single loop
        in      al, dx          ; get status from adapter
        test    al, RHRZ        ; is horizontal retrace low
        jnz     short p11       ; wait until it is
p12:                            ; now wait for either retrace high
        in      al, dx          ; get status
        test    al, RVRT+RHRZ  ; is horizontal or vertical retrace high
        jz      short p12       ; wait until either is active
p13:
        add     esi, 0B8000h    ; 30/08/2014 (Retro UNIX 386 v1)
        mov     ax, [esi]       ; get the character and attribute
        ;
        ; al = character, ah = attribute
        ;
        sti
        ; bl = video page number
u3:
        ;;mov   ax, 0601h       ; scroll one line
        ;;sub   cx, cx          ; upper left corner
        ;;mov   dh, 25-1        ; lower right row
        ;;;mov dl, [CRT_COLS]
        ;mov    dl, 80          ; lower right column
        ;;dec   dl
        ;;mov   dl, 79

        ;;call  scroll_up       ; 04/12/2013
        ;;; 11/03/2015
        ; 02/09/2014
        ;;;mov cx, [crt_ulc] ; Upper left corner  (0000h)
        ;;;mov dx, [crt_lrc] ; Lower right corner (184Fh)
```

```
        ; 11/03/2015
        sub     cx, cx
        mov     dx, 184Fh ; dl= 79 (column), dh = 24 (row)
        ;
        mov     al, 1           ; scroll 1 line up
                ; ah = attribute
        jmp     scroll_up
;u4:
        ;;int   10h             ; video-call return
                                ; scroll up the screen
                                ; tty return
;u5:
        ;retn                   ; return to the caller

u6:                             ; set-cursor-inc
        inc     dh              ; next row
                                ; set cursor
;u7:
        ;;mov   ah, 02h
        ;;jmp   short u4        ; establish the new cursor
        ;call   set_cpos
        ;jmp    short u5
        jmp      set_cpos

        ; check for control characters
u8:
        je      short u9
        cmp     al, 0Ah         ; is it a line feed (0Ah)
        je      short u10
        cmp     al, 07h         ; is it a bell
        je      short u11
        cmp     al, 08h         ; is it a backspace
        ;jne    short u0
        je      short bs        ; 12/12/2013
        ; 12/12/2013 (tab stop)
        cmp     al, 09h         ; is it a tab stop
        jne     short u0
        mov     al, dl
        cbw
        mov     cl, 8
        div     cl
        sub     cl, ah
ts:
        ; 02/09/2014
        ; 01/09/2014
        mov     al, 20h
tsloop:
        push    cx
        push    ax
        xor     bh, bh
        ;mov    bl, [active_page]
        call    m3
        pop     ax  ; ah = attribute/color
        pop     cx
        dec     cl
        jnz     short tsloop
        retn
bs:
        ; back space found

        or      dl, dl          ; is it already at start of line
        ;je     short u7        ; set_cursor
        jz      short set_cpos
        dec     dx              ; no -- just move it back
        ;jmp    short u7
        jmp     short set_cpos

        ; carriage return found
u9:
        mov     dl, 0           ; move to first column
        ;jmp    short u7
        jmp     short set_cpos

        ; line feed found
;u10:
;       cmp     dh, 25-1        ; bottom of screen
;       jne     short u6        ; no, just set the cursor
;        jmp     u1                ; yes, scroll the screen
```

```
beeper:
        ; 30/08/2014 (Retro UNIX 386 v1)
        ; 18/01/2014
        ; 03/12/2013
        ; bell found
u11:
        sti
        cmp     bl, [active_page]
        jne     short u12       ; Do not sound the beep
                                ; if it is not written on the active page
        mov     cx, 1331        ; divisor for 896 hz tone
        mov     bl, 31          ; set count for 31/64 second for beep
        ;call   beep            ; sound the pod bell
        ;jmp    short u5        ; tty_return
        ;retn

TIMER   equ     040h            ; 8254 TIMER - BASE ADDRESS
PORT_B  equ     061h            ; PORT B READ/WRITE DIAGNOSTIC REGISTER
GATE2   equ     00000001b       ; TIMER 2 INPUT CATE CLOCK BIT
SPK2    equ     00000010b       ; SPEAKER OUTPUT DATA ENABLE BIT

beep:
        ; 07/02/2015
        ; 30/08/2014 (Retro UNIX 386 v1)
        ; 18/01/2014
        ; 03/12/2013
        ;
        ; TEST4.ASM - 06/10/85  POST AND BIOS UTILITY ROUTINES
        ;
        ; ROUTINE TO SOUND THE BEEPER USING TIMER 2 FOR TONE
        ;
        ; ENTRY:
        ;    (BL) = DURATION COUNTER ( 1 FOR 1/64 SECOND )
        ;    (CX) = FREQUENCY DIVISOR (1193180/FREQUENCY) (1331 FOR 886 HZ)
        ; EXIT:                        :
        ;    (AX),(BL),(CX) MODIFIED.

        pushf  ; 18/01/2014     ; save interrupt status
        cli                     ; block interrupts during update
        mov     al, 10110110b   ; select timer 2, lsb, msb binary
        out     TIMER+3, al     ; write timer mode register
        jmp     $+2             ; I/O delay
        mov     al, cl          ; divisor for hz (low)
        out     TIMER+2,AL      ; write timer 2 count - lsb
        jmp     $+2             ; I/O delay
        mov     al, ch          ; divisor for hz (high)
        out     TIMER+2, al     ; write timer 2 count - msb
        in      al, PORT_B      ; get current setting of port
        mov     ah, al          ; save that setting
        or      al, GATE2+SPK2  ; gate timer 2 and turn speaker on
        out     PORT_B, al      ; and restore interrupt status
        ;popf   ; 18/01/2014
        sti
g7:                             ; 1/64 second per count (bl)
        mov     ecx, 1035       ; delay count for 1/64 of a second
        call    waitf           ; go to beep delay 1/64 count
        dec     bl              ; (bl) length count expired?
        jnz     short g7        ; no - continue beeping speaker
        ;
        ;pushf                  ; save interrupt status
        cli     ; 18/01/2014    ; block interrupts during update
        in      al, PORT_B      ; get current port value
         ;or      al, not (GATE2+SPK2) ; isolate current speaker bits in case
         or      al, ~(GATE2+SPK2)
         and    ah, al          ; someone turned them off during beep
        mov     al, ah          ; recover value of port
         ;or      al, not (GATE2+SPK2) ; force speaker data off
        or      al, ~(GATE2+SPK2) ; isolate current speaker bits in case
        out     PORT_B, al      ; and stop speaker timer
        ;popf                   ; restore interrupt flag state
        sti
        mov     ecx, 1035       ; force 1/64 second delay (short)
        call    waitf           ; minimum delay between all beeps
        ;pushf                  ; save interrupt status
        cli                     ; block interrupts during update
        in      al, PORT_B      ; get current port value in case
        and     al, GATE2+SPK2  ; someone turned them on
        or      al, ah          ; recover value of port_b
        out     PORT_B, al      ; restore speaker status
```

```
        popf                    ; restore interrupt flag state
u12:
        retn

REFRESH_BIT equu       00010000b      ; REFRESH TEST BIT

WAITF:
waitf:
        ; 30/08/2014 (Retro UNIX 386 v1)
        ; 03/12/2013
        ;
;       push ax                 ; save work register (ah)
;waitf1:
                                ; use timer 1 output bits
;       in      al, PORT_B      ; read current counter output status
;       and     al, REFRESH_BIT        ; mask for refresh determine bit
;       cmp     al, ah          ; did it just change
;       je      short waitf1    ; wait for a change in output line
;       ;
;       mov     ah, al          ; save new lflag state
;       loop    waitf1          ; decrement half cycles till count end
;       ;
;       pop     ax              ; restore (ah)
;       retn                    ; return (cx)=0

; 06/02/2015 (unix386.s <-- dsectrm2.s)
; 17/12/2014 (dsectrm2.s)
; WAITF
; /// IBM PC-XT Model 286 System BIOS Source Code - Test 4 - 06/10/85 ///
;
;---WAITF-----------------------------------------------------------------------
;       FIXED TIME WAIT ROUTINE (HARDWARE CONTROLLED - NOT PROCESSOR)
; ENTRY:
;       (CX) =  COUNT OF 15.085737 MICROSECOND INTERVALS TO WAIT
;               MEMORY REFRESH TIMER 1 OUTPUT USED AS REFERENCE
; EXIT:
;               AFTER (CX) TIME COUNT (PLUS OR MINUS 16 MICROSECONDS)
;       (CX) = 0
;-------------------------------------------------------------------------------

; Refresh period: 30 micro seconds (15-80 us)
; (16/12/2014 - AWARDBIOS 1999 - ATORGS.ASM, WAIT_REFRESH)

;WAITF:                                 ; DELAY FOR (CX)*15.085737 US
        PUSH    AX                      ; SAVE WORK REGISTER (AH)
        ; 16/12/2014
        ;shr    cx, 1                   ; convert to count of 30 micro seconds
        shr     ecx, 1  ; 21/02/2015
;17/12/2014
;WAITF1:
;       IN      AL, PORT_B    ;061h     ; READ CURRENT COUNTER OUTPUT STATUS
;       AND     AL, REFRESH_BIT         ;00010000b ; MASK FOR REFRESH DETERMINE BIT
;       CMP     AL, AH                  ; DID IT JUST CHANGE
;       JE      short WAITF1            ; WAIT FOR A CHANGE IN OUTPUT LINE
;       MOV     AH, AL                  ; SAVE NEW FLAG STATE
;       LOOP    WAITF1                  ; DECREMENT HALF CYCLES TILL COUNT END
        ;
        ; 17/12/2014
        ;
        ; Modification from 'WAIT_REFRESH' procedure of AWARD BIOS - 1999
        ;
;WAIT_REFRESH:  Uses port 61, bit 4 to have CPU speed independent waiting.
;       INPUT:  CX = number of refresh periods to wait
;              (refresh periods = 1 per 30 microseconds on most machines)
WR_STATE_0:
        IN      AL,PORT_B               ; IN AL,SYS1
        TEST    AL,010H
        JZ      SHORT WR_STATE_0
WR_STATE_1:
        IN      AL,PORT_B               ; IN AL,SYS1
        TEST    AL,010H
        JNZ     SHORT WR_STATE_1
         LOOP     WR_STATE_0
        ;
        POP     AX                      ; RESTORE (AH)
        RETn                            ; (CX) = 0
```

```
set_cpos:
        ; 27/06/2015
        ; 01/09/2014
        ; 30/08/2014 (Retro UNIX 386 v1 - beginning)
        ;
        ; 12/12/2013 (Retro UNIX 8086 v1 - last update)
        ; 04/12/2013 (Retro UNIX 8086 v1 - beginning)
        ;
        ; VIDEO.ASM - 06/10/85  VIDEO DISPLAY BIOS
        ;
        ; SET_CPOS
        ;       THIS ROUTINE SETS THE CURRENT CURSOR POSITION TO THE
        ;       NEW X-Y VALUES PASSED
        ; INPUT
        ;       DX - ROW,COLUMN OF NEW CURSOR
        ;       BH - DISPLAY PAGE OF CURSOR
        ; OUTPUT
        ;       CURSOR IS SET AT 6845 IF DISPLAY PAGE IS CURRENT DISPLAY
        ;
         movzx   eax, bl  ; BL = video page number ; 27/06/2015 (movzx)
         shl     al, 1   ; word offset
        mov     esi, cursor_posn
         add     esi, eax
        mov     [esi], dx ; save the pointer
        cmp     [active_page], bl
        jne     short m17
        ;call   m18     ; CURSOR SET
;m17:                   ; SET_CPOS_RETURN
        ; 01/09/2014
;       retn
                ; DX  = row/column
m18:
        call    position ; determine location in regen buffer
        mov     cx, [CRT_START]
        add     cx, ax  ; add char position in regen buffer
                        ; to the start address (offset) for this page
        shr     cx, 1   ; divide by 2 for char only count
        mov     ah, 14  ; register number for cursor
        ;call   m16     ; output value to the 6845
        ;retn

        ;-----  THIS ROUTINE OUTPUTS THE CX REGISTER
        ;       TO THE 6845 REGISTERS NAMED IN (AH)
m16:
        cli
        ;mov    dx, [addr_6845] ; address register
        mov     dx, 03D4h ; I/O address of color card
        mov     al, ah  ; get value
        out     dx, al  ; register set
        inc     dx      ; data register
        jmp     $+2     ; i/o delay
        mov     al, ch  ; data
        out     dx, al
        dec     dx
        mov     al, ah
        inc     al      ; point to other data register
        out     dx, al  ; set for second register
        inc     dx
        jmp     $+2     ; i/o delay
        mov     al, cl  ; second data value
        out     dx, al
        sti
m17:
        retn

set_ctype:
        ; 02/09/2014 (Retro UNIX 386 v1)
        ;
        ; VIDEO.ASM - 06/10/85  VIDEO DISPLAY BIOS

;       CH) = BITS 4-0 = START LINE FOR CURSOR
;        ** HARDWARE WILL ALWAYS CAUSE BLINK
;        ** SETTING BIT 5 OR 6 WILL CAUSE ERRATIC BLINKING
;           OR NO CURSOR AT ALL
;       (CL) = BITS 4-0 = END LINE FOR CURSOR
```

```
;----------------------------------------------
; SET_CTYPE
;       THIS ROUTINE SETS THE CURSOR VALUE
; INPUT
;       (CX) HAS CURSOR VALUE CH-START LINE, CL-STOP LINE
; OUTPUT
;       NONE
;----------------------------------------------

        mov     ah, 10 ; 6845 register for cursor set
        ;mov    [CURSOR_MODE], cx ; save in data area
        ;call   m16     ; output cx register
        ;retn
        jmp     m16


position:
        ; 27/06/2015
        ; 02/09/2014
        ; 30/08/2014 (Retro UNIX 386 v1)
        ; 04/12/2013 (Retro UNIX 8086 v1)
        ;
        ; VIDEO.ASM - 06/10/85  VIDEO DISPLAY BIOS
        ;
        ; POSITION
        ;       THIS SERVICE ROUTINE CALCULATES THE REGEN BUFFER ADDRESS
        ;       OF A CHARACTER IN THE ALPHA MODE
        ; INPUT
        ;       AX = ROW, COLUMN POSITION
        ; OUTPUT
        ;       AX = OFFSET OF CHAR POSITION IN REGEN BUFFER

                ; DX = ROW, COLUMN POSITION
        ;movzx  eax, byte [CRT_COLS] ; 27/06/2015
        xor     eax, eax ; 02/09/2014
        mov     al, 80   ; determine bytes to row
        mul     dh ;     row value
        xor     dh, dh   ; 0
        add     ax, dx   ; add column value to the result
        shl     ax, 1    ; * 2 for attribute bytes
                ; EAX = AX = OFFSET OF CHAR POSITION IN REGEN BUFFER
        retn

find_position:
        ; 27/06/2015
        ; 07/09/2014
        ; 02/09/2014
        ; 30/08/2014 (Retro UNIX 386 v1)
        ; VIDEO.ASM - 06/10/85  VIDEO DISPLAY BIOS
        movzx   ecx, bl ; video page number ; 27/06/2015 (movzx)
        mov     esi, ecx
        shl     si, 1
        mov     dx, [esi + cursor_posn]
        jz      short p21
        xor     si, si
p20:
        ;add    si, [CRT_LEN]
        add     si, 80*25*2 ; add length of buffer for one page
        loop    p20
p21:
        and     dx, dx
        jz      short p22
        call    position ; determine location in regen in page
        add     esi, eax ; add location to start of regen page




p22:
        ;mov    dx, [addr_6845] ; get base address of active display
        ;mov    dx, 03D4h ; I/O address of color card
        ;add    dx, 6   ; point at status port
        mov     dx, 03DAh ; status port
        ; cx = 0
        retn
```

```
scroll_up:
        ; 07/09/2014
        ; 02/09/2014
        ; 01/09/2014 (Retro UNIX 386 v1 - beginning)
        ; 04/04/2014
        ; 04/12/2013
        ;
        ; VIDEO.ASM - 06/10/85  VIDEO DISPLAY BIOS
        ;
        ; SCROLL UP
        ;       THIS ROUTINE MOVES A BLOCK OF CHARACTERS UP
        ;       ON THE SCREEN
        ; INPUT
        ;       (AH) = CURRENT CRT MODE
        ;       (AL) = NUMBER OF ROWS TO SCROLL
        ;       (CX) = ROW/COLUMN OF UPPER LEFT CORNER
        ;       (DX) = ROW/COLUMN OF LOWER RIGHT CORNER
        ;       (BH) = ATTRIBUTE TO BE USED ON BLANKED LINE
        ;       (DS) = DATA SEGMENT
        ;       (ES) = REGEN BUFFER SEGMENT
        ; OUTPUT
        ;       NONE -- THE REGEN BUFFER IS MODIFIED
        ;
        ;       bh = 0  (02/09/2014)
        ;
        ; ((ah = 3))
        ; cl = left upper column
        ; ch = left upper row
        ; dl = right lower column
        ; dh = right lower row
        ;
        ; al = line count
        ; ah = attribute to be used on blanked line
        ; bl = video page number (0 to 7)
        ;

        ; Test  Line Count
        or      al, al
        jz      short al_set
        mov     bh, dh  ; subtract lower row from upper row
        sub     bh, ch
        inc     bh      ; adjust difference by 1
        cmp     bh, al  ; line count = amount of rows in window?
        jne     short al_set ; if not the we're all set
        xor     al, al  ; otherwise set al to zero
al_set:
        xor     bh, bh  ; 0
        push    ax
        ;mov    esi, [crt_base]
         mov     esi, 0B8000h
         cmp     bl, [active_page]
        jne     short n0
        ;
         mov     ax, [CRT_START]
         add     si, ax
         jmp     short n1
n0:
         and     bl, bl
        jz      short n1
        mov     al, bl
n0x:
         ;add    si, [CRT_LEN]
         ;add    esi, 80*25*2
         add     si, 80*25*2
         dec    al
        jnz     short n0x

n1:
         ;Scroll position
        push    dx
        mov     dx, cx  ; now, upper left position in DX
        call    position
        add     esi, eax
        mov     edi, esi
        pop     dx      ; lower right position in DX
        sub     dx, cx
        inc     dh      ; dh = #rows
        inc     dl      ; dl = #cols in block
        pop     ax      ; al = line count, ah = attribute
```

```
        xor     ecx, ecx
        mov     cx, ax
        ;mov    ah, [CRT_COLS]
        mov     ah, 80
        mul     ah      ; determine offset to from address
        add     ax, ax  ; *2 for attribute byte
        ;
        push    ax      ; offset
        push    dx
        ;
        ; 04/04/2014
        mov     dx, 3DAh ; guaranteed to be color card here
n8:                       ; wait_display_enable
         in       al, dx   ; get port
        test    al, RVRT ; wait for vertical retrace
        jz      short n8 ; wait_display_enable
        mov     al, 25h
        mov     dl, 0D8h ; address control port
        out     dx, al ; turn off video during vertical retrace
        pop     dx      ; #rows, #cols
        pop     ax      ; offset
        xchg    ax, cx ;
        ; ecx = offset, al = line count, ah = attribute
;n9:
        or      al, al
         jz       short n3
         add      esi, ecx ; from address for scroll
        mov     bh, dh  ; #rows in block
        sub     bh, al ; #rows to be moved
n2:
        ; Move rows
        mov     cl, dl ; get # of cols to move
        push    esi
        push    edi     ; save start address
n10:
        movsw           ; move that line on screen
        dec     cl
         jnz      short n10
        pop     edi
        pop     esi     ; recover addresses
         ;mov     cl, [CRT_COLS]
        ;add    cl, cl
         ;mov     ecx, 80*2
         mov      cx, 80*2
         add      esi, ecx  ; next line
         add      edi, ecx
        dec     bh       ; count of lines to move
        jnz     short n2 ; row loop
        ; bh = 0
        mov     dh, al  ; #rows
n3:
        ; attribute in ah
        mov     al, ' ' ; fill with blanks
        ; Clear rows
                 ; dh =  #rows
         mov    cl, dl ; get # of cols to clear
         push     edi      ; save address
n11:
         stosw            ; store fill character
        dec     cl
         jnz      short n11
         pop      edi      ; recover address
        ;mov    cl, [CRT_COLS]
        ;add    cl, cl
         ;mov     ecx, 80*2
         mov    cl, 80*2
         add      esi, ecx  ; next line
         add      edi, ecx
        dec     dh
        jnz     short n3
        ;
        cmp     bl, [active_page]
        jne     short n6
        ;mov    al, [CRT_MODE_SET] ; get the value of mode set
        mov     al, 29h ; (ORGS.ASM), M7 mode set table value for mode 3
        mov     dx, 03D8h ; always set color card port
        out     dx, al
n6:
        retn
```

```
write_c_current:
        ; 30/08/2014 (Retro UNIX 386 v1)
        ; 18/01/2014
        ; 04/12/2013
        ;
        ; VIDEO.ASM - 06/10/85  VIDEO DISPLAY BIOS
        ;
        ; WRITE_C_CURRENT
        ;       THIS ROUTINE WRITES THE CHARACTER AT
        ;       THE CURRENT CURSOR POSITION, ATTRIBUTE UNCHANGED
        ; INPUT
        ;       (AH) = CURRENT CRT MODE
        ;       (BH) = DISPLAY PAGE
        ;       (CX) = COUNT OF CHARACTERS TO WRITE
        ;       (AL) = CHAR TO WRITE
        ;       (DS) = DATA SEGMENT
        ;       (ES) = REGEN SEGMENT
        ; OUTPUT
        ;       DISPLAY REGEN BUFFER UPDATED

        cli
        ; bl = video page
        ; al = character
        ; ah = color/attribute
        push    dx
        push    ax      ; save character & attribute/color
        call    find_position  ; get regen location and port address
        ; esi = regen location
        ; dx = status port
        ;
        ; WAIT FOR HORIZONTAL RETRACE OR VERTICAL RETRACE
        ;
p41:                    ; wait for horizontal retrace is low or vertical
        sti             ; enable interrupts first
        cmp     bl, [active_page]
        jne     short p44
        cli             ; block interrupts for single loop
        in      al, dx ; get status from the adapter
        test    al, RVRT ; check for vertical retrace first
        jnz     short p43 ; Do fast write now if vertical retrace
        test    al, RHRZ ; is horizontal retrace low
        jnz     short p41 ; wait until it is
p42:                    ;  wait for either retrace high
        in      al, dx ; get status again
        test    al, RVRT+RHRZ ; is horizontal or vertical retrace high
        jz      short p42 ; wait until either retrace active
p43:
        sti
p44:
        pop     ax      ; restore the character (al) & attribute (ah)
        add     esi, 0B8000h ; 30/08/2014 (crt_base)
                             ; Retro UNIX 386 v1 feature only!
        mov     [esi], ax
        pop     dx
        retn

set_mode:
        ; 02/09/2014 (Retro UNIX 386 v1)
        ;
        ; VIDEO.ASM - 06/10/85  VIDEO DISPLAY BIOS

;-------------------------------------------------------
; SET MODE                                            :
;       THIS ROUTINE INITIALIZES THE ATTACHMENT TO :
;       THE SELECTED MODE, THE SCREEN IS BLANKED.  :
; INPUT                                               :
;       (AL) - MODE SELECTED (RANGE 0-7)              :
; OUTPUT                                              :
;       NONE                                          :
;-------------------------------------------------------

        push    ebx
        push    edx
        push    eax

        ;mov    dx, 03D4h       ; address or color card
        mov     al, 3
```

```
;M8:
        mov     [CRT_MODE], al  ; save mode in global variable
        mov     al, 29h
        ;mov    [CRT_MODE_SET], al ; save the mode set value
        and     al, 037h        ; video off, save high resolution bit
        ;push   dx              ; save port value
        ;add    dx, 4           ; point to control register
        mov     dx, 3D8h
        out     dx, al          ; reset video to off to suppress rolling
        ;pop    dx

;M9:
        xor     ah, ah
        mov     ebx, video_params ; initialization table
        ;mov    ax, [ebx+10]      ; get the cursor mode from the table
        ;xchg   ah, al
        ;mov    [CURSOR_MODE], ax ; save cursor mode
        xor     ah, ah            ; ah is register number during loop

;----- LOOP THROUGH TABLE, OUTPUTTING REGISTER ADDRESS, THEN VALUE FROM TABLE

M10:                    ;  initialization loop
        mov     al, ah  ; get 6845 register number
        out     dx, al
        inc     dx       ; point to data port
        inc     ah      ; next register value
        mov     al, [ebx] ; get table value
        out     dx, al  ; out to chip
        inc     ebx     ; next in table
        dec     dx      ; back to pointer register
        loop    M10     ; do the whole table

;----- FILL REGEN AREA WITH BLANK
        ;xor    ax, ax
        ;mov    [CRT_START], ax  ; start address saved in global
        ;mov    [ACTIVE_PAGE], al ; 0 ; (re)set page value
        ;mov    ecx, 8192 ; number of words in color card
        ; black background, light gray characeter color, space character
        ;mov    ax, 0720h ; fill char for alpha - attribute
;M13:                     ; clear buffer
        ;add    edi, 0B8000h ; [crt_base]
        ;rep    stosw   ; FILL THE REGEN BUFFER WITH BLANKS

;----- ENABLE VIDEO AND CORRECT PORT SETTING
        ;mov    dx, 3D4h ; mov dx, word [ADDR_6845]
                         ; prepare to output to video enable port
        ;add    dx,4     ; point to the mode control gerister
        mov     dx, 3D8h
        ;mov    al, [CRT_MODE_SET] ; get the mode set value
        mov     al, 29h
        out     dx, al   ; set video enable port

;----- DETERMINE NUMBER OF COLUMNS, BOTH FOR ENTIRE DISPLAY
;----- AND THE NUMBER TO BE USED FOR TTY INTERFACE
        ;
        ;mov byte [CRT_COLS], 80h ; initialize number of columns count
        ;

;----- SET CURSOR POSITIONS
        push    edi
        ;mov    word [CRT_LEN], 80*25*2
        push    ecx
        mov     edi, cursor_posn
        mov     ecx, 4 ; clear all cursor positions (16 bytes)
        xor     eax, eax
        rep     stosd   ; fill with zeroes
        pop     ecx
        pop     edi

;----- SET UP OVERSCAN REGISTER
        inc     dx      ; set overscan port to a default
        mov     al, 30h ; 30H valuye for all modes except 640X200 bw
;M14:
        out     dx, al  ; output the correct value to 3D9 port
        ;mov    [CRT_PALETTE], al ; save the value for future use
```

```
;-----  NORMAL RETURN FROM ALL VIDEO RETURNS
        ;
        pop     eax
        pop     edx
        pop     ebx
        retn

tty_sw:
        ; 30/06/2015
        ; 27/06/2015
        ; 07/09/2014
        ; 02/09/2014 (Retro UNIX 386 v1 - beginning)
        ;
        ; (Modified registers : EAX)
        ;
         ;mov     byte [u.quant], 0  ; 04/03/2014
        ;
;act_disp_page:
        ; 30/06/2015
        ; 04/03/2014  (act_disp_page --> tty_sw)
        ; 10/12/2013
        ; 04/12/2013
        ;
        ; VIDEO.ASM - 06/10/85  VIDEO DISPLAY BIOS
        ;
        ; ACT_DISP_PAGE
        ;       THIS ROUTINE SETS THE ACTIVE DISPLAY PAGE, ALLOWING
        ;       THE FULL USE OF THE MEMORY SET ASIDE FOR THE VIDEO ATTACHMENT
        ; INPUT
        ;       AL HAS THE NEW ACTIVE DISPLAY PAGE
        ; OUTPUT
        ;       THE 6845 IS RESET TO DISPLAY THAT PAGE

        ;cli

        push    ebx
        push    cx
        push    dx
        ;
        mov     [active_page], al ; save active page value ; [ptty]
        ;mov    cx, [CRT_LEN] ; get saved length of regen buffer
        mov     cx, 25*80*2
        ; 27/06/2015
        movzx   ebx, al
        ;
        cbw     ; 07/09/2014 (ah=0)
        mul     cx      ; display page times regen length
        ; 10/12/2013
        mov     [CRT_START], ax ; save start address for later
        mov     cx, ax ; start address to cx
        ;sar    cx, 1
        shr     cx, 1   ; divide by 2 for 6845 handling
        mov     ah, 12  ; 6845 register for start address
        call    m16
        ;sal    bx, 1
        ; 01/09/2014
        shl     bl, 1   ; *2 for word offset
        add     ebx, cursor_posn
        mov     dx, [ebx] ; get cursor for this page
        call    m18
        ;
        pop     dx
        pop     cx
        pop     ebx
        ;
        ;sti
        ;
        retn

; % include 'vidata.inc' ; VIDEO DATA ; 11/03/2015


; /// End Of VIDEO FUNCTIONS ///
```

```
; Retro UNIX 386 v1 Kernel - DISKIO.INC
; Last Modification: 22/08/2015
;       (Initialized Disk Parameters Data is in 'DISKDATA.INC')
;       (Uninitialized Disk Parameters Data is in 'DISKBSS.INC')

; DISK I/O SYSTEM - Erdogan Tan (Retro UNIX 386 v1 project)

; ///////// DISK I/O SYSTEM ///////////////

; 06/02/2015
diskette_io:
        pushfd
        push    cs
        call    DISKETTE_IO_1
        retn

;;;;;; DISKETTE I/O ;;;;;;;;;;;;;;;;;;;; 06/02/2015 ;;;
;//////////////////////////////////////////////////////

; DISKETTE I/O - Erdogan Tan (Retro UNIX 386 v1 project)
; 20/02/2015
; 06/02/2015 (unix386.s)
; 16/12/2014 - 02/01/2015 (dsectrm2.s)
;
; Code (DELAY) modifications - AWARD BIOS 1999 (ADISK.EQU, COMMON.MAC)
;
; ADISK.EQU

;----- Wait control constants

;amount of time to wait while RESET is active.

WAITCPU_RESET_ON        EQU     21              ;Reset on must last at least 14us
                                                ;at 250 KBS xfer rate.
                                                ;see INTEL MCS, 1985, pg. 5-456

WAITCPU_FOR_STATUS      EQU     100             ;allow 30 microseconds for
                                                ;status register to become valid
                                                ;before re-reading.

;After sending a byte to NEC, status register may remain
;incorrectly set for 24 us.

WAITCPU_RQM_LOW                 EQU     24      ;number of loops to check for
                                                ;RQM low.

; COMMON.MAC
;
;       Timing macros
;

%macro          SIODELAY 0                      ; SHORT IODELAY
                jmp short $+2
%endmacro

%macro          IODELAY  0                      ; NORMAL IODELAY
                jmp short $+2
                jmp short $+2
%endmacro

%macro          NEWIODELAY 0
                out     0ebh,al
%endmacro

; (According to) AWARD BIOS 1999 - ATORGS.ASM (dw -> equ, db -> equ)
;;; WAIT_FOR_MEM
;WAIT_FDU_INT_LO        equ     017798          ; 2.5 secs in 30 micro units.
;WAIT_FDU_INT_HI        equ     1
WAIT_FDU_INT_LH                 equ     83334           ; 27/02/2015 (2.5 seconds waiting)
;;; WAIT_FOR_PORT
;WAIT_FDU_SEND_LO       equ     16667           ; .5 secons in 30 us units.
;WAIT_FDU_SEND_HI       equ     0
WAIT_FDU_SEND_LH        equ     16667           ; 27/02/2015
;Time to wait while waiting for each byte of NEC results = .5
;seconds.  .5 seconds = 500,000 micros.  500,000/30 = 16,667.
;WAIT_FDU_RESULTS_LO    equ     16667           ; .5 seconds in 30 micro units.
;WAIT_FDU_RESULTS_HI    equ     0
WAIT_FDU_RESULTS_LH     equ     16667   ; 27/02/2015
;;; WAIT_REFRESH
```

```
;amount of time to wait for head settle, per unit in parameter
;table = 1 ms.
WAIT_FDU_HEAD_SETTLE  equ     33             ; 1 ms in 30 micro units.


; //////////////// DISKETTE I/O ////////////////

; 11/12/2014 (copy from IBM PC-XT Model 286 BIOS - POSTEQU.INC)

;----------------------------------------
;       EQUATES USED BY POST AND BIOS :
;----------------------------------------

;--------- 8042 KEYBOARD INTERFACE AND DIAGNOSTIC CONTROL REGISTERS ------------
;PORT_A         EQU     060H            ; 8042 KEYBOARD SCAN CODE/CONTROL PORT
;PORT_B         EQU     061H            ; PORT B READ/WRITE DIAGNOSTIC REGISTER
;REFRESH_BIT    EQU     00010000B       ; REFRESH TEST BIT

;----------------------------------------
;       CMOS EQUATES FOR THIS SYSTEM  :
;-------------------------------------------------------------------------------
;CMOS_PORT      EQU     070H            ; I/O ADDRESS OF CMOS ADDRESS PORT
;CMOS_DATA      EQU     071H            ; I/O ADDRESS OF CMOS DATA PORT
;NMI            EQU     10000000B       ; DISABLE NMI INTERRUPTS MASK -
                                        ;  HIGH BIT OF CMOS LOCATION ADDRESS

;---------- CMOS TABLE LOCATION ADDRESS'S ## -----------------------------------
CMOS_DISKETTE  EQU     010H            ; DISKETTE DRIVE TYPE BYTE          ;
;              EQU     011H            ; - RESERVED                        ;C
CMOS_DISK      EQU     012H            ; FIXED DISK TYPE BYTE              ;H
;              EQU     013H            ; - RESERVED                        ;E
CMOS_EQUIP     EQU     014H            ; EQUIPMENT WORD LOW BYTE           ;C

;---------- DISKETTE EQUATES ---------------------------------------------------
INT_FLAG       EQU     10000000B       ; INTERRUPT OCCURRENCE FLAG
DSK_CHG        EQU     10000000B       ; DISKETTE CHANGE FLAG MASK BIT
DETERMINED     EQU     00010000B       ; SET STATE DETERMINED IN STATE BITS
HOME           EQU     00010000B       ; TRACK 0 MASK
SENSE_DRV_ST   EQU     00000100B       ; SENSE DRIVE STATUS COMMAND
TRK_SLAP       EQU     030H            ; CRASH STOP (48 TPI DRIVES)
QUIET_SEEK     EQU     00AH            ; SEEK TO TRACK 10
;MAX_DRV       EQU     2               ; MAX NUMBER OF DRIVES
HD12_SETTLE    EQU     15              ; 1.2 M HEAD SETTLE TIME
HD320_SETTLE   EQU     20              ; 320 K HEAD SETTLE TIME
MOTOR_WAIT     EQU     37              ; 2 SECONDS OF COUNTS FOR MOTOR TURN OFF

;---------- DISKETTE ERRORS ----------------------------------------------------
;TIME_OUT      EQU     080H            ; ATTACHMENT FAILED TO RESPOND
;BAD_SEEK      EQU     040H            ; SEEK OPERATION FAILED
BAD_NEC        EQU     020H            ; DISKETTE CONTROLLER HAS FAILED
BAD_CRC        EQU     010H            ; BAD CRC ON DISKETTE READ
MED_NOT_FND    EQU     00CH            ; MEDIA TYPE NOT FOUND
DMA_BOUNDARY   EQU     009H            ; ATTEMPT TO DMA ACROSS 64K BOUNDARY
BAD_DMA        EQU     008H            ; DMA OVERRUN ON OPERATION
MEDIA_CHANGE   EQU     006H            ; MEDIA REMOVED ON DUAL ATTACH CARD
RECORD_NOT_FND EQU     004H            ; REQUESTED SECTOR NOT FOUND
WRITE_PROTECT  EQU     003H            ; WRITE ATTEMPTED ON WRITE PROTECT DISK
BAD_ADDR_MARK  EQU     002H            ; ADDRESS MARK NOT FOUND
BAD_CMD        EQU     001H            ; BAD COMMAND PASSED TO DISKETTE I/O

;---------- DISK CHANGE LINE EQUATES -------------------------------------------
NOCHGLN        EQU     001H            ; NO DISK CHANGE LINE AVAILABLE
CHGLN          EQU     002H            ; DISK CHANGE LINE AVAILABLE

;---------- MEDIA/DRIVE STATE INDICATORS ---------------------------------------
TRK_CAPA       EQU     00000001B       ; 80 TRACK CAPABILITY
FMT_CAPA       EQU     00000010B       ; MULTIPLE FORMAT CAPABILITY (1.2M)
DRV_DET        EQU     00000100B       ; DRIVE DETERMINED
MED_DET        EQU     00010000B       ; MEDIA DETERMINED BIT
DBL_STEP       EQU     00100000B       ; DOUBLE STEP BIT
RATE_MSK       EQU     11000000B       ; MASK FOR CLEARING ALL BUT RATE
RATE_500       EQU     00000000B       ; 500 KBS DATA RATE
RATE_300       EQU     01000000B       ; 300 KBS DATA RATE
RATE_250       EQU     10000000B       ; 250 KBS DATA RATE
STRT_MSK       EQU     00001100B       ; OPERATION START RATE MASK
SEND_MSK       EQU     11000000B       ; MASK FOR SEND RATE BITS
```

```
;---------- MEDIA/DRIVE STATE INDICATORS COMPATIBILITY -------------------------
M3D3U           EQU     00000000B       ; 360 MEDIA/DRIVE NOT ESTABLISHED
M3D1U           EQU     00000001B       ; 360 MEDIA,1.2DRIVE NOT ESTABLISHED
M1D1U           EQU     00000010B       ; 1.2 MEDIA/DRIVE NOT ESTABLISHED
MED_UNK         EQU     00000111B       ; NONE OF THE ABOVE

;---------- INTERRUPT EQUATES --------------------------------------------------
;EOI            EQU     020H            ; END OF INTERRUPT COMMAND TO 8259
;INTA00         EQU     020H            ; 8259 PORT
INTA01          EQU     021H            ; 8259 PORT
INTB00          EQU     0A0H            ; 2ND 8259
INTB01          EQU     0A1H            ;

;-------------------------------------------------------------------------------
DMA08           EQU     008H            ; DMA STATUS REGISTER PORT ADDRESS
DMA             EQU     000H            ; DMA CH.0 ADDRESS REGISTER PORT ADDRESS
DMA18           EQU     0D0H            ; 2ND DMA STATUS PORT ADDRESS
DMA1            EQU     0C0H            ; 2ND DMA CH.0 ADDRESS REGISTER ADDRESS
;-------------------------------------------------------------------------------
;TIMER          EQU     040H            ; 8254 TIMER - BASE ADDRESS

;-------------------------------------------------------------------------------
DMA_PAGE        EQU     081H            ; START OF DMA PAGE REGISTERS

; 06/02/2015 (unix386.s, protected mode modifications)
; (unix386.s <-- dsectrm2.s)
; 11/12/2014 (copy from IBM PC-XT Model 286 BIOS - DSEG.INC)

; 10/12/2014
;
;int40h:
;       pushf
;       push    cs
;       ;cli
;       call    DISKETTE_IO_1
;       retn

; DSKETTE ----- 04/21/86 DISKETTE BIOS
; (IBM PC XT Model 286 System BIOS Source Code, 04-21-86)
;

;-- INT13H ---------------------------------------------------------------------
; DISKETTE I/O
;       THIS INTERFACE PROVIDES ACCESS TO THE 5 1/4 INCH 360 KB,
;       1.2 MB, 720 KB AND 1.44 MB DISKETTE DRIVES.
; INPUT
;       (AH) =  00H RESET DISKETTE SYSTEM
;               HARD RESET TO NEC, PREPARE COMMAND, RECALIBRATE REQUIRED
;               ON ALL DRIVES
;-------------------------------------------------------------------------------
;       (AH)= 01H  READ THE STATUS OF THE SYSTEM INTO (AH)
;               @DISKETTE_STATUS FROM LAST OPERATION IS USED
;-------------------------------------------------------------------------------
;       REGISTERS FOR READ/WRITE/VERIFY/FORMAT
;       (DL) - DRIVE NUMBER (0-1 ALLOWED, VALUE CHECKED)
;       (DH) - HEAD NUMBER (0-1 ALLOWED, NOT VALUE CHECKED)
;       (CH) - TRACK NUMBER (NOT VALUE CHECKED)
;               MEDIA   DRIVE   TRACK NUMBER
;               320/360 320/360     0-39
;               320/360 1.2M        0-39
;               1.2M    1.2M        0-79
;               720K    720K        0-79
;               1.44M   1.44M       0-79
;       (CL) -  SECTOR NUMBER (NOT VALUE CHECKED, NOT USED FOR FORMAT)
;               MEDIA   DRIVE   SECTOR NUMBER
;               320/360 320/360      1-8/9
;               320/360 1.2M         1-8/9
;               1.2M    1.2M         1-15
;               720K    720K         1-9
;               1.44M   1.44M        1-18
;       (AL)    NUMBER OF SECTORS (NOT VALUE CHECKED)
;               MEDIA   DRIVE   MAX NUMBER OF SECTORS
;               320/360 320/360         8/9
;               320/360 1.2M            8/9
;               1.2M    1.2M           15
;               720K    720K           9
;               1.44M   1.44M          18
;
```

```
;       (ES:BX) - ADDRESS OF BUFFER (NOT REQUIRED FOR VERIFY)
;
;--------------------------------------------------------------------------------
;       (AH)= 02H  READ THE DESIRED SECTORS INTO MEMORY
;--------------------------------------------------------------------------------
;       (AH)= 03H  WRITE THE DESIRED SECTORS FROM MEMORY
;--------------------------------------------------------------------------------
;       (AH)= 04H  VERIFY THE DESIRED SECTORS
;--------------------------------------------------------------------------------
;       (AH)= 05H  FORMAT THE DESIRED TRACK
;               (ES,BX) MUST POINT TO THE COLLECTION OF DESIRED ADDRESS FIELDS
;               FOR THE TRACK. EACH FIELD IS COMPOSED OF 4 BYTES, (C,H,R,N),
;               WHERE C = TRACK NUMBER, H=HEAD NUMBER, R = SECTOR NUMBER,
;               N= NUMBER OF BYTES PER SECTOR (00=128,01=256,02=512,03=1024),
;               THERE MUST BE ONE ENTRY FOR EVERY SECTOR ON THE TRACK.
;               THIS INFORMATION IS USED TO FIND THE REQUESTED SECTOR DURING
;               READ/WRITE ACCESS.
;               PRIOR TO FORMATTING A DISKETTE, IF THERE EXISTS MORE THAN
;               ONE SUPPORTED MEDIA FORMAT TYPE WITHIN THE DRIVE IN QUESTION,
;               THEN "SET DASD TYPE" (INT 13H, AH = 17H) OR 'SET MEDIA TYPE'
;               (INT 13H, AH =  18H) MUST BE CALLED TO SET THE DISKETTE TYPE
;               THAT IS TO BE FORMATTED. IF "SET DASD TYPE" OR "SET MEDIA TYPE"
;               IS NOT CALLED, THE FORMAT ROUTINE WILL ASSUME THE
;               MEDIA FORMAT TO BE THE MAXIMUM CAPACITY OF THE DRIVE.
;
;               THESE PARAMETERS OF DISK BASE MUST BE CHANGED IN ORDER TO
;               FORMAT THE FOLLOWING MEDIAS:
;               ---------------------------------------------
;               : MEDIA  :      DRIVE       : PARM 1 : PARM 2 :
;               ---------------------------------------------
;               : 320K   : 320K/360K/1.2M :  50H   :   8    :
;               : 360K   : 320K/360K/1.2M :  50H   :   9    :
;               : 1.2M   : 1.2M           :  54H   :  15    :
;               : 720K   : 720K/1.44M     :  50H   :   9    :
;               : 1.44M  : 1.44M          :  6CH   :  18    :
;               ---------------------------------------------
;               NOTES: - PARM 1 = GAP LENGTH FOR FORMAT
;                      - PARM 2 = EOT (LAST SECTOR ON TRACK)
;                      - DISK BASE IS POINTED BY DISK POINTER LOCATED
;                        AT ABSOLUTE ADDRESS 0:78.
;                      - WHEN FORMAT OPERATIONS ARE COMPLETE, THE PARAMETERS
;                        SHOULD BE RESTORED TO THEIR RESPECTIVE INITIAL VALUES.

;--------------------------------------------------------------------------------
;       (AH) = 08H READ DRIVE PARAMETERS
;       REGISTERS
;         INPUT
;           (DL) - DRIVE NUMBER (0-1 ALLOWED, VALUE CHECKED)
;         OUTPUT
;           (ES:DI) POINTS TO DRIVE PARAMETER TABLE
;           (CH) - LOW ORDER 8 OF 10 BITS MAXIMUM NUMBER OF TRACKS
;           (CL) - BITS 7 & 6 - HIGH ORDER TWO BITS OF MAXIMUM TRACKS
;                  BITS 5 THRU 0 - MAXIMUM SECTORS PER TRACK
;           (DH) - MAXIMUM HEAD NUMBER
;           (DL) - NUMBER OF DISKETTE DRIVES INSTALLED
;           (BH) - 0
;           (BL) - BITS 7 THRU 4 - 0
;                  BITS 3 THRU 0 - VALID DRIVE TYPE VALUE IN CMOS
;           (AX) - 0
;        UNDER THE FOLLOWING CIRCUMSTANCES:
;           (1) THE DRIVE NUMBER IS INVALID,
;           (2) THE DRIVE TYPE IS UNKNOWN AND CMOS IS NOT PRESENT,
;           (3) THE DRIVE TYPE IS UNKNOWN AND CMOS IS BAD,
;           (4) OR THE DRIVE TYPE IS UNKNOWN AND THE CMOS DRIVE TYPE IS INVALID
;           THEN ES,AX,BX,CX,DH,DI=0 ; DL=NUMBER OF DRIVES.
;           IF NO DRIVES ARE PRESENT THEN: ES,AX,BX,CX,DX,DI=0.
;           @DISKETTE_STATUS = 0 AND CY IS RESET.
;--------------------------------------------------------------------------------
;       (AH)= 15H  READ DASD TYPE
;       OUTPUT REGISTERS
;       (AH) - ON RETURN IF CARRY FLAG NOT SET, OTHERWISE ERROR
;               00 - DRIVE NOT PRESENT
;               01 - DISKETTE, NO CHANGE LINE AVAILABLE
;               02 - DISKETTE, CHANGE LINE AVAILABLE
;               03 - RESERVED (FIXED DISK)
;       (DL) - DRIVE NUMBER (0-1 ALLOWED, VALUE CHECKED)
```

```
;-------------------------------------------------------------------------------
;       (AH)= 16H  DISK CHANGE LINE STATUS
;       OUTPUT REGISTERS
;       (AH) - 00 - DISK CHANGE LINE NOT ACTIVE
;              06 - DISK CHANGE LINE ACTIVE & CARRY BIT ON
;       (DL) - DRIVE NUMBER (0-1 ALLOWED, VALUE CHECKED)
;-------------------------------------------------------------------------------
;       (AH)= 17H  SET DASD TYPE FOR FORMAT
;       INPUT REGISTERS
;       (AL) -  00 - NOT USED
;               01 - DISKETTE 320/360K IN 360K DRIVE
;               02 - DISKETTE 360K IN 1.2M DRIVE
;               03 - DISKETTE 1.2M IN 1.2M DRIVE
;               04 - DISKETTE 720K IN 720K DRIVE
;       (DL) - DRIVE NUMBER (0-1 ALLOWED, VALUE CHECKED:
;              (DO NOT USE WHEN DISKETTE ATTACH CARD USED)
;-------------------------------------------------------------------------------
;       (AH)= 18H  SET MEDIA TYPE FOR FORMAT
;       INPUT REGISTERS
;       (CH) - LOW ORDER 8 OF 10 BITS MAXIMUM TRACKS
;       (CL) - BITS 7 & 6 - HIGH ORDER TWO BITS OF MAXIMUM TRACKS
;              BITS 5 THRU 0 - MAXIMUM SECTORS PER TRACK
;       (DL) - DRIVE NUMBER (0-1 ALLOWED, VALUE CHACKED)
;       OUTPUT REGISTERS:
;       (ES:DI) - POINTER TO DRIVE PARAMETERS TABLE FOR THIS MEDIA TYPE,
;                 UNCHANGED IF (AH) IS NON-ZERO
;       (AH) - 00H, CY = 0, TRACK AND SECTORS/TRACK COMBINATION IS SUPPORTED
;            - 01H, CY = 1, FUNCTION IS NOT AVAILABLE
;            - 0CH, CY = 1, TRACK AND SECTORS/TRACK COMBINATION IS NOT SUPPORTED
;            - 80H, CY = 1, TIME OUT (DISKETTE NOT PRESENT)
;-------------------------------------------------------------------------------
;       DISK CHANGE STATUS IS ONLY CHECKED WHEN A MEDIA SPECIFIED IS OTHER
;       THAN 360 KB DRIVE. IF THE DISK CHANGE LINE IS FOUND TO BE
;       ACTIVE THE FOLLOWING ACTIONS TAKE PLACE:
;               ATTEMPT TO RESET DISK CHANGE LINE TO INACTIVE STATE.
;               IF ATTEMPT SUCCEEDS SET DASD TYPE FOR FORMAT AND RETURN DISK
;               CHANGE ERROR CODE
;               IF ATTEMPT FAILS RETURN TIMEOUT ERROR CODE AND SET DASD TYPE
;               TO A PREDETERMINED STATE INDICATING MEDIA TYPE UNKNOWN.
;       IF THE DISK CHANGE LINE IN INACTIVE PERFORM SET DASD TYPE FOR FORMAT.
;
; DATA VARIABLE -- @DISK_POINTER
;       DOUBLE WORD POINTER TO THE CURRENT SET OF DISKETTE PARAMETERS
;-------------------------------------------------------------------------------
; OUTPUT FOR ALL FUNCTIONS
;       AH = STATUS OF OPERATION
;               STATUS BITS ARE DEFINED IN THE EQUATES FOR @DISKETTE_STATUS
;               VARIABLE IN THE DATA SEGMENT OF THIS MODULE
;       CY = 0  SUCCESSFUL OPERATION (AH=0 ON RETURN, EXCEPT FOR READ DASD
;               TYPE AH=(15)).
;       CY = 1  FAILED OPERATION (AH HAS ERROR REASON)
;       FOR READ/WRITE/VERIFY
;               DS,BX,DX,CX PRESERVED
;       NOTE: IF AN ERROR IS REPORTED BY THE DISKETTE CODE, THE APPROPRIATE
;               ACTION IS TO RESET THE DISKETTE, THEN RETRY THE OPERATION.
;               ON READ ACCESSES, NO MOTOR START DELAY IS TAKEN, SO THAT
;               THREE RETRIES ARE REQUIRED ON READS TO ENSURE THAT THE
;               PROBLEM IS NOT DUE TO MOTOR START-UP.
;-------------------------------------------------------------------------------
;
```

```
; DISKETTE STATE MACHINE - ABSOLUTE ADDRESS 40:90 (DRIVE A) & 91 (DRIVE B)
;
;    -----------------------------------------------------------------
;    |       |       |       |       |       |       |       |       |
;    |   7   |   6   |   5   |   4   |   3   |   2   |   1   |   0   |
;    |       |       |       |       |       |       |       |       |
;    -----------------------------------------------------------------
;        |       |       |       |       |       |       |       |
;        |       |       |       |       |       -----------------
;        |       |       |       |       |               |
;        |       |       |       |    RESERVED           |
;        |       |       |       |                 PRESENT STATE
;        |       |       |       |       000: 360K IN 360K DRIVE UNESTABLISHED
;        |       |       |       |       001: 360K IN 1.2M DRIVE UNESTABLISHED
;        |       |       |       |       010: 1.2M IN 1.2M DRIVE UNESTABLISHED
;        |       |       |       |       011: 360K IN 360K DRIVE ESTABLISHED
;        |       |       |       |       100: 360K IN 1.2M DRIVE ESTABLISHED
;        |       |       |       |       101: 1.2M IN 1.2M DRIVE ESTABLISHED
;        |       |       |       |       110: RESERVED
;        |       |       |       |       111: NONE OF THE ABOVE
;        |       |       |       |
;        |       |       |       ------> MEDIA/DRIVE ESTABLISHED
;        |       |       |
;        |       |       --------------->       DOUBLE STEPPING REQUIRED (360K IN 1.2M
;        |       |                       DRIVE)
;        |       |
;        ------------------------------->       DATA TRANSFER RATE FOR THIS DRIVE:
;
;                                               00: 500 KBS
;                                               01: 300 KBS
;                                               10: 250 KBS
;                                               11: RESERVED
;
;
;-------------------------------------------------------------------------------
; STATE OPERATION STARTED - ABSOLUTE ADDRESS 40:92 (DRIVE A) & 93 (DRIVE B)
;-------------------------------------------------------------------------------
; PRESENT CYLINDER NUMBER - ABSOLUTE ADDRESS 40:94 (DRIVE A) & 95 (DRIVE B)
;-------------------------------------------------------------------------------

struc MD
        .SPEC1          resb    1       ; SRT=D, HD UNLOAD=0F - 1ST SPECIFY BYTE
        .SPEC2          resb    1       ; HD LOAD=1, MODE=DMA - 2ND SPECIFY BYTE
        .OFF_TIM        resb    1       ; WAIT TIME AFTER OPERATION TILL MOTOR OFF
        .BYT_SEC        resb    1       ; 512 BYTES/SECTOR
        .SEC_TRK        resb    1       ; EOT (LAST SECTOR ON TRACK)
        .GAP            resb    1       ; GAP LENGTH
        .DTL            resb    1       ; DTL
        .GAP3           resb    1       ; GAP LENGTH FOR FORMAT
        .FIL_BYT        resb    1       ; FILL BYTE FOR FORMAT
        .HD_TIM         resb    1       ; HEAD SETTLE TIME (MILLISECONDS)
        .STR_TIM        resb    1       ; MOTOR START TIME (1/8 SECONDS)
        .MAX_TRK        resb    1       ; MAX. TRACK NUMBER
        .RATE           resb    1       ; DATA TRANSFER RATE
endstruc

BIT7OFF EQU     7FH
BIT7ON  EQU     80H

;;int13h: ; 16/02/2015
;; 16/02/2015 - 21/02/2015
int40h:
        pushfd
        push    cs
        call    DISKETTE_IO_1
        retn
```

```
DISKETTE_IO_1:

        STI                             ; INTERRUPTS BACK ON
        PUSH    eBP                     ; USER REGISTER
        PUSH    eDI                     ; USER REGISTER
        PUSH    eDX                     ; HEAD #, DRIVE # OR USER REGISTER
        PUSH    eBX                     ; BUFFER OFFSET PARAMETER OR REGISTER
        PUSH    eCX                     ; TRACK #-SECTOR # OR USER REGISTER
        MOV     eBP,eSP                 ; BP     => PARAMETER LIST DEP. ON AH
                                        ; [BP]   = SECTOR #
                                        ; [BP+1] = TRACK #
                                        ; [BP+2] = BUFFER OFFSET
                                        ; FOR RETURN OF DRIVE PARAMETERS:
                                        ; CL/[BP] = BITS 7&6 HI BITS OF MAX CYL
                                        ;          BITS 0-5 MAX SECTORS/TRACK
                                        ; CH/[BP+1] = LOW 8 BITS OF MAX CYL.
                                        ; BL/[BP+2] = BITS 7-4 = 0
                                        ;             BITS 3-0 = VALID CMOS TYPE
                                        ; BH/[BP+3] = 0
                                        ; DL/[BP+4] = # DRIVES INSTALLED
                                        ; DH/[BP+5] = MAX HEAD #
                                        ; DI/[BP+6] = OFFSET TO DISK BASE
        push    es ; 06/02/2015
        PUSH    DS                      ; BUFFER SEGMENT PARM OR USER REGISTER
        PUSH    eSI                     ; USER REGISTERS
        ;CALL   DDS                     ; SEGMENT OF BIOS DATA AREA TO DS
        ;mov    cx, cs
        ;mov    ds, cx
        mov     cx, KDATA
         mov     ds, cx
         mov     es, cx

        ;CMP    AH,(FNC_TAE-FNC_TAB)/2; CHECK FOR > LARGEST FUNCTION
        cmp     ah,(FNC_TAE-FNC_TAB)/4 ; 18/02/2015
        JB      short OK_FUNC           ; FUNCTION OK
        MOV     AH,14H                  ; REPLACE WITH KNOWN INVALID FUNCTION
OK_FUNC:
        CMP     AH,1                    ; RESET OR STATUS ?
        JBE     short OK_DRV            ; IF RESET OR STATUS DRIVE ALWAYS OK
        CMP     AH,8                    ; READ DRIVE PARMS ?
        JZ      short OK_DRV            ; IF SO DRIVE CHECKED LATER
        CMP     DL,1                    ; DRIVES 0 AND 1 OK
        JBE     short OK_DRV            ; IF 0 OR 1 THEN JUMP
        MOV     AH,14H                  ; REPLACE WITH KNOWN INVALID FUNCTION
OK_DRV:
        xor     ecx, ecx
        ;mov    esi, ecx ; 08/02/2015
        mov     edi, ecx ; 08/02/2015
        MOV     CL,AH                   ; CL = FUNCTION
        ;XOR    CH,CH                   ; CX = FUNCTION
        ;SHL    CL, 1                   ; FUNCTION TIMES 2
        SHL     CL, 2 ; 20/02/2015      ; FUNCTION TIMES 4 (for 32 bit offset)
        MOV     eBX,FNC_TAB             ; LOAD START OF FUNCTION TABLE
        ADD     eBX,eCX                 ; ADD OFFSET INTO TABLE => ROUTINE
        MOV     AH,DH                   ; AX = HEAD #,# OF SECTORS OR DASD TYPE
        XOR     DH,DH                   ; DX = DRIVE #
        MOV     SI,AX                   ; SI = HEAD #,# OF SECTORS OR DASD TYPE
        MOV      DI,DX                      ; DI = DRIVE #
        ;
        ; 11/12/2014
         mov     [cfd], dl                  ; current floppy drive (for 'GET_PARM')
        ;
        MOV     AH, [DSKETTE_STATUS]   ; LOAD STATUS TO AH FOR STATUS FUNCTION
        MOV     byte [DSKETTE_STATUS],0       ; INITIALIZE FOR ALL OTHERS

;       THROUGHOUT THE DISKETTE BIOS, THE FOLLOWING INFORMATION IS CONTAINED IN
;       THE FOLLOWING MEMORY LOCATIONS AND REGISTERS. NOT ALL DISKETTE BIOS
;       FUNCTIONS REQUIRE ALL OF THESE PARAMETERS.
;
;               DI      : DRIVE #
;               SI-HI   : HEAD #
;               SI-LOW  : # OF SECTORS OR DASD TYPE FOR FORMAT
;               ES      : BUFFER SEGMENT
;               [BP]    : SECTOR #
;               [BP+1]  : TRACK #
;               [BP+2]  : BUFFER OFFSET
;
```

```
;       ACROSS CALLS TO SUBROUTINES THE CARRY FLAG (CY=1), WHERE INDICATED IN
;       SUBROUTINE PROLOGUES, REPRESENTS AN EXCEPTION RETURN (NORMALLY AN ERROR
;       CONDITION). IN MOST CASES, WHEN CY = 1, @DSKETTE_STATUS CONTAINS THE
;       SPECIFIC ERROR CODE.
;
                                    ; (AH) = @DSKETTE_STATUS
        CALL    dWORD [eBX]         ; CALL THE REQUESTED FUNCTION
        POP     eSI                 ; RESTORE ALL REGISTERS
        POP     DS
        pop     es      ; 06/02/2015
        POP     eCX
        POP     eBX
        POP     eDX
        POP     eDI
        MOV     eBP, eSP
        PUSH    eAX
        PUSHFd
        POP     eAX
        ;MOV    [BP+6], AX
        mov     [ebp+12], eax   ; 18/02/2015, flags
        POP     eAX
        POP     eBP
        IRETd

;-------------------------------------------------------------------------------
; DW --> dd (06/02/2015)
FNC_TAB dd      DSK_RESET           ; AH = 00H; RESET
        dd      DSK_STATUS          ; AH = 01H; STATUS
        dd      DSK_READ            ; AH = 02H; READ
        dd      DSK_WRITE           ; AH = 03H; WRITE
        dd      DSK_VERF            ; AH = 04H; VERIFY
        dd      DSK_FORMAT          ; AH = 05H; FORMAT
        dd      FNC_ERR             ; AH = 06H; INVALID
        dd      FNC_ERR             ; AH = 07H; INVALID
        dd      DSK_PARMS           ; AH = 08H; READ DRIVE PARAMETERS
        dd      FNC_ERR             ; AH = 09H; INVALID
        dd      FNC_ERR             ; AH = 0AH; INVALID
        dd      FNC_ERR             ; AH = 0BH; INVALID
        dd      FNC_ERR             ; AH = 0CH; INVALID
        dd      FNC_ERR             ; AH = 0DH; INVALID
        dd      FNC_ERR             ; AH = 0EH; INVALID
        dd      FNC_ERR             ; AH = 0FH; INVALID
        dd      FNC_ERR             ; AH = 10H; INVALID
        dd      FNC_ERR             ; AH = 11H; INVALID
        dd      FNC_ERR             ; AH = 12H; INVALID
        dd      FNC_ERR             ; AH = 13H; INVALID
        dd      FNC_ERR             ; AH = 14H; INVALID
        dd      DSK_TYPE            ; AH = 15H; READ DASD TYPE
        dd      DSK_CHANGE          ; AH = 16H; CHANGE STATUS
        dd      FORMAT_SET          ; AH = 17H; SET DASD TYPE
        dd      SET_MEDIA           ; AH = 18H; SET MEDIA TYPE
FNC_TAE EQU      $                     ; END

;-------------------------------------------------------------------------------
; DISK_RESET    (AH = 00H)
;               RESET THE DISKETTE SYSTEM.
;
; ON EXIT:      @DSKETTE_STATUS, CY REFLECT STATUS OF OPERATION
;-------------------------------------------------------------------------------
DSK_RESET:
        MOV     DX,03F2H            ; ADAPTER CONTROL PORT
        CLI                         ; NO INTERRUPTS
        MOV     AL,[MOTOR_STATUS]   ; GET DIGITAL OUTPUT REGISTER REFLECTION
        AND     AL,00111111B        ; KEEP SELECTED AND MOTOR ON BITS
        ROL     AL,4                ; MOTOR VALUE TO HIGH NIBBLE
                                    ; DRIVE SELECT TO LOW NIBBLE
        OR      AL,00001000B        ; TURN ON INTERRUPT ENABLE
        OUT     DX,AL               ; RESET THE ADAPTER
        MOV     byte [SEEK_STATUS],0 ; SET RECALIBRATE REQUIRED ON ALL DRIVES
        ;JMP    $+2                 ; WAIT FOR I/O
        ;JMP    $+2                 ; WAIT FOR I/O (TO INSURE MINIMUM
                                    ;      PULSE WIDTH)
        ; 19/12/2014
        NEWIODELAY
```

```
        ; 17/12/2014
        ; AWARD BIOS 1999 - RESETDRIVES (ADISK.ASM)
        mov     ecx, WAITCPU_RESET_ON ; cx = 21 -- Min. 14 micro seconds !?
wdw1:
        NEWIODELAY   ; 27/02/2015
        loop    wdw1
        ;
        OR      AL,00000100B           ; TURN OFF RESET BIT
        OUT     DX,AL                  ; RESET THE ADAPTER
        ; 16/12/2014
        IODELAY
        ;
        ;STI                           ; ENABLE THE INTERRUPTS
        CALL    WAIT_INT               ; WAIT FOR THE INTERRUPT
        JC      short DR_ERR           ; IF ERROR, RETURN IT
        MOV     CX,11000000B           ; CL = EXPECTED @NEC_STATUS
NXT_DRV:
        PUSH    CX                     ; SAVE FOR CALL
        MOV     eAX, DR_POP_ERR        ; LOAD NEC_OUTPUT ERROR ADDRESS
        PUSH    eAX                    ; "
        MOV     AH,08H                 ; SENSE INTERRUPT STATUS COMMAND
        CALL    NEC_OUTPUT
        POP     eAX                    ; THROW AWAY ERROR RETURN
        CALL    RESULTS                ; READ IN THE RESULTS
        POP     CX                     ; RESTORE AFTER CALL
        JC      short DR_ERR           ; ERROR RETURN
        CMP     CL, [NEC_STATUS]       ; TEST FOR DRIVE READY TRANSITION
        JNZ     short DR_ERR           ; EVERYTHING OK
        INC     CL                     ; NEXT EXPECTED @NEC_STATUS
        CMP     CL,11000011B           ; ALL POSSIBLE DRIVES CLEARED
        JBE     short NXT_DRV          ; FALL THRU IF 11000100B OR >
        ;
        CALL    SEND_SPEC              ; SEND SPECIFY COMMAND TO NEC
RESBAC:
        CALL    SETUP_END              ; VARIOUS CLEANUPS
        MOV     BX,SI                  ; GET SAVED AL TO BL
        MOV     AL,BL                  ; PUT BACK FOR RETURN
        RETn
DR_POP_ERR:
        POP     CX                     ; CLEAR STACK
DR_ERR:
        OR      byte [DSKETTE_STATUS],BAD_NEC ; SET ERROR CODE
        JMP     SHORT RESBAC           ; RETURN FROM RESET

;-------------------------------------------------------------------------------
; DISK_STATUS  (AH = 01H)
;       DISKETTE STATUS.
;
; ON ENTRY:     AH : STATUS OF PREVIOUS OPERATION
;
; ON EXIT:      AH, @DSKETTE_STATUS, CY REFLECT STATUS OF PREVIOUS OPERATION.
;-------------------------------------------------------------------------------
DSK_STATUS:
        MOV     [DSKETTE_STATUS],AH    ; PUT BACK FOR SETUP END
        CALL    SETUP_END              ; VARIOUS CLEANUPS
        MOV     BX,SI                  ; GET SAVED AL TO BL
        MOV     AL,BL                  ; PUT BACK FOR RETURN
        RETn
```

```
;-------------------------------------------------------------------------------
; DISK_READ    (AH = 02H)
;       DISKETTE READ.
;
; ON ENTRY:    DI      : DRIVE #
;              SI-HI   : HEAD #
;              SI-LOW  : # OF SECTORS
;              ES      : BUFFER SEGMENT
;              [BP]    : SECTOR #
;              [BP+1]  : TRACK #
;              [BP+2]  : BUFFER OFFSET
;
; ON EXIT:     @DSKETTE_STATUS, CY REFLECT STATUS OF OPERATION
;-------------------------------------------------------------------------------

; 06/02/2015, ES:BX -> EBX (unix386.s)

DSK_READ:
       AND     byte [MOTOR_STATUS],01111111B ; INDICATE A READ OPERATION
       MOV     AX,0E646H               ; AX = NEC COMMAND, DMA COMMAND
       CALL    RD_WR_VF                ; COMMON READ/WRITE/VERIFY
       RETn

;-------------------------------------------------------------------------------
; DISK_WRITE   (AH = 03H)
;       DISKETTE WRITE.
;
; ON ENTRY:    DI      : DRIVE #
;              SI-HI   : HEAD #
;              SI-LOW  : # OF SECTORS
;              ES      : BUFFER SEGMENT
;              [BP]    : SECTOR #
;              [BP+1]  : TRACK #
;              [BP+2]  : BUFFER OFFSET
;
; ON EXIT:     @DSKETTE_STATUS, CY REFLECT STATUS OF OPERATION
;-------------------------------------------------------------------------------

; 06/02/2015, ES:BX -> EBX (unix386.s)

DSK_WRITE:
       MOV     AX,0C54AH               ; AX = NEC COMMAND, DMA COMMAND
        OR      byte [MOTOR_STATUS],10000000B ; INDICATE WRITE OPERATION
       CALL    RD_WR_VF                ; COMMON READ/WRITE/VERIFY
       RETn

;-------------------------------------------------------------------------------
; DISK_VERF    (AH = 04H)
;       DISKETTE VERIFY.
;
; ON ENTRY:    DI      : DRIVE #
;              SI-HI   : HEAD #
;              SI-LOW  : # OF SECTORS
;              ES      : BUFFER SEGMENT
;              [BP]    : SECTOR #
;              [BP+1]  : TRACK #
;              [BP+2]  : BUFFER OFFSET
;
; ON EXIT:     @DSKETTE_STATUS, CY REFLECT STATUS OF OPERATION
;-------------------------------------------------------------------------------
DSK_VERF:
       AND     byte [MOTOR_STATUS],01111111B ; INDICATE A READ OPERATION
       MOV     AX,0E642H               ; AX = NEC COMMAND, DMA COMMAND
       CALL    RD_WR_VF                ; COMMON READ/WRITE/VERIFY
       RETn
```

```
;-------------------------------------------------------------------------------
; DISK_FORMAT  (AH = 05H)
;       DISKETTE FORMAT.
;
; ON ENTRY:     DI      : DRIVE #
;               SI-HI   : HEAD #
;               SI-LOW  : # OF SECTORS
;               ES      : BUFFER SEGMENT
;               [BP]    : SECTOR #
;               [BP+1]  : TRACK #
;               [BP+2]  : BUFFER OFFSET
;               @DISK_POINTER POINTS TO THE PARAMETER TABLE OF THIS DRIVE
;
; ON EXIT:      @DSKETTE_STATUS, CY REFLECT STATUS OF OPERATION
;-------------------------------------------------------------------------------
DSK_FORMAT:
        CALL    XLAT_NEW                ; TRANSLATE STATE TO PRESENT ARCH.
        CALL    FMT_INIT                ; ESTABLISH STATE IF UNESTABLISHED
        OR       byte [MOTOR_STATUS], 10000000B ; INDICATE WRITE OPERATION
        CALL    MED_CHANGE              ; CHECK MEDIA CHANGE AND RESET IF SO
        JC       short FM_DON              ; MEDIA CHANGED, SKIP
        CALL    SEND_SPEC               ; SEND SPECIFY COMMAND TO NEC
        CALL    CHK_LASTRATE            ; ZF=1 ATTEMPT RATE IS SAME AS LAST RATE
        JZ       short FM_WR               ; YES, SKIP SPECIFY COMMAND
        CALL    SEND_RATE               ; SEND DATA RATE TO CONTROLLER
FM_WR:
        CALL    FMTDMA_SET              ; SET UP THE DMA FOR FORMAT
        JC       short FM_DON              ; RETURN WITH ERROR
        MOV     AH,04DH                 ; ESTABLISH THE FORMAT COMMAND
        CALL    NEC_INIT                ; INITIALIZE THE NEC
        JC       short FM_DON              ; ERROR - EXIT
        MOV      eAX, FM_DON               ; LOAD ERROR ADDRESS
        PUSH    eAX                     ; PUSH NEC_OUT ERROR RETURN
        MOV     DL,3                    ; BYTES/SECTOR VALUE TO NEC
        CALL    GET_PARM
        CALL    NEC_OUTPUT
        MOV     DL,4                    ; SECTORS/TRACK VALUE TO NEC
        CALL    GET_PARM
        CALL    NEC_OUTPUT
        MOV     DL,7                    ; GAP LENGTH VALUE TO NEC
        CALL    GET_PARM
        CALL    NEC_OUTPUT
        MOV     DL,8                    ; FILLER BYTE TO NEC
        CALL    GET_PARM
        CALL    NEC_OUTPUT
        POP     eAX                     ; THROW AWAY ERROR
        CALL    NEC_TERM                ; TERMINATE, RECEIVE STATUS, ETC,
FM_DON:
        CALL    XLAT_OLD                ; TRANSLATE STATE TO COMPATIBLE MODE
        CALL    SETUP_END               ; VARIOUS CLEANUPS
        MOV     BX,SI                   ; GET SAVED AL TO BL
        MOV     AL,BL                   ; PUT BACK FOR RETURN
        RETn

;-------------------------------------------------------------------------------
; FNC_ERR
;       INVALID FUNCTION REQUESTED OR INVALID DRIVE:
;       SET BAD COMMAND IN STATUS.
;
; ON EXIT:      @DSKETTE_STATUS, CY REFLECT STATUS OF OPERATION
;-------------------------------------------------------------------------------
FNC_ERR:                                ; INVALID FUNCTION REQUEST
        MOV     AX,SI                   ; RESTORE AL
        MOV     AH,BAD_CMD              ; SET BAD COMMAND ERROR
        MOV     [DSKETTE_STATUS],AH     ; STORE IN DATA AREA
        STC                             ; SET CARRY INDICATING ERROR
        RETn
```

```
;-------------------------------------------------------------------------------
; DISK_PARMS   (AH = 08H)
;       READ DRIVE PARAMETERS.
;
; ON ENTRY:    DI : DRIVE #
;
; ON EXIT:     CL/[BP]   = BITS 7 & 6 HI 2 BITS OF MAX CYLINDER
;                          BITS 0-5 MAX SECTORS/TRACK
;              CH/[BP+1] = LOW 8 BITS OF MAX CYLINDER
;              BL/[BP+2] = BITS 7-4 = 0
;                          BITS 3-0 = VALID CMOS DRIVE TYPE
;              BH/[BP+3] = 0
;              DL/[BP+4] = # DRIVES INSTALLED (VALUE CHECKED)
;              DH/[BP+5] = MAX HEAD #
;              DI/[BP+6] = OFFSET TO DISK_BASE
;              ES        = SEGMENT OF DISK_BASE
;              AX        = 0
;
;              NOTE : THE ABOVE INFORMATION IS STORED IN THE USERS STACK AT
;                     THE LOCATIONS WHERE THE MAIN ROUTINE WILL POP THEM
;                     INTO THE APPROPRIATE REGISTERS BEFORE RETURNING TO THE
;                     CALLER.
;-------------------------------------------------------------------------------
DSK_PARMS:
        CALL    XLAT_NEW                ; TRANSLATE STATE TO PRESENT ARCH,
     ;  MOV     WORD [BP+2],0           ; DRIVE TYPE = 0
        sub      edx, edx ; 20/02/2015
         mov    [ebp+4], edx ; 20/02/2015
     ;   MOV     AX, [EQUIP_FLAG]           ; LOAD EQUIPMENT FLAG FOR # DISKETTES
     ;   AND     AL,11000001B               ; KEEP DISKETTE DRIVE BITS
     ;   MOV     DL,2                       ; DISKETTE DRIVES = 2
     ;   CMP     AL,01000001B               ; 2 DRIVES INSTALLED ?
     ;   JZ      short STO_DL               ; IF YES JUMP
     ;   DEC     DL                         ; DISKETTE DRIVES = 1
     ;   CMP     AL,00000001B               ; 1 DRIVE INSTALLED ?
     ;   JNZ     short NON_DRV              ; IF NO JUMP
        ;sub    edx, edx
        mov      ax, [fd0_type]
        and      ax, ax
        jz       short NON_DRV
        inc      dl
        and      ah, ah
        jz       short STO_DL
        inc      dl
STO_DL:
        ;MOV    [BP+4],DL               ; STORE NUMBER OF DRIVES
        mov     [ebp+8], edx ; 20/02/2015
        CMP     DI,1                    ; CHECK FOR VALID DRIVE
        JA      short NON_DRV1          ; DRIVE INVALID
        ;MOV    BYTE [BP+5],1           ; MAXIMUM HEAD NUMBER =       1
        mov     byte [ebp+9], 1  ; 20/02/2015
        CALL    CMOS_TYPE               ; RETURN DRIVE TYPE IN AL
        ;;20/02/2015
        ;;JC    short CHK_EST           ; IF CMOS BAD CHECKSUM ESTABLISHED
        ;;OR    AL,AL                   ; TEST FOR NO DRIVE TYPE
        JZ      short CHK_EST           ; JUMP IF SO
        CALL    DR_TYPE_CHECK           ; RTN CS:BX = MEDIA/DRIVE PARAM TBL
        JC      short CHK_EST           ; TYPE NOT IN TABLE (POSSIBLE BAD CMOS)
        ;MOV    [BP+2],AL               ; STORE VALID CMOS DRIVE TYPE
         mov    [ebp+4], al ; 06/02/2015
        MOV      CL, [eBX+MD.SEC_TRK]      ; GET SECTOR/TRACK
         MOV     CH, [eBX+MD.MAX_TRK]       ; GET MAX. TRACK NUMBER
        JMP     SHORT STO_CX            ; CMOS GOOD, USE CMOS
CHK_EST:
        MOV     AH, [DSK_STATE+eDI]     ; LOAD STATE FOR THIS DRIVE
        TEST    AH,MED_DET              ; CHECK FOR ESTABLISHED STATE
        JZ      short NON_DRV1          ; CMOS BAD/INVALID OR UNESTABLISHED
USE_EST:
        AND     AH,RATE_MSK             ; ISOLATE STATE
        CMP     AH,RATE_250             ; RATE 250 ?
        JNE     short USE_EST2          ; NO, GO CHECK OTHER RATE

;----- DATA RATE IS 250 KBS, TRY 360 KB TABLE FIRST

        MOV     AL,01                   ; DRIVE TYPE 1 (360KB)
        CALL    DR_TYPE_CHECK           ; RTN CS:BX = MEDIA/DRIVE PARAM TBL
         MOV     CL, [eBX+MD.SEC_TRK]     ; GET SECTOR/TRACK
         MOV     CH, [eBX+MD.MAX_TRK]     ; GET MAX. TRACK NUMBER
        TEST    byte [DSK_STATE+eDI],TRK_CAPA ; 80 TRACK ?
```

```
        JZ      short STO_CX            ; MUST BE 360KB DRIVE

;-----  IT IS 1.44 MB DRIVE

PARM144:
        MOV     AL,04                   ; DRIVE TYPE 4 (1.44MB)
        CALL    DR_TYPE_CHECK           ; RTN CS:BX = MEDIA/DRIVE PARAM TBL
         MOV     CL, [eBX+MD.SEC_TRK]    ; GET SECTOR/TRACK
         MOV     CH, [eBX+MD.MAX_TRK]    ; GET MAX. TRACK NUMBER
STO_CX:
        MOV     [eBP],eCX               ; SAVE POINTER IN STACK FOR RETURN
ES_DI:
        ;MOV    [BP+6],BX               ; ADDRESS OF MEDIA/DRIVE PARM TABLE
        mov     [ebp+12], ebx ; 06/02/2015
        ;MOV    AX,CS                   ; SEGMENT MEDIA/DRIVE PARAMETER TABLE
        ;MOV    ES,AX                   ; ES IS SEGMENT OF TABLE
DP_OUT:
        CALL    XLAT_OLD                ; TRANSLATE STATE TO COMPATIBLE MODE
        XOR     AX,AX                   ; CLEAR
        CLC
        RETn

;-----  NO DRIYE PRESENT HANDLER

NON_DRV:
        ;MOV    BYTE [BP+4],0           ; CLEAR NUMBER OF DRIVES
        mov     [ebp+8], edx ; 0 ; 20/02/2015
NON_DRV1:
        CMP     DI,80H                  ; CHECK FOR FIXED MEDIA TYPE REQUEST
        JB      short NON_DRV2          ; CONTINUE IF NOT REQUEST FALL THROUGH

;-----  FIXED DISK REQUEST FALL THROUGH ERROR

        CALL    XLAT_OLD                ; ELSE TRANSLATE TO COMPATIBLE MODE
        MOV     AX,SI                   ; RESTORE AL
        MOV     AH,BAD_CMD              ; SET BAD COMMAND ERROR
        STC
        RETn

NON_DRV2:
        ;XOR    AX,AX                   ; CLEAR PARMS IF NO DRIVES OR CMOS BAD
        xor     eax, eax
        MOV     [eBP],AX                ; TRACKS, SECTORS/TRACK = 0
        ;MOV    [BP+5],AH               ; HEAD = 0
        mov     [ebp+9], ah ; 06/02/2015
        ;MOV    [BP+6],AX               ; OFFSET TO DISK_BASE = 0
        mov     [ebp+12], eax
        ;MOV    ES,AX                   ; ES IS SEGMENT OF TABLE
        JMP     SHORT DP_OUT

;-----  DATA RATE IS EITHER 300 KBS OR 500 KBS, TRY 1.2 MB TABLE FIRST

USE_EST2:
        MOV     AL,02                   ; DRIVE TYPE 2 (1.2MB)
        CALL    DR_TYPE_CHECK           ; RTN CS:BX = MEDIA/DRIVE PARAM TBL
         MOV     CL, [eBX+MD.SEC_TRK]    ; GET SECTOR/TRACK
         MOV     CH, [eBX+MD.MAX_TRK]    ; GET MAX. TRACK NUMBER
        CMP     AH,RATE_300             ; RATE 300 ?
        JZ      short STO_CX            ; MUST BE 1.2MB DRIVE
        JMP     SHORT PARM144           ; ELSE, IT IS 1.44MB DRIVE

;--------------------------------------------------------------------------------
; DISK_TYPE (AH = 15H)
;       THIS ROUTINE RETURNS THE TYPE OF MEDIA INSTALLED.
;
;  ON ENTRY:   DI = DRIVE #
;
;  ON EXIT:    AH = DRIVE TYPE, CY=0
;--------------------------------------------------------------------------------
DSK_TYPE:
        CALL    XLAT_NEW                ; TRANSLATE STATE TO PRESENT ARCH.
        MOV     AL, [DSK_STATE+eDI]     ; GET PRESENT STATE INFORMATION
        OR      AL,AL                   ; CHECK FOR NO DRIVE
        JZ      short NO_DRV
        MOV     AH,NOCHGLN              ; NO CHANGE LINE FOR 40 TRACK DRIVE
        TEST    AL,TRK_CAPA             ; IS THIS DRIVE AN 80 TRACK DRIVE?
        JZ      short DT_BACK                   ; IF NO JUMP
        MOV     AH,CHGLN                ; CHANGE LINE FOR 80 TRACK DRIVE
```

```
DT_BACK:
        PUSH    AX                      ; SAVE RETURN VALUE
        CALL    XLAT_OLD                ; TRANSLATE STATE TO COMPATIBLE MODE
        POP     AX                      ; RESTORE RETURN VALUE
        CLC                             ; NO ERROR
        MOV     BX,SI                   ; GET SAVED AL TO BL
        MOV     AL,BL                   ; PUT BACK FOR RETURN
        RETn
NO_DRV:
        XOR     AH,AH                   ; NO DRIVE PRESENT OR UNKNOWN
        JMP     SHORT DT_BACK

;--------------------------------------------------------------------------------
; DISK_CHANGE  (AH = 16H)
;       THIS ROUTINE RETURNS THE STATE OF THE DISK CHANGE LINE.
;
; ON ENTRY:     DI = DRIVE #
;
; ON EXIT:      AH = @DSKETTE_STATUS
;                    00 - DISK CHANGE LINE INACTIVE, CY = 0
;                    06 - DISK CHANGE LINE ACTIVE, CY = 1
;--------------------------------------------------------------------------------
DSK_CHANGE:
        CALL    XLAT_NEW                ; TRANSLATE STATE TO PRESENT ARCH.
        MOV     AL, [DSK_STATE+eDI]     ; GET MEDIA STATE INFORMATION
        OR      AL,AL                   ; DRIVE PRESENT ?
        JZ      short DC_NON            ; JUMP IF NO DRIVE
        TEST    AL,TRK_CAPA             ; 80 TRACK DRIVE ?
        JZ      short SETIT             ; IF SO , CHECK CHANGE LINE
DC0:
         CALL    READ_DSKCHNG              ; GO CHECK STATE OF DISK CHANGE LINE
        JZ      short FINIS             ; CHANGE LINE NOT ACTIVE

SETIT:  MOV     byte [DSKETTE_STATUS], MEDIA_CHANGE ; INDICATE MEDIA REMOVED

FINIS:  CALL    XLAT_OLD                ; TRANSLATE STATE TO COMPATIBLE MODE
        CALL    SETUP_END               ; VARIOUS CLEANUPS
        MOV     BX,SI                   ; GET SAVED AL TO BL
        MOV     AL,BL                   ; PUT BACK FOR RETURN
        RETn
DC_NON:
        OR      byte [DSKETTE_STATUS], TIME_OUT ; SET TIMEOUT, NO DRIVE
        JMP     SHORT FINIS

;--------------------------------------------------------------------------------
; FORMAT_SET   (AH = 17H)
;       THIS ROUTINE IS USED TO ESTABLISH THE TYPE OF MEDIA TO BE USED
;       FOR THE FOLLOWING FORMAT OPERATION.
;
; ON ENTRY:     SI LOW = DASD TYPE FOR FORMAT
;               DI     = DRIVE #
;
; ON EXIT:      @DSKETTE_STATUS REFLECTS STATUS
;               AH = @DSKETTE_STATUS
;               CY = 1 IF ERROR
;--------------------------------------------------------------------------------
FORMAT_SET:
        CALL    XLAT_NEW                ; TRANSLATE STATE TO PRESENT ARCH.
        PUSH    SI                      ; SAVE DASD TYPE
        MOV     AX,SI                   ; AH = ? , AL , DASD TYPE
        XOR     AH,AH                   ; AH , 0 , AL , DASD TYPE
        MOV     SI,AX                   ; SI = DASD TYPE
        AND     byte [DSK_STATE+eDI], ~(MED_DET+DBL_STEP+RATE_MSK) ; CLEAR STATE
        DEC     SI                      ; CHECK FOR 320/360K MEDIA & DRIVE
        JNZ     short NOT_320           ; BYPASS IF NOT
        OR      byte [DSK_STATE+eDI], MED_DET+RATE_250 ; SET TO 320/360
        JMP     SHORT S0

NOT_320:
        CALL    MED_CHANGE              ; CHECK FOR TIME_OUT
        CMP     byte [DSKETTE_STATUS], TIME_OUT
        JZ      short S0                ; IF TIME OUT TELL CALLER
S3:
        DEC     SI                      ; CHECK FOR 320/360K IN 1.2M DRIVE
        JNZ     short NOT_320_12        ; BYPASS IF NOT
        OR      byte [DSK_STATE+eDI], MED_DET+DBL_STEP+RATE_300 ; SET STATE
        JMP     SHORT S0
```

```
NOT_320_12:
        DEC     SI                      ; CHECK FOR 1.2M MEDIA IN 1.2M DRIVE
        JNZ     short NOT_12            ; BYPASS IF NOT
        OR      byte [DSK_STATE+eDI], MED_DET+RATE_500 ; SET STATE VARIABLE
        JMP     SHORT S0                ; RETURN TO CALLER

NOT_12:
        DEC     SI                      ; CHECK FOR SET DASD TYPE 04
        JNZ     short FS_ERR            ; BAD COMMAND EXIT IF NOT VALID TYPE

        TEST    byte [DSK_STATE+eDI], DRV_DET ; DRIVE DETERMINED ?
        JZ      short ASSUME            ; IF STILL NOT DETERMINED ASSUME
        MOV     AL,MED_DET+RATE_300
         TEST    byte [DSK_STATE+eDI], FMT_CAPA ; MULTIPLE FORMAT CAPABILITY ?
        JNZ     short OR_IT_IN          ; IF 1.2 M THEN DATA RATE 300

ASSUME:
        MOV     AL,MED_DET+RATE_250     ; SET UP

OR_IT_IN:
        OR      [DSK_STATE+eDI], AL     ; OR IN THE CORRECT STATE
S0:
        CALL    XLAT_OLD                ; TRANSLATE STATE TO COMPATIBLE MODE
        CALL    SETUP_END               ; VARIOUS CLEANUPS
        POP     BX                      ; GET SAVED AL TO BL
        MOV     AL,BL                   ; PUT BACK FOR RETURN
        RETn

FS_ERR:
        MOV     byte [DSKETTE_STATUS], BAD_CMD ; UNKNOWN STATE,BAD COMMAND
        JMP     SHORT S0

;-------------------------------------------------------------------------------
; SET_MEDIA    (AH = 18H)
;       THIS ROUTINE SETS THE TYPE OF MEDIA AND DATA RATE
;       TO BE USED FOR THE FOLLOWING FORMAT OPERATION.
;
; ON ENTRY:
;       [BP]    = SECTOR PER TRACK
;       [BP+1]  = TRACK #
;       DI      = DRIVE #
;
; ON EXIT:
;       @DSKETTE_STATUS REFLECTS STATUS
;       IF NO ERROR:
;               AH = 0
;               CY = 0
;               ES = SEGMENT OF MEDIA/DRIVE PARAMETER TABLE
;               DI/[BP+6] = OFFSET OF MEDIA/DRIVE PARAMETER TABLE
;       IF ERROR:
;               AH = @DSKETTE_STATUS
;               CY = 1
;-------------------------------------------------------------------------------
SET_MEDIA:
        CALL    XLAT_NEW                ; TRANSLATE STATE TO PRESENT ARCH.
         TEST    byte [DSK_STATE+eDI], TRK_CAPA ; CHECK FOR CHANGE LINE AVAILABLE
        JZ      short SM_CMOS           ; JUMP IF 40 TRACK DRIVE
        CALL    MED_CHANGE              ; RESET CHANGE LINE
        CMP     byte [DSKETTE_STATUS], TIME_OUT ; IF TIME OUT TELL CALLER
        JE      short SM_RTN
        MOV     byte [DSKETTE_STATUS], 0 ; CLEAR STATUS
SM_CMOS:
        CALL    CMOS_TYPE               ; RETURN DRIVE TYPE IN (AL)
        ;;20/02/2015
        ;;JC    short MD_NOT_FND        ; ERROR IN CMOS
        ;;OR    AL,AL                   ; TEST FOR NO DRIVE
        JZ      short SM_RTN            ; RETURN IF SO
        CALL    DR_TYPE_CHECK           ; RTN CS:BX = MEDIA/DRIVE PARAM TBL
        JC      short MD_NOT_FND        ; TYPE NOT IN TABLE (BAD CMOS)
        PUSH    eDI                     ; SAVE REG.
        XOR     eBX,eBX                 ; BX = INDEX TO DR. TYPE TABLE
        MOV     eCX,DR_CNT              ; CX = LOOP COUNT
DR_SEARCH:
        MOV     AH, [DR_TYPE+eBX]       ; GET DRIVE TYPE
        AND     AH,BIT7OFF              ; MASK OUT MSB
        CMP     AL,AH                   ; DRIVE TYPE MATCH ?
        JNE     short NXT_MD            ; NO, CHECK NEXT DRIVE TYPE
DR_FND:
        MOV     eDI, [DR_TYPE+eBX+1]    ; DI = MEDIA/DRIVE PARAM TABLE
```

```
MD_SEARCH:
         MOV     AH, [eDI+MD.SEC_TRK]     ; GET SECTOR/TRACK
        CMP     [eBP],AH                ; MATCH?
        JNE     short NXT_MD            ; NO, CHECK NEXT MEDIA
         MOV     AH, [eDI+MD.MAX_TRK]     ; GET MAX. TRACK #
        CMP     [eBP+1],AH              ; MATCH?
        JE      short MD_FND            ; YES, GO GET RATE
NXT_MD:
        ;ADD    BX,3                    ; CHECK NEXT DRIVE TYPE
         add    ebx, 5 ; 18/02/2015
        LOOP     DR_SEARCH
        POP     eDI                     ; RESTORE REG.
MD_NOT_FND:
        MOV     byte [DSKETTE_STATUS], MED_NOT_FND ; ERROR, MEDIA TYPE NOT FOUND
        JMP     SHORT SM_RTN            ; RETURN
MD_FND:
         MOV     AL, [eDI+MD.RATE]        ; GET RATE
        CMP     AL,RATE_300             ; DOUBLE STEP REQUIRED FOR RATE 300
        JNE     short MD_SET
        OR      AL,DBL_STEP
MD_SET:
        ;MOV    [BP+6],DI               ; SAVE TABLE POINTER IN STACK
        mov     [ebp+12], edi ; 18/02/2015
        OR      AL,MED_DET              ; SET MEDIA ESTABLISHED
        POP     eDI
        AND     byte [DSK_STATE+eDI], ~(MED_DET+DBL_STEP+RATE_MSK) ; CLEAR STATE
        OR      [DSK_STATE+eDI], AL
        ;MOV    AX, CS                  ; SEGMENT OF MEDIA/DRIVE PARAMETER TABLE
        ;MOV    ES, AX                  ; ES IS SEGMENT OF TABLE
SM_RTN:
        CALL    XLAT_OLD                ; TRANSLATE STATE TO COMPATIBLE MODE
        CALL    SETUP_END               ; VARIOUS CLEANUPS
        RETn

;-----------------------------------------------------------------
; DR_TYPE_CHECK                                                  :
;       CHECK IF THE GIVEN DRIVE TYPE IN REGISTER (AL)           :
;       IS SUPPORTED IN BIOS DRIVE TYPE TABLE                    :
; ON ENTRY:                                              :
;       AL = DRIVE TYPE                                          :
; ON EXIT:                                               :
;       CS = SEGMENT MEDIA/DRIVE PARAMETER TABLE (CODE)          :
;       CY = 0  DRIVE TYPE SUPPORTED                     :
;            BX = OFFSET TO MEDIA/DRIVE PARAMETER TABLE          :
;       CY = 1  DRIVE TYPE NOT SUPPORTED                 :
; REGISTERS ALTERED: eBX                                         :
;-----------------------------------------------------------------
DR_TYPE_CHECK:
        PUSH    AX
        PUSH    eCX
        XOR     eBX,eBX                 ; BX = INDEX TO DR_TYPE TABLE
        MOV     eCX,DR_CNT              ; CX = LOOP COUNT
TYPE_CHK:
        MOV     AH,[DR_TYPE+eBX]        ; GET DRIVE TYPE
        CMP     AL,AH                   ; DRIVE TYPE MATCH?
        JE      short DR_TYPE_VALID     ; YES, RETURN WITH CARRY RESET
        ;ADD    BX,3                    ; CHECK NEXT DRIVE TYPE
         add    ebx, 5  ; 16/02/2015 (32 bit address modification)
        LOOP     TYPE_CHK
        ;
        mov     ebx, MD_TBL6            ; 1.44MB fd parameter table
                                        ; Default for GET_PARM (11/12/2014)
        ;
        STC                             ; DRIVE TYPE NOT FOUND IN TABLE
        JMP     SHORT TYPE_RTN
DR_TYPE_VALID:
        MOV     eBX,[DR_TYPE+eBX+1]     ; BX = MEDIA TABLE
TYPE_RTN:
        POP     eCX
        POP     AX
        RETn
```

```
;-----------------------------------------------------------------
; SEND_SPEC                                                      :
;       SEND THE SPECIFY COMMAND TO CONTROLLER USING DATA FROM :
;       THE DRIVE PARAMETER TABLE POINTED BY @DISK_POINTER  :
; ON ENTRY:     @DISK_POINTER = DRIVE PARAMETER TABLE            :
; ON EXIT:      NONE                                             :
; REGISTERS ALTERED: CX, DX                                      :
;-----------------------------------------------------------------
SEND_SPEC:
        PUSH    eAX                     ; SAVE AX
        MOV     eAX, SPECBAC            ; LOAD ERROR ADDRESS
        PUSH    eAX                     ; PUSH NEC_OUT ERROR RETURN
        MOV     AH,03H                  ; SPECIFY COMMAND
        CALL    NEC_OUTPUT              ; OUTPUT THE COMMAND
        SUB     DL,DL                   ; FIRST SPECIFY BYTE
        CALL    GET_PARM                ; GET PARAMETER TO AH
        CALL    NEC_OUTPUT              ; OUTPUT THE COMMAND
        MOV     DL,1                    ; SECOND SPECIFY BYTE
        CALL    GET_PARM                ; GET PARAMETER TO AH
        CALL    NEC_OUTPUT              ; OUTPUT THE COMMAND
        POP     eAX                     ; POP ERROR RETURN
SPECBAC:
        POP     eAX                     ; RESTORE ORIGINAL AX VALUE
        RETn

;-----------------------------------------------------------------
; SEND_SPEC_MD                                                   :
;       SEND THE SPECIFY COMMAND TO CONTROLLER USING DATA FROM      :
;       THE MEDIA/DRIVE PARAMETER TABLE POINTED BY (CS:BX)  :
; ON ENTRY:     CS:BX = MEDIA/DRIVE PARAMETER TABLE              :
; ON EXIT:      NONE                                             :
; REGISTERS ALTERED: AX                                                  :
;-----------------------------------------------------------------
SEND_SPEC_MD:
        PUSH    eAX                     ; SAVE RATE DATA
        MOV     eAX, SPEC_ESBAC                 ; LOAD ERROR ADDRESS
        PUSH    eAX                     ; PUSH NEC_OUT ERROR RETURN
        MOV     AH,03H                  ; SPECIFY COMMAND
        CALL    NEC_OUTPUT              ; OUTPUT THE COMMAND
         MOV     AH, [eBX+MD.SPEC1]        ; GET 1ST SPECIFY BYTE
        CALL    NEC_OUTPUT              ; OUTPUT THE COMMAND
         MOV     AH, [eBX+MD.SPEC2]        ; GET SECOND SPECIFY BYTE
        CALL    NEC_OUTPUT              ; OUTPUT THE COMMAND
        POP     eAX                     ; POP ERROR RETURN
SPEC_ESBAC:
        POP     eAX                     ; RESTORE ORIGINAL AX VALUE
        RETn

;-------------------------------------------------------------------------------
; XLAT_NEW
;       TRANSLATES DISKETTE STATE LOCATIONS FROM COMPATIBLE
;       MODE TO NEW ARCHITECTURE.
;
; ON ENTRY:     DI = DRIVE #
;-------------------------------------------------------------------------------
XLAT_NEW:
        CMP     eDI,1                           ; VALID DRIVE
        JA      short XN_OUT                    ; IF INVALID BACK
        CMP     byte [DSK_STATE+eDI], 0                 ; NO DRIVE ?
        JZ      short DO_DET                    ; IF NO DRIVE ATTEMPT DETERMINE
        MOV     CX,DI                           ; CX = DRIVE NUMBER
        SHL     CL,2                            ; CL = SHIFT COUNT, A=0, B=4
        MOV     AL, [HF_CNTRL]                  ; DRIVE INFORMATION
        ROR     AL,CL                           ; TO LOW NIBBLE
        AND     AL,DRV_DET+FMT_CAPA+TRK_CAPA  ; KEEP DRIVE BITS
         AND     byte [DSK_STATE+eDI], ~(DRV_DET+FMT_CAPA+TRK_CAPA)
        OR      [DSK_STATE+eDI], AL             ; UPDATE DRIVE STATE
XN_OUT:
        RETn
DO_DET:
        CALL    DRIVE_DET                       ; TRY TO DETERMINE
        RETn
```

```
;-------------------------------------------------------------------------------
; XLAT_OLD
;       TRANSLATES DISKETTE STATE LOCATIONS FROM NEW
;       ARCHITECTURE TO COMPATIBLE MODE.
;
; ON ENTRY:    DI = DRIVE
;-------------------------------------------------------------------------------
XLAT_OLD:
        CMP     eDI,1                   ; VALID DRIVE ?
        ;JA     short XO_OUT            ; IF INVALID BACK
        ja      XO_OUT
        CMP     byte [DSK_STATE+eDI],0; NO DRIVE ?
        JZ      short XO_OUT            ; IF NO DRIVE TRANSLATE DONE

;-----  TEST FOR SAVED DRIVE INFORMATION ALREADY SET

        MOV     CX,DI                   ; CX = DRIVE NUMBER
        SHL     CL,2                    ; CL = SHIFT COUNT, A=0, B=4
        MOV     AH,FMT_CAPA             ; LOAD MULTIPLE DATA RATE BIT MASK
        ROR     AH,CL                   ; ROTATE BY MASK
        TEST    [HF_CNTRL], AH          ; MULTIPLE-DATA RATE DETERMINED ?
        JNZ     short SAVE_SET          ; IF SO, NO NEED TO RE-SAVE

;-----  ERASE DRIVE BITS IN @HF_CNTRL FOR THIS DRIVE

        MOV     AH,DRV_DET+FMT_CAPA+TRK_CAPA ; MASK TO KEEP
        ROR     AH,CL                   ; FIX MASK TO KEEP
        NOT     AH                      ; TRANSLATE MASK
        AND     [HF_CNTRL], AH          ; KEEP BITS FROM OTHER DRIVE INTACT

;-----  ACCESS CURRENT DRIVE BITS AND STORE IN @HF_CNTRL

        MOV     AL, [DSK_STATE+eDI]     ; ACCESS STATE
        AND     AL,DRV_DET+FMT_CAPA+TRK_CAPA ; KEEP DRIVE BITS
        ROR     AL,CL                   ; FIX FOR THIS DRIVE
        OR      [HF_CNTRL], AL          ; UPDATE SAVED DRIVE STATE

;-----  TRANSLATE TO COMPATIBILITY MODE

SAVE_SET:
        MOV     AH, [DSK_STATE+eDI]     ; ACCESS STATE
        MOV     BH,AH                   ; TO BH FOR LATER
        AND     AH,RATE_MSK             ; KEEP ONLY RATE
        CMP     AH,RATE_500             ; RATE 500 ?
        JZ      short CHK_144           ; YES 1.2/1.2 OR 1.44/1.44
        MOV     AL,M3D1U                ; AL = 360 IN 1.2 UNESTABLISHED
        CMP     AH,RATE_300             ; RATE 300 ?
        JNZ     short CHK_250           ; NO, 360/360, 720/720 OR 720/1.44
        TEST    BH,DBL_STEP             ; CHECK FOR DOUBLE STEP
        JNZ     short TST_DET           ; MUST BE 360 IN 1.2
UNKNO:
        MOV     AL,MED_UNK              ; NONE OF THE ABOVE
        JMP     SHORT AL_SET            ; PROCESS COMPLETE
CHK_144:
        CALL    CMOS_TYPE               ; RETURN DRIVE TYPE IN (AL)
        ;;20/02/2015
        ;;JC    short UNKNO             ; ERROR, SET 'NONE OF ABOVE'
        jz      short UNKNO ;; 20/02/2015
        CMP     AL,2                    ; 1.2MB DRIVE ?
        JNE     short UNKNO             ; NO, GO SET 'NONE OF ABOVE'
        MOV     AL,M1D1U                ; AL = 1.2 IN 1.2 UNESTABLISHED
        JMP     SHORT TST_DET
CHK_250:
        MOV     AL,M3D3U                ; AL = 360 IN 360 UNESTABLISHED
        CMP     AH,RATE_250             ; RATE 250 ?
        JNZ     short UNKNO             ; IF SO FALL IHRU
        TEST    BH,TRK_CAPA             ; 80 TRACK CAPABILITY ?
        JNZ     short UNKNO             ; IF SO JUMP, FALL THRU TEST DET
TST_DET:
        TEST    BH,MED_DET              ; DETERMINED ?
        JZ      short AL_SET            ; IF NOT THEN SET
        ADD     AL,3                    ; MAKE DETERMINED/ESTABLISHED
AL_SET:
        AND     byte [DSK_STATE+eDI], ~(DRV_DET+FMT_CAPA+TRK_CAPA) ; CLEAR DRIVE
        OR      [DSK_STATE+eDI], AL     ; REPLACE WITH COMPATIBLE MODE
XO_OUT:
        RETn
```

```
;-------------------------------------------------------------------------------
; RD_WR_VF
;       COMMON READ, WRITE AND VERIFY:
;       MAIN LOOP FOR STATE RETRIES.
;
; ON ENTRY:     AH = READ/WRITE/VERIFY NEC PARAMETER
;               AL = READ/WRITE/VERIFY DMA PARAMETER
;
; ON EXIT:      @DSKETTE_STATUS, CY REFLECT STATUS OF OPERATION
;-------------------------------------------------------------------------------
RD_WR_VF:
        PUSH    AX                      ; SAVE DMA, NEC PARAMETERS
        CALL    XLAT_NEW                ; TRANSLATE STATE TO PRESENT ARCH.
        CALL    SETUP_STATE             ; INITIALIZE START AND END RATE
        POP     AX                      ; RESTORE READ/WRITE/VERIFY
DO_AGAIN:
        PUSH    AX                      ; SAVE READ/WRITE/VERIFY PARAMETER
        CALL    MED_CHANGE              ; MEDIA CHANGE AND RESET IF CHANGED
        POP     AX                      ; RESTORE READ/WRITE/VERIFY
         JC      RWV_END                   ; MEDIA CHANGE ERROR OR TIME-OUT
RWV:
        PUSH    AX                      ; SAVE READ/WRITE/VERIFY PARAMETER
        MOV     DH, [DSK_STATE+eDI]     ; GET RATE STATE OF THIS DRIVE
        AND     DH,RATE_MSK             ; KEEP ONLY RATE
        CALL    CMOS_TYPE               ; RETURN DRIVE TYPE IN AL (AL)
        ;;20/02/2015
        ;;JC    short RWV_ASSUME        ; ERROR IN CMOS
        jz      short RWV_ASSUME ; 20/02/2015
        CMP     AL,1                    ; 40 TRACK DRIVE?
        JNE     short RWV_1             ; NO, BYPASS CMOS VALIDITY CHECK
        TEST    byte [DSK_STATE+eDI], TRK_CAPA ; CHECK FOR 40 TRACK DRIVE
        JZ      short RWV_2             ; YES, CMOS IS CORRECT
        MOV     AL,2                    ; CHANGE TO 1.2M
        JMP     SHORT RWV_2
RWV_1:
        JB      short RWV_2             ; NO DRIVE SPECIFIED, CONTINUE
        TEST     byte [DSK_STATE+eDI], TRK_CAPA ; IS IT REALLY 40 TRACK?
        JNZ     short RWV_2             ; NO, 80 TRACK
        MOV     AL,1                    ; IT IS 40 TRACK, FIX CMOS VALUE
        jmp     short rwv_3
RWV_2:
        OR      AL,AL                   ; TEST FOR NO DRIVE
        JZ      short RWV_ASSUME        ; ASSUME TYPE, USE MAX TRACK
rwv_3:
        CALL    DR_TYPE_CHECK           ; RTN CS:BX = MEDIA/DRIVE PARAM TBL.
        JC      short RWV_ASSUME        ; TYPE NOT IN TABLE (BAD CMOS)

;----- SEARCH FOR MEDIA/DRIVE PARAMETER TABLE

        PUSH    eDI                     ; SAVE DRIVE #
        XOR     eBX,eBX                 ; BX = INDEX TO DR_TYPE TABLE
        MOV     eCX,DR_CNT              ; CX = LOOP COUNT
RWV_DR_SEARCH:
        MOV     AH, [DR_TYPE+eBX]       ; GET DRIVE TYPE
        AND     AH,BIT7OFF              ; MASK OUT MSB
        CMP     AL,AH                   ; DRIVE TYPE MATCH?
        JNE     short RWV_NXT_MD        ; NO, CHECK NEXT DRIVE TYPE
RWV_DR_FND:
        MOV     eDI, [DR_TYPE+eBX+1]    ; DI = MEDIA/DRIVE PARAMETER TABLE
RWV_MD_SEARH:
         CMP     DH, [eDI+MD.RATE]         ; MATCH?
        JE      short RWV_MD_FND        ; YES, GO GET 1ST SPECIFY BYTE
RWV_NXT_MD:
        ;ADD    BX,3                    ; CHECK NEXT DRIVE TYPE
        add     eBX, 5
        LOOP    RWV_DR_SEARCH
        POP     eDI                     ; RESTORE DRIVE #

;----- ASSUME PRIMARY DRIVE IS INSTALLED AS SHIPPED

RWV_ASSUME:
        MOV     eBX, MD_TBL1            ; POINT TO 40 TRACK 250 KBS
        TEST    byte [DSK_STATE+eDI], TRK_CAPA ; TEST FOR 80 TRACK
        JZ      short RWV_MD_FND1       ; MUST BE 40 TRACK
        MOV     eBX, MD_TBL3            ; POINT TO 80 TRACK 500 KBS
        JMP     short RWV_MD_FND1       ; GO SPECIFY PARAMTERS
```

```
;----- CS:BX POINTS TO MEDIA/DRIVE PARAMETER TABLE

RWV_MD_FND:
       MOV     eBX,eDI                 ; BX = MEDIA/DRIVE PARAMETER TABLE
       POP     eDI                     ; RESTORE DRIVE #

;----- SEND THE SPECIFY COMMAND TO THE CONTROLLER

RWV_MD_FND1:
       CALL    SEND_SPEC_MD
       CALL    CHK_LASTRATE            ; ZF=1 ATTEMP RATE IS SAME AS LAST RATE
       JZ      short RWV_DBL           ; YES,SKIP SEND RATE COMMAND
       CALL    SEND_RATE               ; SEND DATA RATE TO NEC
RWV_DBL:
       PUSH    eBX                     ; SAVE MEDIA/DRIVE PARAM TBL ADDRESS
       CALL    SETUP_DBL               ; CHECK FOR DOUBLE STEP
       POP     eBX                     ; RESTORE ADDRESS
       JC      short CHK_RET           ; ERROR FROM READ ID, POSSIBLE RETRY
       POP     AX                      ; RESTORE NEC, DMA COMMAND
       PUSH    AX                      ; SAVE NEC COMMAND
       PUSH    eBX                     ; SAVE MEDIA/DRIVE PARAM TBL ADDRESS
       CALL    DMA_SETUP               ; SET UP THE DMA
       POP     eBX
       POP     AX                      ; RESTORE NEC COMMAND
       JC      short RWV_BAC           ; CHECK FOR DMA BOUNDARY ERROR
       PUSH    AX                      ; SAVE NEC COMMAND
       PUSH    eBX                     ; SAVE MEDIA/DRIVE PARAM TBL ADDRESS
       CALL    NEC_INIT                ; INITIALIZE NEC
       POP     eBX                     ; RESTORE ADDRESS
       JC      short CHK_RET           ; ERROR - EXIT
       CALL    RWV_COM                 ; OP CODE COMMON TO READ/WRITE/VERIFY
       JC      short CHK_RET           ; ERROR - EXIT
       CALL    NEC_TERM                ; TERMINATE, GET STATUS, ETC.
CHK_RET:
       CALL    RETRY                   ; CHECK FOR, SETUP RETRY
       POP     AX                      ; RESTORE READ/WRITE/VERIFY PARAMETER
       JNC     short RWV_END           ; CY = 0 NO RETRY
        JMP     DO_AGAIN                  ; CY = 1 MEANS RETRY
RWV_END:
       CALL    DSTATE                  ; ESTABLISH STATE IF SUCCESSFUL
       CALL    NUM_TRANS               ; AL = NUMBER TRANSFERRED
RWV_BAC:                               ; BAD DMA ERROR ENTRY
       PUSH    AX                      ; SAVE NUMBER TRANSFERRED
       CALL    XLAT_OLD                ; TRANSLATE STATE TO COMPATIBLE MODE
       POP     AX                      ; RESTORE NUMBER TRANSFERRED
       CALL    SETUP_END               ; VARIOUS CLEANUPS
       RETn

;--------------------------------------------------------------------------------
; SETUP_STATE: INITIALIZES START AND END RATES.
;--------------------------------------------------------------------------------
SETUP_STATE:
       TEST    byte [DSK_STATE+eDI], MED_DET ; MEDIA DETERMINED ?
       JNZ     short J1C               ; NO STATES IF DETERMINED
        MOV     AX,(RATE_500*256)+RATE_300  ; AH = START RATE, AL = END RATE
       TEST    byte [DSK_STATE+eDI],DRV_DET ; DRIVE ?
       JZ      short AX_SET            ; DO NOT KNOW DRIVE
       TEST    byte [DSK_STATE+eDI], FMT_CAPA ; MULTI-RATE?
       JNZ     short AX_SET            ; JUMP IF YES
        MOV     AX,RATE_250*257           ; START A END RATE 250 FOR 360 DRIVE
AX_SET:
       AND     byte [DSK_STATE+eDI], ~(RATE_MSK+DBL_STEP) ; TURN OFF THE RATE
       OR      [DSK_STATE+eDI], AH     ; RATE FIRST TO TRY
       AND     byte [LASTRATE], ~STRT_MSK ; ERASE LAST TO TRY RATE BITS
       ROR     AL,4                    ; TO OPERATION LAST RATE LOCATION
       OR      [LASTRATE], AL          ; LAST RATE
J1C:
       RETn
```

```
;-------------------------------------------------------------------------------
;  FMT_INIT: ESTABLISH STATE IF UNESTABLISHED AT FORMAT TIME.
;-------------------------------------------------------------------------------
FMT_INIT:
       TEST    byte [DSK_STATE+eDI], MED_DET ; IS MEDIA ESTABLISHED
       JNZ     short F1_OUT            ; IF SO RETURN
       CALL    CMOS_TYPE               ; RETURN DRIVE TYPE IN AL
       ;; 20/02/2015
       ;;JC    short CL_DRV            ; ERROR IN CMOS ASSUME NO DRIVE
       jz      short CL_DRV ;; 20/02/2015
       DEC     AL                      ; MAKE ZERO ORIGIN
       ;;JS    short CL_DRV            ; NO DRIVE IF AL 0
       MOV     AH, [DSK_STATE+eDI]     ; AH = CURRENT STATE
       AND     AH, ~(MED_DET+DBL_STEP+RATE_MSK) ; CLEAR
       OR      AL,AL                   ; CHECK FOR 360
       JNZ     short N_360             ; IF 360 WILL BE 0
       OR      AH,MED_DET+RATE_250     ; ESTABLISH MEDIA
       JMP     SHORT SKP_STATE                 ; SKIP OTHER STATE PROCESSING
N_360:
       DEC     AL                      ; 1.2 M DRIVE
       JNZ     short N_12              ; JUMP IF NOT
F1_RATE:
       OR      AH,MED_DET+RATE_500     ; SET FORMAT RATE
       JMP     SHORT SKP_STATE                 ; SKIP OTHER STATE PROCESSING
N_12:
       DEC     AL                      ; CHECK FOR TYPE 3
       JNZ     short N_720             ; JUMP IF NOT
       TEST    AH,DRV_DET              ; IS DRIVE DETERMINED
       JZ      short ISNT_12           ; TREAT AS NON 1.2 DRIVE
       TEST    AH,FMT_CAPA             ; IS 1.2M
       JZ      short ISNT_12           ; JUMP IF NOT
       OR      AH,MED_DET+RATE_300     ; RATE 300
       JMP     SHORT SKP_STATE                 ; CONTINUE
N_720:
       DEC     AL                      ; CHECK FOR TYPE 4
       JNZ     short CL_DRV            ; NO DRIVE, CMOS BAD
       JMP     SHORT F1_RATE
ISNT_12:
       OR      AH,MED_DET+RATE_250     ; MUST BE RATE 250

SKP_STATE:
       MOV     [DSK_STATE+eDI], AH     ; STORE AWAY
F1_OUT:
       RETn
CL_DRV:
       XOR     AH,AH                   ; CLEAR STATE
       JMP     SHORT SKP_STATE                 ; SAVE IT

;-------------------------------------------------------------------------------
; MED_CHANGE
;       CHECKS FOR MEDIA CHANGE, RESETS MEDIA CHANGE,
;       CHECKS MEDIA CHANGE AGAIN.
;
; ON EXIT:      CY = 1 MEANS MEDIA CHANGE OR TIMEOUT
;               @DSKETTE_STATUS = ERROR CODE
;-------------------------------------------------------------------------------
MED_CHANGE:
       CALL    READ_DSKCHNG            ; READ DISK CHANCE LINE STATE
       JZ      short MC_OUT            ; BYPASS HANDLING DISK CHANGE LINE
       AND     byte [DSK_STATE+eDI], ~MED_DET ; CLEAR STATE FOR THIS DRIVE

;       THIS SEQUENCE ENSURES WHENEVER A DISKETTE IS CHANGED THAT
;       ON THE NEXT OPERATION THE REQUIRED MOTOR START UP TIME WILL
;       BE WAITED. (DRIVE MOTOR MAY GO OFF UPON DOOR OPENING).

       MOV     CX,DI                   ; CL = DRIVE 0
       MOV     AL,1                    ; MOTOR ON BIT MASK
       SHL     AL,CL                   ; TO APPROPRIATE POSITION
       NOT     AL                      ; KEEP ALL BUT MOTOR ON
       CLI                             ; NO INTERRUPTS
       AND     [MOTOR_STATUS], AL      ; TURN MOTOR OFF INDICATOR
       STI                             ; INTERRUPTS ENABLED
       CALL    MOTOR_ON                ; TURN MOTOR ON
```

```
;----- THIS SEQUENCE OF SEEKS IS USED TO RESET DISKETTE CHANGE SIGNAL

       CALL   DSK_RESET            ; RESET NEC
       MOV    CH,01H               ; MOVE TO CYLINDER 1
       CALL   SEEK                 ; ISSUE SEEK
       XOR    CH,CH                ; MOVE TO CYLINDER 0
       CALL   SEEK                 ; ISSUE SEEK
       MOV    byte [DSKETTE_STATUS], MEDIA_CHANGE ; STORE IN STATUS
OK1:
       CALL   READ_DSKCHNG         ; CHECK MEDIA CHANGED AGAIN
       JZ     short OK2            ; IF ACTIVE, NO DISKETTE, TIMEOUT
OK4:
       MOV    byte [DSKETTE_STATUS], TIME_OUT ; TIMEOUT IF DRIVE EMPTY
OK2:
       STC                         ; MEDIA CHANGED, SET CY
       RETn
MC_OUT:
       CLC                         ; NO MEDIA CHANGED, CLEAR CY
       RETn

;-------------------------------------------------------------------------------
; SEND_RATE
;      SENDS DATA RATE COMMAND TO NEC
; ON ENTRY:    DI = DRIVE #
; ON EXIT:     NONE
; REGISTERS ALTERED: DX
;-------------------------------------------------------------------------------
SEND_RATE:
       PUSH   AX                   ; SAVE REG.
       AND    byte [LASTRATE], ~SEND_MSK ; ELSE CLEAR LAST RATE ATTEMPTED
       MOV    AL, [DSK_STATE+eDI]  ; GET RATE STATE OF THIS DRIVE
       AND    AL,SEND_MSK          ; KEEP ONLY RATE BITS
       OR     [LASTRATE], AL       ; SAVE NEW RATE FOR NEXT CHECK
       ROL    AL,2                 ; MOVE TO BIT OUTPUT POSITIONS
       MOV    DX,03F7H             ; OUTPUT NEW DATA RATE
       OUT    DX,AL
       POP    AX                   ; RESTORE REG.
       RETn

;-------------------------------------------------------------------------------
; CHK_LASTRATE
;      CHECK PREVIOUS DATE RATE SNT TO THE CONTROLLER.
; ON ENTRY:
;      DI = DRIVE #
; ON EXIT:
;      ZF =  1 DATA RATE IS THE SAME AS THE LAST RATE SENT TO NEC
;      ZF =  0 DATA RATE IS DIFFERENT FROM LAST RATE
; REGISTERS ALTERED: DX
;-------------------------------------------------------------------------------
CHK_LASTRATE:
       PUSH   AX                   ; SAVE REG
       AND    AH, [LASTRATE]       ; GET LAST DATA RATE SELECTED
       MOV    AL, [DSK_STATE+eDI]  ; GET RATE STATE OF THIS DRIVE
        AND    AX, SEND_MSK*257       ; KEEP ONLY RATE BITS OF BOTH
       CMP    AL, AH               ; COMPARE TO PREVIOUSLY TRIED
                                   ; ZF = 1 RATE IS THE SAME
       POP    AX                   ; RESTORE REG.
       RETn

;-------------------------------------------------------------------------------
; DMA_SETUP
;      THIS ROUTINE SETS UP THE DMA FOR READ/WRITE/VERIFY OPERATIONS.
;
; ON ENTRY:    AL = DMA COMMAND
;
; ON EXIT:     @DSKETTE_STATUS, CY REFLECT STATUS OF OPERATION
;-------------------------------------------------------------------------------
; SI = Head #, # of Sectors or DASD Type

; 22/08/2015
; 08/02/2015 - Protected Mode Modification
; 06/02/2015 - 07/02/2015
; NOTE: Buffer address must be in 1st 16MB of Physical Memory (24 bit limit).
; (DMA Addres = Physical Address)
; (Retro UNIX 386 v1 Kernel/System Mode Virtual Address = Physical Address)
;
; 20/02/2015 modification (source: AWARD BIOS 1999, DMA_SETUP)
; 16/12/2014 (IODELAY)
```

```
DMA_SETUP:
;; 20/02/2015
        mov     edx, [ebp+4]            ; Buffer address
        test    edx, 0FF000000h                 ; 16 MB limit (22/08/2015, bugfix)
        jnz     short dma_bnd_err_stc
        ;
        push    ax                      ; DMA command
        push    edx                     ; *
        mov     dl, 3                   ; GET BYTES/SECTOR PARAMETER
        call    GET_PARM                ;
        mov     cl, ah                  ; SHIFT COUNT (0=128, 1=256, 2=512 ETC)
        mov     ax, si                  ; Sector count
        mov     ah, al                  ; AH =  # OF SECTORS
        sub     al, al                  ; AL = 0, AX = # SECTORS * 256
        shr     ax, 1                   ; AX = # SECTORS * 128
        shl     ax, cl                  ; SHIFT BY PARAMETER VALUE
        dec     ax                      ; -1 FOR DMA VALUE
        mov     cx, ax
        pop     edx                     ; *
        pop     ax
        cmp     al, 42h
         jne     short NOT_VERF
        mov     edx, 0FF0000h
        jmp     short J33
NOT_VERF:
        add     dx, cx                  ; check for overflow
        jc      short dma_bnd_err
        ;
        sub     dx, cx                  ; Restore start address
J33:
        CLI                             ; DISABLE INTERRUPTS DURING DMA SET-UP
        OUT     DMA+12,AL               ; SET THE FIRST/LA5T F/F
        IODELAY                         ; WAIT FOR I/O
        OUT     DMA+11,AL               ; OUTPUT THE MODE BYTE
        mov     eax, edx                ; Buffer address
        OUT     DMA+4,AL                ; OUTPUT LOW ADDRESS
        IODELAY                         ; WAIT FOR I/O
        MOV     AL,AH
        OUT     DMA+4,AL                ; OUTPUT HIGH ADDRESS
        shr     eax, 16
        IODELAY                         ; I/O WAIT STATE
        OUT     081H,AL                 ; OUTPUT highest BITS TO PAGE REGISTER
        IODELAY
        mov     ax, cx                  ; Byte count - 1
        OUT     DMA+5,AL                ; LOW BYTE OF COUNT
        IODELAY                         ; WAIT FOR I/O
        MOV     AL, AH
        OUT     DMA+5,AL                ; HIGH BYTE OF COUNT
        IODELAY
        STI                             ; RE-ENABLE INTERRUPTS
        MOV     AL, 2                   ; MODE FOR 8237
        OUT     DMA+10, AL              ; INITIALIZE THE DISKETTE CHANNEL
        retn
dma_bnd_err_stc:
        stc
dma_bnd_err:
        MOV     byte [DSKETTE_STATUS], DMA_BOUNDARY ; SET ERROR
        RETn                            ; CY SET BY ABOVE IF ERROR

;; 16/12/2014
;;      CLI                             ; DISABLE INTERRUPTS DURING DMA SET-UP
;;      OUT     DMA+12,AL               ; SET THE FIRST/LA5T F/F
;;      ;JMP    $+2                     ; WAIT FOR I/O
;;      IODELAY
;;      OUT     DMA+11,AL               ; OUTPUT THE MODE BYTE
;;      ;SIODELAY
;;       ;CMP   AL, 42H                 ; DMA VERIFY COMMAND
;;       ;JNE   short NOT_VERF          ; NO
;;       ;XOR   AX, AX                  ; START ADDRESS
;;       ;JMP   SHORT J33
;;;NOT_VERF:
;;      ;MOV    AX,ES                   ; GET THE ES VALUE
;;      ;ROL    AX,4                    ; ROTATE LEFT
;;      ;MOV    CH,AL                   ; GET HIGHEST NIBBLE OF ES TO CH
;;      ;AND    AL,11110000B            ; ZERO THE LOW NIBBLE FROM SEGMENT
;;      ;ADD    AX,[BP+2]               ; TEST FOR CARRY FROM ADDITION
;;      mov     eax, [ebp+4] ; 06/02/2015
;;      ;JNC    short J33
;;      ;INC    CH                      ; CARRY MEANS HIGH 4 BITS MUST BE INC
```

```
;;;J33:
;;      PUSH    eAX                     ; SAVE START ADDRESS
;;      OUT     DMA+4,AL                ; OUTPUT LOW ADDRESS
;;      ;JMP    $+2                     ; WAIT FOR I/O
;;      IODELAY
;;      MOV     AL,AH
;;      OUT     DMA+4,AL                ; OUTPUT HIGH ADDRESS
;;      shr     eax, 16       ; 07/02/2015
;;      ;MOV    AL,CH                   ; GET HIGH 4 BITS
;;      ;JMP    $+2                     ; I/O WAIT STATE
;;      IODELAY
;;      ;AND    AL,00001111B
;;      OUT     081H,AL                 ; OUTPUT HIGH 4 BITS TO PAGE REGISTER
;;      ;SIODELAY
;;
;;;----- DETERMINE COUNT
;;      sub     eax, eax ; 08/02/2015
;;      MOV     AX, SI                  ; AL =  # OF SECTORS
;;      XCHG    AL, AH                  ; AH =  # OF SECTORS
;;      SUB     AL, AL                  ; AL = 0, AX = # SECTORS * 256
;;      SHR     AX, 1                   ; AX = # SECTORS * 128
;;      PUSH    AX                      ; SAVE # OF SECTORS * 128
;;      MOV     DL, 3                   ; GET BYTES/SECTOR PARAMETER
;;      CALL    GET_PARM                ; "
;;      MOV     CL,AH                   ; SHIFT COUNT (0=128, 1=256, 2=512 ETC)
;;      POP     AX                      ; AX = # SECTORS * 128
;;      SHL     AX,CL                   ; SHIFT BY PARAMETER VALUE
;;      DEC     AX                      ; -1 FOR DMA VALUE
;;      PUSH    eAX  ; 08/02/2015       ; SAVE COUNT VALUE
;;      OUT     DMA+5,AL                ; LOW BYTE OF COUNT
;;      ;JMP    $+2                     ; WAIT FOR I/O
;;      IODELAY
;;      MOV     AL, AH
;;      OUT     DMA+5,AL                ; HIGH BYTE OF COUNT
;;      ;IODELAY
;;      STI                             ; RE-ENABLE INTERRUPTS
;;      POP     eCX  ; 08/02/2015       ; RECOVER COUNT VALUE
;;      POP     eAX  ; 08/02/2015       ; RECOVER ADDRESS VALUE
;;      ;ADD    AX, CX                  ; ADD, TEST FOR 64K OVERFLOW
;;      add     ecx, eax ; 08/02/2015
;;      MOV     AL, 2                   ; MODE FOR 8237
;;      ;JMP    $+2                     ; WAIT FOR I/O
;;      SIODELAY
;;      OUT     DMA+10, AL              ; INITIALIZE THE DISKETTE CHANNEL
;;      ;JNC    short NO_BAD            ; CHECK FOR ERROR
;;      jc      short dma_bnd_err ; 08/02/2015
;;      and     ecx, 0FFF00000h ; 16 MB limit
;;      jz      short NO_BAD
;;dma_bnd_err:
;;      MOV     byte [DSKETTE_STATUS], DMA_BOUNDARY ; SET ERROR
;;NO_BAD:
;;      RETn                            ; CY SET BY ABOVE IF ERROR

;-------------------------------------------------------------------------------
; FMTDMA_SET
;       THIS ROUTINE SETS UP THE DMA CONTROLLER FOR A FORMAT OPERATION.
;
; ON ENTRY:     NOTHING REQUIRED
;
; ON EXIT:      @DSKETTE_STATUS, CY REFLECT STATUS OF OPERATION
;-------------------------------------------------------------------------------

FMTDMA_SET:
;; 20/02/2015 modification
        mov     edx, [ebp+4]            ; Buffer address
        test    edx, 0FFF00000h                 ; 16 MB limit
        jnz     short dma_bnd_err_stc
        ;
        push    dx                      ; *
        mov     DL, 4                   ; SECTORS/TRACK VALUE IN PARM TABLE
        call    GET_PARM                ; "
        mov     al, ah                  ; AL = SECTORS/TRACK VALUE
        sub     ah, ah                  ; AX = SECTORS/TRACK VALUE
        shl     ax, 2                   ; AX = SEC/TRK * 4 (OFFSET C,H,R,N)
        dec     ax                      ; -1 FOR DMA VALUE
        mov     cx, ax
        pop     dx                      ; *
        add     dx, cx                  ; check for overflow
        jc      short dma_bnd_err
```

```
		;
		sub	dx, cx			; Restore start address
		;
		MOV	AL, 04AH		; WILL WRITE TO THE DISKETTE
		CLI				; DISABLE INTERRUPTS DURING DMA SET-UP
		OUT	DMA+12,AL		; SET THE FIRST/LA5T F/F
		IODELAY				; WAIT FOR I/O
		OUT	DMA+11,AL		; OUTPUT THE MODE BYTE
		mov	eax, edx		; Buffer address
		OUT	DMA+4,AL		; OUTPUT LOW ADDRESS
		IODELAY				; WAIT FOR I/O
		MOV	AL,AH
		OUT	DMA+4,AL		; OUTPUT HIGH ADDRESS
		shr	eax, 16
		IODELAY				; I/O WAIT STATE
		OUT	081H,AL			; OUTPUT highest BITS TO PAGE REGISTER
		IODELAY
		mov	ax, cx			; Byte count - 1
		OUT	DMA+5,AL		; LOW BYTE OF COUNT
		IODELAY				; WAIT FOR I/O
		MOV	AL, AH
		OUT	DMA+5,AL		; HIGH BYTE OF COUNT
		IODELAY
		STI				; RE-ENABLE INTERRUPTS
		MOV	AL, 2			; MODE FOR 8237
		OUT	DMA+10, AL		; INITIALIZE THE DISKETTE CHANNEL
		retn

;; 08/02/2015 - Protected Mode Modification
;;		MOV	AL, 04AH		; WILL WRITE TO THE DISKETTE
;;		CLI				; DISABLE INTERRUPTS DURING DMA SET-UP
;;		OUT	DMA+12,AL		; SET THE FIRST/LA5T F/F
;;		;JMP	$+2			; WAIT FOR I/O
;;		IODELAY
;;		OUT	DMA+11,AL		; OUTPUT THE MODE BYTE
;;		;MOV	AX,ES			; GET THE ES VALUE
;;		;ROL	AX,4			; ROTATE LEFT
;;		;MOV	CH,AL			; GET HIGHEST NIBBLE OF ES TO CH
;;		;AND	AL,11110000B		; ZERO THE LOW NIBBLE FROM SEGMENT
;;		;ADD	AX,[BP+2]		; TEST FOR CARRY FROM ADDITION
;;		;JNC	short J33A
;;		;INC	CH			; CARRY MEANS HIGH 4 BITS MUST BE INC
;;		mov	eax, [ebp+4] ; 08/02/2015
;;;J33A:
;;		PUSH	eAX ; 08/02/2015	; SAVE START ADDRESS
;;		OUT	DMA+4,AL		; OUTPUT LOW ADDRESS
;;		;JMP	$+2			; WAIT FOR I/O
;;		IODELAY
;;		MOV	AL,AH
;;		OUT	DMA+4,AL		; OUTPUT HIGH ADDRESS
;;		shr	eax, 16 ; 08/02/2015
;;		;MOV	AL,CH			; GET HIGH 4 BITS
;;		;JMP	$+2			; I/O WAIT STATE
;;		IODELAY
;;		;AND	AL,00001111B
;;		OUT	081H,AL			; OUTPUT HIGH 4 BITS TO PAGE REGISTER
;;
;;;----- DETERMINE COUNT
;;		sub	eax, eax ; 08/02/2015
;;		MOV	DL, 4			; SECTORS/TRACK VALUE IN PARM TABLE
;;		CALL	GET_PARM		; "
;;		XCHG	AL, AH			; AL = SECTORS/TRACK VALUE
;;		SUB	AH, AH			; AX = SECTORS/TRACK VALUE
;;		SHL	AX, 2			; AX = SEC/TRK * 4 (OFFSET C,H,R,N)
;;		DEC	AX			; -1 FOR DMA VALUE
;;		PUSH	eAX	; 08/02/2015	; SAVE # OF BYTES TO BE TRANSFERED
;;		OUT	DMA+5,AL		; LOW BYTE OF COUNT
;;		;JMP	$+2			; WAIT FOR I/O
;;		IODELAY
;;		MOV	AL, AH
;;		OUT	DMA+5,AL		; HIGH BYTE OF COUNT
;;		STI				; RE-ENABLE INTERRUPTS
;;		POP	eCX	; 08/02/2015	; RECOVER COUNT VALUE
;;		POP	eAX	; 08/02/2015	; RECOVER ADDRESS VALUE
;;		;ADD	AX, CX			; ADD, TEST FOR 64K OVERFLOW
;;		add	ecx, eax ; 08/02/2015
;;		MOV	AL, 2			; MODE FOR 8237
;;		;JMP	$+2			; WAIT FOR I/O
;;		SIODELAY
```

```
;;      OUT     DMA+10, AL              ; INITIALIZE THE DISKETTE CHANNEL
;;      ;JNC    short FMTDMA_OK                ; CHECK FOR ERROR
;;      jc      short fmtdma_bnd_err ; 08/02/2015
;;      and     ecx, 0FFF00000h  ; 16 MB limit
;;      jz      short FMTDMA_OK
;;      stc     ; 20/02/2015
;;fmtdma_bnd_err:
;;      MOV     byte [DSKETTE_STATUS], DMA_BOUNDARY ; SET ERROR
;;FMTDMA_OK:
;;      RETn                            ; CY SET BY ABOVE IF ERROR

;-------------------------------------------------------------------------------
; NEC_INIT
;       THIS ROUTINE SEEKS TO THE REQUESTED TRACK AND INITIALIZES
;       THE NEC FOR THE READ/WRITE/VERIFY/FORMAT OPERATION.
;
; ON ENTRY:     AH = NEC COMMAND TO BE PERFORMED
;
; ON EXIT:      @DSKETTE_STATUS, CY REFLECT STATUS OF OPERATION
;-------------------------------------------------------------------------------
NEC_INIT:
        PUSH    AX                      ; SAVE NEC COMMAND
        CALL    MOTOR_ON                ; TURN MOTOR ON FOR SPECIFIC DRIVE

;-----  DO THE SEEK OPERATION

        MOV     CH,[eBP+1]              ; CH = TRACK #
        CALL    SEEK                    ; MOVE TO CORRECT TRACK
        POP     AX                      ; RECOVER COMMAND
        JC      short ER_1              ; ERROR ON SEEK
        MOV     eBX, ER_1               ; LOAD ERROR ADDRESS
        PUSH    eBX                     ; PUSH NEC_OUT ERROR RETURN

;-----  SEND OUT THE PARAMETERS TO THE CONTROLLER

        CALL    NEC_OUTPUT              ; OUTPUT THE OPERATION COMMAND
        MOV     AX,SI                   ; AH = HEAD #
        MOV     eBX,eDI                 ; BL = DRIVE #
        SAL     AH,2                    ; MOVE IT TO BIT 2
        AND     AH,00000100B            ; ISOLATE THAT BIT
        OR      AH,BL                   ; OR IN THE DRIVE NUMBER
        CALL    NEC_OUTPUT              ; FALL THRU CY SET IF ERROR
        POP     eBX                     ; THROW AWAY ERROR RETURN
ER_1:
        RETn

;-------------------------------------------------------------------------------
; RWV_COM
;       THIS ROUTINE SENDS PARAMETERS TO THE NEC SPECIFIC TO THE
;       READ/WRITE/VERIFY OPERATIONS.
;
; ON ENTRY:     CS:BX = ADDRESS OF MEDIA/DRIVE PARAMETER TABLE
; ON EXIT:      @DSKETTE_STATUS, CY REFLECT STATUS OF OPERATION
;-------------------------------------------------------------------------------
RWV_COM:
        MOV     eAX, ER_2               ; LOAD ERROR ADDRESS
        PUSH    eAX                     ; PUSH NEC_OUT ERROR RETURN
        MOV     AH,[eBP+1]              ; OUTPUT TRACK #
        CALL    NEC_OUTPUT
        MOV     AX,SI                   ; OUTPUT HEAD #
        CALL    NEC_OUTPUT
         MOV     AH,[eBP]                  ; OUTPUT SECTOR #
        CALL    NEC_OUTPUT
        MOV     DL,3                    ; BYTES/SECTOR PARAMETER FROM BLOCK
        CALL    GET_PARM                ; ... TO THE NEC
        CALL    NEC_OUTPUT              ; OUTPUT TO CONTROLLER
        MOV     DL,4                    ; EOT PARAMETER FROM BLOCK
        CALL    GET_PARM                ; ... TO THE NEC
        CALL    NEC_OUTPUT              ; OUTPUT TO CONTROLLER
         MOV     AH, [eBX+MD.GAP]          ; GET GAP LENGTH
_R15:
        CALL    NEC_OUTPUT
        MOV     DL,6                    ; DTL PARAMETER PROM BLOCK
        CALL    GET_PARM                ;  TO THE NEC
        CALL    NEC_OUTPUT              ; OUTPUT TO CONTROLLER
        POP     eAX                     ; THROW AWAY ERROR EXIT
ER_2:
        RETn
```

```
;-------------------------------------------------------------------------------
; NEC_TERM
;       THIS ROUTINE WAITS FOR THE OPERATION THEN ACCEPTS THE STATUS
;       FROM THE NEC FOR THE READ/WRITE/VERIFY/FORWAT OPERATION.
;
; ON EXIT:     @DSKETTE_STATUS, CY REFLECT STATUS OF OPERATION
;-------------------------------------------------------------------------------
NEC_TERM:

;----- LET THE OPERATION HAPPEN

        PUSH    eSI                     ; SAVE HEAD #, # OF SECTORS
        CALL    WAIT_INT                ; WAIT FOR THE INTERRUPT
        PUSHF
        CALL    RESULTS                 ; GET THE NEC STATUS
        JC      short SET_END_POP
        POPF
        JC      short SET_END           ; LOOK FOR ERROR

;----- CHECK THE RESULTS RETURNED BY THE CONTROLLER

        CLD                             ; SET THE CORRECT DIRECTION
        MOV     eSI, NEC_STATUS                 ; POINT TO STATUS FIELD
        lodsb                           ; GET ST0
        AND     AL,11000000B            ; TEST FOR NORMAL TERMINATION
        JZ      short SET_END
        CMP     AL,01000000B            ; TEST FOR ABNORMAL TERMINATION
        JNZ     short J18               ; NOT ABNORMAL, BAD NEC

;----- ABNORMAL TERMINATION, FIND OUT WHY

        lodsb                           ; GET ST1
        SAL     AL,1                    ; TEST FOR EDT FOUND
        MOV     AH,RECORD_NOT_FND
        JC      short J19
        SAL     AL,2
        MOV     AH,BAD_CRC
        JC      short J19
        SAL     AL,1                    ; TEST FOR DMA OVERRUN
        MOV     AH,BAD_DMA
        JC      short J19
        SAL     AL,2                    ; TEST FOR RECORD NOT FOUND
        MOV     AH,RECORD_NOT_FND
        JC      short J19
        SAL     AL,1
        MOV     AH,WRITE_PROTECT        ; TEST FOR WRITE_PROTECT
        JC      short J19
        SAL     AL,1                    ; TEST MISSING ADDRESS MARK
        MOV     AH,BAD_ADDR_MARK
        JC      short J19

;----- NEC MUST HAVE FAILED
J18:
        MOV     AH,BAD_NEC
J19:
        OR      [DSKETTE_STATUS], AH
SET_END:
        CMP     byte [DSKETTE_STATUS], 1 ; SET ERROR CONDITION
        CMC
        POP     eSI
        RETn                            ; RESTORE HEAD #, # OF SECTORS

SET_END_POP:
        POPF
        JMP     SHORT SET_END

;-------------------------------------------------------------------------------
; DSTATE:      ESTABLISH STATE UPON SUCCESSFUL OPERATION.
;-------------------------------------------------------------------------------
DSTATE:
        CMP     byte [DSKETTE_STATUS],0         ; CHECK FOR ERROR
        JNZ     short SETBAC                ; IF ERROR JUMP
        OR      byte [DSK_STATE+eDI],MED_DET ; NO ERROR, MARK MEDIA AS DETERMINED
        TEST    byte [DSK_STATE+eDI],DRV_DET ; DRIVE DETERMINED ?
        JNZ     short SETBAC            ; IF DETERMINED NO TRY TO DETERMINE
        MOV     AL,[DSK_STATE+eDI]      ; LOAD STATE
        AND     AL,RATE_MSK             ; KEEP ONLY RATE
        CMP     AL,RATE_250             ; RATE 250 ?
        JNE     short M_12              ; NO, MUST BE 1.2M OR 1.44M DRIVE
```

```
;-----  CHECK IF IT IS 1.44M

        CALL    CMOS_TYPE               ; RETURN DRIVE TYPE IN (AL)
        ;;20/02/2015
        ;;JC    short M_12              ; CMOS BAD
        jz      short M_12 ;; 20/02/2015
        CMP     AL, 4                   ; 1.44MB DRIVE ?
        JE      short M_12              ; YES
M_720:
        AND     byte [DSK_STATE+eDI], ~FMT_CAPA ; TURN OFF FORMAT CAPABILITY
        OR      byte [DSK_STATE+eDI],DRV_DET  ; MARK DRIVE DETERMINED
        JMP     SHORT SETBAC            ; BACK
M_12:
        OR      byte [DSK_STATE+eDI],DRV_DET+FMT_CAPA
                                        ; TURN ON DETERMINED & FMT CAPA
SETBAC:
        RETn

;-------------------------------------------------------------------------------
; RETRY
;       DETERMINES WHETHER A RETRY IS NECESSARY.
;       IF RETRY IS REQUIRED THEN STATE INFORMATION IS UPDATED FOR RETRY.
;
; ON EXIT:      CY = 1 FOR RETRY, CY = 0 FOR NO RETRY
;-------------------------------------------------------------------------------
RETRY:
        CMP     byte [DSKETTE_STATUS],0         ; GET STATUS OF OPERATION
        JZ      short NO_RETRY          ; SUCCESSFUL OPERATION
        CMP     byte [DSKETTE_STATUS],TIME_OUT ; IF TIME OUT NO RETRY
        JZ      short NO_RETRY
        MOV     AH,[DSK_STATE+eDI]      ; GET MEDIA STATE OF DRIVE
        TEST    AH,MED_DET              ; ESTABLISHED/DETERMINED ?
        JNZ     short NO_RETRY          ; IF ESTABLISHED STATE THEN TRUE ERROR
        AND     AH,RATE_MSK             ; ISOLATE RATE
        MOV     CH,[LASTRATE]           ; GET START OPERATION STATE
        ROL     CH,4                    ; TO CORRESPONDING BITS
        AND     CH,RATE_MSK             ; ISOLATE RATE BITS
        CMP     CH,AH                   ; ALL RATES TRIED
        JE      short NO_RETRY          ; IF YES, THEN TRUE ERROR

;       SETUP STATE INDICATOR FOR RETRY ATTEMPT TO NEXT RATE
;        00000000B (500) -> 10000000B (250)
;        10000000B (250) -> 01000000B (300)
;        01000000B (300) -> 00000000B (500)

        CMP     AH,RATE_500+1           ; SET CY FOR RATE 500
        RCR     AH,1                    ; TO NEXT STATE
        AND     AH,RATE_MSK             ; KEEP ONLY RATE BITS
        AND     byte [DSK_STATE+eDI], ~(RATE_MSK+DBL_STEP)
                                        ; RATE, DBL STEP OFF
        OR      [DSK_STATE+eDI],AH      ; TURN ON NEW RATE
        MOV     byte [DSKETTE_STATUS],0         ; RESET STATUS FOR RETRY
        STC                             ; SET CARRY FOR RETRY
        RETn                            ; RETRY RETURN

NO_RETRY:
        CLC                             ; CLEAR CARRY NO RETRY
        RETn                            ; NO RETRY RETURN

;-------------------------------------------------------------------------------
; NUM_TRANS
;       THIS ROUTINE CALCULATES THE NUMBER OF SECTORS THAT WERE
;       ACTUALLY TRANSFERRED TO/FROM THE DISKETTE.
;
; ON ENTRY:     [BP+1] = TRACK
;               SI-HI  = HEAD
;               [BP]   = START SECTOR
;
; ON EXIT:      AL = NUMBER ACTUALLY TRANSFERRED
;-------------------------------------------------------------------------------
NUM_TRANS:
        XOR     AL,AL                   ; CLEAR FOR ERROR
        CMP     byte [DSKETTE_STATUS],0         ; CHECK FOR ERROR
        JNZ     NT_OUT                  ; IF ERROR 0 TRANSFERRED
        MOV     DL,4                    ; SECTORS/TRACK OFFSET TO DL
        CALL    GET_PARM                ; AH = SECTORS/TRACK
        MOV     BL, [NEC_STATUS+5]      ; GET ENDING SECTOR
        MOV     CX,SI                   ; CH = HEAD # STARTED
```

```
        CMP     CH, [NEC_STATUS+4]     ; GET HEAD ENDED UP ON
        JNZ     DIF_HD                 ; IF ON SAME HEAD, THEN NO ADJUST
        MOV     CH, [NEC_STATUS+3]     ; GET TRACK ENDED UP ON
        CMP     CH,[eBP+1]             ; IS IT ASKED FOR TRACK
        JZ      short SAME_TRK         ; IF SAME TRACK NO INCREASE
        ADD     BL,AH                  ; ADD SECTORS/TRACK
DIF_HD:
        ADD     BL,AH                  ; ADD SECTORS/TRACK
SAME_TRK:
        SUB     BL,[eBP]               ; SUBTRACT START FROM END
        MOV     AL,BL                  ; TO AL
NT_OUT:
        RETn

;-------------------------------------------------------------------------------
; SETUP_END
;       RESTORES @MOTOR_COUNT TO PARAMETER PROVIDED IN TABLE
;       AND LOADS @DSKETTE_STATUS TO AH, AND SETS CY.
;
; ON EXIT:
;       AH, @DSKETTE_STATUS, CY REFLECT STATUS OF OPERATION
;-------------------------------------------------------------------------------
SETUP_END:
        MOV     DL,2                   ; GET THE MOTOR WAIT PARAMETER
        PUSH    AX                     ; SAVE NUMBER TRANSFERRED
        CALL    GET_PARM
        MOV     [MOTOR_COUNT],AH       ; STORE UPON RETURN
        POP     AX                     ; RESTORE NUMBER TRANSFERRED
        MOV     AH, [DSKETTE_STATUS]   ; GET STATUS OF OPERATION
        OR      AH,AH                  ; CHECK FOR ERROR
        JZ      short NUN_ERR                  ; NO ERROR
        XOR     AL,AL                  ; CLEAR NUMBER RETURNED
NUN_ERR:
        CMP     AH,1                   ; SET THE CARRY FLAG TO INDICATE
        CMC                            ; SUCCESS OR FAILURE
        RETn

;-------------------------------------------------------------------------------
; SETUP_DBL
;       CHECK DOUBLE STEP.
;
; ON ENTRY :    DI = DRIVE
;
; ON EXIT :     CY = 1 MEANS ERROR
;-------------------------------------------------------------------------------
SETUP_DBL:
        MOV     AH, [DSK_STATE+eDI]    ; ACCESS STATE
        TEST    AH,MED_DET             ; ESTABLISHED STATE ?
        JNZ     short NO_DBL                   ; IF ESTABLISHED THEN DOUBLE DONE

;-----  CHECK FOR TRACK 0 TO SPEED UP ACKNOWLEDGE OF UNFORMATTED DISKETTE

        MOV     byte [SEEK_STATUS],0   ; SET RECALIBRATE REQUIRED ON ALL DRIVES
        CALL    MOTOR_ON               ; ENSURE MOTOR STAY ON
        MOV     CH,0                   ; LOAD TRACK 0
        CALL    SEEK                   ; SEEK TO TRACK 0
        CALL    READ_ID                ; READ ID FUNCTION
        JC      short SD_ERR           ; IF ERROR NO TRACK 0

;-----  INITIALIZE START AND MAX TRACKS (TIMES 2 FOR BOTH HEADS)

        MOV     CX,0450H               ; START, MAX TRACKS
        TEST    byte [DSK_STATE+eDI],TRK_CAPA ; TEST FOR 80 TRACK CAPABILITY
        JZ      short CNT_OK           ; IF NOT COUNT IS SETUP
        MOV     CL,0A0H                ; MAXIMUM TRACK 1.2 MB

;       ATTEMPT READ ID OF ALL TRACKS, ALL HEADS UNTIL SUCCESS; UPON SUCCESS,
;       MUST SEE IF ASKED FOR TRACK IN SINGLE STEP MODE = TRACK ID READ; IF NOT
;       THEN SET DOUBLE STEP ON.
CNT_OK:
         MOV     byte [MOTOR_COUNT], 0FFH ; ENSURE MOTOR STAYS ON FOR OPERATION
        PUSH    CX                     ; SAVE TRACK, COUNT
        MOV     byte [DSKETTE_STATUS],0        ; CLEAR STATUS, EXPECT ERRORS
        XOR     AX,AX                  ; CLEAR AX
        SHR     CH,1                   ; HALVE TRACK, CY = HEAD
        RCL     AL,3                   ; AX = HEAD IN CORRECT BIT
        PUSH    AX                     ; SAVE HEAD
        CALL    SEEK                   ; SEEK TO TRACK
        POP     AX                     ; RESTORE HEAD
```

```
        OR      DI,AX                   ; DI = HEAD OR'ED DRIVE
        CALL    READ_ID                 ; READ ID HEAD 0
        PUSHF                           ; SAVE RETURN FROM READ_ID
        AND     DI,11111011B            ; TURN OFF HEAD 1 BIT
        POPF                            ; RESTORE ERROR RETURN
        POP     CX                      ; RESTORE COUNT
        JNC     short DO_CHK            ; IF OK, ASKED = RETURNED TRACK ?
        INC     CH                      ; INC FOR NEXT TRACK
        CMP     CH,CL                   ; REACHED MAXIMUM YET
        JNZ     short CNT_OK            ; CONTINUE TILL ALL TRIED

;----- FALL THRU, READ ID FAILED FOR ALL TRACKS

SD_ERR:
        STC                             ; SET CARRY FOR ERROR
        RETn                            ; SETUP_DBL ERROR EXIT

DO_CHK:
        MOV     CL, [NEC_STATUS+3]      ; LOAD RETURNED TRACK
        MOV     [DSK_TRK+eDI], CL       ; STORE TRACK NUMBER
        SHR     CH,1                    ; HALVE TRACK
        CMP     CH,CL                   ; IS IT THE SAME AS ASKED FOR TRACK
        JZ      short NO_DBL            ; IF SAME THEN NO DOUBLE STEP
        OR      byte [DSK_STATE+eDI],DBL_STEP ; TURN ON DOUBLE STEP REQUIRED
NO_DBL:
        CLC                             ; CLEAR ERROR FLAG
        RETn

;-------------------------------------------------------------------------------
; READ_ID
;       READ ID FUNCTION.
;
; ON ENTRY:     DI : BIT 2 = HEAD; BITS 1,0 = DRIVE
;
; ON EXIT:      DI : BIT 2 IS RESET, BITS 1,0 = DRIVE
;               @DSKETTE_STATUS, CY REFLECT STATUS OF OPERATION
;-------------------------------------------------------------------------------
READ_ID:
        MOV     eAX, ER_3               ; MOVE NEC OUTPUT ERROR ADDRESS
        PUSH    eAX
        MOV     AH,4AH                  ; READ ID COMMAND
        CALL    NEC_OUTPUT              ; TO CONTROLLER
        MOV     AX,DI                   ; DRIVE # TO AH, HEAD 0
        MOV     AH,AL
        CALL    NEC_OUTPUT              ; TO CONTROLLER
        CALL    NEC_TERM                ; WAIT FOR OPERATION, GET STATUS
        POP     eAX                     ; THROW AWAY ERROR ADDRESS
ER_3:
        RETn

;-------------------------------------------------------------------------------
; CMOS_TYPE
;       RETURNS DISKETTE TYPE FROM CMOS
;
; ON ENTRY:     DI = DRIVE #
;
; ON EXIT:      AL = TYPE; CY REFLECTS STATUS
;-------------------------------------------------------------------------------

CMOS_TYPE: ; 11/12/2014
mov     al, [eDI+fd0_type]
and     al, al ; 18/12/2014
retn

;CMOS_TYPE:
;       MOV     AL, CMOS_DIAG           ; CMOS DIAGNOSTIC STATUS BYTE ADDRESS
;       CALL    CMOS_READ               ; GET CMOS STATUS
;       TEST    AL,BAD_BAT+BAD_CKSUM    ; BATTERY GOOD AND CHECKSUM VALID
;       STC                             ; SET CY = 1 INDICATING ERROR FOR RETURN
;       JNZ     short BAD_CM            ; ERROR IF EITHER BIT ON
;       MOV     AL,CMOS_DISKETTE        ; ADDRESS OF DISKETTE BYTE IN CMOS
;       CALL    CMOS_READ               ; GET DISKETTE BYTE
;       OR      DI,DI                   ; SEE WHICH DRIVE IN QUESTION
;       JNZ     short TB                ; IF DRIVE 1, DATA IN LOW NIBBLE
;       ROR     AL,4                    ; EXCHANGE NIBBLES IF SECOND DRIVE
;TB:
;       AND     AL,0FH                  ; KEEP ONLY DRIVE DATA, RESET CY, 0
;BAD_CM:
;       RETn                            ; CY, STATUS OF READ
```

```
;-----------------------------------------------------------------------------
; GET_PARM
;       THIS ROUTINE FETCHES THE INDEXED POINTER FROM THE DISK_BASE
;       BLOCK POINTED TO BY THE DATA VARIABLE @DISK_POINTER. A BYTE FROM
;       THAT TABLE IS THEN MOVED INTO AH, THE INDEX OF THAT BYTE BEING
;       THE PARAMETER IN DL.
;
; ON ENTRY:     DL = INDEX OF BYTE TO BE FETCHED
;
; ON EXIT:      AH = THAT BYTE FROM BLOCK
;               AL,DH DESTROYED
;-----------------------------------------------------------------------------
GET_PARM:
        ;PUSH   DS
        PUSH    eSI
        ;SUB    AX,AX                   ; DS = 0, BIOS DATA AREA
        ;MOV    DS,AX
        ;;mov   ax, cs
        ;;mov   ds, ax
        ; 08/02/2015 (protected mode modifications, bx -> ebx)
        XCHG    eDX,eBX                 ; BL = INDEX
        ;SUB    BH,BH                   ; BX = INDEX
        and     ebx, 0FFh
        ;LDS    SI, [DISK_POINTER]      ; POINT TO BLOCK
        ;
        ; 17/12/2014
        mov     ax, [cfd] ; current (AL) and previous fd (AH)
        cmp     al, ah
        je      short gpndc
        mov     [pfd], al ; current drive -> previous drive
        push    ebx ; 08/02/2015
        mov     bl, al
        ; 11/12/2014
        mov     al, [eBX+fd0_type]      ; Drive type (0,1,2,3,4)
        ; 18/12/2014
        and     al, al
        jnz     short gpdtc
        mov     ebx, MD_TBL6            ; 1.44 MB param. tbl. (default)
         jmp     short gpdpu
gpdtc:
        call    DR_TYPE_CHECK
        ; cf = 1 -> eBX points to 1.44MB fd parameter table (default)
gpdpu:
        mov     [DISK_POINTER], ebx
        pop     ebx
gpndc:
        mov     esi, [DISK_POINTER] ; 08/02/2015, si -> esi
        MOV     AH, [eSI+eBX]           ; GET THE WORD
        XCHG    eDX,eBX                 ; RESTORE BX
        POP     eSI
        ;POP    DS
        RETn

;-----------------------------------------------------------------------------
; MOTOR_ON
;       TURN MOTOR ON AND WAIT FOR MOTOR START UP TIME. THE @MOTOR_COUNT
;       IS REPLACED WITH A SUFFICIENTLY HIGH NUMBER (0FFH) TO ENSURE
;       THAT THE MOTOR DOES NOT GO OFF DURING THE OPERATION. IF THE
;       MOTOR NEEDED TO BE TURNED ON, THE MULTI-TASKING HOOK FUNCTION
;       (AX=90FDH, INT 15) IS CALLED TELLING THE OPERATING SYSTEM
;       THAT THE BIOS IS ABOUT TO WAIT FOR MOTOR START UP. IF THIS
;       FUNCTION RETURNS WITH CY = 1, IT MEANS THAT THE MINIMUM WAIT
;       HAS BEEN COMPLETED. AT THIS POINT A CHECK IS MADE TO ENSURE
;       THAT THE MOTOR WASN'T TURNED OFF BY THE TIMER. IF THE HOOK DID
;       NOT WAIT, THE WAIT FUNCTION (AH=086H) IS CALLED TO WAIT THE
;       PRESCRIBED AMOUNT OF TIME. IF THE CARRY FLAG IS SET ON RETURN,
;       IT MEANS THAT THE FUNCTION IS IN USE AND DID NOT PERFORM THE
;       WAIT. A TIMER 1 WAIT LOOP WILL THEN DO THE WAIT.
;
; ON ENTRY:     DI = DRIVE #
; ON EXIT:      AX,CX,DX DESTROYED
;-----------------------------------------------------------------------------
MOTOR_ON:
        PUSH    eBX                     ; SAVE REG.
        CALL    TURN_ON                 ; TURN ON MOTOR
        JC      short MOT_IS_ON                 ; IF CY=1 NO WAIT
        CALL    XLAT_OLD                ; TRANSLATE STATE TO COMPATIBLE MODE
        CALL    XLAT_NEW                ; TRANSLATE STATE TO PRESENT ARCH,
```

```
        ;CALL   TURN_ON                 ; CHECK AGAIN IF MOTOR ON
        ;JC     MOT_IS_ON               ; IF NO WAIT MEANS IT IS ON
M_WAIT:
        MOV     DL,10                   ; GET THE MOTOR WAIT PARAMETER
        CALL    GET_PARM
        ;MOV    AL,AH                   ; AL = MOTOR WAIT PARAMETER
        ;XOR    AH,AH                   ; AX = MOTOR WAIT PARAMETER
        ;CMP    AL,8                    ; SEE IF AT LEAST A SECOND IS SPECIFIED
        cmp     ah, 8
        ;JAE    short GP2               ; IF YES, CONTINUE
        ja      short J13
        ;MOV    AL,8                    ; ONE SECOND WAIT FOR MOTOR START UP
        mov     ah, 8

;-----  AS CONTAINS NUMBER OF 1/8 SECONDS (125000 MICROSECONDS) TO WAIT
GP2:
;-----  FOLLOWING LOOPS REQUIRED WHEN RTC WAIT FUNCTION IS ALREADY IN USE
J13:                                    ; WAIT FOR 1/8 SECOND PER (AL)
        MOV     eCX,8286                ; COUNT FOR 1/8 SECOND AT 15.085737 US
        CALL    WAITF                   ; GO TO FIXED WAIT ROUTINE
        ;DEC    AL                      ; DECREMENT TIME VALUE
        dec     ah
        JNZ     short J13               ; ARE WE DONE YET
MOT_IS_ON:
        POP     eBX                     ; RESTORE REG.
        RETn

;-------------------------------------------------------------------------------
; TURN_ON
;       TURN MOTOR ON AND RETURN WAIT STATE.
;
; ON ENTRY:     DI = DRIVE #
;
; ON EXIT:      CY = 0 MEANS WAIT REQUIRED
;               CY = 1 MEANS NO WAIT REQUIRED
;               AX,BX,CX,DX DESTROYED
;-------------------------------------------------------------------------------
TURN_ON:
        MOV     eBX,eDI                 ; BX = DRIVE #
        MOV     CL,BL                   ; CL = DRIVE #
        ROL     BL,4                    ; BL = DRIVE SELECT
        CLI                             ; NO INTERRUPTS WHILE DETERMINING STATUS
        MOV     byte [MOTOR_COUNT],0FFH       ; ENSURE MOTOR STAYS ON FOR OPERATION
        MOV     AL, [MOTOR_STATUS]      ; GET DIGITAL OUTPUT REGISTER REFLECTION
        AND     AL,00110000B            ; KEEP ONLY DRIVE SELECT BITS
        MOV     AH,1                    ; MASK FOR DETERMINING MOTOR BIT
        SHL     AH,CL                   ; AH = MOTOR ON, A=00000001, B=00000010

;  AL = DRIVE SELECT FROM @MOTOR_STATUS
;  BL = DRIVE SELECT DESIRED
;  AH = MOTOR ON MASK DESIRED

        CMP     AL,BL                   ; REQUESTED DRIVE ALREADY SELECTED ?
        JNZ     short TURN_IT_ON        ; IF NOT SELECTED JUMP
        TEST    AH, [MOTOR_STATUS]      ; TEST MOTOR ON BIT
        JNZ     short NO_MOT_WAIT       ; JUMP IF MOTOR ON AND SELECTED

TURN_IT_ON:
        OR      AH,BL                   ; AH = DRIVE SELECT AND MOTOR ON
        MOV     BH,[MOTOR_STATUS]       ; SAVE COPY OF @MOTOR_STATUS BEFORE
        AND     BH,00001111B            ; KEEP ONLY MOTOR BITS
        AND     byte [MOTOR_STATUS],11001111B ; CLEAR OUT DRIVE SELECT
        OR      [MOTOR_STATUS],AH       ; OR IN DRIVE SELECTED AND MOTOR ON
        MOV     AL,[MOTOR_STATUS]       ; GET DIGITAL OUTPUT REGISTER REFLECTION
        MOV     BL,AL                   ; BL=@MOTOR_STATUS AFTER, BH=BEFORE
        AND     BL,00001111B            ; KEEP ONLY MOTOR BITS
        STI                             ; ENABLE INTERRUPTS AGAIN
        AND     AL,00111111B            ; STRIP AWAY UNWANTED BITS
        ROL     AL,4                    ; PUT BITS IN DESIRED POSITIONS
        OR      AL,00001100B            ; NO RESET, ENABLE DMA/INTERRUPT
        MOV     DX,03F2H                ; SELECT DRIVE AND TURN ON MOTOR
        OUT     DX,AL
        CMP     BL,BH                   ; NEW MOTOR TURNED ON ?
        ;JZ     short NO_MOT_WAIT       ; NO WAIT REQUIRED IF JUST SELECT
        je      short no_mot_w1 ; 27/02/2015
        CLC                             ; (re)SET CARRY MEANING WAIT
        RETn
```

```
NO_MOT_WAIT:
        sti
no_mot_w1: ; 27/02/2015
        STC                             ; SET NO WAIT REQUIRED
        ;STI                            ; INTERRUPTS BACK ON
        RETn

;-------------------------------------------------------------------------------
; HD_WAIT
;       WAIT FOR HEAD SETTLE TIME.
;
; ON ENTRY:     DI = DRIVE #
;
; ON EXIT:      AX,BX,CX,DX DESTROYED
;-------------------------------------------------------------------------------
HD_WAIT:
        MOV     DL,9                    ; GET HEAD SETTLE PARAMETER
        CALL    GET_PARM
        or      ah, ah  ; 17/12/2014
        jnz     short DO_WAT
         TEST     byte [MOTOR_STATUS],10000000B ; SEE IF A WRITE OPERATION
        ;JZ     short ISNT_WRITE        ; IF NOT, DO NOT ENFORCE ANY VALUES
        ;OR     AH,AH                   ; CHECK FOR ANY WAIT?
        ;JNZ    short DO_WAT            ; IF THERE DO NOT ENFORCE
        jz      short HW_DONE
        MOV     AH,HD12_SETTLE          ; LOAD 1.2M HEAD SETTLE MINIMUM
        MOV     AL,[DSK_STATE+eDI]      ; LOAD STATE
        AND     AL,RATE_MSK             ; KEEP ONLY RATE
        CMP     AL,RATE_250             ; 1.2 M DRIVE ?
        JNZ     short DO_WAT            ; DEFAULT HEAD SETTLE LOADED
;GP3:
        MOV     AH,HD320_SETTLE                 ; USE 320/360 HEAD SETTLE
;       JMP     SHORT DO_WAT

;ISNT_WRITE:
;       OR      AH,AH                   ; CHECK FOR NO WAIT
;       JZ      short HW_DONE           ; IF NOT WRITE AND 0 ITS OK

;----- AH CONTAINS NUMBER OF MILLISECONDS TO WAIT
DO_WAT:
;       MOV     AL,AH                   ; AL = # MILLISECONDS
;       ;XOR    AH,AH                   ; AX = # MILLISECONDS
J29:                                    ;       1 MILLISECOND LOOP
        ;mov    cx, WAIT_FDU_HEAD_SETTLE ; 33 ; 1 ms in 30 micro units.
        MOV     eCX,66                  ; COUNT AT 15.085737 US PER COUNT
        CALL    WAITF                   ; DELAY FOR 1 MILLISECOND
        ;DEC    AL                      ; DECREMENT THE COUNT
        dec     ah
        JNZ     short J29               ; DO AL MILLISECOND # OF TIMES
HW_DONE:
        RETn
```

```
;-------------------------------------------------------------------------------
; NEC_OUTPUT
;       THIS ROUTINE SENDS A BYTE TO THE NEC CONTROLLER AFTER TESTING
;       FOR CORRECT DIRECTION AND CONTROLLER READY THIS ROUTINE WILL
;       TIME OUT IF THE BYTE IS NOT ACCEPTED WITHIN A REASONABLE AMOUNT
;       OF TIME, SETTING THE DISKETTE STATUS ON COMPLETION.
;
; ON ENTRY:     AH = BYTE TO BE OUTPUT
;
; ON EXIT:      CY = 0  SUCCESS
;               CY = 1  FAILURE -- DISKETTE STATUS UPDATED
;                       IF A FAILURE HAS OCCURRED, THE RETURN IS MADE ONE LEVEL
;                       HIGHER THAN THE CALLER OF NEC OUTPUT. THIS REMOVES THE
;                       REQUIREMENT OF TESTING AFTER EVERY CALL OF NEC_OUTPUT.
;               AX,CX,DX DESTROYED
;-------------------------------------------------------------------------------

; 09/12/2014 [Erdogan Tan]
;       (from 'PS2 Hardware Interface Tech. Ref. May 88', Page 09-05.)
; Diskette Drive Controller Status Register (3F4h)
;       This read only register facilitates the transfer of data between
;       the system microprocessor and the controller.
; Bit 7 - When set to 1, the Data register is ready to transfer data
;         with the system microprocessor.
; Bit 6 - The direction of data transfer. If this bit is set to 0,
;         the transfer is to the controller.
; Bit 5 - When this bit is set to 1, the controller is in the non-DMA mode.
; Bit 4 - When this bit is set to 1, a Read or Write command is being executed.
; Bit 3 - Reserved.
; Bit 2 - Reserved.
; Bit 1 - When this bit is set to 1, dskette drive 1 is in the seek mode.
; Bit 0 - When this bit is set to 1, dskette drive 1 is in the seek mode.

; Data Register (3F5h)
; This read/write register passes data, commands and parameters, and provides
; diskette status information.

NEC_OUTPUT:
        ;PUSH   BX                      ; SAVE REG.
        MOV     DX,03F4H                ; STATUS PORT
        ;MOV    BL,2                    ; HIGH ORDER COUNTER
        ;XOR    CX,CX                   ; COUNT FOR TIME OUT
        ; 16/12/2014
        ; waiting for (max.) 0.5 seconds
        ;;mov     byte [wait_count], 0 ;; 27/02/2015
        ;
        ; 17/12/2014
        ; Modified from AWARD BIOS 1999 - ADISK.ASM - SEND_COMMAND
        ;
        ;WAIT_FOR_PORT:        Waits for a bit at a port pointed to by DX to
        ;               go on.
        ;INPUT:
        ;       AH=Mask for isolation bits.
        ;       AL=pattern to look for.
        ;       DX=Port to test for
        ;       BH:CX=Number of memory refresh periods to delay.
        ;            (normally 30 microseconds per period.)
        ;
        ;WFP_SHORT:
        ;       Wait for port if refresh cycle is short (15-80 Us range).
        ;

;       mov     bl, WAIT_FDU_SEND_HI+1; 0+1
;       mov     cx, WAIT_FDU_SEND_LO  ; 16667
        mov     ecx, WAIT_FDU_SEND_LH   ; 16667 (27/02/2015)
;
;WFPS_OUTER_LP:
;       ;
;WFPS_CHECK_PORT:
J23:
        IN      AL,DX                   ; GET STATUS
        AND     AL,11000000B            ; KEEP STATUS AND DIRECTION
        CMP     AL,10000000B            ; STATUS 1 AND DIRECTION 0 ?
        JZ      short J27               ; STATUS AND DIRECTION OK
WFPS_HI:
        IN      AL, PORT_B     ;061h    ; SYS1  ; wait for hi to lo
        TEST    AL,010H                 ; transition on memory
        JNZ     SHORT WFPS_HI           ; refresh.
```

```
WFPS_LO:
        IN      AL, PORT_B            ; SYS1
        TEST    AL,010H
        JZ      SHORT WFPS_LO
        ;LOOP   SHORT WFPS_CHECK_PORT
        loop    J23     ; 27/02/2015
;       ;
;       dec     bl
;       jnz     short WFPS_OUTER_LP
;       jmp     short WFPS_TIMEOUT    ; fail
;J23:
;       IN      AL,DX                 ; GET STATUS
;       AND     AL,11000000B          ; KEEP STATUS AND DIRECTION
;       CMP     AL,10000000B          ; STATUS 1 AND DIRECTION 0 ?
;       JZ      short J27             ; STATUS AND DIRECTION OK
        ;LOOP   J23                   ; CONTINUE TILL CX EXHAUSTED
        ;DEC    BL                    ; DECREMENT COUNTER
        ;JNZ    short J23             ; REPEAT TILL DELAY FINISHED, CX = 0

        ;;27/02/2015
        ;16/12/2014
         ;;cmp     byte [wait_count], 10   ; (10/18.2 seconds)
        ;;jb    short J23

;WFPS_TIMEOUT:

;-----  FALL THRU TO ERROR RETURN

        OR      byte [DSKETTE_STATUS],TIME_OUT
        ;POP    BX                    ; RESTORE REG.
        POP     eAX ; 08/02/2015      ; DISCARD THE RETURN ADDRESS
        STC                           ; INDICATE ERROR TO CALLER
        RETn

;-----  DIRECTION AND STATUS OK; OUTPUT BYTE

J27:
        MOV     AL,AH                 ; GET BYTE TO OUTPUT
        INC     DX                    ; DATA PORT = STATUS PORT + 1
        OUT     DX,AL                 ; OUTPUT THE BYTE
        ;;NEWIODELAY  ;; 27/02/2015
        ; 27/02/2015
        PUSHF                         ; SAVE FLAGS
        MOV     eCX, 3                ; 30 TO 45 MICROSECONDS WAIT FOR
        CALL    WAITF                 ; NEC FLAGS UPDATE CYCLE
        POPF                          ; RESTORE FLAGS FOR EXIT
        ;POP    BX                    ; RESTORE REG
        RETn                          ; CY = 0 FROM TEST INSTRUCTION

;-------------------------------------------------------------------------------
; SEEK
;       THIS ROUTINE WILL MOVE THE HEAD ON THE NAMED DRIVE TO THE NAMED
;       TRACK. IF THE DRIVE HAS NOT BEEN ACCESSED SINCE THE DRIVE
;       RESET COMMAND WAS ISSUED, THE DRIVE WILL BE RECALIBRATED.
;
; ON ENTRY:     DI = DRIVE #
;               CH = TRACK #
;
; ON EXIT:      @DSKETTE_STATUS, CY REFLECT STATUS OF OPERATION.
;               AX,BX,CX DX DESTROYED
;-------------------------------------------------------------------------------
SEEK:
        MOV     eBX,eDI               ; BX = DRIVE #
        MOV     AL,1                  ; ESTABLISH MASK FOR RECALIBRATE TEST
        XCHG    CL,BL                 ; SET DRIVE VALULE INTO CL
        ROL     AL,CL                 ; SHIFT MASK BY THE DRIVE VALUE
        XCHG    CL,BL                 ; RECOVER TRACK VALUE
        TEST    AL,[SEEK_STATUS]      ; TEST FOR RECALIBRATE REQUIRED
        JNZ     short J28A            ; JUMP IF RECALIBRATE NOT REQUIRED

        OR      [SEEK_STATUS],AL      ; TURN ON THE NO RECALIBRATE BIT IN FLAG
        CALL    RECAL                 ; RECALIBRATE DRIVE
        JNC     short AFT_RECAL               ; RECALIBRATE DONE

;-----  ISSUE RECALIBRATE FOR 80 TRACK DISKETTES

        MOV     byte [DSKETTE_STATUS],0       ; CLEAR OUT INVALID STATUS
        CALL    RECAL                 ; RECALIBRATE DRIVE
        JC      short RB              ; IF RECALIBRATE FAILS TWICE THEN ERROR
```

```
AFT_RECAL:
        MOV     byte [DSK_TRK+eDI],0    ; SAVE NEW CYLINDER AS PRESENT POSITION
       OR      CH,CH                 ; CHECK FOR SEEK TO TRACK 0
       JZ      short DO_WAIT         ; HEAD SETTLE, CY = 0 IF JUMP

;-----  DRIVE IS IN SYNCHRONIZATION WITH CONTROLLER, SEEK TO TRACK

J28A:  TEST    byte [DSK_STATE+eDI],DBL_STEP ; CHECK FOR DOUBLE STEP REQUIRED
       JZ      short _R7             ; SINGLE STEP REQUIRED BYPASS DOUBLE
       SHL     CH,1                  ; DOUBLE NUMBER OF STEP TO TAKE

_R7:   CMP     CH, [DSK_TRK+eDI]     ; SEE IF ALREADY AT THE DESIRED TRACK
       JE      short RB              ; IF YES, DO NOT NEED TO SEEK

       MOV     eDX, NEC_ERR          ; LOAD RETURN ADDRESS
       PUSH    eDX ; (*)             ; ON STACK FOR NEC OUTPUT ERROR
       MOV     [DSK_TRK+eDI],CH      ; SAVE NEW CYLINDER AS PRESENT POSITION
       MOV     AH,0FH                ; SEEK COMMAND TO NEC
       CALL    NEC_OUTPUT
       MOV     eBX,eDI               ; BX = DRIVE #
       MOV     AH,BL                 ; OUTPUT DRIVE NUMBER
       CALL    NEC_OUTPUT
       MOV     AH, [DSK_TRK+eDI]     ; GET CYLINDER NUMBER
       CALL    NEC_OUTPUT
       CALL    CHK_STAT_2            ; ENDING INTERRUPT AND SENSE STATUS

;-----  WAIT FOR HEAD SETTLE

DO_WAIT:
       PUSHF                         ; SAVE STATUS
       CALL    HD_WAIT               ; WAIT FOR HEAD SETTLE TIME
       POPF                          ; RESTORE STATUS
RB:
NEC_ERR:
       ; 08/02/2015 (code trick here from original IBM PC/AT DISKETTE.ASM)
       ; (*) nec_err -> retn (push edx -> pop edx) -> nec_err -> retn
       RETn                          ; RETURN TO CALLER

;-------------------------------------------------------------------------------
; RECAL
;       RECALIBRATE DRIVE
;
; ON ENTRY:    DI = DRIVE #
;
; ON EXIT:     CY REFLECTS STATUS OF OPERATION.
;-------------------------------------------------------------------------------
RECAL:
       PUSH    CX
       MOV     eAX, RC_BACK          ; LOAD NEC_OUTPUT ERROR
       PUSH    eAX
       MOV     AH,07H                ; RECALIBRATE COMMAND
       CALL    NEC_OUTPUT
       MOV     eBX,eDI               ; BX = DRIVE #
       MOV     AH,BL
       CALL    NEC_OUTPUT            ; OUTPUT THE DRIVE NUMBER
       CALL    CHK_STAT_2            ; GET THE INTERRUPT AND SENSE INT STATUS
       POP     eAX                   ; THROW AWAY ERROR
RC_BACK:
       POP     CX
       RETn

;-------------------------------------------------------------------------------
; CHK_STAT_2
;       THIS ROUTINE HANDLES THE INTERRUPT RECEIVED AFTER RECALIBRATE,
;       OR SEEK TO THE ADAPTER. THE INTERRUPT IS WAITED FOR, THE
;       INTERRUPT STATUS SENSED, AND THE RESULT RETURNED TO THE CALLER.
;
; ON EXIT:     @DSKETTE_STATUS, CY REFLECT STATUS OF OPERATION.
;-------------------------------------------------------------------------------
CHK_STAT_2:
        MOV     eAX, CS_BACK            ; LOAD NEC_OUTPUT ERROR ADDRESS
       PUSH    eAX
       CALL    WAIT_INT              ; WAIT FOR THE INTERRUPT
       JC      short J34             ; IF ERROR, RETURN IT
       MOV     AH,08H                ; SENSE INTERRUPT STATUS COMMAND
       CALL    NEC_OUTPUT
       CALL    RESULTS               ; READ IN THE RESULTS
       JC      short J34
```

```
        MOV     AL,[NEC_STATUS]                 ; GET THE FIRST STATUS BYTE
        AND     AL,01100000B            ; ISOLATE THE BITS
        CMP     AL,01100000B            ; TEST FOR CORRECT VALUE
        JZ      short J35               ; IF ERROR, GO MARK IT
        CLC                             ; GOOD RETURN
J34:
        POP     eAX                     ; THROW AWAY ERROR RETURN
CS_BACK:
        RETn
J35:
        OR      byte [DSKETTE_STATUS], BAD_SEEK
        STC                             ; ERROR RETURN CODE
        JMP     SHORT J34

;-------------------------------------------------------------------------------
; WAIT_INT
;       THIS ROUTINE WAITS FOR AN INTERRUPT TO OCCUR A TIME OUT ROUTINE
;       TAKES PLACE DURING THE WAIT, SO THAT AN ERROR MAY BE RETURNED
;       IF THE DRIVE IS NOT READY.
;
; ON EXIT:      @DSKETTE_STATUS, CY REFLECT STATUS OF OPERATION.
;-------------------------------------------------------------------------------

; 17/12/2014
; 2.5 seconds waiting !
;(AWARD BIOS - 1999, WAIT_FDU_INT_LOW, WAIT_FDU_INT_HI)
; amount of time to wait for completion interrupt from NEC.


WAIT_INT:
        STI                             ; TURN ON INTERRUPTS, JUST IN CASE
        CLC                             ; CLEAR TIMEOUT INDICATOR
       ;MOV     BL,10                   ; CLEAR THE COUNTERS
       ;XOR     CX,CX                   ; FOR 2 SECOND WAIT

        ; Modification from AWARD BIOS - 1999 (ATORGS.ASM, WAIT
        ;
        ;WAIT_FOR_MEM:
        ;       Waits for a bit at a specified memory location pointed
        ;       to by ES:[DI] to become set.
        ;INPUT:
        ;       AH=Mask to test with.
        ;       ES:[DI] = memory location to watch.
        ;       BH:CX=Number of memory refresh periods to delay.
        ;            (normally 30 microseconds per period.)

        ; waiting for (max.) 2.5 secs in 30 micro units.
;       mov     cx, WAIT_FDU_INT_LO             ; 017798
;;      mov     bl, WAIT_FDU_INT_HI
;       mov     bl, WAIT_FDU_INT_HI + 1
        ; 27/02/2015
        mov     ecx, WAIT_FDU_INT_LH    ; 83334 (2.5 seconds)
WFMS_CHECK_MEM:
        test    byte [SEEK_STATUS],INT_FLAG ; TEST FOR INTERRUPT OCCURRING
         jnz     short J37
WFMS_HI:
        IN      AL,PORT_B  ; 061h       ; SYS1, wait for lo to hi
        TEST    AL,010H                 ; transition on memory
        JNZ     SHORT WFMS_HI           ; refresh.
WFMS_LO:
        IN      AL,PORT_B               ;SYS1
        TEST    AL,010H
        JZ      SHORT WFMS_LO
         LOOP    WFMS_CHECK_MEM
;WFMS_OUTER_LP:
;;      or      bl, bl                  ; check outer counter
;;      jz      short J36A              ; WFMS_TIMEOUT
;       dec     bl
;       jz      short J36A
;       jmp     short WFMS_CHECK_MEM

        ;17/12/2014
        ;16/12/2014
;         mov     byte [wait_count], 0    ; Reset (INT 08H) counter
;J36:
;       TEST    byte [SEEK_STATUS],INT_FLAG ; TEST FOR INTERRUPT OCCURRING
;       JNZ     short J37
        ;16/12/2014
        ;LOOP   J36                     ; COUNT DOWN WHILE WAITING
```

```
        ;DEC    BL                      ; SECOND LEVEL COUNTER
        ;JNZ    short J36
;        cmp     byte [wait_count], 46   ; (46/18.2 seconds)
;       jb      short J36

;WFMS_TIMEOUT:
;J36A:
        OR      byte [DSKETTE_STATUS], TIME_OUT ; NOTHING HAPPENED
        STC                             ; ERROR RETURN
J37:
        PUSHF                           ; SAVE CURRENT CARRY
        AND     byte [SEEK_STATUS], ~INT_FLAG ; TURN OFF INTERRUPT FLAG
        POPF                            ; RECOVER CARRY
        RETn                            ; GOOD RETURN CODE

;-------------------------------------------------------------------------------
; RESULTS
;       THIS ROUTINE WILL READ ANYTHING THAT THE NEC CONTROLLER RETURNS
;       FOLLOWING AN INTERRUPT.
;
; ON EXIT:      @DSKETTE_STATUS, CY REFLECT STATUS OF OPERATION.
;               AX,BX,CX,DX DESTROYED
;-------------------------------------------------------------------------------
RESULTS:
        PUSH    eDI
        MOV     eDI, NEC_STATUS                 ; POINTER TO DATA AREA
        MOV     BL,7                    ; MAX STATUS BYTES
        MOV     DX,03F4H                ; STATUS PORT

;----- WAIT FOR REQUEST FOR MASTER

_R10:
        ; 16/12/2014
        ; wait for (max) 0.5 seconds
        ;MOV    BH,2                    ; HIGH ORDER COUNTER
        ;XOR    CX,CX                   ; COUNTER

        ;Time to wait while waiting for each byte of NEC results = .5
        ;seconds.  .5 seconds = 500,000 micros.  500,000/30 = 16,667.
        ; 27/02/2015
        mov     ecx, WAIT_FDU_RESULTS_LH ; 16667
        ;mov    cx, WAIT_FDU_RESULTS_LO  ; 16667
        ;mov    bh, WAIT_FDU_RESULTS_HI+1 ; 0+1

WFPSR_OUTER_LP:
        ;
WFPSR_CHECK_PORT:
J39:                                    ; WAIT FOR MASTER
        IN      AL,DX                   ; GET STATUS
        AND     AL,11000000B            ; KEEP ONLY STATUS AND DIRECTION
        CMP     AL,11000000B            ; STATUS 1 AND DIRECTION 1 ?
        JZ      short J42               ; STATUS AND DIRECTION OK
WFPSR_HI:
        IN      AL, PORT_B     ;061h    ; SYS1  ; wait for hi to lo
        TEST    AL,010H                 ; transition on memory
        JNZ     SHORT WFPSR_HI          ; refresh.
WFPSR_LO:
        IN      AL, PORT_B              ; SYS1
        TEST    AL,010H
        JZ      SHORT WFPSR_LO
         LOOP    WFPSR_CHECK_PORT
        ;; 27/02/2015
        ;;dec   bh
        ;;jnz   short WFPSR_OUTER_LP
        ;jmp    short WFPSR_TIMEOUT     ; fail

        ;;mov   byte [wait_count], 0
;J39:                                   ; WAIT FOR MASTER
;       IN      AL,DX                   ; GET STATUS
;       AND     AL,11000000B            ; KEEP ONLY STATUS AND DIRECTION
;       CMP     AL,11000000B            ; STATUS 1 AND DIRECTION 1 ?
;       JZ      short J42               ; STATUS AND DIRECTION OK
        ;LOOP   J39                     ; LOOP TILL TIMEOUT
        ;DEC    BH                      ; DECREMENT HIGH ORDER COUNTER
        ;JNZ    short J39               ; REPEAT TILL DELAY DONE
        ;
        ;;cmp   byte [wait_count], 10  ; (10/18.2 seconds)
        ;;jb    short J39
```

```
;WFPSR_TIMEOUT:
        OR      byte [DSKETTE_STATUS],TIME_OUT
        STC                             ; SET ERROR RETURN
        JMP     SHORT POPRES            ; POP REGISTERS AND RETURN

;-----  READ IN THE STATUS

J42:
        JMP     $+2                     ; I/O DELAY
        INC     DX                      ; POINT AT DATA PORT
        IN      AL,DX                   ; GET THE DATA
        ; 16/12/2014
        NEWIODELAY
         MOV      [eDI],AL                 ; STORE THE BYTE
        INC     eDI                     ; INCREMENT THE POINTER
        ; 16/12/2014
;       push    cx
;       mov     cx, 30
;wdw2:
;       NEWIODELAY
;       loop    wdw2
;       pop     cx

        MOV     eCX,3                   ; MINIMUM 24 MICROSECONDS FOR NEC
        CALL    WAITF                   ; WAIT 30 TO 45 MICROSECONDS
        DEC     DX                      ; POINT AT STATUS PORT
        IN      AL,DX                   ; GET STATUS
        ; 16/12/2014
        NEWIODELAY
        ;
        TEST    AL,00010000B            ; TEST FOR NEC STILL BUSY
        JZ      short POPRES            ; RESULTS DONE ?

        DEC     BL                      ; DECREMENT THE STATUS COUNTER
         JNZ     short _R10                ; GO BACK FOR MORE
        OR      byte [DSKETTE_STATUS],BAD_NEC ; TOO MANY STATUS BYTES
        STC                             ; SET ERROR FLAG

;-----  RESULT OPERATION IS DONE
POPRES:
        POP     eDI
        RETn                            ; RETURN WITH CARRY SET

;-------------------------------------------------------------------------------
; READ_DSKCHNG
;       READS THE STATE OF THE DISK CHANGE LINE.
;
; ON ENTRY:     DI = DRIVE #
;
; ON EXIT:      DI = DRIVE #
;               ZF = 0 : DISK CHANGE LINE INACTIVE
;               ZF = 1 : DISK CHANGE LINE ACTIVE
;               AX,CX,DX DESTROYED
;-------------------------------------------------------------------------------
READ_DSKCHNG:
        CALL    MOTOR_ON                ; TURN ON THE MOTOR IF OFF
        MOV     DX,03F7H                ; ADDRESS DIGITAL INPUT REGISTER
        IN      AL,DX                   ; INPUT DIGITAL INPUT REGISTER
        TEST    AL,DSK_CHG              ; CHECK FOR DISK CHANGE LINE ACTIVE
        RETn                            ; RETURN TO CALLER WITH ZERO FLAG SET
```

```
;-------------------------------------------------------------------------------
; DRIVE_DET
;       DETERMINES WHETHER DRIVE IS 80 OR 40 TRACKS AND
;       UPDATES STATE INFORMATION ACCORDINGLY.
; ON ENTRY:     DI = DRIVE #
;-------------------------------------------------------------------------------
DRIVE_DET:
        CALL    MOTOR_ON                ; TURN ON MOTOR IF NOT ALREADY ON
        CALL    RECAL                   ; RECALIBRATE DRIVE
        JC      short DD_BAC            ; ASSUME NO DRIVE PRESENT
        MOV     CH,TRK_SLAP             ; SEEK TO TRACK 48
        CALL    SEEK
        JC      short DD_BAC            ; ERROR NO DRIVE
        MOV     CH,QUIET_SEEK+1                 ; SEEK TO TRACK 10
SK_GIN:
        DEC     CH                      ; DECREMENT TO NEXT TRACK
        PUSH    CX                      ; SAVE TRACK
        CALL    SEEK
        JC      short POP_BAC           ; POP AND RETURN
        MOV     eAX, POP_BAC            ; LOAD NEC OUTPUT ERROR ADDRESS
        PUSH    eAX
        MOV     AH,SENSE_DRV_ST                 ; SENSE DRIVE STATUS COMMAND BYTE
        CALL    NEC_OUTPUT              ; OUTPUT TO NEC
        MOV     AX,DI                   ; AL = DRIVE
        MOV     AH,AL                   ; AH = DRIVE
        CALL    NEC_OUTPUT              ; OUTPUT TO NEC
        CALL    RESULTS                 ; GO GET STATUS
        POP     eAX                     ; THROW AWAY ERROR ADDRESS
        POP     CX                      ; RESTORE TRACK
        TEST    byte [NEC_STATUS], HOME         ; TRACK 0 ?
        JZ      short SK_GIN            ; GO TILL TRACK 0
        OR      CH,CH                   ; IS HOME AT TRACK 0
        JZ      short IS_80             ; MUST BE 80 TRACK DRIVE

;       DRIVE IS A 360; SET DRIVE TO DETERMINED;
;       SET MEDIA TO DETERMINED AT RATE 250.

        OR      byte [DSK_STATE+eDI], DRV_DET+MED_DET+RATE_250
        RETn                            ; ALL INFORMATION SET
IS_80:
        OR      byte [DSK_STATE+eDI], TRK_CAPA ; SETUP 80 TRACK CAPABILITY
DD_BAC:
        RETn
POP_BAC:
        POP     CX                      ; THROW AWAY
        RETn

fdc_int:
          ; 30/07/2015
          ; 16/02/2015
;int_0Eh: ; 11/12/2014

;--- HARDWARE INT 0EH -- ( IRQ LEVEL  6 ) --------------------------------------
; DISK_INT
;       THIS ROUTINE HANDLES THE DISKETTE INTERRUPT.
;
; ON EXIT:      THE INTERRUPT FLAG IS SET IN @SEEK_STATUS.
;-------------------------------------------------------------------------------
DISK_INT_1:

        PUSH    AX                      ; SAVE WORK REGISTER
        push    ds
        mov     ax, KDATA
        mov     ds, ax
         OR      byte [SEEK_STATUS], INT_FLAG ; TURN ON INTERRUPT OCCURRED
        MOV      AL,EOI                    ; END OF INTERRUPT MARKER
        OUT     INTA00,AL               ; INTERRUPT CONTROL PORT
        pop     ds
        POP     AX                      ; RECOVER REGISTER
        IRET                            ; RETURN FROM INTERRUPT
```

```
;-------------------------------------------------------------------------------
; DSKETTE_SETUP
;       THIS ROUTINE DOES A PRELIMINARY CHECK TO SEE WHAT TYPE OF
;       DISKETTE DRIVES ARE ATTACH TO THE SYSTEM.
;-------------------------------------------------------------------------------
DSKETTE_SETUP:
        ;PUSH   AX                      ; SAVE REGISTERS
        ;PUSH   BX
        ;PUSH   CX
        PUSH    eDX
        ;PUSH   DI
        ;;PUSH  DS
        ; 14/12/2014
        ;mov    word [DISK_POINTER], MD_TBL6
        ;mov    [DISK_POINTER+2], cs
        ;
        ;OR     byte [RTC_WAIT_FLAG], 1        ; NO RTC WAIT, FORCE USE OF LOOP
        XOR     eDI,eDI                 ; INITIALIZE DRIVE POINTER
        MOV     WORD [DSK_STATE],0      ; INITIALIZE STATES
        AND     byte [LASTRATE],~(STRT_MSK+SEND_MSK) ; CLEAR START & SEND
        OR      byte [LASTRATE],SEND_MSK ; INITIALIZE SENT TO IMPOSSIBLE
        MOV     byte [SEEK_STATUS],0    ; INDICATE RECALIBRATE NEEDED
        MOV     byte [MOTOR_COUNT],0    ; INITIALIZE MOTOR COUNT
        MOV     byte [MOTOR_STATUS],0 ; INITIALIZE DRIVES TO OFF STATE
        MOV     byte [DSKETTE_STATUS],0        ; NO ERRORS
        ;
        ; 28/02/2015
        ;mov    word [cfd], 100h
        call    DSK_RESET
        pop     edx
        retn

;SUP0:
;       CALL    DRIVE_DET               ; DETERMINE DRIVE
;       CALL    XLAT_OLD                ; TRANSLATE STATE TO COMPATIBLE MODE
;       ; 02/01/2015
;       ;INC    DI                      ; POINT TO NEXT DRIVE
;       ;CMP    DI,MAX_DRV              ; SEE IF DONE
;       ;JNZ    short SUP0              ; REPEAT FOR EACH ORIVE
;        cmp     byte [fd1_type], 0
;       jna     short sup1
;       or      di, di
;       jnz     short sup1
;       inc     di
;        jmp     short SUP0
;sup1:
;       MOV     byte [SEEK_STATUS],0    ; FORCE RECALIBRATE
;       ;AND    byte [RTC_WAIT_FLAG],0FEH ; ALLOW FOR RTC WAIT
;       CALL    SETUP_END               ; VARIOUS CLEANUPS
;       ;;POP   DS                      ; RESTORE CALLERS REGISTERS
;       ;POP    DI
;       POP     eDX
;       ;POP    CX
;       ;POP    BX
;       ;POP    AX
;       RETn

;//////////////////////////////////////////////////////
;; END OF DISKETTE I/O ;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;
;
```

```
int13h: ; 21/02/2015
        pushfd
        push    cs
        call    DISK_IO
        retn

;;;;;; DISK I/O ;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;; 21/02/2015 ;;;
;///////////////////////////////////////////////////////////////////////

; DISK I/O - Erdogan Tan (Retro UNIX 386 v1 project)
; 23/02/2015
; 21/02/2015 (unix386.s)
; 22/12/2014 - 14/02/2015 (dsectrm2.s)
;
; Original Source Code:
; DISK ----- 09/25/85 FIXED DISK BIOS
; (IBM PC XT Model 286 System BIOS Source Code, 04-21-86)
;
; Modifications: by reference of AWARD BIOS 1999 (D1A0622)
;               Source Code - ATORGS.ASM, AHDSK.ASM
;


;The wait for controller to be not busy is 10 seconds.
;10,000,000 / 30 = 333,333.  333,333 decimal = 051615h
;;WAIT_HDU_CTLR_BUSY_LO       equ     1615h
;;WAIT_HDU_CTLR_BUSY_HI       equ       05h
WAIT_HDU_CTRL_BUSY_LH equ     51615h   ;21/02/2015

;The wait for controller to issue completion interrupt is 10 seconds.
;10,000,000 / 30 = 333,333.  333,333 decimal = 051615h
;;WAIT_HDU_INT_LO     equ     1615h
;;WAIT_HDU_INT_HI     equ       05h
WAIT_HDU_INT_LH               equ     51615h ; 21/02/2015

;The wait for Data request on read and write longs is
;2000 us. (?)
;;WAIT_HDU_DRQ_LO     equ     1000    ; 03E8h
;;WAIT_HDU_DRQ_HI     equ     0
WAIT_HDU_DRQ_LH               equ     1000    ; 21/02/2015

; Port 61h (PORT_B)
SYS1            equ     61h     ; PORT_B  (diskette.inc)

; 23/12/2014
%define CMD_BLOCK       eBP-8  ; 21/02/2015
```

```
;--- INT 13H -------------------------------------------------------------------
;                                                                             :
; FIXED DISK I/O INTERFACE                                                    :
;                                                                             :
;      THIS INTERFACE PROVIDES ACCESS TO 5 1/4" FIXED DISKS THROUGH           :
;      THE IBM FIXED DISK CONTROLLER.                                          :
;                                                                             :
;      THE  BIOS  ROUTINES  ARE  MEANT  TO  BE  ACCESSED  THROUGH             :
;      SOFTWARE  INTERRUPTS  ONLY.    ANY  ADDRESSES  PRESENT     IN          :
;      THESE  LISTINGS  ARE  INCLUDED        ONLY   FOR  COMPLETENESS,        :
;      NOT  FOR  REFERENCE.  APPLICATIONS   WHICH  REFERENCE  ANY             :
;      ABSOLUTE  ADDRESSES  WITHIN  THE  CODE       SEGMENTS  OF  BIOS        :
;      VIOLATE  THE  STRUCTURE  AND  DESIGN  OF  BIOS.                        :
;                                                                             :
;------------------------------------------------------------------------------:
;                                                                             :
; INPUT  (AH)= HEX COMMAND VALUE                                              :
;                                                                             :
;      (AH)= 00H  RESET DISK (DL = 80H,81H) / DISKETTE                        :
;      (AH)= 01H  READ THE STATUS OF THE LAST DISK OPERATION INTO (AL)        :
;                  NOTE: DL < 80H - DISKETTE                                  :
;                        DL > 80H - DISK                                      :
;      (AH)= 02H  READ THE DESIRED SECTORS INTO MEMORY                        :
;      (AH)= 03H  WRITE THE DESIRED SECTORS FROM MEMORY                       :
;      (AH)= 04H  VERIFY THE DESIRED SECTORS                                   :
;      (AH)= 05H  FORMAT THE DESIRED TRACK                                    :
;      (AH)= 06H  UNUSED                                                      :
;      (AH)= 07H  UNUSED                                                      :
;      (AH)= 08H  RETURN THE CURRENT DRIVE PARAMETERS                                 :
;      (AH)= 09H  INITIALIZE DRIVE PAIR CHARACTERISTICS                       :
;                  INTERRUPT 41 POINTS TO DATA BLOCK FOR DRIVE 0              :
;                  INTERRUPT 46 POINTS TO DATA BLOCK FOR DRIVE 1              :
;      (AH)= 0AH  READ LONG                                                   :
;      (AH)= 0BH  WRITE LONG  (READ & WRITE LONG ENCOMPASS 512 + 4 BYTES ECC) :
;      (AH)= 0CH  SEEK                                                        :
;      (AH)= 0DH  ALTERNATE DISK RESET (SEE DL)                               :
;      (AH)= 0EH  UNUSED                                                      :
;      (AH)= 0FH  UNUSED                                                      :
;      (AH)= 10H  TEST DRIVE READY                                            :
;      (AH)= 11H  RECALIBRATE                                                 :
;      (AH)= 12H  UNUSED                                                      :
;      (AH)= 13H  UNUSED                                                      :
;      (AH)= 14H  CONTROLLER INTERNAL DIAGNOSTIC                              :
;      (AH)= 15H  READ DASD TYPE                                              :
;                                                                             :
;-------------------------------------------------------------------------------
;                                                                             :
;      REGISTERS USED FOR FIXED DISK OPERATIONS                               :
;                                                                             :
;             (DL)    -  DRIVE NUMBER     (80H-81H FOR DISK. VALUE CHECKED):
;             (DH)    -  HEAD NUMBER     (0-15 ALLOWED, NOT VALUE CHECKED):
;             (CH)    -  CYLINDER NUMBER  (0-1023, NOT VALUE CHECKED)(SEE CL):
;             (CL)    -  SECTOR NUMBER    (1-17, NOT VALUE CHECKED)        :
;                                                                             :
;                        NOTE: HIGH 2 BITS OF CYLINDER NUMBER ARE PLACED :
;                              IN THE HIGH 2 BITS OF THE CL REGISTER          :
;                              (10 BITS TOTAL)                                :
;                                                                             :
;             (AL)    -  NUMBER OF SECTORS (MAXIMUM POSSIBLE RANGE 1-80H,:
;                                          FOR READ/WRITE LONG 1-79H)      :
;                                                                             :
;             (ES:BX) -   ADDRESS OF BUFFER FOR READS AND WRITES,             :
;                        (NOT REQUIRED FOR VERIFY)                            :
;                                                                             :
;             FORMAT (AH=5) ES:BX POINTS TO A 512 BYTE BUFFER. THE FIRST :
;                        2*(SECTORS/TRACK) BYTES CONTAIN F,N FOR EACH SECTOR.:
;                        F = 00H FOR A GOOD SECTOR                            :
;                            80H FOR A BAD SECTOR                             :
;                        N = SECTOR NUMBER                                    :
;                        FOR AN INTERLEAVE OF 2 AND 17 SECTORS/TRACK          :
;                        THE TABLE SHOULD BE:                                 :
;                                                                             :
;                DB      00H,01H,00H,0AH,00H,02H,00H,0BH,00H,03H,00H,0CH :
;                DB      00H,04H,00H,0DH,00H,05H,00H,0EH,00H,06H,00H,0FH :
;                DB      00H,07H,00H,10H,00H,08H,00H,11H,00H,09H              :
;                                                                             :
;-------------------------------------------------------------------------------
```

```
;-----------------------------------------------------------------------------
; OUTPUT                                                                    :
;       AH = STATUS OF CURRENT OPERATION                                    :
;            STATUS BITS ARE DEFINED IN THE EQUATES BELOW                   :
;       CY = 0  SUCCESSFUL OPERATION (AH=0 ON RETURN)                       :
;       CY = 1  FAILED OPERATION (AH HAS ERROR REASON)                      :
;                                                                           :
;       NOTE:  ERROR 11H  INDICATES THAT THE DATA READ HAD A RECOVERABLE   :
;              ERROR WHICH WAS CORRECTED BY THE ECC ALGORITHM.  THE DATA   :
;              IS PROBABLY GOOD,   HOWEVER THE BIOS ROUTINE INDICATES AN   :
;              ERROR TO ALLOW THE CONTROLLING PROGRAM A CHANCE TO DECIDE   :
;              FOR ITSELF.  THE  ERROR  MAY  NOT  RECUR  IF  THE DATA IS   :
;              REWRITTEN.                                                   :
;                                                                           :
;       IF DRIVE PARAMETERS WERE REQUESTED (DL >= 80H),                     :
;          INPUT:                                                           :
;            (DL) = DRIVE NUMBER                                            :
;          OUTPUT:                                                          :
;            (DL) = NUMBER OF CONSECUTIVE ACKNOWLEDGING DRIVES ATTACHED (1-2):
;                   (CONTROLLER CARD ZERO TALLY ONLY)                       :
;            (DH) = MAXIMUM USEABLE VALUE FOR HEAD NUMBER                   :
;            (CH) = MAXIMUM USEABLE VALUE FOR CYLINDER NUMBER               :
;            (CL) = MAXIMUM USEABLE VALUE FOR SECTOR NUMBER                 :
;                   AND CYLINDER NUMBER HIGH BITS                           :
;                                                                           :
;       IF READ DASD TYPE WAS REQUESTED,                                    :
;                                                                           :
;       AH = 0 - NOT PRESENT                                                :
;            1 - DISKETTE - NO CHANGE LINE AVAILABLE                        :
;            2 - DISKETTE - CHANGE LINE AVAILABLE                           :
;            3 - FIXED DISK                                                 :
;                                                                           :
;       CX,DX = NUMBER OF 512 BYTE BLOCKS WHEN AH = 3                       :
;                                                                           :
;       REGISTERS WILL BE PRESERVED EXCEPT WHEN THEY ARE USED TO RETURN     :
;       INFORMATION.                                                        :
;                                                                           :
;       NOTE: IF AN ERROR IS REPORTED BY THE DISK CODE, THE APPROPRIATE     :
;              ACTION IS TO RESET THE DISK, THEN RETRY THE OPERATION.      :
;                                                                           :
;-----------------------------------------------------------------------------

SENSE_FAIL     EQU     0FFH           ; NOT IMPLEMENTED
NO_ERR         EQU     0E0H           ; STATUS ERROR/ERROR REGISTER=0
WRITE_FAULT    EQU     0CCH           ; WRITE FAULT ON SELECTED DRIVE
UNDEF_ERR      EQU     0BBH           ; UNDEFINED ERROR OCCURRED
NOT_RDY        EQU     0AAH           ; DRIVE NOT READY
TIME_OUT       EQU     80H            ; ATTACHMENT FAILED TO RESPOND
BAD_SEEK       EQU     40H            ; SEEK OPERATION FAILED
BAD_CNTLR      EQU     20H            ; CONTROLLER HAS FAILED
DATA_CORRECTED EQU     11H            ; ECC CORRECTED DATA ERROR
BAD_ECC        EQU     10H            ; BAD ECC ON DISK READ
BAD_TRACK      EQU     0BH            ; NOT IMPLEMENTED
BAD_SECTOR     EQU     0AH            ; BAD SECTOR FLAG DETECTED
;DMA_BOUNDARY  EQU     09H            ; DATA EXTENDS TOO FAR
INIT_FAIL      EQU     07H            ; DRIVE PARAMETER ACTIVITY FAILED
BAD_RESET      EQU     05H            ; RESET FAILED
;RECORD_NOT_FND        EQU     04H            ; REQUESTED SECTOR NOT FOUND
;BAD_ADDR_MARK EQU     02H            ; ADDRESS MARK NOT FOUND
;BAD_CMD       EQU     01H            ; BAD COMMAND PASSED TO DISK I/O
```

```
;---------------------------------------------------------
;                                                    :
; FIXED DISK PARAMETER TABLE                         :
;  -  THE TABLE IS COMPOSED OF A BLOCK DEFINED AS:  :
;                                                    :
;  +0   (1 WORD) - MAXIMUM NUMBER OF CYLINDERS       :
;  +2   (1 BYTE) - MAXIMUM NUMBER OF HEADS           :
;  +3   (1 WORD) - NOT USED/SEE PC-XT                :
;  +5   (1 WORD) - STARTING WRITE PRECOMPENSATION CYL :
;  +7   (1 BYTE) - MAXIMUM ECC DATA BURST LENGTH     :
;  +8   (1 BYTE) - CONTROL BYTE                      :
;                  BIT    7 DISABLE RETRIES -OR-     :
;                  BIT    6 DISABLE RETRIES          :
;                  BIT    3 MORE THAN 8 HEADS        :
;  +9   (3 BYTES)- NOT USED/SEE PC-XT                :
; +12   (1 WORD) - LANDING ZONE                      :
; +14   (1 BYTE) - NUMBER OF SECTORS/TRACK           :
; +15   (1 BYTE) - RESERVED FOR FUTURE USE           :
;                                                    :
;        - TO DYNAMICALLY DEFINE A SET OF PARAMETERS:
;          BUILD A TABLE FOR UP TO 15 TYPES AND PLACE :
;          THE CORRESPONDING VECTOR INTO INTERRUPT 41 :
;          FOR DRIVE 0 AND INTERRUPT 46 FOR DRIVE 1.:
;                                                    :
;---------------------------------------------------------

;---------------------------------------------------------
;                                                    :
; HARDWARE SPECIFIC VALUES                           :
;                                                    :
;  -  CONTROLLER I/O PORT                            :
;                                                    :
;     > WHEN READ FROM:                              :
;       HF_PORT+0 - READ DATA (FROM CONTROLLER TO CPU):
;       HF_PORT+1 - GET ERROR REGISTER               :
;       HF_PORT+2 - GET SECTOR COUNT                 :
;       HF_PORT+3 - GET SECTOR NUMBER                :
;       HF_PORT+4 - GET CYLINDER LOW                 :
;       HF_PORT+5 - GET CYLINDER HIGH (2 BITS)       :
;       HF_PORT+6 - GET SIZE/DRIVE/HEAD              :
;       HF_PORT+7 - GET STATUS REGISTER              :
;                                                    :
;     > WHEN WRITTEN TO:                             :
;       HF_PORT+0 - WRITE DATA (FROM CPU TO CONTROLLER) :
;       HF_PORT+1 - SET PRECOMPENSATION CYLINDER     :
;       HF_PORT+2 - SET SECTOR COUNT                 :
;       HF_PORT+3 - SET SECTOR NUMBER                :
;       HF_PORT+4 - SET CYLINDER LOW                 :
;       HF_PORT+5 - SET CYLINDER HIGH (2 BITS)       :
;       HF_PORT+6 - SET SIZE/DRIVE/HEAD              :
;       HF_PORT+7 - SET COMMAND REGISTER             :
;                                                    :
;---------------------------------------------------------

;HF_PORT       EQU    01F0H  ; DISK PORT
;HF1_PORT      equ    0170h
;HF_REG_PORT   EQU    03F6H
;HF1_REG_PORT  equ    0376h

HDC1_BASEPORT  equ    1F0h
HDC2_BASEPORT  equ    170h

align 2

;-----         STATUS REGISTER

ST_ERROR       EQU    00000001B     ;
ST_INDEX       EQU    00000010B     ;
ST_CORRCTD     EQU    00000100B     ; ECC CORRECTION SUCCESSFUL
ST_DRQ         EQU    00001000B     ;
ST_SEEK_COMPL  EQU    00010000B     ; SEEK COMPLETE
ST_WRT_FLT     EQU    00100000B     ; WRITE FAULT
ST_READY       EQU    01000000B     ;
ST_BUSY        EQU    10000000B     ;
```

```
;-----          ERROR REGISTER

ERR_DAM         EQU     00000001B       ; DATA ADDRESS MARK NOT FOUND
ERR_TRK_0       EQU     00000010B       ; TRACK 0 NOT FOUND ON RECAL
ERR_ABORT       EQU     00000100B       ; ABORTED COMMAND
;               EQU     00001000B       ; NOT USED
ERR_ID          EQU     00010000B       ; ID NOT FOUND
;               EQU     00100000B       ; NOT USED
ERR_DATA_ECC    EQU     01000000B
ERR_BAD_BLOCK   EQU     10000000B


RECAL_CMD       EQU     00010000B       ; DRIVE RECAL  (10H)
READ_CMD        EQU     00100000B       ;       READ   (20H)
WRITE_CMD       EQU     00110000B       ;       WRITE  (30H)
VERIFY_CMD      EQU     01000000B       ;       VERIFY (40H)
FMTTRK_CMD      EQU     01010000B       ; FORMAT TRACK (50H)
INIT_CMD        EQU     01100000B       ;   INITIALIZE (60H)
SEEK_CMD        EQU     01110000B       ;       SEEK   (70H)
DIAG_CMD        EQU     10010000B       ; DIAGNOSTIC   (90H)
SET_PARM_CMD    EQU     10010001B       ; DRIVE PARMS  (91H)
NO_RETRIES      EQU     00000001B       ; CHD MODIFIER (01H)
ECC_MODE        EQU     00000010B       ; CMD MODIFIER (02H)
BUFFER_MODE     EQU     00001000B       ; CMD MODIFIER (08H)

;MAX_FILE       EQU     2
;S_MAX_FILE     EQU     2
MAX_FILE        equ     4               ; 22/12/2014
S_MAX_FILE      equ     4               ; 22/12/2014

DELAY_1         EQU     25H             ; DELAY FOR OPERATION COMPLETE
DELAY_2         EQU     0600H           ; DELAY FOR READY
DELAY_3         EQU     0100H           ; DELAY FOR DATA REQUEST

HF_FAIL         EQU     08H             ; CMOS FLAG IN BYTE 0EH

;-----          COMMAND BLOCK REFERENCE

;CMD_BLOCK       EQU      BP-8             ; @CMD_BLOCK REFERENCES BLOCK HEAD IN SS
                                        ;  (BP) POINTS TO COMMAND BLOCK TAIL
                                        ;       AS DEFINED BY THE "ENTER" PARMS
; 19/12/2014
ORG_VECTOR      equ     4*13h           ; INT 13h vector
DISK_VECTOR     equ     4*40h           ; INT 40h vector (for floppy disks)
;HDISK_INT      equ     4*76h           ; Primary HDC - Hardware interrupt (IRQ14)
;HDISK_INT1     equ     4*76h           ; Primary HDC - Hardware interrupt (IRQ14)
;HDISK_INT2     equ     4*77h           ; Secondary HDC - Hardware interrupt (IRQ15)
;HF_TBL_VEC     equ     4*41h           ; Pointer to 1st fixed disk parameter table
;HF1_TBL_VEC    equ     4*46h           ; Pointer to 2nd fixed disk parameter table
```

```
align 2

;-----------------------------------------------------------------
; FIXED DISK I/O SETUP                                           :
;                                                                :
;  -  ESTABLISH TRANSFER VECTORS FOR THE FIXED DISK              :
;  -  PERFORM POWER ON DIAGNOSTICS                               :
;     SHOULD AN ERROR OCCUR A "1701" MESSAGE IS DISPLAYED        :
;                                                                :
;-----------------------------------------------------------------

DISK_SETUP:
	;CLI
	;;MOV	AX,ABS0			; GET ABSOLUTE SEGMENT
	;xor	ax,ax
	;MOV	DS,AX			; SET SEGMENT REGISTER
	;MOV	AX, [ORG_VECTOR]	; GET DISKETTE VECTOR
	;MOV	[DISK_VECTOR],AX	;  INTO INT 40H
	;MOV	AX, [ORG_VECTOR+2]
	;MOV	[DISK_VECTOR+2],AX
	;MOV	word [ORG_VECTOR],DISK_IO	; FIXED DISK HANDLER
	;MOV	[ORG_VECTOR+2],CS
	; 1st controller (primary master, slave)   - IRQ 14
	;;MOV	word [HDISK_INT],HD_INT		; FIXED DISK INTERRUPT
	;mov	word [HDISK_INT1],HD_INT	;
	;;MOV	[HDISK_INT+2],CS
	;mov	[HDISK_INT1+2],CS
	; 2nd controller (secondary master, slave) - IRQ 15
	;mov	word [HDISK_INT2],HD1_INT	;
	;mov	[HDISK_INT2+2],CS
	;
	;;MOV	word [HF_TBL_VEC],HD0_DPT	; PARM TABLE DRIVE 80
	;;MOV	word [HF_TBL_VEC+2],DPT_SEGM
	;;MOV	word [HF1_TBL_VEC],HD1_DPT	; PARM TABLE DRIVE 81
	;;MOV	word [HF1_TBL_VEC+2],DPT_SEGM
	;push	cs
	;pop	ds
	;mov	word [HDPM_TBL_VEC],HD0_DPT	; PARM TABLE DRIVE 80h
	;mov	word [HDPM_TBL_VEC+2],DPT_SEGM
	mov	dword [HDPM_TBL_VEC], (DPT_SEGM*16)+HD0_DPT
	;mov	word [HDPS_TBL_VEC],HD1_DPT	; PARM TABLE DRIVE 81h
	;mov	word [HDPS_TBL_VEC+2],DPT_SEGM
	mov	dword [HDPS_TBL_VEC], (DPT_SEGM*16)+HD1_DPT
	;mov	word [HDSM_TBL_VEC],HD2_DPT	; PARM TABLE DRIVE 82h
	;mov	word [HDSM_TBL_VEC+2],DPT_SEGM
	mov	dword [HDSM_TBL_VEC], (DPT_SEGM*16)+HD2_DPT
	;mov	word [HDSS_TBL_VEC],HD3_DPT	; PARM TABLE DRIVE 83h
	;mov	word [HDSS_TBL_VEC+2],DPT_SEGM
	mov	dword [HDSS_TBL_VEC], (DPT_SEGM*16)+HD3_DPT
	;
	;;IN	AL,INTB01		; TURN ON SECOND INTERRUPT CHIP
	;;;AND	AL,0BFH
	;;and	al, 3Fh			; enable IRQ 14 and IRQ 15
	;;;JMP	$+2
	;;IODELAY
	;;OUT	INTB01,AL
	;;IODELAY
	;;IN	AL,INTA01		; LET INTERRUPTS PASS THRU TO
	;;AND	AL,0FBH			;  SECOND CHIP
	;;;JMP	$+2
	;;IODELAY
	;;OUT	INTA01,AL
	;
	;STI
	;;PUSH	DS			; MOVE ABS0 POINTER TO
	;;POP	ES			; EXTRA SEGMENT POINTER
	;;;CALL	DDS			; ESTABLISH DATA SEGMENT
	;;MOV	byte [DISK_STATUS1],0	; RESET THE STATUS INDICATOR
	;;MOV	byte [HF_NUM],0		; ZERO NUMBER OF FIXED DISKS
	;;MOV	byte [CONTROL_BYTE],0
	;;MOV	byte [PORT_OFF],0	; ZERO CARD OFFSET
	; 20/12/2014 - private code by Erdogan Tan
		     ; (out of original PC-AT, PC-XT BIOS code)
	;mov	si, hd0_type
	mov	esi, hd0_type
	;mov	cx, 4
	mov	ecx, 4
```

```
hde_l:
        lodsb
        cmp     al, 80h                 ; 8?h = existing
        jb      short _L4
        inc     byte [HF_NUM]           ; + 1 hard (fixed) disk drives
_L4: ; 26/02/2015
        loop    hde_l
;_L4:                                   ; 0 <= [HF_NUM] =< 4
        ;
        ;; 31/12/2014 - cancel controller diagnostics here
        ;;;mov  cx, 3  ; 26/12/2014 (Award BIOS 1999)
        ;;mov   cl, 3
        ;;
        ;;MOV   DL,80H                  ; CHECK THE CONTROLLER
;;hdc_dl:
        ;;MOV   AH,14H                  ; USE CONTROLLER DIAGNOSTIC COMMAND
        ;;INT   13H                     ; CALL BIOS WITH DIAGNOSTIC COMMAND
        ;;;JC   short CTL_ERRX          ; DISPLAY ERROR MESSAGE IF BAD RETURN
        ;;;jc   short POD_DONE ;22/12/2014
        ;;jnc   short hdc_reset0
        ;;loop  hdc_dl
        ;;; 27/12/2014
        ;;stc
        ;;retn
        ;
;;hdc_reset0:
        ; 18/01/2015
        mov     cl, [HF_NUM]
        and     cl, cl
        jz      short POD_DONE
        ;
        mov     dl, 7Fh
hdc_reset1:
        inc     dl
        ;; 31/12/2015
        ;;push  dx
        ;;push  cx
        ;;push  ds
        ;;sub   ax, ax
        ;;mov   ds, ax
        ;;MOV   AX, [TIMER_LOW]                 ; GET START TIMER COUNTS
        ;;pop   ds
        ;;MOV   BX,AX
        ;;ADD   AX,6*182                ; 60 SECONDS* 18.2
        ;;MOV   CX,AX
        ;;mov   word [wait_count], 0  ; 22/12/2014 (reset wait counter)
        ;;
        ;; 31/12/2014 - cancel HD_RESET_1
        ;;CALL  HD_RESET_1              ; SET UP DRIVE 0, (1,2,3)
        ;;pop   cx
        ;;pop   dx
        ;;
        ; 18/01/2015
        mov     ah, 0Dh ; ALTERNATE RESET
        ;int    13h
        call    int13h
        loop    hdc_reset1
POD_DONE:
        RETn

;;----- POD_ERROR

;;CTL_ERRX:
;       ;MOV    SI,OFFSET F1782         ; CONTROLLER ERROR
;       ;CALL   SET_FAIL                ; DO NOT IPL FROM DISK
;       ;CALL   E_MSG                   ; DISPLAY ERROR AND SET (BP) ERROR FLAG
;       ;JMP    short POD_DONE

;;HD_RESET_1:
;;      ;PUSH   BX                      ; SAVE TIMER LIMITS
;;      ;PUSH   CX
;;RES_1: MOV    AH,09H                  ; SET DRIVE PARAMETERS
;;      INT     13H
;;      JC      short RES_2
;;      MOV     AH,11H                  ; RECALIBRATE DRIVE
;;      INT     13H
;;      JNC     short RES_CK            ; DRIVE OK
;;RES_2: ;CALL POD_TCHK                 ; CHECK TIME OUT
;;      cmp     word [wait_count], 6*182 ; waiting time (in timer ticks)
```

```
;;                                      ; (30 seconds)
;;       ;cmc
;;       ;JNC    short RES_1
;;       jb      short RES_1
;;;RES_FL: ;MOV       SI,OFFSET F1781       ; INDICATE DISK 1 FAILURE;
;;       ;TEST   DL,1
;;       ;JNZ    RES_E1
;;       ;MOV    SI,OFFSET F1780        ; INDICATE DISK 0 FAILURE
;;       ;CALL   SET_FAIL               ; DO NOT TRY TO IPL DISK 0
;;       ;JMP    SHORT RES_E1
;;RES_ER: ; 22/12/2014
;;RES_OK:
;;       ;POP    CX                     ; RESTORE TIMER LIMITS
;;       ;POP    BX
;;       RETn
;;
;;RES_RS: MOV    AH,00H                 ; RESET THE DRIVE
;;       INT     13H
;;RES_CK: MOV    AH,08H                 ; GET MAX CYLINDER,HEAD,SECTOR
;;       MOV     BL,DL                  ; SAVE DRIVE CODE
;;       INT     13H
;;       JC      short RES_ER
;;       MOV     [NEC_STATUS],CX        ; SAVE MAX CYLINDER, SECTOR
;;       MOV     DL,BL                  ; RESTORE DRIVE CODE
;;RES_3: MOV     AX,0401H               ; VERIFY THE LAST SECTOR
;;       INT     13H
;;       JNC     short RES_OK           ; VERIFY OK
;;       CMP     AH,BAD_SECTOR          ; OK ALSO IF JUST ID READ
;;       JE      short RES_OK
;;       CMP     AH,DATA_CORRECTED
;;       JE      short RES_OK
;;       CMP     AH,BAD_ECC
;;       JE      short RES_OK
;;       ;CALL   POD_TCHK               ; CHECK FOR TIME OUT
;;       cmp     word [wait_count], 6*182 ; waiting time (in timer ticks)
;;                                      ; (60 seconds)
;;       cmc
;;       JC      short RES_ER           ; FAILED
;;       MOV     CX,[NEC_STATUS]        ; GET SECTOR ADDRESS, AND CYLINDER
;;       MOV     AL,CL                  ; SEPARATE OUT SECTOR NUMBER
;;       AND     AL,3FH
;;       DEC     AL                     ; TRY PREVIOUS ONE
;;       JZ      short RES_RS           ; WE'VE TRIED ALL SECTORS ON TRACK
;;       AND     CL,0C0H                ; KEEP CYLINDER BITS
;;       OR      CL,AL                  ; MERGE SECTOR WITH CYLINDER BITS
;;       MOV     [NEC_STATUS],CX        ; SAVE CYLINDER, NEW SECTOR NUMBER
;;       JMP     short RES_3            ; TRY AGAIN
;;;RES_ER: MOV SI,OFFSET F1791          ; INDICATE DISK 1 ERROR
;;       ;TEST   DL,1
;;       ;JNZ    short RES_E1
;;       ;MOV    SI,OFFSET F1790        ; INDICATE DISK 0 ERROR
;;;RES_E1:
;;       ;CALL   E_MSG                  ; DISPLAY ERROR AND SET (BP) ERROR FLAG
;;;RES_OK:
;;       ;POP    CX                     ; RESTORE TIMER LIMITS
;;       ;POP    BX
;;       ;RETn
;
;;SET_FAIL:
;        ;MOV    AX,X*(CMOS_DIAG+NMI)   ; GET CMOS ERROR BYTE
;        ;CALL   CMOS_READ
;        ;OR     AL,HF_FAIL             ; SET DO NOT IPL FROM DISK FLAG
;        ;XCHG   AH,AL                  ; SAVE IT
;        ;CALL   CMOS_WRITE             ; PUT IT OUT
;        ;RETn
;
;;POD_TCHK:                             ; CHECK FOR 30 SECOND TIME OUT
;        ;POP    AX                     ; SAVE RETURN
;        ;POP    CX                     ; GET TIME OUT LIMITS
;        ;POP    BX
;        ;PUSH   BX                     ; AND SAVE THEM AGAIN
;        ;PUSH   CX
;        ;PUSH   AX
;        ;push   ds
;        ;xor    ax, ax
;        ;mov    ds, ax                 ; RESTORE RETURN
;        ;MOV    AX, [TIMER_LOW]               ; AX = CURRENT TIME
;        ;                              ; BX = START TIME
;        ;                              ; CX = END TIME
```

```
;       ;pop    ds
;       ;CMP    BX,CX
;       ;JB     short TCHK1            ; START < END
;       ;CMP    BX,AX
;       ;JB     short TCHKG            ; END < START < CURRENT
;       ;JMP    SHORT TCHK2            ; END, CURRENT < START
;;TCHK1: CMP   AX,BX
;;      JB      short TCHKNG           ; CURRENT < START < END
;;TCHK2: CMP   AX,CX
;;      JB      short TCHKG            ; START < CURRENT < END
;;                                     ; OR CURRENT < END < START
;;TCHKNG: STC                          ; CARRY SET INDICATES TIME OUT
;;      RETn
;;TCHKG: CLC                           ; INDICATE STILL TIME
;;      RETn
;;
;;int_13h:

;----------------------------------------
;       FIXED DISK BIOS ENTRY POINT   :
;----------------------------------------

DISK_IO:
        CMP     DL,80H                 ; TEST FOR FIXED DISK DRIVE
        ;JAE    short A1               ; YES, HANDLE HERE
        ;;;INT  40H                    ; DISKETTE HANDLER
        ;;call  int40h
        jb      DISKETTE_IO_1
;RET_2:
        ;RETf   2                      ; BACK TO CALLER
;       retf    4
A1:
        STI                            ; ENABLE INTERRUPTS
        ;; 04/01/2015
        ;;OR    AH,AH
        ;;JNZ   short A2
        ;;INT   40H                    ; RESET NEC WHEN AH=0
        ;;SUB   AH,AH
        CMP     DL,(80H + S_MAX_FILE - 1)
        JA      short RET_2
        ; 18/01/2015
        or      ah,ah
        jz      short A4
        cmp     ah, 0Dh ; Alternate reset
        jne     short A2
        sub     ah,ah   ; Reset
        jmp     short A4
A2:
        CMP     AH,08H                 ; GET PARAMETERS IS A SPECIAL CASE
        ;JNZ    short A3
         ;JMP    GET_PARM_N
        je      GET_PARM_N
A3:     CMP     AH,15H                 ; READ DASD TYPE IS ALSO
        ;JNZ    short A4
         ;JMP    READ_DASD_TYPE
         je      READ_DASD_TYPE
        ; 02/02/2015
        cmp     ah, 1Dh                ;(Temporary for Retro UNIX 386 v1)
        ; 12/01/2015
        cmc
        jnc     short A4
        ; 30/01/2015
        ;mov     byte [CS:DISK_STATUS1],BAD_CMD  ; COMMAND ERROR
         mov     byte [DISK_STATUS1], BAD_CMD
        ;jmp    short RET_2
RET_2:
        retf    4
A4:                                    ; SAVE REGISTERS DURING OPERATION
        ENTER   8,0                    ; SAVE (BP) AND MAKE ROOM FOR @CMD_BLOCK
        PUSH    eBX                    ;  IN THE STACK, THE COMMAND BLOCK IS:
        PUSH    eCX                    ;   @CMD_BLOCK == BYTE PTR [BP]-8
        PUSH    eDX
        PUSH    DS
        PUSH    ES
        PUSH    eSI
        PUSH    eDI
        ;;04/01/2015
        ;;OR    AH,AH                  ; CHECK FOR RESET
        ;;JNZ   short A5
```

```
        ;;MOV   DL,80H                  ; FORCE DRIVE 80 FOR RESET
;;A5:
        ;push   cs
        ;pop    ds
        ; 21/02/2015
        push    ax
        mov     ax, KDATA
        mov     ds, ax
        mov     es, ax
        pop     ax
        CALL    DISK_IO_CONT            ; PERFORM THE OPERATION
        ;;CALL  DDS                     ; ESTABLISH SEGMENT
        MOV     AH,[DISK_STATUS1]       ; GET STATUS FROM OPERATION
        CMP     AH,1                    ; SET THE CARRY FLAG TO INDICATE
        CMC                             ; SUCCESS OR FAILURE
        POP     eDI                     ; RESTORE REGISTERS
        POP     eSI
         POP     ES
         POP     DS
        POP     eDX
        POP     eCX
        POP     eBX
        LEAVE                           ; ADJUST (SP) AND RESTORE (BP)
        ;RETf   2                       ; THROW AWAY SAVED FLAGS
        retf    4
; 21/02/2015
;        dw --> dd
M1:                                     ; FUNCTION TRANSFER TABLE
        dd      DISK_RESET              ; 000H
        dd      RETURN_STATUS           ; 001H
        dd      DISK_READ               ; 002H
        dd      DISK_WRITE              ; 003H
        dd      DISK_VERF               ; 004H
        dd      FMT_TRK                 ; 005H
        dd      BAD_COMMAND             ; 006H  FORMAT BAD SECTORS
        dd      BAD_COMMAND             ; 007H  FORMAT DRIVE
        dd      BAD_COMMAND             ; 008H  RETURN PARAMETERS
        dd      INIT_DRV                ; 009H
        dd      RD_LONG                 ; 00AH
        dd      WR_LONG                 ; 00BH
        dd      DISK_SEEK               ; 00CH
        dd      DISK_RESET              ; 00DH
        dd      BAD_COMMAND             ; 00EH  READ BUFFER
        dd      BAD_COMMAND             ; 00FH  WRITE BUFFER
        dd      TST_RDY                 ; 010H
        dd      HDISK_RECAL             ; 011H
        dd      BAD_COMMAND             ; 012H  MEMORY DIAGNOSTIC
        dd      BAD_COMMAND             ; 013H  DRIVE DIAGNOSTIC
        dd      CTLR_DIAGNOSTIC         ; 014H  CONTROLLER DIAGNOSTIC
        ; 02/02/2015 (Temporary - Retro UNIX 386 v1 - DISK I/O test)
        dd      BAD_COMMAND             ; 015h
        dd      BAD_COMMAND             ; 016h
        dd      BAD_COMMAND             ; 017h
        dd      BAD_COMMAND             ; 018h
        dd      BAD_COMMAND             ; 019h
        dd      BAD_COMMAND             ; 01Ah
        dd      DISK_READ               ; 01Bh ; LBA read
        dd      DISK_WRITE              ; 01Ch ; LBA write
M1L      EQU     $-M1

DISK_IO_CONT:
        ;;CALL  DDS                     ; ESTABLISH SEGMENT
        CMP     AH,01H                  ; RETURN STATUS
        ;;JNZ   short SU0
         ;;JMP     RETURN_STATUS
        je      RETURN_STATUS
SU0:
        MOV     byte [DISK_STATUS1],0 ; RESET THE STATUS INDICATOR
        ;;PUSH  BX                      ; SAVE DATA ADDRESS
        ;mov    si, bx ;; 14/02/2015
        mov     esi, ebx ; 21/02/2015
        MOV     BL,[HF_NUM]             ; GET NUMBER OF DRIVES
        ;; 04/01/2015
        ;;PUSH  AX
        AND     DL,7FH                  ; GET DRIVE AS 0 OR 1
                                        ; (get drive number as 0 to 3)
        CMP     BL,DL
         ;;JBE   BAD_COMMAND_POP           ; INVALID DRIVE
         jbe    BAD_COMMAND ;; 14/02/2015
```

```
        ;;03/01/2015
        sub     ebx, ebx
        mov     bl, dl
        ;sub    bh, bh
        mov     [LBAMode], bh  ; 0
        ;;test  byte [bx+hd0_type], 1 ; LBA ready ?
        ;test   byte [ebx+hd0_type], 1
        ;jz     short su1             ; no
        ;inc    byte [LBAMode]
;su1:
        ; 21/02/2015 (32 bit modification)
        ;04/01/2015
        push    ax ; ***
        ;PUSH   ES ; **
        PUSH    DX ; *
        push    ax
        CALL    GET_VEC               ; GET DISK PARAMETERS
        ; 02/02/2015
        ;mov    ax, [ES:BX+16] ; I/O port base address (1F0h, 170h)
        mov     ax, [ebx+16]
        mov     [HF_PORT], ax
        ;mov    dx, [ES:BX+18] ; control port address (3F6h, 376h)
        mov     dx, [ebx+18]
        mov     [HF_REG_PORT], dx
        ;mov    al, [ES:BX+20] ; head register upper nibble (A0h,B0h,E0h,F0h)
        mov     al, [ebx+20]
        ; 23/02/2015
        test    al, 40h  ; LBA bit (bit 6)
        jz      short su1
        inc     byte [LBAMode] ; 1
su1:
        shr     al, 4
        and     al, 1
        mov     [hf_m_s], al
        ;
        ; 03/01/2015
        ;MOV    AL,byte [ES:BX+8]     ; GET CONTROL BYTE MODIFIER
        mov     al, [ebx+8]
        ;MOV    DX,[HF_REG_PORT]      ; Device Control register
        OUT     DX,AL                 ; SET EXTRA HEAD OPTION
                                      ; Control Byte:  (= 08h, here)
                                      ; bit 0 - 0
                                      ; bit 1 - nIEN (1 = disable irq)
                                      ; bit 2 - SRST (software RESET)
                                      ; bit 3 - use extra heads (8 to 15)
                                      ;         -always set to 1-
                                      ; (bits 3 to 7 are reserved
                                      ;          for ATA devices)
        MOV     AH,[CONTROL_BYTE]     ; SET EXTRA HEAD OPTION IN
        AND     AH,0C0H               ; CONTROL BYTE
        OR      AH,AL
        MOV     [CONTROL_BYTE],AH
        ; 04/01/2015
        pop     ax
        pop     dx ; * ;; 14/02/2015
        and     ah, ah ; Reset function ?
        jnz     short su2
        ;;pop   dx ; * ;; 14/02/2015
        ;pop    es ; **
        pop     ax ; ***
        ;;pop   bx
         jmp     DISK_RESET
su2:
        cmp     byte [LBAMode], 0
        jna     short su3
        ;
        ; 02/02/2015 (LBA read/write function calls)
        cmp     ah, 1Bh
        jb      short lbarw1
        cmp     ah, 1Ch
        ja      short invldfnc
        ;;pop   dx ; * ; 14/02/2015
        ;mov    ax, cx ; Lower word of LBA address (bits 0-15)
        mov     eax, ecx ; LBA address (21/02/2015)
        ;; 14/02/2015
        mov     cl, dl ; 14/02/2015
        ;;mov   dx, bx
        ;mov    dx, si ; higher word of LBA address (bits 16-23)
        ;;mov   bx, di
```

```
        ;mov    si, di ; Buffer offset
        jmp     short lbarw2
lbarw1:
        ; convert CHS to LBA
        ;
        ; LBA calculation - AWARD BIOS - 1999 - AHDSK.ASM
        ; LBA = "# of Heads" * Sectors/Track * Cylinder + Head * Sectors/Track
        ;       + Sector - 1
        push    dx ; * ;; 14/02/2015
        ;xor    dh, dh
        xor     edx, edx
        ;mov    dl, [ES:BX+14] ; sectors per track (logical)
        mov     dl, [ebx+14]
        ;xor    ah, ah
        xor     eax, eax
        ;mov    al, [ES:BX+2]  ; heads (logical)
        mov     al, [ebx+2]
        dec     al
        inc     ax              ; 0 =  256
        mul     dx
                ; AX = # of Heads" * Sectors/Track
        mov     dx, cx
        ;and    cx, 3Fh  ; sector  (1 to 63)
        and     ecx, 3fh
        xchg    dl, dh
        shr     dh, 6
                ; DX = cylinder (0 to 1023)
        ;mul    dx
                ; DX:AX = # of Heads" * Sectors/Track * Cylinder
        mul     edx
        dec     cl  ; sector - 1
        ;add    ax, cx
        ;adc    dx, 0
                ; DX:AX = # of Heads" * Sectors/Track * Cylinder + Sector -1
        add     eax, ecx
        pop     cx ; * ; ch = head, cl = drive number (zero based)
        ;push   dx
        ;push   ax
        push    eax
        ;mov    al, [ES:BX+14] ; sectors per track (logical)
        mov     al, [ebx+14]
        mul     ch
                ;  AX = Head * Sectors/Track
         cwd
        ;pop    dx
        pop     edx
        ;add    ax, dx
        ;pop    dx
        ;adc    dx, 0 ; add carry bit
        add     eax, edx
lbarw2:
        sub     edx, edx ; 21/02/2015
        mov     dl, cl ; 21/02/2015
         mov     byte [CMD_BLOCK], 0 ; Features Register
                              ; NOTE: Features register (1F1h, 171h)
                              ; is not used for ATA device R/W functions.
                              ; It is old/obsolete 'write precompensation'
                              ; register and error register
                              ; for old ATA/IDE devices.
        ; 18/01/2014
        ;mov    ch, [hf_m_s]   ; Drive 0 (master) or 1 (slave)
        mov     cl, [hf_m_s]
        ;shl    ch, 4          ; bit 4 (drive bit)
        ;or     ch, 0E0h       ; bit 5 = 1
                              ; bit 6 = 1 = LBA mode
                              ; bit 7 = 1
        or      cl, 0Eh ; 1110b
        ;and    dh, 0Fh        ; LBA byte 4 (bits 24 to 27)
        and     eax, 0FFFFFFFh
        shl     ecx, 28 ; 21/02/2015
        ;or     dh, ch
        or      eax, ecx
        ;;mov   [CMD_BLOCK+2], al ; LBA byte 1 (bits 0 to 7)
                                ; (Sector Number Register)
        ;;mov   [CMD_BLOCK+3], ah ; LBA byte 2 (bits 8 to 15)
                                ; (Cylinder Low Register)
        ;mov    [CMD_BLOCK+2], ax ; LBA byte 1, 2
        ;mov    [CMD_BLOCK+4], dl ; LBA byte 3 (bits 16 to 23)
                                ; (Cylinder High Register)
```

```
        ;;mov   [CMD_BLOCK+5], dh ; LBA byte 4 (bits 24 to 27)
                                ; (Drive/Head Register)

        ;mov    [CMD_BLOCK+4], dx ; LBA byte 4, LBA & DEV select bits
        mov     [CMD_BLOCK+2], eax ; 21/02/2015
        ;14/02/2015
        ;mov    dl, cl ; Drive number (INIT_DRV)
        jmp     short su4
su3:
        ; 02/02/2015
        ; (Temporary functions 1Bh & 1Ch are not valid for CHS mode)
        cmp     ah, 14h
        jna     short chsfnc
invldfnc:
         ; 14/02/2015
        ;pop    es ; **
         pop     ax ; ***
         ;jmp     short BAD_COMMAND_POP
         jmp     short BAD_COMMAND
chsfnc:
        ;MOV    AX,[ES:BX+5]          ; GET WRITE PRE-COMPENSATION CYLINDER
        mov     ax, [ebx+5]
        SHR     AX,2
        MOV     [CMD_BLOCK],AL
        ;;MOV   AL,[ES:BX+8]          ; GET CONTROL BYTE MODIFIER
        ;;PUSH  DX
        ;;MOV   DX,[HF_REG_PORT]
        ;;OUT   DX,AL                 ; SET EXTRA HEAD OPTION
        ;;POP   DX ; *
        ;;POP   ES ; **
        ;;MOV   AH,[CONTROL_BYTE]     ; SET EXTRA HEAD OPTION IN
        ;;AND   AH,0C0H               ; CONTROL BYTE
        ;;OR    AH,AL
        ;;MOV   [CONTROL_BYTE],AH
        ;
        MOV     AL,CL                 ; GET SECTOR NUMBER
        AND     AL,3FH
        MOV     [CMD_BLOCK+2],AL
        MOV     [CMD_BLOCK+3],CH      ; GET CYLINDER NUMBER
        MOV     AL,CL
        SHR     AL,6
        MOV     [CMD_BLOCK+4],AL      ; CYLINDER HIGH ORDER 2 BITS
        ;;05/01/2015
        ;;MOV   AL,DL                 ; DRIVE NUMBER
        mov     al, [hf_m_s]
        SHL     AL,4
        AND     DH,0FH                ; HEAD NUMBER
        OR      AL,DH
        ;OR     AL,80H or 20H
        OR      AL,80h+20h            ; ECC AND 512 BYTE SECTORS
        MOV     [CMD_BLOCK+5],AL      ; ECC/SIZE/DRIVE/HEAD
su4:
        ;POP    ES ; **
         ;; 14/02/2015
         ;;POP   AX
         ;;MOV   [CMD_BLOCK+1],AL        ; SECTOR COUNT
         ;;PUSH  AX
         ;;MOV   AL,AH                   ; GET INTO LOW BYTE
         ;;XOR   AH,AH                   ; ZERO HIGH BYTE
         ;;SAL   AX,1                    ; *2 FOR TABLE LOOKUP
         pop     ax ; ***
         mov     [CMD_BLOCK+1], al
         sub    ebx, ebx
        mov      bl, ah
         ;xor     bh, bh
         ;sal     bx, 1
         sal    bx, 2  ; 32 bit offset (21/02/2015)
        ;;MOV    SI,AX                  ; PUT INTO SI FOR BRANCH
         ;;CMP   AX,M1L                  ; TEST WITHIN RANGE
         ;;JNB   short BAD_COMMAND_POP
         ;cmp     bx, M1L
        cmp     ebx, M1L
        jnb     short BAD_COMMAND
         ;xchg    bx, si
         xchg   ebx, esi
        ;;;POP  AX                    ; RESTORE AX
        ;;;POP  BX                    ; AND DATA ADDRESS

        ;;PUSH  CX
```

```
        ;;PUSH  AX                      ; ADJUST ES:BX
        ;MOV    CX,BX                   ; GET 3 HIGH ORDER NIBBLES OF BX
        ;SHR    CX,4
        ;MOV    AX,ES
        ;ADD    AX,CX
        ;MOV    ES,AX
        ;AND    BX,000FH                ; ES:BX CHANGED TO ES:000X
        ;;POP   AX
        ;;POP   CX
        ;;JMP   word [CS:SI+M1]
        ;jmp    word [SI+M1]
        jmp     dword [esi+M1]
;;BAD_COMMAND_POP:
;;      POP     AX
;;      POP     BX
BAD_COMMAND:
         MOV     byte [DISK_STATUS1],BAD_CMD  ; COMMAND ERROR
        MOV     AL,0
        RETn

;---------------------------------------
;       RESET THE DISK SYSTEM  (AH=00H) :
;---------------------------------------

; 18-1-2015 : one controller reset (not other one)

DISK_RESET:
        CLI
        IN      AL,INTB01               ; GET THE MASK REGISTER
        ;JMP    $+2
        IODELAY
        ;AND    AL,0BFH                 ; ENABLE FIXED DISK INTERRUPT
        and     al,3Fh                  ; 22/12/2014 (IRQ 14 & IRQ 15)
        OUT     INTB01,AL
        STI                             ; START INTERRUPTS
        ; 14/02/2015
        mov     di, dx
        ; 04/01/2015
        ;xor    di,di
drst0:
        MOV     AL,04H  ; bit 2 - SRST
        ;MOV    DX,HF_REG_PORT
        MOV     DX,[HF_REG_PORT]
        OUT     DX,AL                   ; RESET
;       MOV     CX,10                   ; DELAY COUNT
;DRD:   DEC     CX
;       JNZ     short DRD               ; WAIT 4.8 MICRO-SEC
        ;mov    cx,2                    ; wait for 30 micro seconds
         mov    ecx, 2 ; 21/02/2015
        call     WAITF                     ; (Award Bios 1999 - WAIT_REFRESH,
                                            ; 40 micro seconds)
        mov     al,[CONTROL_BYTE]
        AND     AL,0FH                  ; SET HEAD OPTION
        OUT     DX,AL                   ; TURN RESET OFF
        CALL    NOT_BUSY
        JNZ     short DRERR             ; TIME OUT ON RESET
        MOV     DX,[HF_PORT]
        inc     dl  ; HF_PORT+1
        ; 02/01/2015 - Award BIOS 1999 - AHDSK.ASM
         ;mov     cl, 10
         mov     ecx, 10 ; 21/02/2015
drst1:
        IN      AL,DX                   ; GET RESET STATUS
        CMP     AL,1
        ; 04/01/2015
        jz      short drst2
        ;JNZ    short DRERR             ; BAD RESET STATUS
                ; Drive/Head Register - bit 4
        loop    drst1
DRERR:
        MOV     byte [DISK_STATUS1],BAD_RESET ; CARD FAILED
        RETn
```

```
drst2:
        ; 14/02/2015
        mov     dx,di
;drst3:
;       ; 05/01/2015
;       shl     di,1
;       ; 04/01/2015
;       mov     ax,[di+hd_cports]
;       cmp     ax,[HF_REG_PORT]
;       je      short drst4
;       mov     [HF_REG_PORT], ax
;       ; 03/01/2015
;       mov     ax,[di+hd_ports]
;        mov     [HF_PORT], ax
;       ; 05/01/2014
;       shr     di,1
;       ; 04/01/2015
;       jmp     short drst0    ; reset other controller
;drst4:
;       ; 05/01/2015
;       shr     di,1
;       mov     al,[di+hd_dregs]
;       and     al,10h ; bit 4 only
;       shr     al,4 ; bit 4  -> bit 0
;       mov     [hf_m_s], al ; (0 = master, 1 = slave)
        ;
        mov     al, [hf_m_s] ; 18/01/2015
        test    al,1
;       jnz     short drst6
         jnz     short drst4
        AND      byte [CMD_BLOCK+5],0EFH ; SET TO DRIVE 0
;drst5:
drst3:
        CALL    INIT_DRV              ; SET MAX HEADS
        ;mov    dx,di
        CALL    HDISK_RECAL           ; RECAL TO RESET SEEK SPEED
        ; 04/01/2014
;       inc     di
;       mov     dx,di
;       cmp     dl,[HF_NUM]
;       jb      short drst3
;DRE:
        MOV     byte [DISK_STATUS1],0 ; IGNORE ANY SET UP ERRORS
        RETn
;drst6:
drst4:          ; Drive/Head Register - bit 4
        OR       byte [CMD_BLOCK+5],010H ; SET TO DRIVE 1
         ;jmp    short drst5
         jmp     short drst3

;----------------------------------------
;       DISK STATUS ROUTINE  (AH = 01H) :
;----------------------------------------

RETURN_STATUS:
        MOV     AL,[DISK_STATUS1]     ; OBTAIN PREVIOUS STATUS
         MOV     byte [DISK_STATUS1],0   ; RESET STATUS
        RETn

;----------------------------------------
;       DISK READ ROUTINE    (AH = 02H) :
;----------------------------------------

DISK_READ:
        MOV     byte [CMD_BLOCK+6],READ_CMD
         JMP     COMMANDI

;----------------------------------------
;       DISK WRITE ROUTINE   (AH = 03H) :
;----------------------------------------

DISK_WRITE:
        MOV     byte [CMD_BLOCK+6],WRITE_CMD
         JMP     COMMANDO
```

```
;----------------------------------------
;       DISK VERIFY        (AH = 04H) :
;----------------------------------------

DISK_VERF:
        MOV     byte [CMD_BLOCK+6],VERIFY_CMD
        CALL    COMMAND
        JNZ     short VERF_EXIT                ; CONTROLLER STILL BUSY
        CALL    _WAIT                  ; (Original: CALL WAIT)
        JNZ     short VERF_EXIT                ; TIME OUT
        CALL    CHECK_STATUS
VERF_EXIT:
        RETn

;----------------------------------------
;       FORMATTING         (AH = 05H) :
;----------------------------------------

FMT_TRK:                               ; FORMAT TRACK (AH = 005H)
        MOV     byte [CMD_BLOCK+6],FMTTRK_CMD
        ;PUSH   ES
        ;PUSH   BX
        push    ebx
        CALL    GET_VEC                ; GET DISK PARAMETERS ADDRESS
        ;MOV    AL,[ES:BX+14]          ; GET SECTORS/TRACK
        mov     al, [ebx+14]
        MOV     [CMD_BLOCK+1],AL       ; SET SECTOR COUNT IN COMMAND
        pop     ebx
        ;POP    BX
        ;POP    ES
         JMP     CMD_OF                   ; GO EXECUTE THE COMMAND

;----------------------------------------
;       READ DASD TYPE     (AH = 15H) :
;----------------------------------------

READ_DASD_TYPE:
READ_D_T:                              ; GET DRIVE PARAMETERS
        PUSH    DS                     ; SAVE REGISTERS
        ;PUSH   ES
        PUSH    eBX
        ;CALL   DDS                    ; ESTABLISH ADDRESSING
        ;push   cs
        ;pop    ds
         mov    bx, KDATA
        mov     ds, bx
        ;mov    es, bx
        MOV      byte [DISK_STATUS1],0
        MOV     BL,[HF_NUM]            ; GET NUMBER OF DRIVES
        AND     DL,7FH                 ; GET DRIVE NUMBER
        CMP     BL,DL
        JBE     short RDT_NOT_PRESENT  ; RETURN DRIVE NOT PRESENT
        CALL    GET_VEC                ; GET DISK PARAMETER ADDRESS
        ;MOV    AL,[ES:BX+2]           ; HEADS
        mov     al, [ebx+2]
        ;MOV    CL,[ES:BX+14]
        mov     cl, [ebx+14]
        IMUL    CL                     ; * NUMBER OF SECTORS
        ;MOV    CX,[ES:BX]             ; MAX NUMBER OF CYLINDERS
        mov     cx ,[ebx]
        ;
        ; 02/01/2015
        ; ** leave the last cylinder as reserved for diagnostics **
        ; (Also in Award BIOS - 1999, AHDSK.ASM, FUN15 -> sub ax, 1)
        DEC     CX                     ; LEAVE ONE FOR DIAGNOSTICS
        ;
        IMUL    CX                     ; NUMBER OF SECTORS
        MOV     CX,DX                  ; HIGH ORDER HALF
        MOV     DX,AX                  ; LOW ORDER HALF
        ;SUB    AX,AX
        sub     al, al
        MOV     AH,03H                 ; INDICATE FIXED DISK
RDT2:   POP     eBX                    ; RESTORE REGISTERS
        ;POP    ES
        POP     DS
        CLC                            ; CLEAR CARRY
        ;RETf   2
        retf    4
```

```
RDT_NOT_PRESENT:
       SUB     AX,AX                  ; DRIVE NOT PRESENT RETURN
       MOV     CX,AX                  ; ZERO BLOCK COUNT
       MOV     DX,AX
       JMP     short RDT2

;----------------------------------------
;      GET PARAMETERS      (AH = 08H) :
;----------------------------------------

GET_PARM_N:
;GET_PARM:                            ; GET DRIVE PARAMETERS
       PUSH    DS                     ; SAVE REGISTERS
       ;PUSH   ES
       PUSH    eBX
       ;MOV    AX,ABS0                ; ESTABLISH ADDRESSING
       ;MOV    DS,AX
       ;TEST   DL,1                   ; CHECK FOR DRIVE 1
       ;JZ     short G0
       ;LES    BX,@HF1_TBL_VEC
       ;JMP    SHORT G1
;G0:   LES     BX,@HF_TBL_VEC
;G1:
       ;CALL   DDS                    ; ESTABLISH SEGMENT
       ; 22/12/2014
       ;push   cs
       ;pop    ds
       mov     bx, KDATA
       mov     ds, bx
       ;mov    es, bx
       ;
       SUB     DL,80H
       CMP     DL,MAX_FILE            ; TEST WITHIN RANGE
       JAE     short G4
       ;
       xor     ebx, ebx ; 21/02/2015
       ; 22/12/2014
       mov     bl, dl
       ;xor    bh, bh
       shl     bl, 2                  ; convert index to offset
       ;add    bx, HF_TBL_VEC
       add     ebx, HF_TBL_VEC
       ;mov    ax, [bx+2]
       ;mov    es, ax                 ; dpt segment
       ;mov    bx, [bx]               ; dpt offset
       mov     ebx, [ebx] ; 32 bit offset

       MOV     byte [DISK_STATUS1],0
        ;MOV      AX,[ES:BX]                ; MAX NUMBER OF CYLINDERS
       mov     ax, [ebx]
       ;;SUB   AX,2                   ; ADJUST FOR 0-N
       dec     ax                     ; max. cylinder number
       MOV     CH,AL
       AND     AX,0300H               ; HIGH TWO BITS OF CYLINDER
       SHR     AX,1
       SHR     AX,1
       ;OR     AL,[ES:BX+14]          ; SECTORS
       or      al, [ebx+14]
       MOV     CL,AL
       ;MOV    DH,[ES:BX+2]           ; HEADS
       mov     dh, [ebx+2]
       DEC     DH                     ; 0-N RANGE
       MOV     DL,[HF_NUM]            ; DRIVE COUNT
       SUB     AX,AX
        ;27/12/2014
       ; ES:DI = Address of disk parameter table from BIOS
       ;(Programmer's Guide to the AMIBIOS - 1993)
       ;mov    di, bx                 ; HDPT offset
       mov     edi, ebx
G5:
       POP     eBX                    ; RESTORE REGISTERS
       ;POP    ES
       POP     DS
       ;RETf   2
       retf    4
G4:
       MOV      byte [DISK_STATUS1],INIT_FAIL ; OPERATION FAILED
       MOV     AH,INIT_FAIL
       SUB     AL,AL
```

```
        SUB     DX,DX
        SUB     CX,CX
        STC                             ; SET ERROR FLAG
        JMP     short G5

;-----------------------------------------
;       INITIALIZE DRIVE     (AH = 09H) :
;-----------------------------------------
        ; 03/01/2015
        ; According to ATA-ATAPI specification v2.0 to v5.0
        ; logical sector per logical track
        ; and logical heads - 1 would be set but
        ; it is seen as it will be good
        ; if physical parameters will be set here
        ; because, number of heads <= 16.
        ; (logical heads usually more than 16)
        ; NOTE: ATA logical parameters (software C, H, S)
        ;       == INT 13h physical parameters

;INIT_DRV:
;       MOV     byte [CMD_BLOCK+6],SET_PARM_CMD
;       CALL    GET_VEC                 ; ES:BX -> PARAMETER BLOCK
;       MOV     AL,[ES:BX+2]            ; GET NUMBER OF HEADS
;       DEC     AL                      ; CONVERT TO 0-INDEX
;       MOV     AH,[CMD_BLOCK+5]        ; GET SDH REGISTER
;       AND     AH,0F0H                 ; CHANGE HEAD NUMBER
;       OR      AH,AL                   ; TO MAX HEAD
;       MOV     [CMD_BLOCK+5],AH
;       MOV     AL,[ES:BX+14]           ; MAX SECTOR NUMBER
;       MOV     [CMD_BLOCK+1],AL
;       SUB     AX,AX
;       MOV     [CMD_BLOCK+3],AL        ; ZERO FLAGS
;       CALL    COMMAND                 ; TELL CONTROLLER
;       JNZ     short INIT_EXIT                 ; CONTROLLER BUSY ERROR
;       CALL    NOT_BUSY                ; WAIT FOR IT TO BE DONE
;       JNZ     short INIT_EXIT                 ; TIME OUT
;       CALL    CHECK_STATUS
;INIT_EXIT:
;       RETn

; 04/01/2015
; 02/01/2015 - Derived from from AWARD BIOS 1999
;                               AHDSK.ASM - INIT_DRIVE
INIT_DRV:
        ;xor    ah,ah
        xor     eax, eax ; 21/02/2015
        mov     al,11 ; Physical heads from translated HDPT
         cmp     [LBAMode], ah    ; 0
        ja      short idrv0
        mov     al,2  ; Physical heads from standard HDPT
idrv0:
        ; DL = drive number (0 based)
        call    GET_VEC
        ;push   bx
        push    ebx ; 21/02/2015
        ;add    bx,ax
        add     ebx, eax
        ;; 05/01/2015
        mov     ah, [hf_m_s] ; drive number (0= master, 1= slave)
        ;;and   ah,1
        shl     ah,4
        or      ah,0A0h  ; Drive/Head register - 10100000b (A0h)
        ;mov    al,[es:bx]
        mov     al, [ebx] ; 21/02/2015
        dec     al       ; last head number
        ;and    al,0Fh
        or      al,ah    ; lower 4 bits for head number
        ;
        mov     byte [CMD_BLOCK+6],SET_PARM_CMD
        mov     [CMD_BLOCK+5],al
        ;pop    bx
        pop     ebx
        sub     eax, eax ; 21/02/2015
        mov     al,4 ; Physical sec per track from translated HDPT
        cmp     byte [LBAMode], 0
        ja      short idrv1
        mov     al,14 ; Physical sec per track from standard HDPT
```

```
idrv1:
        ;xor    ah,ah
        ;add    bx,ax
        add     ebx, eax ; 21/02/2015
        ;mov    al,[es:bx]
                        ; sector number
        mov     al, [ebx]
        mov     [CMD_BLOCK+1],al
        sub     al,al
        mov     [CMD_BLOCK+3],al  ; ZERO FLAGS
        call    COMMAND          ; TELL CONTROLLER
        jnz     short INIT_EXIT          ; CONTROLLER BUSY ERROR
        call    NOT_BUSY         ; WAIT FOR IT TO BE DONE
        jnz     short INIT_EXIT          ; TIME OUT
        call    CHECK_STATUS
INIT_EXIT:
        RETn

;----------------------------------------
;       READ LONG           (AH = 0AH) :
;----------------------------------------

RD_LONG:
        ;MOV    @CMD_BLOCK+6,READ_CMD OR ECC_MODE
         mov     byte [CMD_BLOCK+6],READ_CMD + ECC_MODE
         JMP     COMMANDI

;----------------------------------------
;       WRITE LONG          (AH = 0BH) :
;----------------------------------------

WR_LONG:
        ;MOV    @CMD_BLOCK+6,WRITE_CMD OR ECC_MODE
         MOV     byte [CMD_BLOCK+6],WRITE_CMD + ECC_MODE
         JMP     COMMANDO

;----------------------------------------
;       SEEK                (AH = 0CH) :
;----------------------------------------

DISK_SEEK:
         MOV     byte [CMD_BLOCK+6],SEEK_CMD
        CALL    COMMAND
        JNZ     short DS_EXIT          ; CONTROLLER BUSY ERROR
        CALL    _WAIT
         JNZ     DS_EXIT                  ; TIME OUT ON SEEK
        CALL    CHECK_STATUS
         CMP     byte [DISK_STATUS1],BAD_SEEK
        JNE     short DS_EXIT
         MOV     byte [DISK_STATUS1],0
DS_EXIT:
        RETn

;----------------------------------------
;       TEST DISK READY      (AH = 10H) :
;----------------------------------------

TST_RDY:                               ; WAIT FOR CONTROLLER
        CALL    NOT_BUSY
        JNZ     short TR_EX
        MOV     AL,[CMD_BLOCK+5]       ; SELECT DRIVE
        MOV     DX,[HF_PORT]
        add     dl,6
        OUT     DX,AL
        CALL    CHECK_ST               ; CHECK STATUS ONLY
        JNZ     short TR_EX
        MOV     byte [DISK_STATUS1],0 ; WIPE OUT DATA CORRECTED ERROR
TR_EX:
        RETn
```

```
;----------------------------------------
;       RECALIBRATE         (AH = 11H) :
;----------------------------------------

HDISK_RECAL:
        MOV     byte [CMD_BLOCK+6],RECAL_CMD ; 10h, 16
        CALL    COMMAND                 ; START THE OPERATION
        JNZ     short RECAL_EXIT        ; ERROR
        CALL    _WAIT                   ; WAIT FOR COMPLETION
        JZ      short RECAL_X           ; TIME OUT ONE OK ?
        CALL    _WAIT                   ; WAIT FOR COMPLETION LONGER
        JNZ     short RECAL_EXIT        ; TIME OUT TWO TIMES IS ERROR
RECAL_X:
        CALL    CHECK_STATUS
        CMP     byte [DISK_STATUS1],BAD_SEEK ; SEEK NOT COMPLETE
        JNE     short RECAL_EXIT        ; IS OK
        MOV     byte [DISK_STATUS1],0
RECAL_EXIT:
        CMP     byte [DISK_STATUS1],0
        RETn

;----------------------------------------
;      CONTROLLER DIAGNOSTIC (AH = 14H) :
;----------------------------------------

CTLR_DIAGNOSTIC:
        CLI                             ; DISABLE INTERRUPTS WHILE CHANGING MASK
        IN      AL,INTB01               ; TURN ON SECOND INTERRUPT CHIP
        ;AND    AL,0BFH
        and     al, 3Fh                 ; enable IRQ 14 & IRQ 15
        ;JMP    $+2
        IODELAY
        OUT     INTB01,AL
        IODELAY
        IN      AL,INTA01               ; LET INTERRUPTS PASS THRU TO
        AND     AL,0FBH                 ;  SECOND CHIP
        ;JMP    $+2
        IODELAY
        OUT     INTA01,AL
        STI
        CALL    NOT_BUSY                ; WAIT FOR CARD
        JNZ     short CD_ERR            ; BAD CARD
        ;MOV    DX, HF_PORT+7
        mov     dx, [HF_PORT]
        add     dl, 7
        MOV     AL,DIAG_CMD             ; START DIAGNOSE
        OUT     DX,AL
        CALL    NOT_BUSY                ; WAIT FOR IT TO COMPLETE
        MOV     AH,TIME_OUT
        JNZ     short CD_EXIT           ; TIME OUT ON DIAGNOSTIC
        ;MOV    DX,HF_PORT+1            ; GET ERROR REGISTER
        mov     dx, [HF_PORT]
        inc     dl
        IN      AL,DX
        MOV     [HF_ERROR],AL           ; SAVE IT
        MOV     AH,0
        CMP     AL,1                    ; CHECK FOR ALL OK
        JE      SHORT CD_EXIT
CD_ERR: MOV     AH,BAD_CNTLR
CD_EXIT:
        MOV     [DISK_STATUS1],AH
        RETn
```

```
;----------------------------------------
; COMMANDI                              :
;       REPEATEDLY INPUTS DATA TILL     :
;       NSECTOR RETURNS ZERO            :
;----------------------------------------
COMMANDI:
        CALL    CHECK_DMA               ; CHECK 64K BOUNDARY ERROR
        JC      short CMD_ABORT
        ;MOV    DI,BX
        mov     edi, ebx ; 21/02/2015
        CALL    COMMAND                 ; OUTPUT COMMAND
        JNZ     short CMD_ABORT
CMD_I1:
        CALL    _WAIT                   ; WAIT FOR DATA REQUEST INTERRUPT
        JNZ     short TM_OUT            ; TIME OUT
        ;MOV    CX,256                  ; SECTOR SIZE IN WORDS
        mov     ecx, 256 ; 21/02/2015
        ;MOV    DX,HF_PORT
        mov     dx,[HF_PORT]
        CLI
        CLD
        REP     INSW                    ; GET THE SECTOR
        STI
        TEST    byte [CMD_BLOCK+6],ECC_MODE ; CHECK FOR NORMAL INPUT
        JZ      CMD_I3
        CALL    WAIT_DRQ                ; WAIT FOR DATA REQUEST
        JC      short TM_OUT
        ;MOV    DX,HF_PORT
        mov     dx,[HF_PORT]
        ;MOV    CX,4                    ; GET ECC BYTES
        mov     ecx, 4 ; mov cx, 4
CMD_I2: IN      AL,DX
        ;MOV    [ES:DI],AL              ; GO SLOW FOR BOARD
        mov     [edi], al ; 21/02/2015
        INC     eDI
        LOOP    CMD_I2
CMD_I3: CALL    CHECK_STATUS
        JNZ     short CMD_ABORT                 ; ERROR RETURNED
        DEC     byte [CMD_BLOCK+1]      ; CHECK FOR MORE
        JNZ     SHORT CMD_I1
CMD_ABORT:
TM_OUT: RETn

;----------------------------------------
; COMMANDO                              :
;       REPEATEDLY OUTPUTS DATA TILL    :
;       NSECTOR RETURNS ZERO            :
;----------------------------------------
COMMANDO:
        CALL    CHECK_DMA               ; CHECK 64K BOUNDARY ERROR
        JC      short CMD_ABORT
CMD_OF: MOV     eSI,eBX ; 21/02/2015
        CALL    COMMAND                 ; OUTPUT COMMAND
        JNZ     short CMD_ABORT
        CALL    WAIT_DRQ                ; WAIT FOR DATA REQUEST
        JC      short TM_OUT                    ; TOO LONG
CMD_O1: ;PUSH   DS
        ;PUSH   ES                      ; MOVE ES TO DS
        ;POP    DS
        ;MOV    CX,256                  ; PUT THE DATA OUT TO THE CARD
        ;MOV    DX,HF_PORT
        ; 01/02/2015
        mov     dx, [HF_PORT]
        ;push   es
        ;pop    ds
        ;mov    cx, 256
        mov     ecx, 256 ; 21/02/2015
        CLI
        CLD
        REP     OUTSW
        STI
        ;POP    DS                      ; RESTORE DS
        TEST    byte [CMD_BLOCK+6],ECC_MODE ; CHECK FOR NORMAL OUTPUT
        JZ      short CMD_O3
        CALL    WAIT_DRQ                ; WAIT FOR DATA REQUEST
        JC      short TM_OUT
        ;MOV    DX,HF_PORT
        mov     dx, [HF_PORT]
        ;MOV    CX,4                    ; OUTPUT THE ECC BYTES
```

```
        mov     ecx, 4  ; mov cx, 4
CMD_02: ;MOV    AL,[ES:SI]
        mov     al, [esi]
        OUT     DX,AL
        INC     eSI
        LOOP    CMD_02
CMD_03:
        CALL    _WAIT                   ; WAIT FOR SECTOR COMPLETE INTERRUPT
        JNZ     short TM_OUT            ; ERROR RETURNED
        CALL    CHECK_STATUS
        JNZ     short CMD_ABORT
        TEST    byte [HF_STATUS],ST_DRQ         ; CHECK FOR MORE
        JNZ     SHORT CMD_01
        ;MOV    DX,HF_PORT+2            ; CHECK RESIDUAL SECTOR COUNT
        mov     dx, [HF_PORT]
        ;add    dl, 2
        inc     dl
        inc     dl
        IN      AL,DX                   ;
        TEST    AL,0FFH                 ;
        JZ      short CMD_04                    ; COUNT = 0  OK
        MOV     byte [DISK_STATUS1],UNDEF_ERR
                                        ; OPERATION ABORTED - PARTIAL TRANSFER
CMD_04:
        RETn

;---------------------------------------------------------
; COMMAND                                               :
;       THIS ROUTINE OUTPUTS THE COMMAND BLOCK                 :
; OUTPUT                                                :
;       BL = STATUS                                     :
;       BH = ERROR REGISTER                             :
;---------------------------------------------------------
COMMAND:
        PUSH    eBX                     ; WAIT FOR SEEK COMPLETE AND READY
        ;;MOV   CX,DELAY_2              ; SET INITIAL DELAY BEFORE TEST
COMMAND1:
        ;;PUSH  CX                      ; SAVE LOOP COUNT
        CALL    TST_RDY                 ; CHECK DRIVE READY
        ;;POP   CX
        JZ      short COMMAND2          ; DRIVE IS READY
         CMP      byte [DISK_STATUS1],TIME_OUT ; TST_RDY TIMED OUT--GIVE UP
        ;JZ     short CMD_TIMEOUT
        ;;LOOP  COMMAND1                ; KEEP TRYING FOR A WHILE
        ;JMP    SHORT COMMAND4          ; ITS NOT GOING TO GET READY
        jne     short COMMAND4
CMD_TIMEOUT:
        MOV     byte [DISK_STATUS1],BAD_CNTLR
COMMAND4:
        POP     eBX
         CMP      byte [DISK_STATUS1],0   ; SET CONDITION CODE FOR CALLER
        RETn
COMMAND2:
        POP     eBX
        PUSH    eDI
        MOV     byte [HF_INT_FLAG],0    ; RESET INTERRUPT FLAG
        CLI                             ; INHIBIT INTERRUPTS WHILE CHANGING MASK
        IN      AL,INTB01               ; TURN ON SECOND INTERRUPT CHIP
        ;AND    AL,0BFH
        and     al, 3Fh                 ; Enable IRQ 14 & 15
        ;JMP    $+2
        IODELAY
        OUT     INTB01,AL
        IN      AL,INTA01               ; LET INTERRUPTS PASS THRU TO
        AND     AL,0FBH                 ;  SECOND CHIP
        ;JMP    $+2
        IODELAY
        OUT     INTA01,AL
        STI
        XOR     eDI,eDI                 ; INDEX THE COMMAND TABLE
        ;MOV    DX,HF_PORT+1            ; DISK ADDRESS
        mov     dx, [HF_PORT]
        inc     dl
        TEST    byte [CONTROL_BYTE],0C0H ; CHECK FOR RETRY SUPPRESSION
        JZ      short COMMAND3
        MOV     AL, [CMD_BLOCK+6]       ; YES-GET OPERATION CODE
        AND     AL,0F0H                 ; GET RID OF MODIFIERS
        CMP     AL,20H                  ; 20H-40H IS READ, WRITE, VERIFY
        JB      short COMMAND3
```

```
        CMP     AL,40H
        JA      short COMMAND3
        OR      byte [CMD_BLOCK+6],NO_RETRIES
                                      ; VALID OPERATION FOR RETRY SUPPRESS
COMMAND3:
        MOV     AL,[CMD_BLOCK+eDI]    ; GET THE COMMAND STRING BYTE
        OUT     DX,AL                 ; GIVE IT TO CONTROLLER
        IODELAY
        INC     eDI                   ; NEXT BYTE IN COMMAND BLOCK
        INC     DX                    ; NEXT DISK ADAPTER REGISTER
        cmp     di, 7   ; 1/1/2015    ; ALL DONE?
        JNZ     short COMMAND3        ; NO--GO DO NEXT ONE
        POP     eDI
        RETn                          ; ZERO FLAG IS SET

;CMD_TIMEOUT:
;       MOV     byte [DISK_STATUS1],BAD_CNTLR
;COMMAND4:
;       POP     BX
;       CMP     [DISK_STATUS1],0      ; SET CONDITION CODE FOR CALLER
;       RETn

;----------------------------------------
;       WAIT FOR INTERRUPT            :
;----------------------------------------
;WAIT:
_WAIT:
        STI                           ; MAKE SURE INTERRUPTS ARE ON
        ;SUB    CX,CX                 ; SET INITIAL DELAY BEFORE TEST
        ;CLC
        ;MOV    AX,9000H              ; DEVICE WAIT INTERRUPT
        ;INT    15H
        ;JC     WT2                   ; DEVICE TIMED OUT
        ;MOV    BL,DELAY_1            ; SET DELAY COUNT

        ;mov    bl, WAIT_HDU_INT_HI
        ;; 21/02/2015
        ;;mov   bl, WAIT_HDU_INT_HI + 1
        ;;mov   cx, WAIT_HDU_INT_LO
        mov     ecx, WAIT_HDU_INT_LH
                                      ; (AWARD BIOS -> WAIT_FOR_MEM)
;-----  WAIT LOOP

WT1:
        ;TEST   byte [HF_INT_FLAG],80H; TEST FOR INTERRUPT
        test    byte [HF_INT_FLAG],0C0h
        ;LOOPZ  WT1
        JNZ     short WT3             ; INTERRUPT--LETS GO
        ;DEC    BL
        ;JNZ    short WT1             ; KEEP TRYING FOR A WHILE

WT1_hi:
        in      al, SYS1 ; 61h (PORT_B)      ; wait for lo to hi
        test    al, 10h               ; transition on memory
        jnz     short WT1_hi          ; refresh.
WT1_lo:
        in      al, SYS1              ; 061h (PORT_B)
        test    al, 10h
        jz      short WT1_lo
        loop    WT1
        ;;or    bl, bl
        ;;jz    short WT2
        ;;dec   bl
        ;;jmp   short WT1
        ;dec    bl
        ;jnz    short WT1

WT2:    MOV     byte [DISK_STATUS1],TIME_OUT ; REPORT TIME OUT ERROR
        JMP     SHORT WT4
WT3:    MOV     byte [DISK_STATUS1],0
        MOV     byte [HF_INT_FLAG],0
WT4:    CMP     byte [DISK_STATUS1],0 ; SET CONDITION CODE FOR CALLER
        RETn
```

```
;----------------------------------------
;       WAIT FOR CONTROLLER NOT BUSY   :
;----------------------------------------
NOT_BUSY:
        STI                             ; MAKE SURE INTERRUPTS ARE ON
        ;PUSH   eBX
        ;SUB    CX,CX                   ; SET INITIAL DELAY BEFORE TEST
        mov     DX, [HF_PORT]
        add     dl, 7                   ; Status port (HF_PORT+7)
        ;MOV    BL,DELAY_1
                                        ; wait for 10 seconds
        ;mov    cx, WAIT_HDU_INT_LO     ; 1615h
        ;;mov   bl, WAIT_HDU_INT_HI     ;   05h
        ;mov    bl, WAIT_HDU_INT_HI + 1
        mov     ecx, WAIT_HDU_INT_LH    ; 21/02/2015
        ;
;;       mov     byte [wait_count], 0    ; Reset wait counter
NB1:
        IN      AL,DX                   ; CHECK STATUS
        ;TEST   AL,ST_BUSY
        and     al, ST_BUSY
        ;LOOPNZ NB1
        JZ      short NB2               ; NOT BUSY--LETS GO
        ;DEC    BL
        ;JNZ    short NB1               ; KEEP TRYING FOR A WHILE

NB1_hi: IN      AL,SYS1                 ; wait for hi to lo
        TEST    AL,010H                 ; transition on memory
        JNZ     SHORT NB1_hi            ; refresh.
NB1_lo: IN      AL,SYS1
        TEST    AL,010H
        JZ      short NB1_lo
        LOOP    NB1
        ;dec    bl
        ;jnz    short NB1
        ;
;;       cmp     byte [wait_count], 182  ; 10 seconds (182 timer ticks)
;;      jb      short NB1
        ;
        ;MOV    [DISK_STATUS1],TIME_OUT       ; REPORT TIME OUT ERROR
        ;JMP    SHORT NB3
        mov     al, TIME_OUT
NB2:
        ;MOV    byte [DISK_STATUS1],0
;NB3:
        ;POP    eBX
        mov     [DISK_STATUS1], al      ;;; will be set after return
        ;CMP    byte [DISK_STATUS1],0 ; SET CONDITION CODE FOR CALLER
        or      al, al                  ; (zf = 0 --> timeout)
        RETn

;----------------------------------------
;       WAIT FOR DATA REQUEST          :
;----------------------------------------
WAIT_DRQ:
        ;MOV    CX,DELAY_3
        ;MOV    DX,HF_PORT+7
        mov     dx, [HF_PORT]
        add     dl, 7
        ;;MOV   bl, WAIT_HDU_DRQ_HI     ; 0
        ;MOV    cx, WAIT_HDU_DRQ_LO     ; 1000 (30 milli seconds)
                                        ; (but it is written as 2000
                                        ; micro seconds in ATORGS.ASM file
                                        ; of Award Bios - 1999, D1A0622)
        mov     ecx, WAIT_HDU_DRQ_LH ; 21/02/2015
WQ_1:   IN      AL,DX                   ; GET STATUS
        TEST    AL,ST_DRQ               ; WAIT FOR DRQ
        JNZ     short WQ_OK
        ;LOOP   WQ_1                    ; KEEP TRYING FOR A SHORT WHILE
WQ_hi:
        IN      AL,SYS1                 ; wait for hi to lo
        TEST    AL,010H                 ; transition on memory
        JNZ     SHORT WQ_hi             ; refresh.
WQ_lo:  IN       AL,SYS1
        TEST    AL,010H
        JZ      SHORT WQ_lo
        LOOP    WQ_1
```

```
		MOV	byte [DISK_STATUS1],TIME_OUT ; ERROR
		STC
WQ_OK:
		RETn
;WQ_OK: ;CLC
;		RETn

;----------------------------------------
;	CHECK FIXED DISK STATUS		:
;----------------------------------------
CHECK_STATUS:
		CALL	CHECK_ST		; CHECK THE STATUS BYTE
		JNZ	short CHECK_S1		; AN ERROR WAS FOUND
		TEST	AL,ST_ERROR		; WERE THERE ANY OTHER ERRORS
		JZ	short CHECK_S1		; NO ERROR REPORTED
		CALL	CHECK_ER		; ERROR REPORTED
CHECK_S1:
		CMP	byte [DISK_STATUS1],0 ; SET STATUS FOR CALLER
		RETn

;----------------------------------------
;	CHECK FIXED DISK STATUS BYTE	:
;----------------------------------------
CHECK_ST:
		;MOV	DX,HF_PORT+7		; GET THE STATUS
		mov	dx, [HF_PORT]
		add	dl, 7
		IN	AL,DX
		MOV	[HF_STATUS],AL
		MOV	AH,0
		TEST	AL,ST_BUSY		; IF STILL BUSY
		JNZ	short CKST_EXIT			;  REPORT OK
		MOV	AH,WRITE_FAULT
		TEST	AL,ST_WRT_FLT		; CHECK FOR WRITE FAULT
		JNZ	short CKST_EXIT
		MOV	AH,NOT_RDY
		TEST	AL,ST_READY		; CHECK FOR NOT READY
		JZ	short CKST_EXIT
		MOV	AH,BAD_SEEK
		TEST	AL,ST_SEEK_COMPL	; CHECK FOR SEEK NOT COMPLETE
		JZ	short CKST_EXIT
		MOV	AH,DATA_CORRECTED
		TEST	AL,ST_CORRCTD		; CHECK FOR CORRECTED ECC
		JNZ	short CKST_EXIT
		MOV	AH,0
CKST_EXIT:
		MOV	[DISK_STATUS1],AH	; SET ERROR FLAG
		CMP	AH,DATA_CORRECTED	; KEEP GOING WITH DATA CORRECTED
		JZ	short CKST_EX1
		CMP	AH,0
CKST_EX1:
		RETn

;----------------------------------------
;	CHECK FIXED DISK ERROR REGISTER :
;----------------------------------------
CHECK_ER:
		;MOV	DX, HF_PORT+1		; GET THE ERROR REGISTER
		mov	dx, [HF_PORT]		;
		inc	dl
		IN	AL,DX
		MOV	[HF_ERROR],AL
		PUSH	eBX ; 21/02/2015
		MOV	eCX,8			; TEST ALL 8 BITS
CK1:		SHL	AL,1			; MOVE NEXT ERROR BIT TO CARRY
		JC	short CK2		; FOUND THE ERROR
		LOOP	CK1			; KEEP TRYING
CK2:		MOV	eBX, ERR_TBL		; COMPUTE ADDRESS OF
		ADD	eBX,eCX			; ERROR CODE
		;;MOV	AH,BYTE [CS:BX]			; GET ERROR CODE
		;mov	ah, [bx]
		mov	ah, [ebx] ; 21/02/2015
CKEX:		MOV	[DISK_STATUS1],AH	; SAVE ERROR CODE
		POP	eBX
		CMP	AH,0
		RETn
```

```
;-------------------------------------------------------
; CHECK_DMA                                           :
;  -CHECK ES:BX AND # SECTORS TO MAKE SURE THAT IT WILL :
;   FIT WITHOUT SEGMENT OVERFLOW.                     :
;  -ES:BX HAS BEEN REVISED TO THE FORMAT SSSS:000X  :
;  -OK IF # SECTORS < 80H (7FH IF LONG READ OR WRITE) :
;  -OK IF # SECTORS = 80H (7FH) AND BX <= 00H (04H) :
;  -ERROR OTHERWISE                                   :
;-------------------------------------------------------
CHECK_DMA:
        PUSH    AX                      ; SAVE REGISTERS
        MOV     AX,8000H                ; AH = MAX # SECTORS AL = MAX OFFSET
         TEST    byte [CMD_BLOCK+6],ECC_MODE
        JZ      short CKD1
        MOV     AX,7F04H                ; ECC IS 4 MORE BYTES
CKD1:   CMP     AH, [CMD_BLOCK+1]       ; NUMBER OF SECTORS
        JA      short CKDOK             ; IT WILL FIT
        JB      short CKDERR            ; TOO MANY
        CMP     AL,BL                   ; CHECK OFFSET ON MAX SECTORS
        JB      short CKDERR            ; ERROR
CKDOK:  CLC                             ; CLEAR CARRY
        POP     AX
        RETn                            ; NORMAL RETURN
CKDERR: STC                             ; INDICATE ERROR
         MOV     byte [DISK_STATUS1],DMA_BOUNDARY
        POP     AX
        RETn

;----------------------------------------
;       SET UP ES:BX-> DISK PARMS       :
;----------------------------------------

; INPUT -> DL = 0 based drive number
; OUTPUT -> ES:BX = disk parameter table address

GET_VEC:
        ;SUB    AX,AX                   ; GET DISK PARAMETER ADDRESS
        ;MOV    ES,AX
        ;TEST   DL,1
        ;JZ     short GV_0
;       LES     BX,[HF1_TBL_VEC]        ; ES:BX -> DRIVE PARAMETERS
;       JMP     SHORT GV_EXIT
;GV_0:
;       LES     BX,[HF_TBL_VEC]                 ; ES:BX -> DRIVE PARAMETERS
;
        ;xor    bh, bh
        xor     ebx, ebx
        mov     bl, dl
        ;;02/01/2015
        ;;shl   bl, 1                   ; port address offset
        ;;mov   ax, [bx+hd_ports]       ; Base port address (1F0h, 170h)
        ;;shl   bl, 1                   ; dpt pointer offset
        shl     bl, 2   ;;
        ;add    bx, HF_TBL_VEC          ; Disk parameter table pointer
        add     ebx, HF_TBL_VEC ; 21/02/2015
        ;push   word [bx+2]             ; dpt segment
        ;pop    es
        ;mov    bx, [bx]                ; dpt offset
        mov     ebx, [ebx]
;GV_EXIT:
        RETn
```

```
hdc1_int: ; 21/02/2015
;--- HARDWARE INT 76H -- ( IRQ LEVEL  14 ) ----------------------
;                                                              :
;       FIXED DISK INTERRUPT ROUTINE                           :
;                                                              :
;----------------------------------------------------------------
; 22/12/2014
; IBM PC-XT Model 286 System BIOS Source Code - DISK.ASM (HD_INT)
;        '11/15/85'
; AWARD BIOS 1999 (D1A0622)
;       Source Code - ATORGS.ASM (INT_HDISK, INT_HDISK1)
;int_76h:
HD_INT:
       PUSH    AX
       PUSH    DS
       ;CALL   DDS
       ; 21/02/2015 (32 bit, 386 pm modification)
       mov     ax, KDATA
       mov     ds, ax
       ;;MOV   @HF_INT_FLAG,0FFH       ; ALL DONE
        ;mov      byte [CS:HF_INT_FLAG], 0FFh
       mov     byte [HF_INT_FLAG], 0FFh
       push    dx
       mov     dx, HDC1_BASEPORT+7     ; Status Register (1F7h)
                                       ; Clear Controller
Clear_IRQ1415:                         ; (Award BIOS - 1999)
       in      al, dx                  ;
       pop     dx
       NEWIODELAY
       MOV     AL,EOI                  ; NON-SPECIFIC END OF INTERRUPT
       OUT     INTB00,AL               ; FOR CONTROLLER #2
       ;JMP    $+2                     ; WAIT
       NEWIODELAY
       OUT     INTA00,AL               ; FOR CONTROLLER #1
       POP     DS
       ;STI                            ; RE-ENABLE INTERRUPTS
       ;MOV    AX,9100H                ; DEVICE POST
       ;INT    15H                     ;  INTERRUPT
irq15_iret: ; 25/02/2015
       POP     AX
       IRETd                           ; RETURN FROM INTERRUPT

hdc2_int: ; 21/02/2015
;++++ HARDWARE INT 77H ++ ( IRQ LEVEL  15 ) +++++++++++++++++++++
;                                                              :
;       FIXED DISK INTERRUPT ROUTINE                           :
;                                                              :
;++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++
;int_77h:
HD1_INT:
       PUSH    AX
       ; Check if that is a spurious IRQ (from slave PIC)
       ; 25/02/2015 (source: http://wiki.osdev.org/8259_PIC)
       mov     al, 0Bh  ; In-Service Register
       out     0A0h, al
        jmp short $+2
       jmp short $+2
       in      al, 0A0h
       and     al, 80h ; bit 7 (is it real IRQ 15 or fake?)
       jz      short irq15_iret ; Fake (spurious)IRQ, do not send EOI)
       PUSH    DS
       ;CALL   DDS
       ; 21/02/2015 (32 bit, 386 pm modification)
       mov     ax, KDATA
       mov     ds, ax
       ;;MOV   @HF_INT_FLAG,0FFH       ; ALL DONE
        ;or       byte [CS:HF_INT_FLAG],0C0h
       or      byte [HF_INT_FLAG], 0C0h
       push    dx
       mov     dx, HDC2_BASEPORT+7     ; Status Register (177h)
                                       ; Clear Controller (Award BIOS 1999)
       jmp     short Clear_IRQ1415

;%include 'diskdata.inc' ; 11/03/2015
;%include 'diskbss.inc' ; 11/03/2015

;////////////////////////////////////////////////////////////////////
;; END OF DISK I/O SYTEM ///
```

```
; MEMORY.ASM - Retro UNIX 386 v1 MEMORY MANAGEMENT FUNCTIONS (PROCEDURES)
; Retro UNIX 386 v1 Kernel (unix386.s, v0.2.0.14) - MEMORY.INC
; Last Modification: 18/10/2015 (!not completed!)
;
; Source code for NASM - Netwide Assembler (2.11)

; ///////// MEMORY MANAGEMENT FUNCTIONS (PROCEDURES) ///////////////

;;04/11/2014 (unix386.s)
;PDE_A_PRESENT equ    1             ; Present flag for PDE
;PDE_A_WRITE   equ    2             ; Writable (write permission) flag
;PDE_A_USER    equ    4             ; User (non-system/kernel) page flag
;;
;PTE_A_PRESENT equ    1             ; Present flag for PTE (bit 0)
;PTE_A_WRITE   equ    2             ; Writable (write permission) flag (bit 1)
;PTE_A_USER    equ    4             ; User (non-system/kernel) page flag (bit 2)
;PTE_A_ACCESS   equ   32            ; Accessed flag (bit 5) ; 09/03/2015

; 27/04/2015
; 09/03/2015
PAGE_SIZE      equ 4096             ; page size in bytes
PAGE_SHIFT     equ 12               ; page table shift count
PAGE_D_SHIFT   equ 22 ; 12 + 10     ; page directory shift count
PAGE_OFF       equ 0FFFh            ; 12 bit byte offset in page frame
PTE_MASK       equ 03FFh            ; page table entry mask
PTE_DUPLICATED  equ 200h            ; duplicated page sign (AVL bit 0)
PDE_A_CLEAR    equ 0F000h           ; to clear PDE attribute bits
PTE_A_CLEAR    equ 0F000h           ; to clear PTE attribute bits
LOGIC_SECT_SIZE equ 512                    ; logical sector size
ERR_MAJOR_PF   equ 0E0h             ; major error: page fault
ERR_MINOR_IM   equ 1                ; insufficient (out of) memory
ERR_MINOR_DSK  equ 2                ; disk read/write error
ERR_MINOR_PV   equ 3                ; protection violation
SWP_DISK_READ_ERR equ 4
SWP_DISK_NOT_PRESENT_ERR equ 5
SWP_SECTOR_NOT_PRESENT_ERR equ 6
SWP_NO_FREE_SPACE_ERR equ 7
SWP_DISK_WRITE_ERR equ 8
SWP_NO_PAGE_TO_SWAP_ERR equ 9
PTE_A_ACCESS_BIT equ 5  ; Bit 5 (accessed flag)
SECTOR_SHIFT    equ 3   ; sector shift (to convert page block number)

;
;; Retro Unix 386 v1 - paging method/principles
;;
;; 10/10/2014
;; RETRO UNIX 386 v1 - PAGING METHOD/PRINCIPLES
;;
;; KERNEL PAGE MAP: 1 to 1 physical memory page map
;;     (virtual address = physical address)
;; KERNEL PAGE TABLES:
;;     Kernel page directory and all page tables are
;;     on memory as initialized, as equal to physical memory
;;     layout. Kernel pages can/must not be swapped out/in.
;;
;;     what for: User pages may be swapped out, when accessing
;;     a page in kernel/system mode, if it would be swapped out,
;;     kernel would have to swap it in! But it is also may be
;;     in use by a user process. (In system/kernel mode
;;     kernel can access all memory pages even if they are
;;     reserved/allocated for user processes. Swap out/in would
;;     cause conflicts.)
;;
;;     As result of these conditions,
;;     all kernel pages must be initialized as equal to
;;     physical layout for preventing page faults.
;;     Also, calling "allocate page" procedure after
;;     a page fault can cause another page fault (double fault)
;;     if all kernel page tables would not be initialized.
;;
;;     [first_page] = Beginning of users space, as offset to
;;     memory allocation table. (double word aligned)
;;
;;     [next_page] = first/next free space to be searched
;;     as offset to memory allocation table. (dw aligned)
;;
;;     [last_page] = End of memory (users space), as offset
;;     to memory allocation table. (double word aligned)
;;
```

```
;; USER PAGE TABLES:
;;      Demand paging (& 'copy on write' allocation method) ...
;;              'ready only' marked copies of the
;;              parent process's page table entries (for
;;              same physical memory).
;;              (A page will be copied to a new page after
;;               if it causes R/W page fault.)
;;
;;      Every user process has own (different)
;;      page directory and page tables.
;;
;;      Code starts at virtual address 0, always.
;;      (Initial value of EIP is 0 in user mode.)
;;      (Programs can be written/developed as simple
;;       flat memory programs.)
;;
;; MEMORY ALLOCATION STRATEGY:
;;      Memory page will be allocated by kernel only
;;              (in kernel/system mode only).
;;      * After a
;;        - 'not present' page fault
;;        - 'writing attempt on read only page' page fault
;;      * For loading (opening, reading) a file or disk/drive
;;      * As responce to 'allocate additional memory blocks'
;;        request by running process.
;;      * While creating a process, allocating a new buffer,
;;        new page tables etc.
;;
;;      At first,
;;      - 'allocate page' procedure will be called;
;;         if it will return with a valid (>0) physical address
;;         (that means the relevant M.A.T. bit has been RESET)
;;         relevant memory page/block will be cleared (zeroed).
;;      - 'allocate page' will be called for allocating page
;;         directory, page table and running space (data/code).
;;      - every successful 'allocate page' call will decrease
;;        'free_pages' count (pointer).
;;      - 'out of (insufficient) memory error' will be returned
;;        if 'free_pages' points to a ZERO.
;;      - swapping out and swapping in (if it is not a new page)
;;        procedures will be called as responce to 'out of memory'
;;        error except errors caused by attribute conflicts.
;;       (swapper functions)
;;
;;      At second,
;;      - page directory entry will be updated then page table
;;        entry will be updated.
;;
;; MEMORY ALLOCATION TABLE FORMAT:
;;      - M.A.T. has a size according to available memory as
;;        follows:
;;                - 1 (allocation) bit per 1 page (4096 bytes)
;;                - a bit with value of 0 means allocated page
;;                - a bit with value of 1 means a free page
;;      - 'free_pages' pointer holds count of free pages
;;        depending on M.A.T.
;;              (NOTE: Free page count will not be checked
;;              again -on M.A.T.- after initialization.
;;              Kernel will trust on initial count.)
;;      - 'free_pages' count will be decreased by allocation
;;        and it will be increased by deallocation procedures.
;;
;;      - Available memory will be calculated during
;;        the kernel's initialization stage (in real mode).
;;        Memory allocation table and kernel page tables
;;        will be formatted/sized as result of available
;;        memory calculation before paging is enabled.
;;
;; For 4GB Available/Present Memory: (max. possible memory size)
;;      - Memory Allocation Table size will be 128 KB.
;;      - Memory allocation for kernel page directory size
;;        is always 4 KB. (in addition to total allocation size
;;        for page tables)
;;      - Memory allocation for kernel page tables (1024 tables)
;;        is 4 MB (1024*4*1024 bytes).
;;      - User (available) space will be started
;;        at 6th MB of the memory (after 1MB+4MB).
;;      - The first 640 KB is for kernel's itself plus
;;        memory allocation table and kernel's page directory
```

```
;;        (D0000h-EFFFFh may be used as kernel space...)
;;      - B0000h to B7FFFh address space (32 KB) will be used
;;        for buffers.
;;      - ROMBIOS, VIDEO BUFFER and VIDEO ROM space are reserved.
;;        (A0000h-AFFFFh, C0000h-CFFFFh, F0000h-FFFFFh)
;;      - Kernel page tables start at 100000h (2nd MB)
;;
;; For 1GB Available Memory:
;;      - Memory Allocation Table size will be 32 KB.
;;      - Memory allocation for kernel page directory size
;;        is always 4 KB. (in addition to total allocation size
;;        for page tables)
;;      - Memory allocation for kernel page tables (256 tables)
;;        is 1 MB (256*4*1024 bytes).
;;      - User (available) space will be started
;;        at 3th MB of the memory (after 1MB+1MB).
;;      - The first 640 KB is for kernel's itself plus
;;        memory allocation table and kernel's page directory
;;        (D0000h-EFFFFh may be used as kernel space...)
;;      - B0000h to B7FFFh address space (32 KB) will be used
;;        for buffers.
;;      - ROMBIOS, VIDEO BUFFER and VIDEO ROM space are reserved.
;;        (A0000h-AFFFFh, C0000h-CFFFFh, F0000h-FFFFFh)
;;      - Kernel page tables start at 100000h (2nd MB).
;;
;;


;;*************************************************************************************
;;
;; RETRO UNIX 386 v1 - Paging (Method for Copy On Write paging principle)
;; DEMAND PAGING - PARENT&CHILD PAGE TABLE DUPLICATION PRINCIPLES (23/04/2015)

;; Main factor: "sys fork" system call
;;
;;             FORK
;;                        |----> parent - duplicated PTEs, read only pages
;;  writable pages ---->|
;;                        |----> child - duplicated PTEs, read only pages
;;
;; AVL bit (0) of Page Table Entry is used as duplication sign
;;
;; AVL Bit 0 [PTE Bit 9] = 'Duplicated PTE belongs to child' sign/flag (if it is set)
;; Note: Dirty bit (PTE bit 6) may be used instead of AVL bit 0 (PTE bit 9)
;;       -while R/W bit is 0-.
;;
;; Duplicate page tables with writable pages (the 1st sys fork in the process):
;; # Parent's Page Table Entries are updated to point same pages as read only,
;;   as duplicated PTE bit  -AVL bit 0, PTE bit 9- are reset/clear.
;; # Then Parent's Page Table is copied to Child's Page Table.
;; # Child's Page Table Entries are updated as duplicated child bit
;;   -AVL bit 0, PTE bit 9- is set.
;;
;; Duplicate page tables with read only pages (several sys fork system calls):
;; # Parent's read only pages are copied to new child pages.
;;   Parent's PTE attributes are not changed.
;;   (Because, there is another parent-child fork before this fork! We must not
;;    destroy/mix previous fork result).
;; # Child's Page Table Entries (which are corresponding to Parent's
;;   read only pages) are set as writable (while duplicated PTE bit is clear).
;; # Parent's PTEs with writable page attribute are updated to point same pages
;;   as read only, (while) duplicated PTE bit is reset (clear).
;; # Parent's Page Table Entries (with writable page attribute) are duplicated
;;   as Child's Page Table Entries without copying actual page.
;; # Child 's Page Table Entries (which are corresponding to Parent's writable
;;   pages) are updated as duplicated PTE bit (AVL bit 0, PTE bit 9- is set.
;;
;; !? WHAT FOR (duplication after duplication):
;; In UNIX method for sys fork (a typical 'fork' application in /etc/init)
;; program/executable code continues from specified location as child process,
;; returns back previous code location as parent process, every child after
;; every sys fork uses last image of code and data just prior the fork.
;; Even if the parent code changes data, the child will not see the changed data
;; after the fork. In Retro UNIX 8086 v1, parent's process segment (32KB)
;; was copied to child's process segment (all of code and data) according to
;; original UNIX v1 which copies all of parent process code and data -core-
;; to child space -core- but swaps that core image -of child- on to disk.
;; If I (Erdogan Tan) would use a method of to copy parent's core
;; (complete running image of parent process) to the child process;
```

```
;; for big sizes, i would force Retro UNIX 386 v1 to spend many memory pages
;; and times only for a sys fork. (It would excessive reservation for sys fork,
;; because sys fork usually is prior to sys exec; sys exec always establishes
;; a new/fresh core -running space-, by clearing all code/data content).
;; 'Read Only' page flag ensures page fault handler is needed only for a few write
;; attempts between sys fork and sys exec, not more... (I say so by thinking
;; of "/etc/init" content, specially.) sys exec will clear page tables and
;; new/fresh pages will be used to load and run new executable/program.
;; That is what for i have preferred "copy on write", "duplication" method
;; for sharing same read only pages between parent and child processes.
;; That is a pitty i have to use new private flag (AVL bit 0, "duplicated PTE
;; belongs to child" sign) for cooperation on duplicated pages between a parent
;; and it's child processes; otherwise parent process would destroy data belongs
;; to its child or vice versa; or some pages would remain unclaimed
;; -deallocation problem-.
;; Note: to prevent conflicts, read only pages must not be swapped out...
;;
;; WHEN PARENT TRIES TO WRITE IT'S READ ONLY (DUPLICATED) PAGE:
;; # Page fault handler will do those:
;;   - 'Duplicated PTE' flag (PTE bit 9) is checked (on the failed PTE).
;;   - If it is reset/clear, there is a child uses same page.
;;   - Parent's read only page -previous page- is copied to a new writable page.
;;   - Parent's PTE is updated as writable page, as unique page (AVL=0)
;;   - (Page fault handler whill check this PTE later, if child process causes to
;;     page fault due to write attempt on read only page. Of course, the previous
;;     read only page will be converted to writable and unique page which belongs
;;     to child process.)
;; WHEN CHILD TRIES TO WRITE IT'S READ ONLY (DUPLICATED) PAGE:
;; # Page fault handler will do those:
;;   - 'Duplicated PTE' flag (PTE bit 9) is checked (on the failed PTE).
;;   - If it is set, there is a parent uses -or was using- same page.
;;   - Same PTE address within parent's page table is checked if it has same page
;;     address or not.
;;   - If parent's PTE has same address, child will continue with a new writable page.
;;     Parent's PTE will point to same (previous) page as writable, unique (AVL=0).
;;   - If parent's PTE has different address, child will continue with it's
;;     own/same page but read only flag (0) will be changed to writable flag (1) and
;;     'duplicated PTE (belongs to child)' flag/sign will be cleared/reset.
;;
;; NOTE: When a child process is terminated, read only flags of parent's page tables
;;       will be set as writable (and unique) in case of child process was using
;;       same pages with duplicated child PTE sign... Depending on sys fork and
;;       duplication method details, it is not possible multiple child processes
;;       were using same page with duplicated PTEs.
;;
;;*************************************************************************************

;; 08/10/2014
;; 11/09/2014 - Retro UNIX 386 v1 PAGING (further) draft
;;             by Erdogan Tan (Based on KolibriOS 'memory.inc')

;; 'allocate_page' code is derived and modified from KolibriOS
;; 'alloc_page' procedure in 'memory.inc'
;; (25/08/2014, Revision: 5057) file
;; by KolibriOS Team (2004-2012)

allocate_page:
        ; 01/07/2015
        ; 05/05/2015
        ; 30/04/2015
        ; 16/10/2014
        ; 08/10/2014
        ; 09/09/2014 (Retro UNIX 386 v1 - beginning)
        ;
        ; INPUT -> none
        ;
        ; OUTPUT ->
        ;       EAX = PHYSICAL (real/flat) ADDRESS OF THE ALLOCATED PAGE
        ;       (corresponding MEMORY ALLOCATION TABLE bit is RESET)
        ;
        ;       CF = 1 and EAX = 0
        ;                  if there is not a free page to be allocated
        ;
        ; Modified Registers -> none (except EAX)
        ;
        mov     eax, [free_pages]
        and     eax, eax
        jz      short out_of_memory
        ;
```

```
        push    ebx
        push    ecx
        ;
        mov     ebx, MEM_ALLOC_TBL   ; Memory Allocation Table offset
        mov     ecx, ebx
                                   ; NOTE: 32 (first_page) is initial
                                   ; value of [next_page].
                                   ; It points to the first available
                                   ; page block for users (ring 3) ...
                                   ; (MAT offset 32 = 1024/32)
                                   ; (at the of the first 4 MB)
        add     ebx, [next_page] ; Free page searching starts from here
                               ; next_free_page >> 5
        add     ecx, [last_page] ; Free page searching ends here
                               ; (total_pages - 1) >> 5
al_p_scan:
        cmp     ebx, ecx
        ja      short al_p_notfound
        ;
        ; 01/07/2015
        ; AMD64 Architecture ProgrammerÆs Manual
        ; Volume 3:
        ; General-Purpose and System Instructions
        ;
        ; BSF - Bit Scan Forward
        ;
        ;   Searches the value in a register or a memory location
        ;   (second operand) for the least-significant set bit.
        ;   If a set bit is found, the instruction clears the zero flag (ZF)
        ;   and stores the index of the least-significant set bit in a destination
        ;   register (first operand). If the second operand contains 0,
        ;   the instruction sets ZF to 1 and does not change the contents of the
        ;   destination register. The bit index is an unsigned offset from bit 0
        ;   of the searched value
        ;
        bsf     eax, [ebx] ; Scans source operand for first bit set (1).
                           ; Clear ZF if a bit is found set (1) and
                           ; loads the destination with an index to
                           ; first set bit. (0 -> 31)
                           ; Sets ZF to 1 if no bits are found set.
        jnz     short al_p_found ; ZF = 0 -> a free page has been found
                         ;
                         ; NOTE:  a Memory Allocation Table bit
                         ;        with value of 1 means
                         ;        the corresponding page is free
                         ;        (Retro UNIX 386 v1 feaure only!)
        add     ebx, 4
                         ; We return back for searching next page block
                         ; NOTE: [free_pages] is not ZERO; so,
                         ;       we always will find at least 1 free page here.
         jmp     short al_p_scan
        ;
al_p_notfound:
        sub     ecx, MEM_ALLOC_TBL
        mov     [next_page], ecx ; next/first free page = last page
                               ; (deallocate_page procedure will change it)
        xor     eax, eax
        mov     [free_pages], eax ; 0
        pop     ecx
        pop     ebx
        ;
out_of_memory:
        call    swap_out
        jnc     short al_p_ok  ; [free_pages] = 0, re-allocation by swap_out
        ;
        sub     eax, eax ; 0
        stc
        retn

al_p_found:
        mov     ecx, ebx
        sub     ecx, MEM_ALLOC_TBL
        mov     [next_page], ecx ; Set first free page searching start
                               ; address/offset (to the next)
         dec     dword [free_pages] ; 1 page has been allocated (X = X-1)
        ;
        btr     [ebx], eax      ; The destination bit indexed by the source value
                               ; is copied into the Carry Flag and then cleared
                               ; in the destination.
```

```
                        ; Reset the bit which is corresponding to the
                        ; (just) allocated page.
        ; 01/07/2015 (4*8 = 32, 1 allocation byte = 8 pages)
        shl     ecx, 3          ; (page block offset * 32) + page index
        add     eax, ecx        ; = page number
        shl     eax, 12         ; physical address of the page (flat/real value)
        ; EAX = physical address of memory page
        ;
        ; NOTE: The relevant page directory and page table entry will be updated
        ;       according to this EAX value...
        pop     ecx
        pop     ebx
al_p_ok:
        retn

make_page_dir:
        ; 18/04/2015
        ; 12/04/2015
        ; 23/10/2014
        ; 16/10/2014
        ; 09/10/2014 ; (Retro UNIX 386 v1 - beginning)
        ;
        ; INPUT ->
        ;       none
        ; OUTPUT ->
        ;       (EAX = 0)
        ;       cf = 1 -> insufficient (out of) memory error
        ;       cf = 0 ->
        ;       u.pgdir = page directory (physical) address of the current
        ;                 process/user.
        ;
        ; Modified Registers -> EAX
        ;
        call    allocate_page
        jc      short mkpd_error
        ;
        mov     [u.pgdir], eax    ; Page dir address for current user/process
                                ; (Physical address)
clear_page:
        ; 18/04/2015
        ; 09/10/2014 ; (Retro UNIX 386 v1 - beginning)
        ;
        ; INPUT ->
        ;       EAX = physical address of the page
        ; OUTPUT ->
        ;       all bytes of the page will be cleared
        ;
        ; Modified Registers -> none
        ;
        push    edi
        push    ecx
        push    eax
        mov     ecx, PAGE_SIZE / 4
        mov     edi, eax
        xor     eax, eax
        rep     stosd
        pop     eax
        pop     ecx
        pop     edi
mkpd_error:
mkpt_error:
        retn

make_page_table:
        ; 23/06/2015
        ; 18/04/2015
        ; 12/04/2015
        ; 16/10/2014
        ; 09/10/2014 ; (Retro UNIX 386 v1 - beginning)
        ;
        ; INPUT ->
        ;       EBX = virtual (linear) address
        ;       ECX = page table attributes (lower 12 bits)
        ;             (higher 20 bits must be ZERO)
        ;             (bit 0 must be 1)
        ;       u.pgdir = page directory (physical) address
        ; OUTPUT ->
        ;       EDX = Page directory entry address
        ;       EAX = Page table address
```

```
        ;       cf = 1 -> insufficient (out of) memory error
        ;       cf = 0 -> page table address in the PDE (EDX)
        ;
        ; Modified Registers -> EAX, EDX
        ;
        call    allocate_page
        jc      short mkpt_error
        call    set_pde
        jmp     short clear_page

make_page:
        ; 24/07/2015
        ; 23/06/2015 ; (Retro UNIX 386 v1 - beginning)
        ;
        ; INPUT ->
        ;       EBX = virtual (linear) address
        ;       ECX = page attributes (lower 12 bits)
        ;             (higher 20 bits must be ZERO)
        ;             (bit 0 must be 1)
        ;       u.pgdir = page directory (physical) address
        ; OUTPUT ->
        ;       EBX = Virtual address
        ;       (EDX = PTE value)
        ;       EAX = Physical address
        ;       cf = 1 -> insufficient (out of) memory error
        ;
        ; Modified Registers -> EAX, EDX
        ;
        call    allocate_page
        jc      short mkp_err
        call    set_pte
        jnc     short clear_page ; 18/04/2015
mkp_err:
        retn

set_pde:        ; Set page directory entry (PDE)
        ; 20/07/2015
        ; 18/04/2015
        ; 12/04/2015
        ; 23/10/2014
        ; 10/10/2014 ; (Retro UNIX 386 v1 - beginning)
        ;
        ; INPUT ->
        ;       EAX = physical address
        ;             (use present value if EAX = 0)
        ;       EBX = virtual (linear) address
        ;       ECX = page table attributes (lower 12 bits)
        ;             (higher 20 bits must be ZERO)
        ;             (bit 0 must be 1)
        ;       u.pgdir = page directory (physical) address
        ; OUTPUT ->
        ;       EDX = PDE address
        ;       EAX = page table address (physical)
        ;       ;(CF=1 -> Invalid page address)
        ;
        ; Modified Registers -> EDX
        ;
        mov     edx, ebx
        shr     edx, PAGE_D_SHIFT ; 22
        shl     edx, 2 ; offset to page directory (1024*4)
        add     edx, [u.pgdir]
        ;
        and     eax, eax
        jnz     short spde_1
        ;
        mov     eax, [edx]  ; old PDE value
        ;test   al, 1
        ;jz     short spde_2
        and     ax, PDE_A_CLEAR ; 0F000h  ; clear lower 12 bits
spde_1:
        ;and    cx, 0FFFh
        mov     [edx], eax
        or      [edx], cx
        retn
;spde_2: ; error
;       stc
;       retn
```

```
set_pte:        ; Set page table entry (PTE)
        ; 24/07/2015
        ; 20/07/2015
        ; 23/06/2015
        ; 18/04/2015
        ; 12/04/2015
        ; 10/10/2014 ; (Retro UNIX 386 v1 - beginning)
        ;
        ; INPUT ->
        ;       EAX = physical page address
        ;             (use present value if EAX = 0)
        ;       EBX = virtual (linear) address
        ;       ECX = page attributes (lower 12 bits)
        ;             (higher 20 bits must be ZERO)
        ;             (bit 0 must be 1)
        ;       u.pgdir = page directory (physical) address
        ; OUTPUT ->
        ;       EAX = physical page address
        ;       (EDX = PTE value)
        ;       EBX = virtual address
        ;
        ;       CF = 1 -> error
        ;
        ; Modified Registers -> EAX, EDX
        ;
        push    eax
        mov     eax, [u.pgdir] ; 20/07/2015
        call    get_pde
                ; EDX = PDE address
                ; EAX = PDE value
        pop     edx ; physical page address
        jc      short spte_err ; PDE not present
        ;
        push    ebx ; 24/07/2015
        and     ax, PDE_A_CLEAR ; 0F000h ; clear lower 12 bits
                            ; EDX = PT address (physical)
        shr     ebx, PAGE_SHIFT ; 12
        and     ebx, PTE_MASK  ; 03FFh
                        ; clear higher 10 bits (PD bits)
        shl     ebx, 2   ; offset to page table (1024*4)
        add     ebx, eax
        ;
        mov     eax, [ebx] ; Old PTE value
        test    al, 1
        jz      short spte_0
        or      edx, edx
        jnz     short spte_1
        and     ax, PTE_A_CLEAR ; 0F000h ; clear lower 12 bits
        mov     edx, eax
        jmp     short spte_2
spte_0:
        ; If this PTE contains a swap (disk) address,
        ; it can be updated by using 'swap_in' procedure
        ; only!
        and     eax, eax
        jz      short spte_1
        ; 24/07/2015
        ; swapped page ! (on disk)
        pop     ebx
spte_err:
        stc
        retn
spte_1:
        mov     eax, edx
spte_2:
        or      edx, ecx
        ; 23/06/2015
        mov     [ebx], edx ; PTE value in EDX
        ; 24/07/2015
        pop     ebx
        retn
```

```
get_pde:        ; Get present value of the relevant PDE
        ; 20/07/2015
        ; 18/04/2015
        ; 12/04/2015
        ; 10/10/2014 ; (Retro UNIX 386 v1 - beginning)
        ;
        ; INPUT ->
        ;       EBX = virtual (linear) address
        ;       EAX = page directory (physical) address
        ; OUTPUT ->
        ;       EDX = Page directory entry address
        ;       EAX = Page directory entry value
        ;       CF = 1 -> PDE not present or invalid ?
        ; Modified Registers -> EDX, EAX
        ;
        mov     edx, ebx
        shr     edx, PAGE_D_SHIFT ; 22  (12+10)
        shl     edx, 2 ; offset to page directory (1024*4)
        add     edx, eax ; page directory address (physical)
        mov     eax, [edx]
        test    al, PDE_A_PRESENT ; page table is present or not !
        jnz     short gpte_retn
        stc
gpde_retn:
        retn

get_pte:
                ; Get present value of the relevant PTE
        ; 29/07/2015
        ; 20/07/2015
        ; 18/04/2015
        ; 12/04/2015
        ; 10/10/2014 ; (Retro UNIX 386 v1 - beginning)
        ;
        ; INPUT ->
        ;       EBX = virtual (linear) address
        ;       EAX = page directory (physical) address
        ; OUTPUT ->
        ;       EDX = Page table entry address (if CF=0)
        ;             Page directory entry address (if CF=1)
        ;            (Bit 0 value is 0 if PT is not present)
        ;       EAX = Page table entry value (page address)
        ;       CF = 1 -> PDE not present or invalid ?
        ; Modified Registers -> EAX, EDX
        ;
        call    get_pde
        jc      short gpde_retn       ; page table is not present
        ;jnc    short gpte_1
        ;retn
;gpte_1:
        and     ax, PDE_A_CLEAR ; 0F000h ; clear lower 12 bits
        mov     edx, ebx
        shr     edx, PAGE_SHIFT ; 12
        and     edx, PTE_MASK  ; 03FFh
                        ; clear higher 10 bits (PD bits)
        shl     edx, 2 ; offset from start of page table (1024*4)
        add     edx, eax
        mov     eax, [edx]
gpte_retn:
        retn

deallocate_page_dir:
        ; 15/09/2015
        ; 05/08/2015
        ; 30/04/2015
        ; 28/04/2015
        ; 17/10/2014
        ; 12/10/2014 (Retro UNIX 386 v1 - beginning)
        ;
        ; INPUT ->
        ;       EAX = PHYSICAL ADDRESS OF THE PAGE DIRECTORY (CHILD)
        ;       EBX = PHYSICAL ADDRESS OF THE PARENT'S PAGE DIRECTORY
        ; OUTPUT ->
        ;       All of page tables in the page directory
        ;       and page dir's itself will be deallocated
        ;       except 'read only' duplicated pages (will be converted
        ;       to writable pages).
        ;
        ; Modified Registers -> EAX
```

```
        ;
        push    esi
        push    ecx
        push    eax
        mov     esi, eax
        xor     ecx, ecx
        ; The 1st PDE points to Kernel Page Table 0 (the 1st 4MB),
        ; it must not be deallocated
        mov     [esi], ecx ; 0 ; clear PDE 0
dapd_0:
        lodsd
        test    al, PDE_A_PRESENT ; bit 0, present flag (must be 1)
        jz      short dapd_1
        and     ax, PDE_A_CLEAR ; 0F000h ; clear lower 12 (attribute) bits
        call    deallocate_page_table
dapd_1:
        inc     ecx ; page directory entry index
        cmp     ecx, PAGE_SIZE / 4 ; 1024
        jb      short dapd_0
dapd_2:
        pop     eax
        call    deallocate_page       ; deallocate the page dir's itself
        pop     ecx
        pop     esi
        retn

deallocate_page_table:
        ; 19/09/2015
        ; 15/09/2015
        ; 05/08/2015
        ; 30/04/2015
        ; 28/04/2015
        ; 24/10/2014
        ; 23/10/2014
        ; 12/10/2014 (Retro UNIX 386 v1 - beginning)
        ;
        ; INPUT ->
        ;       EAX = PHYSICAL (real/flat) ADDRESS OF THE PAGE TABLE
        ;       EBX = PHYSICAL ADDRESS OF THE PARENT'S PAGE DIRECTORY
        ;       (ECX = page directory entry index)
        ; OUTPUT ->
        ;       All of pages in the page table and page table's itself
        ;       will be deallocated except 'read only' duplicated pages
        ;       (will be converted to writable pages).
        ;
        ; Modified Registers -> EAX
        ;
        push    esi
        push    edi
        push    edx
        push    eax ; *
        mov     esi, eax
        xor     edi, edi ; 0
dapt_0:
        lodsd
        test    al, PTE_A_PRESENT ; bit 0, present flag (must be 1)
        jz      short dapt_1
        ;
        test    al, PTE_A_WRITE   ; bit 1, writable (r/w) flag
                                 ; (must be 1)
        jnz     short dapt_3
        ; Read only -duplicated- page (belongs to a parent or a child)
         test    ax, PTE_DUPLICATED ; Was this page duplicated
                                   ; as child's page ?
        jz      short dapt_4 ; Clear PTE but don't deallocate the page!
        ; check the parent's PTE value is read only & same page or not..
        ; ECX = page directory entry index (0-1023)
        push    ebx
        push    ecx
        shl     cx, 2 ; *4
        add     ebx, ecx ; PDE offset (for the parent)
        mov     ecx, [ebx]
        test    cl, PDE_A_PRESENT ; present (valid) or not ?
        jz      short dapt_2   ; parent process does not use this page
        and     cx, PDE_A_CLEAR ; 0F000h ; Clear attribute bits
        ; EDI = page table entry index (0-1023)
        mov     edx, edi
        shl     dx, 2 ; *4
        add     edx, ecx ; PTE offset (for the parent)
```

```
        mov     ebx, [edx]
        test    bl, PTE_A_PRESENT ; present or not ?
        jz      short dapt_2   ; parent process does not use this page
        and     ax, PTE_A_CLEAR ; 0F000h ; Clear attribute bits
        and     bx, PTE_A_CLEAR ; 0F000h ; Clear attribute bits
        cmp     eax, ebx       ; parent's and child's pages are same ?
        jne     short dapt_2   ; not same page
                               ; deallocate the child's page
         or      byte [edx], PTE_A_WRITE ; convert to writable page (parent)
        pop     ecx
        pop     ebx
        jmp     short dapt_4
dapt_1:
        or      eax, eax       ; swapped page ?
        jz      short dapt_5   ; no
                               ; yes
        shr     eax, 1
        call    unlink_swap_block ; Deallocate swapped page block
                                 ; on the swap disk (or in file)
        jmp     short dapt_5
dapt_2:
        pop     ecx
        pop     ebx
dapt_3:
        ;and    ax, PTE_A_CLEAR ; 0F000h ; clear lower 12 (attribute) bits
        call    deallocate_page
dapt_4:
        mov     dword [esi-4], 0 ; clear/reset PTE (child, dupl. as parent)
dapt_5:
        inc     edi ; page table entry index
        cmp     edi, PAGE_SIZE / 4 ; 1024
        jb      short dapt_0
        ;
        pop     eax ; *
        pop     edx
        pop     edi
        pop     esi
        ;
        ;call   deallocate_page       ; deallocate the page table's itself
        ;retn

deallocate_page:
        ; 15/09/2015
        ; 28/04/2015
        ; 10/03/2015
        ; 17/10/2014
        ; 12/10/2014 (Retro UNIX 386 v1 - beginning)
        ;
        ; INPUT ->
        ;       EAX = PHYSICAL (real/flat) ADDRESS OF THE ALLOCATED PAGE
        ; OUTPUT ->
        ;       [free_pages] is increased
        ;       (corresponding MEMORY ALLOCATION TABLE bit is SET)
        ;       CF = 1 if the page is already deallocated
        ;              (or not allocated) before.
        ;
        ; Modified Registers -> EAX
        ;
        push    ebx
        push    edx
        ;
        shr     eax, PAGE_SHIFT      ; shift physical address to
                                   ; 12 bits right
                                   ; to get page number
        mov     edx, eax
        ; 15/09/2015
        shr     edx, 3             ; to get offset to M.A.T.
                                   ; (1 allocation bit = 1 page)
                                   ; (1 allocation bytes = 8 pages)
        and     dl, 0FCh           ; clear lower 2 bits
                                   ; (to get 32 bit position)
        ;
        mov     ebx, MEM_ALLOC_TBL   ; Memory Allocation Table address
        add     ebx, edx
        and     eax, 1Fh           ; lower 5 bits only
                                   ; (allocation bit position)
```

```
        cmp     edx, [next_page]     ; is the new free page address lower
                                     ; than the address in 'next_page' ?
                                     ; (next/first free page value)
        jnb     short dap_1          ; no
        mov     [next_page], edx     ; yes
dap_1:
        bts     [ebx], eax           ; unlink/release/deallocate page
                                     ; set relevant bit to 1.
                                     ; set CF to the previous bit value
        ;cmc                         ; complement carry flag
        ;jc     short dap_2          ; do not increase free_pages count
                                     ; if the page is already deallocated
                                     ; before.
         inc     dword [free_pages]
dap_2:
        pop     edx
        pop     ebx
        retn

;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;
;;                                                              ;;
;; Copyright (C) KolibriOS team 2004-2012. All rights reserved. ;;
;; Distributed under terms of the GNU General Public License    ;;
;;                                                              ;;
;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;

;;$Revision: 5057 $


;;align 4
;;proc alloc_page

;;        pushfd
;;        cli
;;        push    ebx
;;;//-
;;        cmp     [pg_data.pages_free], 1
;;        jle     .out_of_memory
;;;//-
;;
;;        mov     ebx, [page_start]
;;        mov     ecx, [page_end]
;;.l1:
;;        bsf     eax, [ebx];
;;        jnz     .found
;;        add     ebx, 4
;;        cmp     ebx, ecx
;;        jb      .l1
;;        pop     ebx
;;        popfd
;;        xor     eax, eax
;;        ret
;;.found:
;;;//-
;;        dec     [pg_data.pages_free]
;;        jz      .out_of_memory
;;;//-
;;        btr     [ebx], eax
;;        mov     [page_start], ebx
;;        sub     ebx, sys_pgmap
;;        lea     eax, [eax+ebx*8]
;;        shl     eax, 12
;;;//-        dec [pg_data.pages_free]
;;        pop     ebx
;;        popfd
;;        ret
;;;//-
;;.out_of_memory:
;;        mov     [pg_data.pages_free], 1
;;        xor     eax, eax
;;        pop     ebx
;;        popfd
;;        ret
;;;//-
;;endp
```

```
duplicate_page_dir:
        ; 21/09/2015
        ; 31/08/2015
        ; 20/07/2015
        ; 28/04/2015
        ; 27/04/2015
        ; 18/04/2015
        ; 12/04/2015
        ; 18/10/2014
        ; 16/10/2014 (Retro UNIX 386 v1 - beginning)
        ;
        ; INPUT ->
        ;       [u.pgdir] = PHYSICAL (real/flat) ADDRESS of the parent's
        ;                   page directory.
        ; OUTPUT ->
        ;       EAX =  PHYSICAL (real/flat) ADDRESS of the child's
        ;              page directory.
        ;       (New page directory with new page table entries.)
        ;       (New page tables with read only copies of the parent's
        ;       pages.)
        ;       EAX = 0 -> Error (CF = 1)
        ;
        ; Modified Registers -> none (except EAX)
        ;
        call    allocate_page
        jc      short dpd_err
        ;
        push    ebp ; 20/07/2015
        push    esi
        push    edi
        push    ebx
        push    ecx
        mov     esi, [u.pgdir]
        mov     edi, eax
        push    eax ; save child's page directory address
        ; 31/08/2015
        ; copy PDE 0 from the parent's page dir to the child's page dir
        ; (use same system space for all user page tables)
        movsd
        mov     ebp, 1024*4096 ; pass the 1st 4MB (system space)
        mov     ecx, (PAGE_SIZE / 4) - 1 ; 1023
dpd_0:
        lodsd
        ;or     eax, eax
         ;jnz     short dpd_1
        test    al, PDE_A_PRESENT ;  bit 0 =  1
        jnz     short dpd_1
        ; 20/07/2015 (virtual address at the end of the page table)
        add     ebp, 1024*4096 ; page size * PTE count
        jmp     short dpd_2
dpd_1:
        and     ax, PDE_A_CLEAR ; 0F000h ; clear attribute bits
        mov     ebx, eax
        ; EBX = Parent's page table address
        call    duplicate_page_table
        jc      short dpd_p_err
        ; EAX = Child's page table address
        or      al, PDE_A_PRESENT + PDE_A_WRITE + PDE_A_USER
                        ; set bit 0, bit 1 and bit 2 to 1
                        ; (present, writable, user)
dpd_2:
        stosd
        loop    dpd_0
        ;
        pop     eax  ; restore child's page directory address
dpd_3:
        pop     ecx
        pop     ebx
        pop     edi
        pop     esi
        pop     ebp ; 20/07/2015
dpd_err:
        retn
```

```
dpd_p_err:
        ; release the allocated pages missing (recover free space)
        pop     eax  ; the new page directory address (physical)
        mov     ebx, [u.pgdir] ; parent's page directory address
        call    deallocate_page_dir
        sub     eax, eax ; 0
        stc
        jmp     short dpd_3

duplicate_page_table:
        ; 21/09/2015
        ; 20/07/2015
        ; 05/05/2015
        ; 28/04/2015
        ; 27/04/2015
        ; 18/04/2015
        ; 18/10/2014
        ; 16/10/2014 (Retro UNIX 386 v1 - beginning)
        ;
        ; INPUT ->
        ;       EBX = PHYSICAL (real/flat) ADDRESS of the parent's page table.
        ;       EBP = page table entry index (from 'duplicate_page_dir')
        ; OUTPUT ->
        ;       EAX = PHYSICAL (real/flat) ADDRESS of the child's page table.
        ;             (with 'read only' attribute of page table entries)
        ;       EBP = (recent) page table index (for 'add_to_swap_queue')
        ;       CF = 1 -> error
        ;
        ; Modified Registers -> EBP (except EAX)
        ;
        call    allocate_page
        jc      short dpt_err
        ;
        push    eax ; *
        push    esi
        push    edi
        push    edx
        push    ecx
        ;
        mov     esi, ebx
        mov     edi, eax
        mov     edx, eax
        add     edx, PAGE_SIZE
dpt_0:
        lodsd
        and     eax, eax
        jz      short dpt_3
        test    al, PTE_A_PRESENT ;  bit 0 =  1
        jnz     short dpt_1
        ; 20/07/2015
        ; ebp = virtual (linear) address of the memory page
        call    reload_page ; 28/04/2015
        jc      short dpt_p_err
dpt_1:
        ; 21/09/2015
        mov     ecx, eax
        and     ax, PTE_A_CLEAR ; 0F000h ; clear attribute bits
        test    cl, PTE_A_WRITE ; writable page ?
        jnz     short dpt_2
        ; Read only (parent) page
        ;       - there is a third process which uses this page -
        ; Allocate a new page for the child process
        call    allocate_page
        jc      short dpt_p_err
        push    edi
        push    esi
        mov     esi, ecx
        mov     edi, eax
        mov     ecx, PAGE_SIZE/4
        rep     movsd   ; copy page (4096 bytes)
        pop     esi
        pop     edi
        ;
        push    ebx
        push    eax
        ; 20/07/2015
        mov     ebx, ebp
        ; ebx = virtual address of the memory page
        call    add_to_swap_queue
```

```
        pop     eax
        pop     ebx
        ; 21/09/2015
        or      al, PTE_A_USER+PTE_A_WRITE+PTE_A_PRESENT
                ; user + writable + present page
        jmp     short dpt_3
dpt_2:
        ;or     ax, PTE_A_USER+PTE_A_PRESENT
        or      al, PTE_A_USER+PTE_A_PRESENT
                    ; (read only page!)
        mov     [esi-4], eax ; update parent's PTE
        or       ax, PTE_DUPLICATED  ; (read only page & duplicated PTE!)
dpt_3:
        stosd  ; EDI points to child's PTE
        ;
        add     ebp, 4096 ; 20/07/2015 (next page)
        ;
        cmp     edi, edx
        jb      short dpt_0
dpt_p_err:
        pop     ecx
        pop     edx
        pop     edi
        pop     esi
        pop     eax ; *
dpt_err:
        retn

page_fault_handler:    ; CPU EXCEPTION 0Eh (14) : Page Fault !
        ; 21/09/2015
        ; 19/09/2015
        ; 17/09/2015
        ; 28/08/2015
        ; 20/07/2015
        ; 28/06/2015
        ; 03/05/2015
        ; 30/04/2015
        ; 18/04/2015
        ; 12/04/2015
        ; 30/10/2014
        ; 11/09/2014
        ; 10/09/2014 (Retro UNIX 386 v1 - beginning)
        ;
        ; Note: This is not an interrupt/exception handler.
        ;       This is a 'page fault remedy' subroutine
        ;       which will be called by standard/uniform
        ;       exception handler.
        ;
        ; INPUT ->
        ;       [error_code] = 32 bit ERROR CODE (lower 5 bits are valid)
        ;
        ;       cr2 = the virtual (linear) address
        ;             which has caused to page fault (19/09/2015)
        ;
        ; OUTPUT ->
        ;       (corresponding PAGE TABLE ENTRY is mapped/set)
        ;       EAX = 0 -> no error
        ;       EAX > 0 -> error code in EAX (also CF = 1)
        ;
        ; Modified Registers -> none (except EAX)
        ;
        ;
        ; ERROR CODE:
        ;        31  .....     4   3   2   1   0
        ;       +---+-- --+---+---+---+---+---+
        ;       |   Reserved  | I | R | U | W | P |
        ;       +---+-- --+---+---+---+---+---+
        ;
        ; P : PRESENT -       When set, the page fault was caused by
        ;              a page-protection violation. When not set,
        ;              it was caused by a non-present page.
        ; W : WRITE   -       When set, the page fault was caused by
        ;              a page write. When not set, it was caused
        ;              by a page read.
        ; U : USER    -       When set, the page fault was caused
        ;              while CPL = 3.
        ;              This does not necessarily mean that
        ;              the page fault was a privilege violation.
        ; R : RESERVD -       When set, the page fault was caused by
```

```
;     WRITE    reading a 1 in a reserved field.
; I : INSTRUC - When set, the page fault was caused by
;     FETCH    an instruction fetch
;
;; x86 (32 bit) VIRTUAL ADDRESS TRANSLATION
;  31                22                    12 11                     0
; +-------------------+-------------------+-----------------------+
; | PAGE DIR. ENTRY # | PAGE TAB. ENTRY # |        OFFSET         |
; +-------------------+-------------------+-----------------------+
;

;; CR3 REGISTER (Control Register 3)
;  31                                    12             5 4 3 2   0
; +---------------------------------------+-------------+---+-----+
; |                                       |             |P|P|     |
; |   PAGE DIRECTORY TABLE BASE ADDRESS   |  reserved   |C|W|rsvrd|
; |                                       |         |D|T|     |
; +---------------------------------------+-------------+---+-----+
;
;      PWT    - WRITE THROUGH
;      PCD    - CACHE DISABLE
;
;
;; x86 PAGE DIRECTORY ENTRY (4 KByte Page)
;  31                                    12 11  9 8 7 6 5 4 3 2 1 0
; +---------------------------------------+-----+---+-+-+---+-+-+-+
; |                                       |     | | | | |P|P|U|R| |
; |     PAGE TABLE BASE ADDRESS 31..12    | AVL |G|0|D|A|C|W|/|/|P|
; |                                       |     | | | | |D|T|S|W| |
; +---------------------------------------+-----+---+-+-+---+-+-+-+
;
 ;       P      - PRESENT
 ;       R/W    - READ/WRITE
 ;       U/S    - USER/SUPERVISOR
;      PWT    - WRITE THROUGH
;      PCD    - CACHE DISABLE
;      A      - ACCESSED
 ;       D      - DIRTY (IGNORED)
;      PAT    - PAGE ATTRIBUTE TABLE INDEX (CACHE BEHAVIOR)
;      G      - GLOBAL       (IGNORED)
 ;       AVL    - AVAILABLE FOR SYSTEMS PROGRAMMER USE
;
;
;; x86 PAGE TABLE ENTRY (4 KByte Page)
;  31                                    12 11  9 8 7 6 5 4 3 2 1 0
; +---------------------------------------+-----+---+-+-+---+-+-+-+
; |                                       |     | |P| | |P|P|U|R| |
; |     PAGE FRAME BASE ADDRESS 31..12    | AVL |G|A|D|A|C|W|/|/|P|
; |                                       |     | |T| | |D|T|S|W| |
; +---------------------------------------+-----+---+-+-+---+-+-+-+
;
 ;       P      - PRESENT
 ;       R/W    - READ/WRITE
 ;       U/S    - USER/SUPERVISOR
;      PWT    - WRITE THROUGH
;      PCD    - CACHE DISABLE
;      A      - ACCESSED
 ;       D      - DIRTY
;      PAT    - PAGE ATTRIBUTE TABLE INDEX (CACHE BEHAVIOR)
;      G      - GLOBAL
 ;       AVL    - AVAILABLE FOR SYSTEMS PROGRAMMER USE
;
;
;; 80386 PAGE TABLE ENTRY (4 KByte Page)
;  31                                    12 11  9 8 7 6 5 4 3 2 1 0
; +---------------------------------------+-----+-+-+-+-+---+-+-+-+
; |                                       |     | | | | | | |U|R| |
; |     PAGE FRAME BASE ADDRESS 31..12    | AVL |0|0|D|A|0|0|/|/|P|
; |                                       |     | | | | | | |S|W| |
; +---------------------------------------+-----+-+-+-+-+---+-+-+-+
;
 ;       P      - PRESENT
 ;       R/W    - READ/WRITE
 ;       U/S    - USER/SUPERVISOR
 ;       D      - DIRTY
 ;       AVL    - AVAILABLE FOR SYSTEMS PROGRAMMER USE
;
 ;       NOTE: 0 INDICATES INTEL RESERVED. DO NOT DEFINE.
;
```

```
        ;
        ;; Invalid Page Table Entry
        ; 31                                                          1 0
        ; +--------------------------------------------------------------+-+
        ; |                                                              | |
        ; |                          AVAILABLE                           |0|
        ; |                                                              | |
        ; +--------------------------------------------------------------+-+
        ;

        push    ebx
        push    edx
        push    ecx
        ;
        ; 21/09/2015 (debugging)
        inc     dword [u.pfcount] ; page fault count for running process
        inc     dword [PF_Count] ; total page fault count
        ; 28/06/2015
        ;mov    edx, [error_code] ; Lower 5 bits are valid
        mov     dl, [error_code]
        ;
        test    dl, 1   ; page fault was caused by a non-present page
                        ; sign
        jz      short pfh_alloc_np
        ;
        ; If it is not a 'write on read only page' type page fault
        ; major page fault error with minor reason must be returned without
        ; fixing the problem. 'sys_exit with error' will be needed
        ; after return here!
        ; Page fault will be remedied, by copying page contents
        ; to newly allocated page with write permission;
        ; sys_fork -> sys_exec -> copy on write, demand paging method is
        ; used for working with minimum possible memory usage.
        ; sys_fork will duplicate page directory and tables of parent
        ; process with 'read only' flag. If the child process attempts to
        ; write on these read only pages, page fault will be directed here
        ; for allocating a new page with same data/content.
        ;
        ; IMPORTANT : Retro UNIX 386 v1 (and SINGLIX and TR-DOS)
        ; will not force to separate CODE and DATA space
        ; in a process/program...
        ; CODE segment/section may contain DATA!
        ; It is flat, smoth and simplest programming method already as in
        ; Retro UNIX 8086 v1 and MS-DOS programs.
        ;
        test    dl, 2   ; page fault was caused by a page write
                        ; sign
         jz      pfh_p_err
        ; 31/08/2015
        test    dl, 4   ; page fault was caused while CPL = 3 (user mode)
                        ; sign.  (U+W+P = 4+2+1 = 7)
         jz     pfh_pv_err
        ;
        ; make a new page and copy the parent's page content
        ; as the child's new page content
        ;
        mov     ebx, cr2 ; CR2 contains the linear address
                         ; which has caused to page fault
        call    copy_page
         jc      pfh_im_err ; insufficient memory
        ;
         jmp     pfh_cpp_ok
        ;
pfh_alloc_np:
        call    allocate_page  ; (allocate a new page)
         jc      pfh_im_err    ; 'insufficient memory' error
pfh_chk_cpl:
        ; EAX = Physical (base) address of the allocated (new) page
                ; (Lower 12 bits are ZERO, because
                ;       the address is on a page boundary)
        and     dl, 4   ; CPL = 3 ?
        jnz     short pfh_um
                        ; Page fault handler for kernel/system mode (CPL=0)
        mov     ebx, cr3 ; CR3 (Control Register 3) contains physical address
                         ; of the current/active page directory
                         ; (Always kernel/system mode page directory, here!)
                         ; Note: Lower 12 bits are 0. (page boundary)
        jmp     short pfh_get_pde
        ;
```

```
pfh_um:               ; Page fault handler for user/appl. mode (CPL=3)
       mov    ebx, [u.pgdir] ; Page directory of current/active process
                      ; Physical address of the USER's page directory
                      ; Note: Lower 12 bits are 0. (page boundary)
pfh_get_pde:
       or     dl, 3   ; USER + WRITE + PRESENT or SYSTEM + WRITE + PRESENT
       mov    ecx, cr2 ; CR2 contains the virtual address
                       ; which has been caused to page fault
                       ;
       shr    ecx, 20 ; shift 20 bits right
       and    cl, 0FCh ; mask lower 2 bits to get PDE offset
       ;
       add    ebx, ecx ; now, EBX points to the relevant page dir entry
       mov    ecx, [ebx] ; physical (base) address of the page table
       test   cl, 1    ; check bit 0 is set (1) or not (0).
       jz     short pfh_set_pde ; Page directory entry is not valid,
                                ; set/validate page directory entry
       and    cx, PDE_A_CLEAR ; 0F000h ; Clear attribute bits
       mov    ebx, ecx ; Physical address of the page table
       mov    ecx, eax ; new page address (physical)
       jmp    short pfh_get_pte
pfh_set_pde:
       ;; NOTE: Page directories and page tables never be swapped out!
       ;;      (So, we know this PDE is empty or invalid)
       ;
       or     al, dl   ; lower 3 bits are used as U/S, R/W, P flags
       mov    [ebx], eax ; Let's put the new page directory entry here !
       xor    al, al   ; clear lower (3..8) bits
       mov    ebx, eax
       call   allocate_page   ; (allocate a new page)
       jc     short pfh_im_err   ; 'insufficient memory' error
pfh_spde_1:
       ; EAX = Physical (base) address of the allocated (new) page
       mov    ecx, eax
       call   clear_page ; Clear page content
pfh_get_pte:
       mov    eax, cr2 ; virtual address
                       ; which has been caused to page fault
       mov    edi, eax ; 20/07/2015
       shr    eax, 12  ; shift 12 bit right to get
                       ; higher 20 bits of the page fault address
       and    eax, 3FFh ; mask PDE# bits, the result is PTE# (0 to 1023)
       shl    eax, 2 ; shift 2 bits left to get PTE offset
       add    ebx, eax ; now, EBX points to the relevant page table entry
       mov    eax, [ebx] ; get previous value of pte
              ; bit 0 of EAX is always 0 (otherwise we would not be here)
       and    eax, eax
       jz     short pfh_gpte_1
       ; 20/07/2015
       xchg   ebx, ecx ; new page address (physical)
       push   ebp ; 20/07/2015
       mov    ebp, cr2
              ; ECX = physical address of the page table entry
              ; EBX = Memory page address (physical!)
              ; EAX = Swap disk (offset) address
              ; EBP = virtual address (page fault address)
       call   swap_in
       pop    ebp
       jc      short pfh_err_retn
       xchg   ecx, ebx
              ; EBX = physical address of the page table entry
              ; ECX = new page
pfh_gpte_1:
       or     cl, dl ; lower 3 bits are used as U/S, R/W, P flags
       mov    [ebx], ecx ; Let's put the new page table entry here !
pfh_cpp_ok:
       ; 20/07/2015
       mov    ebx, cr2
       call   add_to_swap_queue
       ;
       ; The new PTE (which contains the new page) will be added to
       ; the swap queue, here.
       ; (Later, if memory will become insufficient,
       ; one page will be swapped out which is at the head of
       ; the swap queue by using FIFO and access check methods.)
       ;
       xor    eax, eax  ; 0
       ;
```

```
pfh_err_retn:
        pop     ecx
        pop     edx
        pop     ebx
        retn

pfh_im_err:
        mov     eax, ERR_MAJOR_PF + ERR_MINOR_IM ; Error code in AX
                        ; Major (Primary) Error: Page Fault
                        ; Minor (Secondary) Error: Insufficient Memory !
        jmp     short pfh_err_retn


pfh_p_err: ; 09/03/2015
pfh_pv_err:
        ; Page fault was caused by a protection-violation
        mov     eax, ERR_MAJOR_PF + ERR_MINOR_PV ; Error code in AX
                        ; Major (Primary) Error: Page Fault
                        ; Minor (Secondary) Error: Protection violation !
        stc
        jmp     short pfh_err_retn

copy_page:
        ; 22/09/2015
        ; 21/09/2015
        ; 19/09/2015
        ; 07/09/2015
        ; 31/08/2015
        ; 20/07/2015
        ; 05/05/2015
        ; 03/05/2015
        ; 18/04/2015
        ; 12/04/2015
        ; 30/10/2014
        ; 18/10/2014 (Retro UNIX 386 v1 - beginning)
        ;
        ; INPUT ->
        ;       EBX = Virtual (linear) address of source page
        ;            (Page fault address)
        ; OUTPUT ->
        ;       EAX = PHYSICAL (real/flat) ADDRESS OF THE ALLOCATED PAGE
        ;       (corresponding PAGE TABLE ENTRY is mapped/set)
        ;       EAX = 0 (CF = 1)
        ;               if there is not a free page to be allocated
        ;       (page content of the source page will be copied
        ;       onto the target/new page)
        ;
        ; Modified Registers -> ecx, ebx (except EAX)
        ;
        push    esi
        push    edi
        ;push   ebx
        ;push   ecx
        xor     esi, esi
        shr     ebx, 12 ; shift 12 bits right to get PDE & PTE numbers
        mov     ecx, ebx ; save page fault address (as 12 bit shifted)
        shr     ebx, 8   ; shift 8 bits right and then
        and     bl, 0FCh ; mask lower 2 bits to get PDE offset
        mov     edi, ebx ; save it for the parent of current process
        add     ebx, [u.pgdir] ; EBX points to the relevant page dir entry
        mov     eax, [ebx] ; physical (base) address of the page table
        and     ax, PTE_A_CLEAR ; 0F000h ; clear attribute bits
        mov     ebx, ecx   ; (restore higher 20 bits of page fault address)
        and     ebx, 3FFh  ; mask PDE# bits, the result is PTE# (0 to 1023)
        shl     bx, 2      ; shift 2 bits left to get PTE offset
        add     ebx, eax   ; EBX points to the relevant page table entry
        ; 07/09/2015
         test    word [ebx], PTE_DUPLICATED ; (Does current process share this
                                ; read only page as a child process?)
        jnz     short cpp_0 ; yes
        mov     ecx, [ebx] ; PTE value
        and     cx, PTE_A_CLEAR ; 0F000h  ; clear page attributes
        jmp     short cpp_1
cpp_0:
        mov     esi, edi
        add     esi, [u.ppgdir] ; the parent's page directory entry
        mov     eax, [esi] ; physical (base) address of the page table
        and     ax, PTE_A_CLEAR ; 0F000h ; clear attribute bits
        mov     esi, ecx   ; (restore higher 20 bits of page fault address)
```

```
        and     esi, 3FFh  ; mask PDE# bits, the result is PTE# (0 to 1023)
        shl     si, 2     ; shift 2 bits left to get PTE offset
        add     esi, eax   ; EDX points to the relevant page table entry
        mov     ecx, [esi] ; PTE value of the parent process
        ; 21/09/2015
        mov     eax, [ebx] ; PTE value of the child process
        and     ax, PTE_A_CLEAR ; 0F000h ; clear page attributes
        ;
        test    cl, PTE_A_PRESENT ; is it a present/valid page ?
        jz      short cpp_3 ; the parent's page is not same page
        ;
        and     cx, PTE_A_CLEAR ; 0F000h ; clear page attributes
        cmp     eax, ecx   ; Same page?
        jne     short cpp_3 ; Parent page and child page are not same
                           ; Convert child's page to writable page
cpp_1:
        call    allocate_page
        jc      short cpp_4 ; 'insufficient memory' error
        and     esi, esi    ; check ESI is valid or not
        jz      short cpp_2
                ; Convert read only page to writable page
                ;(for the parent of the current process)
        ;and    word [esi], PTE_A_CLEAR ; 0F000h
        ; 22/09/2015
        mov     [esi], ecx
        or      byte [esi], PTE_A_PRESENT + PTE_A_WRITE + PTE_A_USER
                                ; 1+2+4 = 7
cpp_2:
        mov     edi, eax ; new page address of the child process
        ; 07/09/2015
        mov     esi, ecx ; the page address of the parent process
        mov     ecx, PAGE_SIZE / 4
        rep     movsd ; 31/08/2015
cpp_3:
        or      al, PTE_A_PRESENT + PTE_A_WRITE + PTE_A_USER ; 1+2+4 = 7
        mov     [ebx], eax ; Update PTE
        sub     al, al ; clear attributes
cpp_4:
        ;pop    ecx
        ;pop    ebx
        pop     edi
        pop     esi
        retn

;; 28/04/2015
;; 24/10/2014
;; 21/10/2014 (Retro UNIX 386 v1 - beginning)
;; SWAP_PAGE_QUEUE (4096 bytes)
;;
;;   0000   0001   0002   0003   ....   1020   1021   1022   1023
;; +------+------+------+------+-    -+------+------+------+------+
;; |  pg1 |  pg2 |  pg3 |  pg4 | .... |pg1021|pg1022|pg1023|pg1024|
;; +------+------+------+------+-    -+------+------+------+------+
;;
;; [swpq_last] = 0 to 4096 (step 4) -> the last position on the queue
;;
;; Method:
;;      Swap page queue is a list of allocated pages with physical
;;      addresses (system mode virtual adresses = physical addresses).
;;      It is used for 'swap_in' and 'swap_out' procedures.
;;      When a new page is being allocated, swap queue is updated
;;      by 'swap_queue_shift' procedure, header of the queue (offset 0)
;;      is checked for 'accessed' flag. If the 1st page on the queue
;;      is 'accessed' or 'read only', it is dropped from the list;
;;      other pages from the 2nd to the last (in [swpq_last]) shifted
;;      to head then the 2nd page becomes the 1st and '[swpq_last]'
;;      offset value becomes it's previous offset value - 4.
;;      If the 1st page of the swap page queue is not 'accessed'
;;      the queue/list is not shifted.
;;      After the queue/list shift, newly allocated page is added
;;      to the tail of the queue at the [swpq_count*4] position.
;;      But, if [swpq_count] > 1023, the newly allocated page
;;      will not be added to the tail of swap page queue.
;;
;;      During 'swap_out' procedure, swap page queue is checked for
;;      the first non-accessed, writable page in the list,
;;      from the head to the tail. The list is shifted to left
;;      (to the head) till a non-accessed page will be found in the list.
```

```
;;      Then, this page is swapped out (to disk) and then it is dropped
;;      from the list by a final swap queue shift. [swpq_count] value
;;      is changed. If all pages on the queue' are 'accessed',
;;      'insufficient memory' error will be returned ('swap_out'
;;      procedure will be failed)...
;;
;;      Note: If the 1st page of the queue is an 'accessed' page,
;;      'accessed' flag of the page will be reset (0) and that page
;;      (PTE) will be added to the tail of the queue after
;;      the check, if [swpq_count] < 1023. If [swpq_count] = 1024
;;      the queue will be rotated and the PTE in the head will be
;;      added to the tail after resetting 'accessed' bit.
;;
;;
;;
;; SWAP DISK/FILE (with 4096 bytes swapped page blocks)
;;
;;  00000000  00000004  00000008  0000000C   ...   size-8    size-4
;; +---------+---------+---------+---------+-- --+---------+---------+
;; |descriptr| page(1) | page(2) | page(3) | ... |page(n-1)| page(n) |
;; +---------+---------+---------+---------+-- --+---------+---------+
;;
;; [swpd_next] = the first free block address in swapped page records
;;              for next free block search by 'swap_out' procedure.
;; [swpd_size] = swap disk/file size in sectors (512 bytes)
;;              NOTE: max. possible swap disk size is 1024 GB
;;              (entire swap space must be accessed by using
;;              31 bit offset address)
;; [swpd_free] = free block (4096 bytes) count in swap disk/file space
;; [swpd_start] = absolute/start address of the swap disk/file
;;               0 for file, or beginning sector of the swap partition
;; [swp_drv] = logical drive description table addr. of swap disk/file
;;
;;
;; Method:
;;     When the memory (ram) becomes insufficient, page allocation
;;     procedure swaps out a page from memory to the swap disk
;;     (partition) or swap file to get a new free page at the memory.
;;     Swapping out is performed by using swap page queue.
;;
;;     Allocation block size of swap disk/file is equal to page size
;;     (4096 bytes). Swapping address (in sectors) is recorded
;;     into relevant page file entry as 31 bit physical (logical)
;;     offset address as 1 bit shifted to left for present flag (0).
;;     Swapped page address is between 1 and swap disk/file size - 4.
;;     Absolute physical (logical) address of the swapped page is
;;     calculated by adding offset value to the swap partition's
;;     start address. If the swap device (disk) is a virtual disk
;;     or it is a file, start address of the swap disk/volume is 0,
;;     and offset value is equal to absolute (physical or logical)
;;     address/position. (It has not to be ZERO if the swap partition
;;     is in a partitioned virtual hard disk.)
;;
;;     Note: Swap addresses are always specified/declared in sectors,
;;     not in bytes or       in blocks/zones/clusters (4096 bytes) as unit.
;;
;;     Swap disk/file allocation is mapped via 'Swap Allocation Table'
;;     at memory as similar to 'Memory Allocation Table'.
;;
;;     Every bit of Swap Allocation Table repsesents one swap block
;;     (equal to page size) respectively. Bit 0 of the S.A.T. byte 0
;;     is reserved for swap disk/file block 0 as descriptor block
;;     (also for compatibility with PTE). If bit value is ZERO,
;;     it means relevant (respective) block is in use, and,
;;     of course, if bit value is 1, it means relevant (respective)
;;      swap disk/file block is free.
;;     For example: bit 1 of the byte 128 repsesents block 1025
;;     (128*8+1) or sector (offset) 8200 on the swap disk or
;;     byte (offset/position) 4198400 in the swap file.
;;     4GB swap space is represented via 128KB Swap Allocation Table.
;;     Initial layout of Swap Allocation Table is as follows:
;;     ------------------------------------------------------------
;;     0111111111111111111111111 .... 11111111111111111111111111111
;;     ------------------------------------------------------------
;;     (0 is reserved block, 1s represent free blocks respectively.)
;;     (Note: Allocation cell/unit of the table is bit, not byte)
;;
```

```
;;      ............................................................
;;
;;      'swap_out' procedure checks 'free_swap_blocks' count at first,
;;      then it searches Swap Allocation Table if free count is not
;;      zero. From begining the [swpd_next] dword value, the first bit
;;      position with value of 1 on the table is converted to swap
;;      disk/file offset address, in sectors (not 4096 bytes block).
;;      'ldrv_write' procedure is called with ldrv (logical drive
;;      number of physical swap disk or virtual swap disk)
;;      number, sector offset (not absolute sector -LBA- number),
;;      and sector count (8, 512*8 = 4096) and buffer adress
;;      (memory page). That will be a direct disk write procedure.
;;      (for preventing late memory allocation, significant waiting).
;;      If disk write procedure returns with error or free count of
;;      swap blocks is ZERO, 'swap_out' procedure will return with
;;      'insufficient memory error' (cf=1).
;;
;;      (Note: Even if free swap disk/file blocks was not zero,
;;      any disk write error will not be fixed by 'swap_out' procedure,
;;      in other words, 'swap_out' will not check the table for other
;;      free blocks after a disk write error. It will return to
;;      the caller with error (CF=1) which means swapping is failed.
;;
;;      After writing the page on to swap disk/file address/sector,
;;      'swap_out' procesure returns with that swap (offset) sector
;;      address (cf=0).
;;
;;      ............................................................
;;
;;      'swap_in' procedure loads addressed (relevant) swap disk or
;;      file sectors at specified memory page. Then page allocation
;;      procedure updates relevant page table entry with 'present'
;;      attribute. If swap disk or file reading fails there is nothing
;;      to do, except to terminate the process which is the owner of
;;      the swapped page.
;;
;;      'swap_in' procedure sets the relevant/respective bit value
;;      in the Swap Allocation Table (as free block). 'swap_in' also
;;      updates [swpd_first] pointer if it is required.
;;
;;      ............................................................
;;
;;      Note: If [swap_enabled] value is ZERO, that means there is not
;;      a swap disk or swap file in use... 'swap_in' and 'swap_out'
;;      procedures ans 'swap page que' procedures will not be active...
;;      'Insufficient memory' error will be returned by 'swap_out'
;;      and 'general protection fault' will be returned by 'swap_in'
;;      procedure, if it is called mistakenly (a wrong value in a PTE).
;;

swap_in:
        ; 31/08/2015
        ; 20/07/2015
        ; 28/04/2015
        ; 18/04/2015
        ; 24/10/2014 (Retro UNIX 386 v1 - beginning)
        ;
        ; INPUT ->
        ;       EBX = PHYSICAL (real/flat) ADDRESS OF THE MEMORY PAGE
        ;       EBP = VIRTUAL (LINEAR) ADDRESS (page fault address)
        ;       EAX = Offset Address for the swapped page on the
        ;             swap disk or in the swap file.
        ;
        ; OUTPUT ->
        ;       EAX = 0 if loading at memory has been successful
        ;
        ;       CF = 1 -> swap disk reading error (disk/file not present
        ;                 or sector not present or drive not ready
        ;            EAX = Error code
        ;            [u.error] = EAX
        ;                     = The last error code for the process
        ;                       (will be reset after returning to user)
        ;
        ; Modified Registers -> EAX
        ;

        cmp     dword [swp_drv], 0
        jna     short swpin_dnp_err
```

```
        cmp     eax, [swpd_size]
        jnb     short swpin_snp_err

        push    esi
        push    ebx
        push    ecx
        mov     esi, [swp_drv]
        mov     ecx, PAGE_SIZE / LOGIC_SECT_SIZE  ; 8 !
                ; Note: Even if corresponding physical disk's sector
                ; size different than 512 bytes, logical disk sector
                ; size is 512 bytes and disk reading procedure
                ; will be performed for reading 4096 bytes
                ; (2*2048, 8*512).
        ; ESI = Logical disk description table address
        ; EBX = Memory page (buffer) address (physical!)
        ; EAX = Sector adress (offset address, logical sector number)
        ; ECX = Sector count ; 8 sectors
        push    eax
        call    logical_disk_read
        pop     eax
        jnc     short swpin_read_ok
        ;
        mov     eax, SWP_DISK_READ_ERR ; drive not ready or read error
        mov     [u.error], eax
        jmp     short swpin_retn
        ;
swpin_read_ok:
        ; EAX = Offset address (logical sector number)
        call    unlink_swap_block  ; Deallocate swap block
        ;
        ; EBX = Memory page (buffer) address (physical!)
        ; 20/07/2015
        mov     ebx, ebp ; virtual address (page fault address)
         and     bx, ~PAGE_OFF ; ~0FFFh ; reset bits, 0 to 11
        mov     bl, [u.uno] ; current process number
        ; EBX = Virtual address & process number combination
        call    swap_queue_shift
        sub     eax, eax  ; 0 ; Error Code = 0  (no error)
        ;
swpin_retn:
        pop     ecx
        pop     ebx
        pop     esi
        retn

swpin_dnp_err:
        mov     eax, SWP_DISK_NOT_PRESENT_ERR
swpin_err_retn:
        mov     [u.error], eax
        stc
        retn

swpin_snp_err:
        mov     eax, SWP_SECTOR_NOT_PRESENT_ERR
        jmp     short swpin_err_retn

swap_out:
        ; 31/08/2015
        ; 05/05/2015
        ; 30/04/2015
        ; 28/04/2015
        ; 18/04/2015
        ; 24/10/2014 (Retro UNIX 386 v1 - beginning)
        ;
        ; INPUT ->
        ;       none
        ;
        ; OUTPUT ->
        ;       EAX = Physical page address (which is swapped out
        ;             for allocating a new page)
        ;       CF = 1 -> swap disk writing error (disk/file not present
        ;                 or sector not present or drive not ready
        ;            EAX = Error code
        ;            [u.error] = EAX
        ;                     = The last error code for the process
        ;                       (will be reset after returning to user)
        ;
        ; Modified Registers -> non (except EAX)
        ;
```

```
        cmp     word [swpq_count], 1
         jc      short swpout_im_err ; 'insufficient memory'

         ;cmp     dword [swp_drv], 1
        ;jc     short swpout_dnp_err ; 'swap disk/file not present'

         cmp     dword [swpd_free], 1
        jc      short swpout_nfspc_err ; 'no free space on swap disk'

        push    ebx
swpout_1:
        xor     ebx, ebx
        call    swap_queue_shift
        and     eax, eax        ; entry count (before shifting)
        jz      short swpout_npts_err  ; There is no any PTE in
                                     ; the swap queue
        mov     ebx, swap_queue              ; Addres of the head of
                                     ; the swap queue
        mov     eax, [ebx]           ; The PTE in the queue head

        ;test   al, PTE_A_PRESENT      ; bit 0 = 1
        ;jz     short swpout_1       ; non-present page already
                                     ; must not be in the queue

        ;test   al, PTE_A_WRITE              ; bit 1 = 0
        ;jz     short swpout_1               ; read only page (must not be
                                     ; swapped out)

        test    al, PTE_A_ACCESS       ; bit 5 = 1 (Accessed)
        jnz     short swpout_1               ; accessed page (must not be
                                     ; swapped out, at this stage)
        ;
        and     ax, PTE_A_CLEAR ; 0F000h ; clear attribute bits
        ;
        push    edx
        mov     edx, ebx             ; Page table entry address
        mov     ebx, eax             ; Buffer (Page) Address
        ;
        call    link_swap_block
        jnc     short swpout_2       ; It may not be needed here
        pop     edx                  ; because [swpd_free] value
        pop     ebx
        jmp     short swpout_nfspc_err ; was checked at the beginging.
swpout_2:
        push    esi
        push    ecx
        push    eax ; sector address
        mov     esi, [swp_drv]
        mov     ecx, PAGE_SIZE / LOGIC_SECT_SIZE  ; 8 !
                ; Note: Even if corresponding physical disk's sector
                ; size different than 512 bytes, logical disk sector
                ; size is 512 bytes and disk writing procedure
                ; will be performed for writing 4096 bytes
                ; (2*2048, 8*512).
        ; ESI = Logical disk description table address
        ; EBX = Buffer address
        ; EAX = Sector adress (offset address, logical sector number)
        ; ECX = Sector count ; 8 sectors
        call    logical_disk_write
        pop     ecx ; sector address
        jnc     short swpout_write_ok
        ;
        ;; call unlink_swap_block ; this block must be left as 'in use'
swpout_dw_err:
        mov     eax, SWP_DISK_WRITE_ERR ; drive not ready or write error
        mov     [u.error], eax
        jmp     short swpout_retn
        ;
swpout_write_ok:
        ; EBX = Buffer (page) address
        ; EDX = Page Table entry address
        ; ECX = Swap disk sector (file block) address (31 bit)
        shl     ecx, 1  ; 31 bit sector address from bit 1 to bit 31
        mov     [edx], ecx
                ; bit 0 = 0 (swapped page)
        mov     eax, ebx
swpout_retn:
        pop     ecx
        pop     esi
```

```
        pop     edx
        pop     ebx
        retn

; Note: Swap_queue will not be updated in 'swap_out' procedure
;       after the page is swapped out. (the PTE at the queue head
;       -with 'non-present' attribute- will be dropped from the
;       the queue in next 'swap_out' or in next 'swap_queue_shift'.

;swpout_dnp_err:
;       mov     eax, SWP_DISK_NOT_PRESENT_ERR ; disk not present
;       jmp     short swpout_err_retn
swpout_nfspc_err:
        mov     eax, SWP_NO_FREE_SPACE_ERR ; no free space
swpout_err_retn:
        mov     [u.error], eax
        ;stc
        retn
swpout_npts_err:
        mov     eax, SWP_NO_PAGE_TO_SWAP_ERR
        pop     ebx
        jmp     short swpout_err_retn
swpout_im_err:
        mov     eax, ERR_MINOR_IM ; insufficient (out of) memory
        jmp     short swpout_err_retn

swap_queue_shift:
        ; 20/07/2015
        ; 28/04/2015
        ; 18/04/2015
        ; 23/10/2014 (Retro UNIX 386 v1 - beginning)
        ;
        ; INPUT ->
        ;       EBX = Virtual (linear) address (bit 12 to 31)
        ;             and process number combination (bit 0 to 11)
        ;       EBX = 0 -> shift/drop from the head (offset 0)
        ; OUTPUT ->
        ;       If EBX input > 0
        ;           the queue will be shifted 4 bytes (dword),
        ;           from the tail to the head, up to entry offset
        ;           which points to EBX input value or nothing
        ;           to do if EBX value is not found in the queue.
        ;           (The entry -with EBX value- will be removed
        ;            from the queue if it is found.)
        ;       If EBX input = 0
        ;           the queue will be shifted 4 bytes (dword),
        ;           from the tail to the head, if the PTE address
        ;           in head of the queue is marked as "accessed"
        ;           or it is marked as "non present".
        ;           (If "accessed" flag of the PTE -in the head-
        ;           is set -to 1-, it will be reset -to 0- and then,
        ;           the queue will be rotated -without dropping
        ;           the PTE from the queue-, for 4 bytes on head
        ;           to tail direction. The PTE in the head will be
        ;           moved in the tail, other PTEs will be shifted on
        ;           head direction.)
        ;
        ;       EAX = [swpq_count] (before the shifting)
        ;           (EAX = 0 -> next 'swap_out' stage
        ;            is not applicable)
        ;
        ; Modified Registers -> EAX
        ;
        movzx   eax, word [swpq_count]  ; Max. 1024
        and     ax, ax
        jz      short swpqs_retn
        push    edi
        push    esi
        push    ebx
        push    ecx
        push    eax
        mov     esi, swap_queue
        mov     ecx, eax
        or      ebx, ebx
        jz      short swpqs_7
```

```
swpqs_1:
        lodsd
        cmp     eax, ebx
        je      short swpqs_2
        loop    swpqs_1
        jmp     short swpqs_6
swpqs_2:
        mov     edi, esi
        sub     edi, 4
swpqs_3:
        dec     word [swpq_count]
        jz      short swpqs_5
swpqs_4:
        dec     ecx
        rep     movsd   ; shift up (to the head)
swpqs_5:
        xor     eax, eax
        mov     [edi], eax
swpqs_6:
        pop     eax
        pop     ecx
        pop     ebx
        pop     esi
        pop     edi
swpqs_retn:
        retn
swpqs_7:
        mov     edi, esi ; head
        lodsd
        ; 20/07/2015
        mov     ebx, eax
        and     ebx, ~PAGE_OFF ; ~0FFFh
                      ; ebx = virtual address (at page boundary)
        and     eax, PAGE_OFF ; 0FFFh
                      ; ax = process number (1 to 4095)
        cmp     al, [u.uno]
                ; Max. 16 (nproc) processes for Retro UNIX 386 v1
        jne     short swpqs_8
        mov     eax, [u.pgdir]
        jmp     short swpqs_9
swpqs_8:
        ;shl    ax, 2
        shl     al, 2
        mov     eax, [eax+p.upage-4]
        or      eax, eax
        jz      short swpqs_3 ; invalid upage
        add     eax, u.pgdir - user
                        ; u.pgdir value for the process
                        ; is in [eax]
        mov     eax, [eax]
        and     eax, eax
        jz      short swpqs_3 ; invalid page directory
swpqs_9:
        push    edx
        ; eax = page directory
        ; ebx = virtual address
        call    get_pte
        mov     ebx, edx ; PTE address
        pop     edx
        jc      short swpqs_3 ; empty PDE
        ; EAX = PTE value
        test    al, PTE_A_PRESENT ; bit 0 = 1
        jz      short swpqs_3 ; Drop non-present page
                              ; from the queue (head)
        test    al, PTE_A_WRITE         ; bit 1 = 0
        jz      short swpqs_3 ; Drop read only page
                              ; from the queue (head)
        ;test   al, PTE_A_ACCESS  ; bit 5 = 1 (Accessed)
        ;jz     short swpqs_6 ; present
                              ; non-accessed page
         btr     eax, PTE_A_ACCESS_BIT ; reset 'accessed' bit
        jnc     short swpqs_6  ; non-accessed page
        mov     [ebx], eax     ; save changed attribute
        ;
        ; Rotation (head -> tail)
        dec     ecx      ; entry count -> last entry number
        jz      short swpqs_6
                ; esi = head + 4
                ; edi = head
```

```
        mov     eax, [edi] ; 20/07/2015
        rep     movsd    ; n = 1 to k-1, [n - 1] = [n]
        mov     [edi], eax ; head -> tail ; [k] = [1]
        jmp     short swpqs_6

add_to_swap_queue:
; temporary - 16/09/2015
retn
        ; 20/07/2015
        ; 24/10/2014 (Retro UNIX 386 v1 - beginning)
        ;
        ; Adds new page to swap queue
        ; (page directories and page tables must not be added
        ; to swap queue)
        ;
        ; INPUT ->
        ;       EBX = Virtual address (for current process, [u.uno])
        ;
        ; OUTPUT ->
        ;       EAX = [swpq_count]
        ;             (after the PTE has been added)
        ;       EAX = 0 -> Swap queue is full, (1024 entries)
        ;             the pte could not be added.
        ;
        ; Modified Registers -> EAX
        ;
        push    ebx
         and     bx, ~PAGE_OFF ; ~0FFFh ; reset bits, 0 to 11
        mov     bl, [u.uno] ; current process number
        call    swap_queue_shift ; drop from the queue if
                                ; it is already in the queue
                ; Then add it to the tail of the queue
        movzx   eax, word [swpq_count]
        cmp     ax, 1024
        jb      short atsq_1
        sub     ax, ax
        pop     ebx
        retn
atsq_1:
        push    esi
        mov     esi, swap_queue
        and     ax, ax
        jz      short atsq_2
        shl     ax, 2   ; convert to offset
        add     esi, eax
        shr     ax, 2
atsq_2:
        inc     ax
        mov     [esi], ebx ; Virtual address + [u.uno] combination
        mov     [swpq_count], ax
        pop     esi
        pop     ebx
        retn

unlink_swap_block:
        ; 15/09/2015
        ; 30/04/2015
        ; 18/04/2015
        ; 24/10/2014 (Retro UNIX 386 v1 - beginning)
        ;
        ; INPUT ->
        ;       EAX = swap disk/file offset address
        ;             (bit 1 to bit 31)
        ; OUTPUT ->
        ;       [swpd_free] is increased
        ;       (corresponding SWAP DISK ALLOC. TABLE bit is SET)
        ;
        ; Modified Registers -> EAX
        ;
        push    ebx
        push    edx
        ;
        shr     eax, SECTOR_SHIFT+1  ;3+1 ; shift sector address to
                                    ; 3 bits right
                                    ; to get swap block/page number
        mov     edx, eax
        ; 15/09/2015
        shr     edx, 3              ; to get offset to S.A.T.
                                    ; (1 allocation bit = 1 page)
```

```
                                ; (1 allocation bytes = 8 pages)
        and     dl, 0FCh        ; clear lower 2 bits
                                ; (to get 32 bit position)
        ;
        mov     ebx, swap_alloc_table ; Swap Allocation Table address
        add     ebx, edx
        and     eax, 1Fh        ; lower 5 bits only
                                ; (allocation bit position)
        cmp     eax, [swpd_next]  ; is the new free block addr. lower
                                ; than the address in 'swpd_next' ?
                                ; (next/first free block value)
        jnb     short uswpbl_1  ; no
        mov     [swpd_next], eax  ; yes
uswpbl_1:
        bts     [ebx], eax      ; unlink/release/deallocate block
                                ; set relevant bit to 1.
                                ; set CF to the previous bit value
        cmc                     ; complement carry flag
        jc      short uswpbl_2  ; do not increase swfd_free count
                                ; if the block is already deallocated
                                ; before.
         inc     dword [swpd_free]
uswpbl_2:
        pop     edx
        pop     ebx
        retn

link_swap_block:
        ; 01/07/2015
        ; 18/04/2015
        ; 24/10/2014 (Retro UNIX 386 v1 - beginning)
        ;
        ; INPUT -> none
        ;
        ; OUTPUT ->
        ;       EAX = OFFSET ADDRESS OF THE ALLOCATED BLOCK (4096 bytes)
        ;             in sectors (corresponding
        ;             SWAP DISK ALLOCATION TABLE bit is RESET)
        ;
        ;       CF = 1 and EAX = 0
        ;                if there is not a free block to be allocated
        ;
        ; Modified Registers -> none (except EAX)
        ;

        ;mov    eax, [swpd_free]
        ;and    eax, eax
        ;jz     short out_of_swpspc
        ;
        push    ebx
        push    ecx
        ;
        mov     ebx, swap_alloc_table ; Swap Allocation Table offset
        mov     ecx, ebx
        add     ebx, [swpd_next] ; Free block searching starts from here
                               ; next_free_swap_block >> 5
        add     ecx, [swpd_last] ; Free block searching ends here
                               ; (total_swap_blocks - 1) >> 5
lswbl_scan:
        cmp     ebx, ecx
        ja      short lswbl_notfound
        ;
        bsf     eax, [ebx] ; Scans source operand for first bit set (1).
                          ; Clears ZF if a bit is found set (1) and
                          ; loads the destination with an index to
                          ; first set bit. (0 -> 31)
                          ; Sets ZF to 1 if no bits are found set.
        ; 01/07/2015
        jnz     short lswbl_found ; ZF = 0 -> a free block has been found
                        ;
                        ; NOTE:  a Swap Disk Allocation Table bit
                        ;        with value of 1 means
                        ;        the corresponding page is free
                        ;        (Retro UNIX 386 v1 feaure only!)
        add     ebx, 4
                        ; We return back for searching next page block
                        ; NOTE: [swpd_free] is not ZERO; so,
                        ;       we always will find at least 1 free block here.
        jmp     short lswbl_scan
```

```
        ;
lswbl_notfound:
        sub     ecx, swap_alloc_table
        mov     [swpd_next], ecx ; next/first free page = last page
                                ; (unlink_swap_block procedure will change it)
        xor     eax, eax
        mov     [swpd_free], eax
        stc
lswbl_ok:
        pop     ecx
        pop     ebx
        retn
        ;
;out_of_swpspc:
;       stc
;       retn

lswbl_found:
        mov     ecx, ebx
        sub     ecx, swap_alloc_table
        mov     [swpd_next], ecx ; Set first free block searching start
                                ; address/offset (to the next)
         dec     dword [swpd_free] ; 1 block has been allocated (X = X-1)
        ;
        btr     [ebx], eax      ; The destination bit indexed by the source value
                                ; is copied into the Carry Flag and then cleared
                                ; in the destination.
                                ;
                                ; Reset the bit which is corresponding to the
                                ; (just) allocated block.
        shl     ecx, 5          ; (block offset * 32) + block index
        add     eax, ecx        ; = block number
        shl     eax, SECTOR_SHIFT ; 3, sector (offset) address of the block
                                ; 1 block =  8 sectors
        ;
        ; EAX = offset address of swap disk/file sector (beginning of the block)
        ;
        ; NOTE: The relevant page table entry will be updated
        ;       according to this EAX value...
        ;
        jmp     short lswbl_ok

logical_disk_read:
        ; 20/07/2015
        ; 09/03/2015 (temporary code here)
        ;
        ; INPUT ->
        ;       ESI = Logical disk description table address
        ;       EBX = Memory page (buffer) address (physical!)
        ;       EAX = Sector adress (offset address, logical sector number)
        ;       ECX = Sector count
        ;
        ;
        retn

logical_disk_write:
        ; 20/07/2015
        ; 09/03/2015 (temporary code here)
        ;
        ; INPUT ->
        ;       ESI = Logical disk description table address
        ;       EBX = Memory page (buffer) address (physical!)
        ;       EAX = Sector adress (offset address, logical sector number)
        ;       ECX = Sector count
        ;
        retn
```

```
get_physical_addr:
        ; 18/10/2015
        ; 29/07/2015
        ; 20/07/2015
        ; 04/06/2015
        ; 20/05/2015
        ; 28/04/2015
        ; 18/04/2015
        ; Get physical address
        ;     (allocates a new page for user if it is not present)
        ;
        ; (This subroutine is needed for mapping user's virtual
        ; (buffer) address to physical address (of the buffer).)
        ; ('sys write', 'sys read' system calls...)
        ;
        ; INPUT ->
        ;       EBX = virtual address
        ;       u.pgdir = page directory (physical) address
        ;
        ; OUTPUT ->
        ;       EAX = physical address
        ;       EBX = linear address
        ;       EDX = physical address of the page frame
        ;             (with attribute bits)
        ;       ECX = byte count within the page frame
        ;
        ; Modified Registers -> EAX, EBX, ECX, EDX
        ;
        add     ebx, CORE ; 18/10/2015
        mov     eax, [u.pgdir]
        call    get_pte
                ; EDX = Page table entry address (if CF=0)
                 ;       Page directory entry address (if CF=1)
                ;       (Bit 0 value is 0 if PT is not present)
                ; EAX = Page table entry value (page address)
                ;      CF = 1 -> PDE not present or invalid ?
        jnc     short gpa_1
        ;
        call    allocate_page
        jc      short gpa_im_err  ; 'insufficient memory' error
gpa_0:
        call    clear_page
        ; EAX = Physical (base) address of the allocated (new) page
        or      al, PDE_A_PRESENT + PDE_A_WRITE + PDE_A_USER ; 4+2+1 = 7
                          ; lower 3 bits are used as U/S, R/W, P flags
                          ; (user, writable, present page)
        mov     [edx], eax ; Let's put the new page directory entry here !
        mov     eax, [u.pgdir]
        call    get_pte
        jc      short gpa_im_err ; 'insufficient memory' error
gpa_1:
        ; EAX = PTE value, EDX = PTE address
        test    al, PTE_A_PRESENT
        jnz     short gpa_3
        or      eax, eax
        jz      short gpa_4  ; Allocate a new page
        ; 20/07/2015
        push    ebp
        mov     ebp, ebx ; virtual (linear) address
        ; reload swapped page
        call    reload_page ; 28/04/2015
        pop     ebp
        jc      short gpa_retn
gpa_2:
        ; 20/07/2015
        ; 20/05/2015
        ; add this page to swap queue
        push    eax
        ; EBX = virtual address
        call    add_to_swap_queue
        pop     eax
                ; PTE address in EDX
                ; virtual address in EBX
        ; EAX = memory page address
        or      al, PTE_A_PRESENT + PTE_A_USER + PTE_A_WRITE
                                ; present flag, bit 0 = 1
                                ; user flag, bit 2 = 1
                                ; writable flag, bit 1 = 1
        mov     [edx], eax  ; Update PTE value
```

```
gpa_3:  ; 18/10/2015
        mov     ecx, ebx
        and     ecx, PAGE_OFF
        mov     edx, eax
        and     ax, PTE_A_CLEAR
        add     eax, ecx
        neg     ecx ; 1 -> -1 (0FFFFFFFFh), 4095 (0FFFh) -> -4095
        add     ecx, PAGE_SIZE
        clc
gpa_retn:
        retn
gpa_4:
        call    allocate_page
        jc      short gpa_im_err ; 'insufficient memory' error
        call    clear_page
        jmp     short gpa_2

gpa_im_err:
        mov     eax, ERR_MINOR_IM ; Insufficient memory (minor) error!
                                  ; Major error = 0 (No protection fault)
        retn

reload_page:
        ; 20/07/2015
        ; 28/04/2015 (Retro UNIX 386 v1 - beginning)
        ;
        ; Reload (Restore) swapped page at memory
        ;
        ; INPUT ->
        ;       EBP = Virtual (linear) memory address
        ;       EAX = PTE value (swap disk sector address)
        ;       (Swap disk sector address = bit 1 to bit 31 of EAX)
        ; OUTPUT ->
        ;       EAX = PHYSICAL (real/flat) ADDRESS OF RELOADED PAGE
        ;
        ;       CF = 1 and EAX = error code
        ;
        ; Modified Registers -> none (except EAX)
        ;
        shr     eax, 1    ; Convert PTE value to swap disk address
        push    ebx       ;
        mov     ebx, eax ; Swap disk (offset) address
        call    allocate_page
        jc      short rlp_im_err
        xchg    eax, ebx
        ; EBX = Physical memory (page) address
        ; EAX = Swap disk (offset) address
        ; EBP = Virtual (linear) memory address
        call    swap_in
        jc      short rlp_swp_err  ; (swap disk/file read error)
        mov     eax, ebx
rlp_retn:
        pop     ebx
        retn

rlp_im_err:
        mov     eax, ERR_MINOR_IM ; Insufficient memory (minor) error!
                                  ; Major error = 0 (No protection fault)
        jmp     short rlp_retn

rlp_swp_err:
        mov     eax, SWP_DISK_READ_ERR ; Swap disk read error !
        jmp     short rlp_retn

copy_page_dir:
        ; 19/09/2015
        ; temporary - 07/09/2015
        ; 07/09/2015 (Retro UNIX 386 v1 - beginning)
        ;
        ; INPUT ->
        ;       [u.pgdir] = PHYSICAL (real/flat) ADDRESS of the parent's
        ;                   page directory.
        ; OUTPUT ->
        ;       EAX =  PHYSICAL (real/flat) ADDRESS of the child's
        ;              page directory.
        ;       (New page directory with new page table entries.)
        ;       (New page tables with read only copies of the parent's
        ;       pages.)
        ;       EAX = 0 -> Error (CF = 1)
```

```
        ;
        ; Modified Registers -> none (except EAX)
        ;
        call    allocate_page
        jc      short cpd_err
        ;
        push    ebp ; 20/07/2015
        push    esi
        push    edi
        push    ebx
        push    ecx
        mov     esi, [u.pgdir]
        mov     edi, eax
        push    eax ; save child's page directory address
        ; copy PDE 0 from the parent's page dir to the child's page dir
        ; (use same system space for all user page tables)
        movsd
        mov     ebp, 1024*4096 ; pass the 1st 4MB (system space)
        mov     ecx, (PAGE_SIZE / 4) - 1 ; 1023
cpd_0:
        lodsd
        ;or     eax, eax
         ;jnz      short cpd_1
        test    al, PDE_A_PRESENT ;  bit 0 =  1
        jnz     short cpd_1
        ; (virtual address at the end of the page table)
        add     ebp, 1024*4096 ; page size * PTE count
        jmp     short cpd_2
cpd_1:
        and     ax, PDE_A_CLEAR ; 0F000h ; clear attribute bits
        mov     ebx, eax
        ; EBX = Parent's page table address
        call    copy_page_table
        jc      short cpd_p_err
        ; EAX = Child's page table address
        or      al, PDE_A_PRESENT + PDE_A_WRITE + PDE_A_USER
                        ; set bit 0, bit 1 and bit 2 to 1
                        ; (present, writable, user)
cpd_2:
        stosd
        loop    cpd_0
        pop     eax  ; restore child's page directory address
cpd_3:
        pop     ecx
        pop     ebx
        pop     edi
        pop     esi
        pop     ebp
cpd_err:
        retn
cpd_p_err:
        ; release the allocated pages missing (recover free space)
        pop     eax  ; the new page directory address (physical)
        mov     ebx, [u.pgdir] ; parent's page directory address
        call    deallocate_page_dir
        sub     eax, eax ; 0
        stc
        jmp     short cpd_3

copy_page_table:
        ; 19/09/2015
        ; temporary - 07/09/2015
        ; 07/09/2015 (Retro UNIX 386 v1 - beginning)
        ;
        ; INPUT ->
        ;       EBX = PHYSICAL (real/flat) ADDRESS of the parent's page table.
        ;       EBP = page table entry index (from 'copy_page_dir')
        ; OUTPUT ->
        ;       EAX = PHYSICAL (real/flat) ADDRESS of the child's page table.
        ;       EBP = (recent) page table index (for 'add_to_swap_queue')
        ;       CF = 1 -> error
        ;
        ; Modified Registers -> EBP (except EAX)
        ;
        call    allocate_page
        jc      short cpt_err
        ;
```

```
        push    eax ; *
        ;push   ebx
        push    esi
        push    edi
        push    edx
        push    ecx
        ;
        mov     esi, ebx
        mov     edi, eax
        mov     edx, eax
        add     edx, PAGE_SIZE
cpt_0:
        lodsd
        test    al, PTE_A_PRESENT ;  bit 0 =  1
        jnz     short cpt_1
        and     eax, eax
        jz      short cpt_2
        ; ebp = virtual (linear) address of the memory page
        call    reload_page ; 28/04/2015
        jc      short cpt_p_err
cpt_1:
        and     ax, PTE_A_CLEAR ; 0F000h ; clear attribute bits
        mov     ecx, eax
        ; Allocate a new page for the child process
        call    allocate_page
        jc      short cpt_p_err
        push    edi
        push    esi
        mov     esi, ecx
        mov     edi, eax
        mov     ecx, PAGE_SIZE/4
        rep     movsd   ; copy page (4096 bytes)
        pop     esi
        pop     edi
        ;
        push    ebx
        push    eax
        mov     ebx, ebp
        ; ebx = virtual address of the memory page
        call    add_to_swap_queue
        pop     eax
        pop     ebx
        ;
        ;or     ax, PTE_A_USER+PTE_A_PRESENT
        or      al, PTE_A_USER+PTE_A_WRITE+PTE_A_PRESENT
cpt_2:
        stosd  ; EDI points to child's PTE
        ;
        add     ebp, 4096 ; 20/07/2015 (next page)
        ;
        cmp     edi, edx
        jb      short cpt_0
cpt_p_err:
        pop     ecx
        pop     edx
        pop     edi
        pop     esi
        ;pop    ebx
        pop     eax ; *
cpt_err:
        retn

; /// End Of MEMORY MANAGEMENT FUNCTIONS ///

;; Data:

; 09/03/2015
;swpq_count: dw 0 ; count of pages on the swap que
;swp_drv:    dd 0 ; logical drive description table address of the swap drive/disk
;swpd_size:  dd 0 ; size of swap drive/disk (volume) in sectors (512 bytes).

;swpd_free:  dd 0 ; free page blocks (4096 bytes) on swap disk/drive (logical)
;swpd_next:  dd 0 ; next free page block
;swpd_last:  dd 0 ; last swap page block
```

```
; Retro UNIX 386 v1 Kernel - SYSDEFS.INC
; Last Modification: 09/12/2015
;
; ///////// RETRO UNIX 386 V1 SYSTEM DEFINITIONS //////////////
; (Modified from
;       Retro UNIX 8086 v1 system definitions in 'UNIX.ASM', 01/09/2014)
; ((UNIX.ASM (RETRO UNIX 8086 V1 Kernel), 11/03/2013 - 01/09/2014))
;       UNIX.ASM (MASM 6.11) --> SYSDEFS.INC (NASM 2.11)
; ----------------------------------------------------------------------------
;
; Derived 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>
;
; ****************************************************************************

nproc   equ     16  ; number of processes
nfiles  equ     50
ntty    equ      8   ; 8+1 -> 8 (10/05/2013)
nbuf    equ     4   ; 6 ;; 21/08/2015 - 'namei' buffer problem when nbuf > 4
                ; NOTE: If fd0 super block buffer addres is beyond of the 1st
                ; 32K, DMA r/w routine or someting else causes a jump to
                ; kernel panic routine (in 'alloc' routine, in u5.s)
                ; because of invalid buffer content (r/w error).
                ; When all buffers are set before the end of the 1st 32k,
                ; there is no problem!? (14/11/2015)

;csgmnt equ     2000h   ; 26/05/2013 (segment of process 1)
;core   equ     0           ; 19/04/2013
;ecore  equ     32768 - 64  ; 04/06/2013 (24/05/2013)
        ; (if total size of argument list and arguments is 128 bytes)
        ; maximum executable file size = 32768-(64+40+128-6) = 32530 bytes
        ; maximum stack size = 40 bytes (+6 bytes for 'IRET' at 32570)
        ; initial value of user's stack pointer = 32768-64-128-2 = 32574
        ;       (sp=32768-args_space-2 at the beginning of execution)
        ; argument list offset = 32768-64-128 = 32576 (if it is 128 bytes)
        ; 'u' structure offset (for the '/core' dump file) = 32704
        ; '/core' dump file size = 32768 bytes

; 08/03/2014
;sdsegmnt equ  6C0h  ; 256*16 bytes (swap data segment size for 16 processes)

; 19/04/2013 Retro UNIX 8086 v1 feaure only !
;;sdsegmnt equ        740h  ; swap data segment (for user structures and registers)

; 30/08/2013
time_count equ 4 ; 10 --> 4 01/02/2014

; 05/02/2014
; process status
;SFREE  equ 0
;SRUN   equ 1
;SWAIT  equ 2
;SZOMB  equ 3
;SSLEEP equ 4 ; Retro UNIX 8086 V1 extension (for sleep and wakeup)

; 09/03/2015
userdata equ 80000h ; user structure data address for current user ; temporary
swap_queue equ 90000h - 2000h ; swap queue address ; temporary
swap_alloc_table equ 0D0000h  ;  swap allocation table address ; temporary

; 17/09/2015
ESPACE equ 48 ; [u.usp] (at 'sysent') - [u.sp] value for error return
```

```
; 21/09/2015 (36)
; 01/07/2015 (35)
; 14/07/2013 (0-34)
; UNIX v1 system calls
_rele   equ 0
_exit   equ 1
_fork   equ 2
_read   equ 3
_write  equ 4
_open   equ 5
_close  equ 6
_wait   equ 7
_creat  equ 8
_link   equ 9
_unlink equ 10
_exec   equ 11
_chdir  equ 12
_time   equ 13
_mkdir  equ 14
_chmod  equ 15
_chown  equ 16
_break  equ 17
_stat   equ 18
_seek   equ 19
_tell   equ 20
_mount  equ 21
_umount equ 22
_setuid equ 23
_getuid equ 24
_stime  equ 25
_quit   equ 26
_intr   equ 27
_fstat  equ 28
_emt    equ 29
_mdate  equ 30
_stty   equ 31
_gtty   equ 32
_ilgins equ 33
_sleep  equ 34 ; Retro UNIX 8086 v1 feature only !
_msg    equ 35 ; Retro UNIX 386 v1 feature only !
_geterr equ 36 ; Retro UNIX 386 v1 feature only !

%macro sys 1-4
    ; 03/09/2015
    ; 13/04/2015
    ; Retro UNIX 386 v1 system call.
    %if %0 >= 2
        mov ebx, %2
        %if %0 >= 3
            mov ecx, %3
            %if %0 = 4
               mov edx, %4
            %endif
        %endif
    %endif
    mov eax, %1
    int 30h
%endmacro

; 13/05/2015 - ERROR CODES
ERR_FILE_NOT_OPEN  equ 10 ; 'file not open !' error
ERR_FILE_ACCESS    equ 11 ; 'permission denied !' error
; 14/05/2015
ERR_DIR_ACCESS     equ 11 ; 'permission denied !' error
ERR_FILE_NOT_FOUND equ 12 ; 'file not found !' error
ERR_TOO_MANY_FILES equ 13 ; 'too many open files !' error
ERR_DIR_EXISTS     equ 14 ; 'directory already exists !' error
; 16/05/2015
ERR_DRV_NOT_RDY    equ 15 ; 'drive not ready !' error
; 18/05/2015
ERR_DEV_NOT_RDY    equ 15 ; 'device not ready !' error
ERR_DEV_ACCESS     equ 11 ; 'permission denied !' error
ERR_DEV_NOT_OPEN   equ 10 ; 'device not open !' error
; 07/06/2015
ERR_FILE_EOF      equ 16 ; 'end of file !' error
ERR_DEV_VOL_SIZE   equ 16 ; 'out of volume' error
; 09/06/2015
ERR_DRV_READ      equ 17 ; 'disk read error !'
ERR_DRV_WRITE     equ 18 ; 'disk write error !'
```

```
; 16/06/2015
ERR_NOT_DIR       equ 19 ; 'not a (valid) directory !' error
ERR_FILE_SIZE     equ 20 ; 'file size error !'
; 22/06/2015
ERR_NOT_SUPERUSER  equ 11 ; 'permission denied !' error
ERR_NOT_OWNER      equ 11 ; 'permission denied !' error
ERR_NOT_FILE       equ 11 ; 'permission denied !' error
; 23/06/2015
ERR_FILE_EXISTS    equ 14 ; 'file already exists !' error
ERR_DRV_NOT_SAME   equ 21 ; 'not same drive !' error
ERR_DIR_NOT_FOUND  equ 12 ; 'directory not found !' error
ERR_NOT_EXECUTABLE equ 22 ; 'not executable file !' error
; 27/06/2015
ERR_INV_PARAMETER  equ 23 ; 'invalid parameter !' error
ERR_INV_DEV_NAME   equ 24 ; 'invalid device name !' error
; 29/06/2015
ERR_TIME_OUT      equ 25 ; 'time out !' error
ERR_DEV_NOT_RESP   equ 25 ; 'device not responding !' error

; 26/08/2015
; 24/07/2015
; 24/06/2015
MAX_ARG_LEN       equ 256 ; max. length of sys exec arguments
; 01/07/2015
MAX_MSG_LEN       equ 255 ; max. msg length for 'sysmsg'
;
```

```
; Retro UNIX 386 v1 Kernel (v0.2) - SYS0.INC
; Last Modification: 21/11/2015
; ----------------------------------------------------------------------------
; Derived from 'Retro UNIX 8086 v1' source code by Erdogan Tan
; (v0.1 - Beginning: 11/07/2012)
;
; Derived 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>
;
; Retro UNIX 8086 v1 - U0.ASM (28/07/2014) //// UNIX v1 -> u0.s
;
; ****************************************************************************

sys_init:
        ; 18/10/2015
        ; 28/08/2015
        ; 24/08/2015
        ; 14/08/2015
        ; 24/07/2015
        ; 02/07/2015
        ; 01/07/2015
        ; 23/06/2015
        ; 15/04/2015
        ; 13/04/2015
        ; 11/03/2015 (Retro UNIX 386 v1 - Beginning)
        ; 28/07/2014 (Retro UNIX 8086 v1)
        ;
        ;call   ldrv_init ; Logical drive description tables initialization
        ;
        ; 14/02/2014
        ; 14/07/2013
        mov     ax, 41
        mov     [rootdir], ax
        mov     [u.cdir], ax
        and     al, 1 ; 15/04/2015
        mov     [u.uno], al
        mov     [mpid], ax
        mov     [p.pid], ax
        mov     [p.stat], al ; SRUN, 05/02/2014
        ;
        mov     al, time_count ; 30/08/2013
        mov     [u.quant], al ; 14/07/2013
        ; 02/07/2015
        mov     eax, [k_page_dir]
        ;sub    eax, eax
        mov     [u.pgdir], eax ; reset
        ; 18/10/2015
        ;mov    [u.ppgdir], eax ; 0
        ;
        call    epoch
        mov     [s.time], eax ; 13/03/2015
        ; 17/07/2013
        call    bf_init ; buffer initialization
        ; 23/06/2015
        call    allocate_page
        ;;jc    error
        jc       panic   ; jc short panic (01/07/2015)
        mov     [u.upage], eax ; user structure page
        mov     [p.upage], eax
        ;
        call    clear_page
        ;
        ; 14/08/2015
        cli
        ; 14/03/2015
        ; 17/01/2014
        call    sp_init ; serial port initialization
        ; 14/08/2015
        sti
        ;
```

```
        ; 30/06/2015
        ;mov    esi, kernel_init_ok_msg
        ;call   print_msg
        ;
        xor     bl, bl ; video page 0
vp_clr_nxt:  ; clear video pages (reset cursor positions)
        call    vp_clr  ; 17/07/2013
        inc     bl
        cmp     bl, 8
        jb      short vp_clr_nxt
        ;
        ; 24/07/2015
        ;push    KDATA
         ;push    esp
        ;mov    [tss.esp0], esp
         ;mov     word [tss.ss0], KDATA
        ;
        ; 24/08/2015
        ;; temporary (01/07/2015)
        mov     byte [u.quant], time_count ; 4
                                  ; it is not needed here !
        ;;inc   byte [u.kcall] ; 'the caller is kernel' sign
        dec     byte [sysflg] ; FFh = ready for system call
                                 ; 0 = executing a system call
        ;;sys   _msg, kernel_init_ok_msg, 255, 0
        ;
        ;;; 06/08/2015
        ;;;call getch ; wait for a key stroke
        ;;mov   ecx, 0FFFFFFFh
;;sys_init_msg_wait:
;;      push    ecx
;;      mov     al, 1
;;      mov     ah, [ptty] ; active (current) video page
;;      call    getc_n
;;      pop     ecx
;;      jnz     short sys_init_msg_ok
;;      loop    sys_init_msg_wait
        ;
;;sys_init_msg_ok:
        ; 28/08/2015 (initial settings for the 1st 'rswap')
        push    KDATA ; ss
        push    esp
        pushfd
        push    KCODE ; cs
        push    init_exec ; eip
        mov     [u.sp], esp
        push    ds
        push    es
        push    fs
        push    gs
        pushad
        mov     [u.usp], esp
        call    wswap ; save current user (u) structure, user registers
                      ; and interrupt return components (for IRET)
        popad
        pop     ax ; gs
        pop     ax ; fs
        pop     ax ; es
        pop     ax ; ds
        pop     eax ; eip (init_exec)
        pop     ax ; cs (KCODE)
        pop     eax ; E-FLAGS
        pop     eax ; esp
        pop     ax ; ss (KDATA)
        ;
        xor     eax, eax ; 0
        mov     [u.ppgdir], eax ; reset (to zero) for '/etc/init'
        ;
        ; 02/07/2015
        ; [u.pgdir ] = [k_page_dir]
        ; [u.ppgdir] = 0 (page dir of the parent process)
        ;     (The caller is os kernel sign for 'sysexec')
init_exec:
        ; 13/03/2013
        ; 24/07/2013
        mov     ebx, init_file
        mov     ecx, init_argp
        ; EBX contains 'etc/init' asciiz file name address
        ; ECX contains address of argument list pointer
```

```
		;
		;dec	byte [sysflg] ; FFh = ready for system call
					; 0 = executing a system call
		sys	_exec  ; execute file
		jnc	short panic
		;
		mov	esi, etc_init_err_msg
		call	print_msg
		jmp	short key_to_reboot

;align 4
init_argp:
		dd	init_file, 0  ; 23/06/2015 (dw -> dd)
init_file:
		; 24/08/2015
		db	'/etc/init', 0
panic:
		; 13/03/2015 (Retro UNIX 386 v1)
		; 07/03/2014 (Retro UNIX 8086 v1)
		mov	esi, panic_msg
		call	print_msg
key_to_reboot:
		; 15/11/2015
		call	getch
			; wait for a character from the current tty
		;
		mov	al, 0Ah
		mov	bl, [ptty] ; [active_page]
		mov	ah, 07h ; Black background,
				; light gray forecolor
		call	write_tty
		jmp	cpu_reset

print_msg:
		; 01/07/2015
		; 13/03/2015 (Retro UNIX 386 v1)
		; 07/03/2014 (Retro UNIX 8086 v1)
		; (Modified registers: EAX, EBX, ECX, EDX, ESI, EDI)
		;
		;
		lodsb
pmsg1:
		push	esi
		movzx	ebx, byte [ptty]
		mov	ah, 07h ; Black background, light gray forecolor
		call	write_tty
		pop	esi
		lodsb
		and	al, al
		jnz	short pmsg1
		retn

ctrlbrk:
		; 12/11/2015
		; 13/03/2015 (Retro UNIX 386 v1)
		; 06/12/2013 (Retro UNIX 8086 v1)
		;
		; INT 1Bh (control+break) handler
		;
		; Retro Unix 8086 v1 feature only!
		;
		cmp	word [u.intr], 0
		jna	short cbrk4
cbrk0:
		; 12/11/2015
		; 06/12/2013
		cmp	word [u.quit], 0
		jz	short cbrk4
		;
		; 20/09/2013
		push	ax
		mov	al, [ptty]
		;
		; 12/11/2015
		;
		; ctrl+break (EOT, CTRL+D) from serial port
		; or ctrl+break from console (pseudo) tty
		; (!redirection!)
		;
```

```
        cmp     al, 8 ; serial port tty nums > 7
         jb      short cbrk1 ; console (pseudo) tty
        ;
        ; Serial port interrupt handler sets [ptty]
        ; to the port's tty number (as temporary).
        ;
        ; If active process is using a stdin or
        ; stdout redirection (by the shell),
         ; console tty keyboard must be available
        ; to terminate running process,
        ; in order to prevent a deadlock.
        ;
        push    edx
        movzx   edx, byte [u.uno]
        cmp      al, [edx+p.ttyc-1] ; console tty (rw)
        pop     edx
        je      short cbrk2
cbrk1:
        inc     al  ; [u.ttyp] : 1 based tty number
        ; 06/12/2013
        cmp     al, [u.ttyp] ; recent open tty (r)
        je      short cbrk2
         cmp     al, [u.ttyp+1] ; recent open tty (w)
        jne     short cbrk3
cbrk2:
        ;; 06/12/2013
        ;mov    ax, [u.quit]
        ;and    ax, ax
        ;jz     short cbrk3
        ;
        xor     ax, ax ; 0
        dec     ax
        ; 0FFFFh = 'ctrl+brk' keystroke
        mov     [u.quit], ax
cbrk3:
        pop     ax
cbrk4:
        retn

com2_int:
        ; 07/11/2015
        ; 24/10/2015
        ; 23/10/2015
        ; 14/03/2015 (Retro UNIX 386 v1 - Beginning)
        ; 28/07/2014 (Retro UNIX 8086 v1)
        ; < serial port 2 interrupt handler >
        ;
        mov     [esp], eax ; overwrite call return address
        ;push   eax
        mov     ax, 9
        jmp     short comm_int
com1_int:
        ; 07/11/2015
        ; 24/10/2015
        mov     [esp], eax ; overwrite call return address
        ; 23/10/2015
        ;push   eax
        mov     ax, 8
comm_int:
        ; 20/11/2015
        ; 18/11/2015
        ; 17/11/2015
        ; 16/11/2015
        ; 09/11/2015
        ; 08/11/2015
        ; 07/11/2015
        ; 06/11/2015 (serial4.asm, 'serial')
        ; 01/11/2015
        ; 26/10/2015
        ; 23/10/2015
        push    ebx
        push    esi
        push    edi
        push    ds
        push    es
        ; 18/11/2015
        mov     ebx, cr3
        push    ebx ; ****
        ;
```

```
        push    ecx ; ***
        push    edx ; **
        ;
        mov     ebx, KDATA
        mov     ds, bx
        mov     es, bx
        ;
        mov     ecx, [k_page_dir]
        mov     cr3, ecx
        ; 20/11/2015
        ; Interrupt identification register
        mov     dx, 2FAh ; COM2
        ;
        cmp     al, 8
        ja      short com_i0
        ;
        ; 20/11/2015
        ; 17/11/2015
        ; 16/11/2015
        ; 15/11/2015
        ; 24/10/2015
        ; 14/03/2015 (Retro UNIX 386 v1 - Beginning)
        ; 28/07/2014 (Retro UNIX 8086 v1)
        ; < serial port 1 interrupt handler >
        ;
        inc     dh ; 3FAh ; COM1 Interrupt id. register
com_i0:
        ;push   eax ; *
        ; 07/11/2015
        mov     byte [ccomport], al
        ; 09/11/2015
        movzx   ebx, ax ; 8 or 9
        ; 17/11/2015
        ; reset request for response status
        mov     [ebx+req_resp-8], ah ; 0
        ;
        ; 20/11/2015
        in      al, dx          ; read interrupt id. register
        JMP     $+2             ; I/O DELAY
        and     al, 4           ; received data available?
        jz      short com_eoi   ; (transmit. holding reg. empty)
        ;
        ; 20/11/2015
        sub     dl, 3FAh-3F8h   ; data register (3F8h, 2F8h)
        in      al, dx          ; read character
        ;JMP    $+2             ; I/O DELAY
        ; 08/11/2015
        ; 07/11/2015
        mov     esi, ebx
        mov     edi, ebx
        add     esi, rchar - 8 ; points to last received char
        add     edi, schar - 8 ; points to last sent char
        mov     [esi], al ; received char (current char)
        ; query
        and     al, al
        jnz     short com_i2
        ; response
        ; 17/11/2015
        ; set request for response status
         inc     byte [ebx+req_resp-8] ; 1
        ;
        add     dx, 3FDh-3F8h   ; (3FDh, 2FDh)
        in      al, dx          ; read line status register
        JMP     $+2             ; I/O DELAY
        and     al, 20h         ; transmitter holding reg. empty?
        jz      short com_eoi   ; no
        mov     al, 0FFh        ; response
        sub     dx, 3FDh-3F8h   ; data port (3F8h, 2F8h)
        out     dx, al          ; send on serial port
        ; 17/11/2015
        cmp     byte [edi], 0    ; query ? (schar)
        jne     short com_i1     ; no
        mov     [edi], al       ; 0FFh (responded)
com_i1:
        ; 17/11/2015
        ; reset request for response status (again)
         dec     byte [ebx+req_resp-8] ; 0
        jmp     short com_eoi
```

```
com_i2:
        ; 08/11/2015
        cmp     al, 0FFh        ; (response ?)
        je      short com_i3    ; (check for response signal)
        ; 07/11/2015
        cmp     al, 04h ; EOT
        jne     short com_i4
        ; EOT = 04h (End of Transmit) - 'CTRL + D'
        ;(an EOT char is supposed as a ctrl+brk from the terminal)
        ; 08/11/2015
                ; ptty -> tty 0 to 7 (pseudo screens)
        xchg    bl, [ptty]  ; tty number (8 or 9)
        call    ctrlbrk
        xchg    [ptty], bl ; (restore ptty value and BL value)
        ;mov    al, 04h ; EOT
        ; 08/11/2015
        jmp     short com_i4
com_i3:
        ; 08/11/2015
        ; If 0FFh has been received just after a query
        ; (schar, ZERO), it is a response signal.
        ; 17/11/2015
         cmp     byte [edi], 0 ; query ? (schar)
        ja      short com_i4 ; no
        ; reset query status (schar)
        mov     [edi], al ; 0FFh
        inc     al ; 0
com_i4:
        ; 27/07/2014
        ; 09/07/2014
        shl     bl, 1
        add     ebx, ttychr
        ; 23/07/2014 (always overwrite)
        ;;cmp   word [ebx], 0
        ;;ja    short com_eoi
        ;
        mov     [ebx], ax   ; Save ascii code
                           ; scan code = 0
com_eoi:
        ;mov    al, 20h
        ;out    20h, al    ; end of interrupt
        ;
        ; 07/11/2015
        ;pop    eax ; *
        mov     al, byte [ccomport] ; current COM port
         ; al = tty number (8 or 9)
         call   wakeup
com_iret:
        ; 23/10/2015
        pop     edx ; **
        pop     ecx ; ***
        ; 18/11/2015
        ;pop    eax ; ****
        ;mov    cr3, eax
        ;jmp    iiret
        jmp     iiretp

;iiretp: ; 01/09/2015
;       ; 28/08/2015
;       pop     eax ; (*) page directory
;       mov     cr3, eax
;iiret:
;       ; 22/08/2014
;       mov     al, 20h ; END OF INTERRUPT COMMAND TO 8259
;       out     20h, al ; 8259 PORT
;       ;
;       pop     es
;       pop     ds
;       pop     edi
;       pop     esi
;       pop     ebx ; 29/08/2014
;       pop     eax
;       iretd
```

```
sp_init:
        ; 07/11/2015
        ; 29/10/2015
        ; 26/10/2015
        ; 23/10/2015
        ; 29/06/2015
        ; 14/03/2015 (Retro UNIX 386 v1 - 115200 baud)
        ; 28/07/2014 (Retro UNIX 8086 v1 - 9600 baud)
        ; Initialization of Serial Port Communication Parameters
        ; (COM1 base port address = 3F8h, COM1 Interrupt = IRQ 4)
        ; (COM2 base port address = 2F8h, COM1 Interrupt = IRQ 3)
        ;
        ; ((Modified registers: EAX, ECX, EDX, EBX))
        ;
        ; INPUT:  (29/06/2015)
        ;      AL = 0 for COM1
        ;           1 for COM2
        ;      AH = Communication parameters
        ;
        ;  (*) Communication parameters (except BAUD RATE):
        ;      Bit     4      3       2       1       0
        ;             -PARITY--    STOP BIT  -WORD LENGTH-
        ;  this one -->       00 = none     0 = 1 bit  11 = 8 bits
        ;             01 = odd      1 = 2 bits       10 = 7 bits
        ;             11 = even
        ;  Baud rate setting bits: (29/06/2015)
        ;             Retro UNIX 386 v1 feature only !
        ;      Bit     7    6    5  | Baud rate
        ;             ------------------------
        ;      value   0    0    0  | Default (Divisor = 1)
        ;             0    0    1  | 9600 (12)
        ;             0    1    0  | 19200 (6)
        ;             0    1    1  | 38400 (3)
        ;             1    0    0  | 14400 (8)
        ;             1    0    1  | 28800 (4)
        ;             1    1    0  | 57600 (2)
        ;             1    1    1  | 115200 (1)

        ; References:
        ; (1) IBM PC-XT Model 286 BIOS Source Code
        ;     RS232.ASM --- 10/06/1985 COMMUNICATIONS BIOS (RS232)
        ; (2) Award BIOS 1999 - ATORGS.ASM
        ; (3) http://wiki.osdev.org/Serial_Ports
        ;
        ; Set communication parameters for COM1 (= 03h)
        ;
        mov     ebx, com1p            ; COM1 parameters
        mov     dx, 3F8h              ; COM1
         ; 29/10/2015
        mov     cx, 301h  ; divisor = 1 (115200 baud)
        call    sp_i3   ; call A4
        test    al, 80h
        jz      short sp_i0 ; OK..
                ; Error !
        ;mov    dx, 3F8h
        sub     dl, 5 ; 3FDh -> 3F8h
        mov     cx, 30Eh  ; divisor = 12 (9600 baud)
        call    sp_i3   ; call A4
        test    al, 80h
        jnz     short sp_i1
sp_i0:
         ; (Note: Serial port interrupts will be disabled here...)
         ; (INT 14h initialization code disables interrupts.)
        ;
        mov     byte [ebx], 0E3h ; 11100011b
        call    sp_i5 ; 29/06/2015
sp_i1:
        inc     ebx
        mov     dx, 2F8h              ; COM2
         ; 29/10/2015
        mov     cx, 301h  ; divisor = 1 (115200 baud)
        call    sp_i3   ; call A4
        test    al, 80h
        jz      short sp_i2 ; OK..
                ; Error !
        ;mov    dx, 2F8h
        sub     dl, 5 ; 2FDh -> 2F8h
        mov     cx, 30Eh  ; divisor = 12 (9600 baud)
        call    sp_i3   ; call A4
```

```
        test    al, 80h
        jnz     short sp_i7
sp_i2:
        mov     byte [ebx], 0E3h ; 11100011b
sp_i6:
        ;; COM2 - enabling IRQ 3
        ; 07/11/2015
        ; 26/10/2015
        pushf
        cli
        mov     dx, 2FCh                ; modem control register
        in      al, dx                  ; read register
        JMP     $+2                     ; I/O DELAY
        or      al, 8                   ; enable bit 3 (OUT2)
        out     dx, al                  ; write back to register
        JMP     $+2                     ; I/O DELAY
        mov     dx, 2F9h                ; interrupt enable register
        in      al, dx                  ; read register
        JMP     $+2                     ; I/O DELAY
        ;or     al, 1                   ; receiver data interrupt enable and
        or      al, 3                   ; transmitter empty interrupt enable
        out     dx, al                  ; write back to register
        JMP     $+2                     ; I/O DELAY
        in      al, 21h                 ; read interrupt mask register
        JMP     $+2                     ; I/O DELAY
        and     al, 0F7h                ; enable IRQ 3 (COM2)
        out     21h, al                 ; write back to register
        ;
        ; 23/10/2015
        mov     eax, com2_int
        mov     [com2_irq3], eax
        ; 26/10/2015
        popf
sp_i7:
        retn
sp_i3:
;A4:    ;----- INITIALIZE THE COMMUNICATIONS PORT
        ; 28/10/2015
        inc     dl      ; 3F9h (2F9h)   ; 3F9h, COM1 Interrupt enable register
        mov     al, 0
        out     dx, al                  ; disable serial port interrupt
        JMP     $+2                     ; I/O DELAY
        add     dl, 2   ; 3FBh (2FBh)   ; COM1 Line control register (3FBh)
        mov     al, 80h
        out     dx, al                  ; SET DLAB=1 ; divisor latch access bit
        ;----- SET BAUD RATE DIVISOR
        ; 26/10/2015
        sub     dl, 3    ; 3F8h (2F8h) ; register for least significant byte
                                        ; of the divisor value
        mov     al, cl ; 1
        out     dx, al                  ; 1 = 115200 baud (Retro UNIX 386 v1)
                                        ; 2 = 57600 baud
                                        ; 3 = 38400 baud
                                        ; 6 = 19200 baud
                                        ; 12 = 9600 baud (Retro UNIX 8086 v1)
        JMP     $+2                     ; I/O DELAY
        sub     al, al
        inc     dl       ; 3F9h (2F9h) ; register for most significant byte
                                        ; of the divisor value
        out     dx, al ; 0
        JMP     $+2                     ; I/O DELAY
        ;
        mov     al, ch ; 3              ; 8 data bits, 1 stop bit, no parity
        ;and    al, 1Fh ; Bits 0,1,2,3,4
        add     dl, 2   ; 3FBh (2FBh)   ; Line control register
        out     dx, al
        JMP     $+2                     ; I/O DELAY
        ; 29/10/2015
        dec     dl      ; 3FAh (2FAh)   ; FIFO Control register (16550/16750)
        xor     al, al                  ; 0
        out     dx, al                  ; Disable FIFOs (reset to 8250 mode)
        JMP     $+2
sp_i4:
;A18:   ;----- COMM PORT STATUS ROUTINE
        ; 29/06/2015 (line status after modem status)
        add     dl, 4   ; 3FEh (2FEh)   ; Modem status register
sp_i4s:
        in      al, dx                  ; GET MODEM CONTROL STATUS
        JMP     $+2                     ; I/O DELAY
```

```
        mov     ah, al                  ; PUT IN (AH) FOR RETURN
        dec     dl      ; 3FDh (2FDh)   ; POINT TO LINE STATUS REGISTER
                                        ; dx = 3FDh for COM1, 2FDh for COM2
        in      al, dx                  ; GET LINE CONTROL STATUS
        ; AL = Line status, AH = Modem status
        retn

sp_status:
        ; 29/06/2015
        ; 27/06/2015 (Retro UNIX 386 v1)
        ; Get serial port status
        mov     dx, 3FEh                ; Modem status register (COM1)
        sub     dh, al                  ; dh = 2 for COM2 (al = 1)
                                        ; dx = 2FEh for COM2
        jmp     short sp_i4s

sp_setp: ; Set serial port communication parameters
        ; 07/11/2015
        ; 29/10/2015
        ; 29/06/2015
        ; Retro UNIX 386 v1 feature only !
        ;
        ; INPUT:
        ;       AL = 0 for COM1
        ;            1 for COM2
        ;       AH = Communication parameters (*)
        ; OUTPUT:
        ;       CL = Line status
        ;       CH = Modem status
        ;   If cf = 1 -> Error code in [u.error]
        ;               'invalid parameter !'
        ;                       or
        ;               'device not ready !' error
        ;
        ;  (*) Communication parameters (except BAUD RATE):
        ;       Bit     4       3       2       1       0
        ;               -PARITY--    STOP BIT  -WORD LENGTH-
        ;  this one -->        00 = none    0 = 1 bit  11 = 8 bits
        ;               01 = odd     1 = 2 bits       10 = 7 bits
        ;               11 = even
        ;  Baud rate setting bits: (29/06/2015)
        ;               Retro UNIX 386 v1 feature only !
        ;       Bit     7    6    5  | Baud rate
        ;               ------------------------
        ;       value   0    0    0  | Default (Divisor = 1)
        ;               0    0    1  | 9600 (12)
        ;               0    1    0  | 19200 (6)
        ;               0    1    1  | 38400 (3)
        ;               1    0    0  | 14400 (8)
        ;               1    0    1  | 28800 (4)
        ;               1    1    0  | 57600 (2)
        ;               1    1    1  | 115200 (1)
        ;
        ; (COM1 base port address = 3F8h, COM1 Interrupt = IRQ 4)
        ; (COM2 base port address = 2F8h, COM1 Interrupt = IRQ 3)
        ;
        ; ((Modified registers: EAX, ECX, EDX, EBX))
        ;
        mov     dx, 3F8h
        mov     ebx, com1p ; COM1 control byte offset
        cmp     al, 1
        ja      short sp_invp_err
        jb      short sp_setp1 ;  COM1 (AL = 0)
        dec     dh ; 2F8h
        inc     ebx ; COM2 control byte offset
sp_setp1:
        ; 29/10/2015
        mov     [ebx], ah
        movzx   ecx, ah
        shr     cl, 5 ; -> baud rate index
        and     ah, 1Fh ; communication parameters except baud rate
        mov     al, [ecx+b_div_tbl]
        mov     cx, ax
        call    sp_i3
        mov     cx, ax ; CL = Line status, CH = Modem status
        test    al, 80h
        jz      short sp_setp2
         mov     byte [ebx], 0E3h ; Reset to initial value (11100011b)
```

```
stp_dnr_err:
        mov     dword [u.error], ERR_DEV_NOT_RDY ; 'device not ready !'
        ; CL = Line status, CH = Modem status
        stc
        retn
sp_setp2:
        cmp     dh, 2 ; COM2 (2F?h)
         jna     sp_i6
                      ; COM1 (3F?h)
sp_i5:
        ; 07/11/2015
        ; 26/10/2015
        ; 29/06/2015
        ;
        ;; COM1 - enabling IRQ 4
        pushf
        cli
        mov     dx, 3FCh              ; modem control register
        in      al, dx                ; read register
        JMP     $+2                   ; I/O DELAY
        or      al, 8                 ; enable bit 3 (OUT2)
        out     dx, al                ; write back to register
        JMP     $+2                   ; I/O DELAY
        mov     dx, 3F9h              ; interrupt enable register
        in      al, dx                ; read register
        JMP     $+2                   ; I/O DELAY
        ;or     al, 1                 ; receiver data interrupt enable and
        or      al, 3                 ; transmitter empty interrupt enable
        out     dx, al                ; write back to register
        JMP     $+2                   ; I/O DELAY
        in      al, 21h               ; read interrupt mask register
        JMP     $+2                   ; I/O DELAY
        and     al, 0EFh              ; enable IRQ 4 (COM1)
        out     21h, al               ; write back to register
        ;
        ; 23/10/2015
        mov     eax, com1_int
        mov     [com1_irq4], eax
        ; 26/10/2015
        popf
        retn

sp_invp_err:
        mov     dword [u.error], ERR_INV_PARAMETER ; 'invalid parameter !'
        xor     ecx, ecx
        dec     ecx ; 0FFFFh
        stc
        retn

; 29/10/2015
b_div_tbl: ; Baud rate divisor table (115200/divisor)
        db 1, 12, 6, 3, 8, 4, 1

; Retro UNIX 8086 v1 - UNIX.ASM (01/09/2014)
epoch:
        ; 15/03/2015 (Retro UNIX 386 v1 - 32 bit version)
        ; 09/04/2013 (Retro UNIX 8086 v1 - UNIX.ASM)
        ; 'epoch' procedure prototype:
        ;                   UNIXCOPY.ASM, 10/03/2013
        ; 14/11/2012
        ; unixboot.asm (boot file configuration)
        ; version of "epoch" procedure in "unixproc.asm"
        ; 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
        ;
         ;  ((Modified registers: EAX, EDX, ECX, EBX))
        ;
        call    get_rtc_time          ; Return Current Time
         xchg   ch,cl
         mov    [hour], cx
         xchg   dh,dl
         mov    [second], dx
        ;
         call   get_rtc_date          ; Return Current Date
         xchg   ch,cl
```

```
		mov	[year], cx
		xchg	dh,dl
		mov	[month], dx
		;
		mov	cx, 3030h
		;
		mov	al, [hour] ; Hour
			; AL <= BCD number)
		db	0D4h,10h		; Undocumented inst. AAM
						; AH = AL / 10h
						; AL = AL MOD 10h
		aad	; AX= AH*10+AL
		mov	[hour], al
		mov	al, [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	[minute], al
		mov	al, [second] ; Second
			; AL <= BCD number)
		db	0D4h,10h		; Undocumented inst. AAM
						; AH = AL / 10h
						; AL = AL MOD 10h
		aad	; AX= AH*10+AL
		mov	[second], al
		mov	ax, [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	[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	[year], ax
		mov	al, [month] ; Month
			; AL <= BCD number)
		db	0D4h,10h		; Undocumented inst. AAM
						; AH = AL / 10h
						; AL = AL MOD 10h
		aad	; AX= AH*10+AL
		mov	[month], al
		mov	 al, [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	 [day], al

convert_to_epoch:
		; 15/03/2015 (Retro UNIX 386 v1 - 32 bit modification)
		; 09/04/2013 (retro UNIX 8086 v1)
		;
		; ((Modified registers: EAX, EDX, EBX))
		;
		; Derived from DALLAS Semiconductor
		; Application Note 31 (DS1602/DS1603)
		; 6 May 1998
		sub	eax, eax
		mov	ax, [year]
		sub	ax, 1970
		mov	edx, 365
		mul	edx
		xor	ebx, ebx
		mov	bl, [month]
		dec	bl
		shl	bl, 1
		;sub	edx, edx
```

```
        mov     dx, [EBX+DMonth]
         mov     bl, [day]
        dec     bl
        add     eax, edx
        add     eax, ebx
                        ; EAX = days since 1/1/1970
        mov     dx, [year]
        sub     dx, 1969
        shr     dx, 1
        shr     dx, 1
                ; (year-1969)/4
        add     eax, edx
                        ; + leap days since 1/1/1970
        cmp     byte [month], 2         ; if past february
        jna     short cte1
        mov     dx, [year]
        and     dx, 3 ; year mod 4
        jnz     short cte1
                        ; and if leap year
        add     eax, 1 ; add this year's leap day (february 29)
cte1:                   ; compute seconds since 1/1/1970
        mov     edx, 24
        mul     edx
        mov     dl, [hour]
        add     eax, edx
                ; EAX = hours since 1/1/1970 00:00:00
        ;mov    ebx, 60
        mov     bl, 60
        mul     ebx
        mov     dl, [minute]
        add     eax, edx
                ; EAX = minutes since 1/1/1970 00:00:00
        ;mov    ebx, 60
        mul     ebx
        mov     dl, [second]
        add     eax, edx
                ; EAX -> seconds since 1/1/1970 00:00:00
        retn

get_rtc_time:
        ; 15/03/2015
        ; Derived from IBM PC-XT Model 286 BIOS Source Code
        ; BIOS2.ASM ---- 10/06/1985 BIOS INTERRUPT ROUTINES
        ; INT 1Ah
        ; (AH) = 02H  READ THE REAL TIME CLOCK AND RETURN WITH,    :
        ;       (CH) = HOURS IN BCD (00-23)                 :
        ;       (CL) = MINUTES IN BCD (00-59)                      :
        ;       (DH) = SECONDS IN BCD (00-59)                      :
        ;       (DL) = DAYLIGHT SAVINGS ENABLE (00-01).            :
        ;
RTC_20:                                 ; GET RTC TIME
        cli
        CALL    UPD_IPR                 ; CHECK FOR UPDATE IN PROCESS
        JC      short RTC_29            ; EXIT IF ERROR (CY= 1)

        MOV     AL,CMOS_SECONDS         ; SET ADDRESS OF SECONDS
        CALL    CMOS_READ               ; GET SECONDS
        MOV     DH,AL                   ; SAVE
        MOV     AL,CMOS_REG_B           ; ADDRESS ALARM REGISTER
        CALL    CMOS_READ               ; READ CURRENT VALUE OF DSE BIT
        AND     AL,00000001B            ; MASK FOR VALID DSE BIT
        MOV     DL,AL                   ; SET [DL] TO ZERO FOR NO DSE BIT
        MOV     AL,CMOS_MINUTES         ; SET ADDRESS OF MINUTES
        CALL    CMOS_READ               ; GET MINUTES
        MOV     CL,AL                   ; SAVE
        MOV     AL,CMOS_HOURS           ; SET ADDRESS OF HOURS
        CALL    CMOS_READ               ; GET HOURS
        MOV     CH,AL                   ; SAVE
        CLC                             ; SET CY= 0
RTC_29:
        sti
        RETn                            ; RETURN WITH RESULT IN CARRY FLAG
```

```
get_rtc_date:
        ; 15/03/2015
        ; Derived from IBM PC-XT Model 286 BIOS Source Code
        ; BIOS2.ASM ---- 10/06/1985 BIOS INTERRUPT ROUTINES
        ; INT 1Ah
        ; (AH) = 04H  READ THE DATE FROM THE REAL TIME CLOCK AND RETURN WITH,:
        ;      (CH) = CENTURY IN BCD (19 OR 20)                     :
        ;      (CL) = YEAR IN BCD (00-99)                           :
        ;      (DH) = MONTH IN BCD (01-12)                          :
        ;      (DL) = DAY IN BCD (01-31).
        ;
RTC_40:                                 ; GET RTC DATE
        cli
        CALL    UPD_IPR                 ; CHECK FOR UPDATE IN PROCESS
        JC      short RTC_49            ; EXIT IF ERROR (CY= 1)

        MOV     AL,CMOS_DAY_MONTH       ; ADDRESS DAY OF MONTH
        CALL    CMOS_READ               ; READ DAY OF MONTH
        MOV     DL,AL                   ; SAVE
        MOV     AL,CMOS_MONTH           ; ADDRESS MONTH
        CALL    CMOS_READ               ; READ MONTH
        MOV     DH,AL                   ; SAVE
        MOV     AL,CMOS_YEAR            ; ADDRESS YEAR
        CALL    CMOS_READ               ; READ YEAR
        MOV     CL,AL                   ; SAVE
        MOV     AL,CMOS_CENTURY         ; ADDRESS CENTURY LOCATION
        CALL    CMOS_READ               ; GET CENTURY BYTE
        MOV     CH,AL                   ; SAVE
        CLC                             ; SET CY=0
RTC_49:
        sti
        RETn                            ; RETURN WITH RESULTS IN CARRY FLAG

set_date_time:
convert_from_epoch:
        ; 15/03/2015 (Retro UNIX 386 v1 - 32 bit version)
        ; 20/06/2013 (Retro UNIX 8086 v1)
        ; 'convert_from_epoch' procedure prototype:
        ;                  UNIXCOPY.ASM, 10/03/2013
        ;
        ; ((Modified registers: EAX, EDX, ECX, EBX))
        ;
        ; Derived from DALLAS Semiconductor
        ; Application Note 31 (DS1602/DS1603)
        ; 6 May 1998
        ;
        ; INPUT:
        ; EAX = Unix (Epoch) Time
        ;
        xor     edx, edx
        mov     ecx, 60
        div     ecx
        ;mov    [imin], eax    ; whole minutes
                         ; since 1/1/1970
        mov     [second], dx  ; leftover seconds
        sub     edx, edx
        div     ecx
        ;mov    [ihrs], eax    ; whole hours
        ;                    ; since 1/1/1970
        mov     [minute], dx  ; leftover minutes
        xor     edx, edx
        ;mov    cx, 24
        mov     cl, 24
        div     ecx
        ;mov    [iday], ax    ; whole days
                            ; since 1/1/1970
        mov     [hour], dx    ; leftover hours
        add     eax, 365+366 ; whole day since
                            ; 1/1/1968
        ;mov    [iday], ax
        push    eax
        sub     edx, edx
        mov     ecx, (4*365)+1 ; 4 years = 1461 days
        div     ecx
        pop     ecx
        ;mov    [lday], ax    ; count of quadyrs (4 years)
        push    dx
        ;mov    [qday], dx    ; days since quadyr began
        cmp     dx, 31 + 29   ; if past feb 29 then
```

```
        cmc                     ; add this quadyr's leap day
        adc     eax, 0          ; to # of qadyrs (leap days)
        ;mov    [lday], ax      ; since 1968
        ;mov    cx, [iday]
        xchg    ecx, eax        ; ECX = lday, EAX = iday
        sub     eax, ecx        ; iday - lday
        mov     ecx, 365
        xor     edx, edx
        ; EAX = iday-lday, EDX = 0
        div     ecx
        ;mov    [iyrs], ax      ; whole years since 1968
        ;jday = iday - (iyrs*365) - lday
        ;mov [jday], dx          ; days since 1/1 of current year
        ;add    eax, 1968
        add     ax, 1968        ; compute year
        mov     [year], ax
        mov     cx, dx
        ;mov    dx, [qday]
        pop     dx
        cmp     dx, 365         ; if qday <= 365 and qday >= 60
        ja      short cfe1      ; jday = jday +1
        cmp     dx, 60          ; if past 2/29 and leap year then
         cmc                    ; add a leap day to the # of whole
        adc     cx, 0           ; days since 1/1 of current year
cfe1:
        ;mov    [jday], cx
        mov     bx, 12          ; estimate month
        mov     dx, 366         ; mday, max. days since 1/1 is 365
        and     ax, 11b         ; year mod 4 (and dx, 3)
cfe2:   ; Month calculation     ; 0 to 11  (11 to 0)
        cmp     cx, dx          ; mday = # of days passed from 1/1
        jnb     short cfe3
        dec     bx              ; month = month - 1
        shl     bx, 1
        mov     dx, [EBX+DMonth] ; # elapsed days at 1st of month
        shr     bx, 1           ; bx = month - 1 (0 to 11)
        cmp     bx, 1           ; if month > 2 and year mod 4  = 0
        jna     short cfe2      ; then mday = mday + 1
        or      al, al          ; if past 2/29 and leap year then
        jnz     short cfe2      ; add leap day (to mday)
        inc     dx              ; mday = mday + 1
        jmp     short cfe2
cfe3:
        inc     bx              ; -> bx = month, 1 to 12
        mov     [month], bx
        sub     cx, dx          ; day = jday - mday + 1
        inc     cx
        mov     [day], cx

        ; eax, ebx, ecx, edx is changed at return
        ; output ->
        ; [year], [month], [day], [hour], [minute], [second]

        ; 15/03/2015 (Retro UNIX 386 v1 - 32 bit version)
        ; 20/06/2013 (Retro UNIX 8086 v1)
set_date:
         mov     al, [year+1]
        aam     ; ah = al / 10, al = al mod 10
        db      0D5h,10h        ; Undocumented inst. AAD
                                ; AL = AH * 10h + AL
        mov     ch, al ; century (BCD)
        mov     al, [year]
        aam     ; ah = al / 10, al = al mod 10
        db      0D5h,10h        ; Undocumented inst. AAD
                                ; AL = AH * 10h + AL
        mov     cl, al ; year (BCD)
         mov    al, [month]
        aam     ; ah = al / 10, al = al mod 10
        db      0D5h,10h        ; Undocumented inst. AAD
                                ; AL = AH * 10h + AL
        mov     dh, al ; month (BCD)
        mov     al, [day]
        aam     ; ah = al / 10, al = al mod 10
        db      0D5h,10h        ; Undocumented inst. AAD
                                ; AL = AH * 10h + AL
        mov     dh, al ; day (BCD)
        ; Set real-time clock date
        call    set_rtc_date
```

```
set_time:
         ; Read real-time clock time
        ; (get day light saving time bit status)
        cli
        CALL    UPD_IPR                ; CHECK FOR UPDATE IN PROCESS
        ; cf = 1 -> al = 0
         jc       short stime1
        MOV     AL,CMOS_REG_B          ; ADDRESS ALARM REGISTER
        CALL    CMOS_READ              ; READ CURRENT VALUE OF DSE BIT
stime1:
        sti
        AND     AL,00000001B           ; MASK FOR VALID DSE BIT
        MOV     DL,AL                  ; SET [DL] TO ZERO FOR NO DSE BIT
        ; DL = 1 or 0 (day light saving time)
        ;
        mov     al, [hour]
        aam     ; ah = al / 10, al = al mod 10
        db      0D5h,10h     ; Undocumented inst. AAD
                            ; AL = AH * 10h + AL
        mov     ch, al ; hour (BCD)
         mov     al, [minute]
        aam     ; ah = al / 10, al = al mod 10
        db      0D5h,10h     ; Undocumented inst. AAD
                            ; AL = AH * 10h + AL
        mov     cl, al       ; minute (BCD)
         mov     al, [second]
        aam     ; ah = al / 10, al = al mod 10
        db      0D5h,10h     ; Undocumented inst. AAD
                            ; AL = AH * 10h + AL
        mov     dh, al      ; second (BCD)
        ; Set real-time clock time
        ; call  set_rtc_time
set_rtc_time:
        ; 15/04/2015 (257, POSTEQU.INC -> H EQU 256, X EQU H+1)
        ; 15/03/2015
        ; Derived from IBM PC-XT Model 286 BIOS Source Code
        ; BIOS2.ASM ---- 10/06/1985 BIOS INTERRUPT ROUTINES
        ; INT 1Ah
        ; (AH) = 03H  SET THE REAL TIME CLOCK USING,                 :
        ;       (CH) = HOURS IN BCD (00-23)                          :
        ;       (CL) = MINUTES IN BCD (00-59)                        :
        ;       (DH) = SECONDS IN BCD (00-59)                        :
        ;       (DL) = 01 IF DAYLIGHT SAVINGS ENABLE OPTION, ELSE 00.     :
        ;                                                            :
        ;  NOTE: (DL)= 00 IF DAYLIGHT SAVINGS TIME ENABLE IS NOT ENABLED. :
        ;         (DL)= 01 ENABLES TWO SPECIAL UPDATES THE LAST SUNDAY IN  :
        ;          APRIL   (1:59:59 --> 3:00:00 AM) AND THE LAST SUNDAY IN :
        ;          OCTOBER (1:59:59 --> 1:00:00 AM) THE FIRST TIME.   :
        ;
RTC_30:                                ; SET RTC TIME
        cli
        CALL    UPD_IPR                ; CHECK FOR UPDATE IN PROCESS
        JNC     short RTC_35           ; GO AROUND IF CLOCK OPERATING
        CALL    RTC_STA                ; ELSE TRY INITIALIZING CLOCK
RTC_35:
        MOV     AH,DH                  ; GET TIME BYTE - SECONDS
        MOV     AL,CMOS_SECONDS        ; ADDRESS SECONDS
        CALL    CMOS_WRITE             ; UPDATE SECONDS
        MOV     AH,CL                  ; GET TIME BYTE - MINUTES
        MOV     AL,CMOS_MINUTES        ; ADDRESS MINUTES
        CALL    CMOS_WRITE             ; UPDATE MINUTES
        MOV     AH,CH                  ; GET TIME BYTE - HOURS
        MOV     AL,CMOS_HOURS          ; ADDRESS HOURS
        CALL    CMOS_WRITE             ; UPDATE ADDRESS
        ;MOV    AX,X*CMOS_REG_B        ; ADDRESS ALARM REGISTER
        MOV     AX,257*CMOS_REG_B      ;
        CALL    CMOS_READ              ; READ CURRENT TIME
        AND     AL,01100010B           ; MASK FOR VALID BIT POSITIONS
        OR      AL,00000010B           ; TURN ON 24 HOUR MODE
        AND     DL,00000001B           ; USE ONLY THE DSE BIT
        OR      AL,DL                  ; GET DAY LIGHT SAVINGS TIME BIT (OSE)
        XCHG    AH,AL                  ; PLACE IN WORK REGISTER AND GET ADDRESS
        CALL    CMOS_WRITE             ; SET NEW ALARM BITS
        CLC                            ; SET CY= 0
        sti
        RETn                           ; RETURN WITH CY= 0
```

```
set_rtc_date:
        ; 15/04/2015 (257, POSTEQU.INC -> H EQU 256, X EQU H+1)
        ; 15/03/2015
        ; Derived from IBM PC-XT Model 286 BIOS Source Code
        ; BIOS2.ASM ---- 10/06/1985 BIOS INTERRUPT ROUTINES
        ; INT 1Ah
        ; (AH) = 05H  SET THE DATE INTO THE REAL TIME CLOCK USING, :
        ;     (CH) = CENTURY IN BCD (19 OR 20)                        :
        ;     (CL) = YEAR IN BCD (00-99)                      :
        ;     (DH) = MONTH IN BCD (01-12)                     :
        ;     (DL) = DAY IN BCD (01-31).
        ;
RTC_50:                                 ; SET RTC DATE
        cli
        CALL    UPD_IPR                 ; CHECK FOR UPDATE IN PROCESS
        JNC     short RTC_55            ; GO AROUND IF NO ERROR
        CALL    RTC_STA                 ; ELSE INITIALIZE CLOCK
RTC_55:
        MOV     AX,CMOS_DAY_WEEK        ; ADDRESS OF DAY OF WEEK BYTE
        CALL    CMOS_WRITE              ; LOAD ZEROS TO DAY OF WEEK
        MOV     AH,DL                   ; GET DAY OF MONTH BYTE
        MOV     AL,CMOS_DAY_MONTH       ; ADDRESS DAY OF MONTH BYTE
        CALL    CMOS_WRITE              ; WRITE OF DAY OF MONTH REGISTER
        MOV     AH,DH                   ; GET MONTH
        MOV     AL,CMOS_MONTH           ; ADDRESS MONTH BYTE
        CALL    CMOS_WRITE              ; WRITE MONTH REGISTER
        MOV     AH,CL                   ; GET YEAR BYTE
        MOV     AL,CMOS_YEAR            ; ADDRESS YEAR REGISTER
        CALL    CMOS_WRITE              ; WRITE YEAR REGISTER
        MOV     AH,CH                   ; GET CENTURY BYTE
        MOV     AL,CMOS_CENTURY         ; ADDRESS CENTURY BYTE
        CALL    CMOS_WRITE              ; WRITE CENTURY LOCATION
        ;MOV    AX,X*CMOS_REG_B         ; ADDRESS ALARM REGISTER
        MOV     AX,257*CMOS_REG_B       ;
        CALL    CMOS_READ               ; READ CURRENT SETTINGS
        AND     AL,07FH                 ; CLEAR 'SET BIT'
        XCHG    AH,AL                   ; MOVE TO WORK REGISTER
        CALL    CMOS_WRITE              ; AND START CLOCK UPDATING
        CLC                             ; SET CY= 0
        sti
        RETn                            ; RETURN CY=0

        ; 15/03/2015
RTC_STA:                                ; INITIALIZE REAL TIME CLOCK
        mov     ah, 26h
        mov     al, CMOS_REG_A          ; ADDRESS REGISTER A AND LOAD DATA MASK
        CALL    CMOS_WRITE              ; INITIALIZE STATUS REGISTER A
        mov     ah, 82h
        mov     al, CMOS_REG_B          ; SET "SET BIT" FOR CLOCK INITIALIZATION
        CALL    CMOS_WRITE              ; AND 24 HOUR MODE TO REGISTER B
        MOV     AL,CMOS_REG_C           ; ADDRESS REGISTER C
        CALL    CMOS_READ               ; READ REGISTER C TO INITIALIZE
        MOV     AL,CMOS_REG_D           ; ADDRESS REGISTER D
        CALL    CMOS_READ               ; READ REGISTER D TO INITIALIZE
        RETn

        ; 15/03/2015
        ; IBM PC/XT Model 286 BIOS source code ----- 10/06/85 (test4.asm)
CMOS_WRITE:                     ; WRITE (AH) TO LOCATION (AL)
        pushf                   ; SAVE INTERRUPT ENABLE STATUS AND FLAGS
        ;push   ax              ; SAVE WORK REGISTER VALUES
        rol     al, 1           ; MOVE NMI BIT TO LOW POSITION
        stc                     ; FORCE NMI BIT ON IN CARRY FLAG
        rcr     al, 1           ; HIGH BIT ON TO DISABLE NMI - OLD IN CY
        cli                     ; DISABLE INTERRUPTS
        out     CMOS_PORT, al   ; ADDRESS LOCATION AND DISABLE NMI
        mov     al, ah          ; GET THE DATA BYTE TO WRITE
        out     CMOS_DATA, al   ; PLACE IN REQUESTED CMOS LOCATION
        mov     al, CMOS_SHUT_DOWN*2 ; GET ADDRESS OF DEFAULT LOCATION
        rcr     al, 1           ; PUT ORIGINAL NMI MASK BIT INTO ADDRESS
        out     CMOS_PORT, al   ; SET DEFAULT TO READ ONLY REGISTER
        nop                     ; I/O DELAY
        in      al, CMOS_DATA   ; OPEN STANDBY LATCH
        ;pop    ax              ; RESTORE WORK REGISTERS
        popf
        RETn
```

```
bf_init:
        ; 14/08/2015
        ; 02/07/2015
        ; 01/07/2015
        ; 15/04/2015 (Retro UNIX 386 v1 - Beginning)
        ; Buffer (pointer) initialization !
        ;
        ; 17/07/2013 - 24/07/2013
        ; Retro UNIX 8086 v1 (U9.ASM)
        ; (Retro UNIX 8086 v1 feature only !)
        ;
        mov     edi, bufp
        mov     eax, buffer + (nbuf*520)
        sub     edx, edx
        dec     dl
        xor     ecx, ecx
        dec     ecx
bi0:
        sub     eax, 520 ; 8 header + 512 data
        stosd
        mov     esi, eax
        mov     [esi], edx ; 000000FFh
                           ; Not a valid device sign
        mov     [esi+4], ecx ; 0FFFFFFFFh
                     ; Not a valid block number sign
        cmp     eax, buffer
        ja      short bi0
        mov     eax, sb0
        stosd
        mov     eax, sb1
        stosd
        mov     esi, eax ; offset sb1
        mov     [esi], edx ; 000000FFh
                           ; Not a valid device sign
        mov     [esi+4], ecx ; 0FFFFFFFFh
                     ; Not a valid block number sign
        ; 14/08/2015
        ;call   rdev_init
        ;retn

rdev_init: ; root device, super block buffer initialization
        ; 14/08/2015
        ; Retro UNIX 386 v1 feature only !
        ;
        ; NOTE: Disk partitions (file systems), logical
        ; drive initialization, partition's start sector etc.
        ; will be coded here, later in 'ldrv_init'

        movzx   eax, byte [boot_drv]
rdi_0:
        cmp     al, 80h
        jb      short rdi_1
        sub     al, 7Eh ; 80h = 2 (hd0), 81h = 3 (hd1)
rdi_1:
        mov     [rdev], al
         mov    ebx, sb0 ; super block buffer
        mov     [ebx], eax
        mov     al, 1 ; eax = 1
        mov     [ebx+4], eax ; super block address on disk
        call    diskio
        retn

; 23/10/2015
com1_irq4:
        dd dummy_retn
com2_irq3:
        dd dummy_retn

dummy_retn:
        retn
```

```
; Retro UNIX 386 v1 Kernel (v0.2) - SYS1.INC
; Last Modification: 23/11/2015
; ----------------------------------------------------------------------------
; Derived from 'Retro UNIX 8086 v1' source code by Erdogan Tan
; (v0.1 - Beginning: 11/07/2012)
;
; Derived 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>
;
; Retro UNIX 8086 v1 - U1.ASM (12/07/2014) //// UNIX v1 -> u1.s
;
; ****************************************************************************

unkni: ; / used for all system calls
sysent: ; < enter to system call >
        ;19/10/2015
       ; 21/09/2015
       ; 01/07/2015
       ; 19/05/2015
       ; 16/04/2015 (Retro UNIX 386 v1 - Beginning)
       ; 10/04/2013 - 18/01/2014 (Retro UNIX 8086 v1)
       ;
       ; 'unkni' or 'sysent' is sytem entry from various traps.
       ; The trap type is determined and an indirect jump is made to
       ; the appropriate system call handler. If there is a trap inside
       ; the system a jump to panic is made. All user registers are saved
       ; and u.sp points to the end of the users stack. The sys (trap)
       ; instructor is decoded to get the the system code part (see
       ; trap instruction in the PDP-11 handbook) and from this
       ; the indirect jump address is calculated. If a bad system call is
       ; made, i.e., the limits of the jump table are exceeded, 'badsys'
       ; is called. If the call is legitimate control passes to the
       ; appropriate system routine.
       ;
       ; Calling sequence:
       ;       Through a trap caused by any sys call outside the system.
       ; Arguments:
       ;       Arguments of particular system call.
       ; ...............................................................
       ;
       ; Retro UNIX 8086 v1 modification:
       ;       System call number is in EAX register.
       ;
       ;       Other parameters are in EDX, EBX, ECX, ESI, EDI, EBP
       ;       registers depending of function details.
       ;
       ; 16/04/2015
        mov     [ss:u.sp], esp ; Kernel stack points to return address
       ; save user registers
       push    ds
       push    es
       push    fs
       push    gs
       pushad  ; eax, ecx, edx, ebx, esp -before pushad-, ebp, esi, edi
       ;
       ; ESPACE = esp - [ss:u.sp] ; 4*12 = 48 ; 17/09/2015
       ;       (ESPACE is size of space in kernel stack
       ;       for saving/restoring user registers.)
       ;
       push    eax ; 01/07/2015
       mov      ax, KDATA
        mov     ds, ax
        mov     es, ax
        mov     fs, ax
        mov     gs, ax
       mov     eax, [k_page_dir]
       mov     cr3, eax
       pop     eax ; 01/07/2015
       ; 19/10/2015
       cld
       inc     byte [sysflg]
               ; incb sysflg / indicate a system routine is in progress
        sti    ; 18/01/2014
       jnz      panic ; 24/05/2013
               ; beq 1f
               ; jmp panic ; / called if trap inside system
;1:
```

```
        ; 16/04/2015
        mov     [u.r0], eax
        mov     [u.usp], esp ; kernel stack points to user's registers
        ;
                ; mov $s.syst+2,clockp
                ; mov r0,-(sp) / save user registers
                ; mov sp,u.r0 / pointer to bottom of users stack
                          ; / in u.r0
                ; mov r1,-(sp)
                ; mov r2,-(sp)
                ; mov r3,-(sp)
                ; mov r4,-(sp)
                ; mov r5,-(sp)
                ; mov ac,-(sp) / "accumulator" register for extended
                             ; / arithmetic unit
                ; mov mq,-(sp) / "multiplier quotient" register for the
                             ; / extended arithmetic unit
                ; mov sc,-(sp) / "step count" register for the extended
                             ; / arithmetic unit
                ; mov sp,u.sp / u.sp points to top of users stack
                ; mov 18.(sp),r0 / store pc in r0
                ; mov -(r0),r0 / sys inst in r0      10400xxx
                ; sub $sys,r0 / get xxx code
        shl     eax, 2
                ; asl r0 / multiply by 2 to jump indirect in bytes
        cmp     eax, end_of_syscalls - syscalls
                ; cmp r0,$2f-1f / limit of table (35) exceeded
        ;jnb    short badsys
                ; bhis badsys / yes, bad system call
        cmc
        pushf
        push    eax
        mov     ebp, [u.sp] ; Kernel stack at the beginning of sys call
        mov     al, 0FEh ; 11111110b
        adc     al, 0 ; al = al + cf
        and     [ebp+8], al ; flags (reset carry flag)
                ; bic $341,20.(sp) / set users processor priority to 0
                               ; / and clear carry bit
        pop     ebp ; eax
        popf
         jc       badsys
        mov     eax, [u.r0]
        ; system call registers: EAX, EDX, ECX, EBX, ESI, EDI
        jmp     dword [ebp+syscalls]
                ; jmp *1f(r0) / jump indirect thru table of addresses
                            ; / to proper system routine.
syscalls: ; 1:
        ; 21/09/2015
        ; 01/07/2015
        ; 16/04/2015 (32 bit address modification)
        dd sysrele     ; / 0
        dd sysexit     ; / 1
        dd sysfork     ; / 2
        dd sysread     ; / 3
        dd syswrite    ; / 4
        dd sysopen     ; / 5
        dd sysclose    ; / 6
        dd syswait     ; / 7
        dd syscreat    ; / 8
        dd syslink     ; / 9
        dd sysunlink   ; / 10
        dd sysexec     ; / 11
        dd syschdir    ; / 12
        dd systime     ; / 13
        dd sysmkdir    ; / 14
        dd syschmod    ; / 15
        dd syschown    ; / 16
        dd sysbreak    ; / 17
        dd sysstat     ; / 18
        dd sysseek     ; / 19
        dd systell     ; / 20
        dd sysmount    ; / 21
        dd sysumount   ; / 22
        dd syssetuid   ; / 23
        dd sysgetuid   ; / 24
        dd sysstime    ; / 25
        dd sysquit     ; / 26
        dd sysintr     ; / 27
        dd sysfstat    ; / 28
```

```
        dd sysemt      ; / 29
        dd sysmdate    ; / 30
        dd sysstty     ; / 31
        dd sysgtty     ; / 32
        dd sysilgins   ; / 33
        dd syssleep    ; 34 ; Retro UNIX 8086 v1 feature only !
                            ; 11/06/2014
        dd sysmsg      ; 35 ; Retro UNIX 386 v1 feature only !
                            ; 01/07/2015
        dd sysgeterr   ; 36 ; Retro UNIX 386 v1 feature only !
                            ; 21/09/2015 - get last error number
end_of_syscalls:

error:
        ; 17/09/2015
        ; 03/09/2015
        ; 01/09/2015
        ; 09/06/2015
        ; 13/05/2015
        ; 16/04/2015 (Retro UNIX 386 v1 - Beginning)
        ; 10/04/2013 - 07/08/2013 (Retro UNIX 8086 v1)
        ;
        ; 'error' merely sets the error bit off the processor status (c-bit)
        ; then falls right into the 'sysret', 'sysrele' return sequence.
        ;
        ; INPUTS -> none
        ; OUTPUTS ->
        ;       processor status - carry (c) bit is set (means error)
        ;
        ; 26/05/2013 (Stack pointer must be reset here!
        ;             Because, jumps to error procedure
        ;             disrupts push-pop nesting balance)
        ;
        mov     ebp, [u.sp] ; interrupt (system call) return (iretd) address
        or      byte [ebp+8], 1  ; set carry bit of flags register
                               ; (system call will return with cf = 1)
                ; bis $1,20.(r1) / set c bit in processor status word below
                              ; / users stack
        ; 17/09/2015
        sub     ebp, ESPACE ; 48 ; total size of stack frame ('sysdefs.inc')
                               ; for saving/restoring user registers
        ;cmp    ebp, [u.usp]
        ;je     short err0
        mov     [u.usp], ebp
;err0:
        ; 01/09/2015
        mov     esp, [u.usp]       ; Retro Unix 8086 v1 modification!
                                   ; 10/04/2013
                                   ; (If an I/O error occurs during disk I/O,
                                   ; related procedures will jump to 'error'
                                   ; procedure directly without returning to
                                   ; the caller procedure. So, stack pointer
                                     ; must be restored here.)
        ; 13/05/2015
        ; NOTE: (The last) error code is in 'u.error', it can be retrieved by
        ;       'get last error' system call later.

        ; 03/09/2015 - 09/06/2015 - 07/08/2013
        mov     byte [u.kcall], 0 ; namei_r, mkdir_w reset

sysret: ; < return from system call>
        ; 10/09/2015
        ; 29/07/2015
        ; 25/06/2015
        ; 16/04/2015 (Retro UNIX 386 v1 - Beginning)
        ; 10/04/2013 - 23/02/2014 (Retro UNIX 8086 v1)
        ;
        ; 'sysret' first checks to see if process is about to be
        ; terminated (u.bsys). If it is, 'sysexit' is called.
        ; If not, following happens:
        ;       1) The user's stack pointer is restored.
        ;       2) r1=0 and 'iget' is called to see if last mentioned
        ;          i-node has been modified. If it has, it is written out
        ;          via 'ppoke'.
        ;       3) If the super block has been modified, it is written out
        ;          via 'ppoke'.
        ;       4) If the dismountable file system's super block has been
        ;          modified, it is written out to the specified device
        ;          via 'ppoke'.
```

```
        ;       5) A check is made if user's time quantum (uquant) ran out
        ;          during his execution. If so, 'tswap' is called to give
        ;          another user a chance to run.
        ;       6) 'sysret' now goes into 'sysrele'.
        ;           (See 'sysrele' for conclusion.)
        ;
        ; Calling sequence:
        ;       jump table or 'br sysret'
        ; Arguments:
        ;       -
        ; ...............................................................
        ;
        ; ((AX=r1 for 'iget' input))
        ;
        xor     ax, ax ; 04/05/2013
sysret0: ; 29/07/2015 (eax = 0, jump from sysexec)
        inc     al ; 04/05/2013
        cmp     [u.bsys], al ; 1
                ; tstb u.bsys / is a process about to be terminated because
         jnb     sysexit ; 04/05/2013
                ; bne sysexit / of an error? yes, go to sysexit
        ;mov    esp, [u.usp] ; 24/05/2013 (that is not needed here)
                ; mov u.sp,sp / no point stack to users stack
        dec     al ; mov ax, 0
                ; clr r1 / zero r1 to check last mentioned i-node
        call    iget
                ; jsr r0,iget / if last mentioned i-node has been modified
                            ; / it is written out
        xor     ax, ax ; 0
        cmp     [smod], al ; 0
                ; tstb  smod / has the super block been modified
        jna     short sysret1
                ; beq   1f / no, 1f
        mov     [smod], al ; 0
                ; clrb smod / yes, clear smod
        mov     ebx, sb0 ;; 07/08//2013
        or      word [ebx], 200h ;;
        ;or     word [sb0], 200h ; write bit, bit 9
                ; bis $1000,sb0 / set write bit in I/O queue for super block
                             ; / output
        ; AX = 0
        call    poke ; 07/08/2013
        ; call ppoke
        ; AX = 0
                ; jsr r0,ppoke / write out modified super block to disk
sysret1: ;1:
        cmp     [mmod], al ; 0
                ; tstb  mmod / has the super block for the dismountable file
                           ; / system
        jna     short sysrel0
                ; beq 1f / been modified?  no, 1f
        mov     [mmod], al ; 0
                ; clrb  mmod / yes, clear mmod
         ;mov    ax, [mntd]
         ;;mov   al, [mdev] ; 26/04/2013
        mov     ebx, sb1 ;; 07/08//2013
         ;;mov  [ebx], al
        ;mov     [sb1], al
                ; movb mntd,sb1 / set the I/O queue
        or      word [ebx], 200h
        ;or     word [sb1], 200h ; write bit, bit 9
                ; bis $1000,sb1 / set write bit in I/O queue for detached sb
        call    poke ; 07/08/2013
        ;call   ppoke
                ; jsr r0,ppoke / write it out to its device
         ;xor    al, al ; 26/04/2013
;1:
                ; tstb uquant / is the time quantum 0?
                ; bne 1f / no, don't swap it out
```

```
sysrele: ; < release >
        ; 14/10/2015
        ; 01/09/2015
        ; 24/07/2015
        ; 14/05/2015
        ; 16/04/2015 (Retro UNIX 386 v1 - Beginning)
        ; 10/04/2013 - 07/03/2014 (Retro UNIX 8086 v1)
        ;
        ; 'sysrele' first calls 'tswap' if the time quantum for a user is
        ; zero (see 'sysret'). It then restores the user's registers and
        ; turns off the system flag. It then checked to see if there is
        ; an interrupt from the user by calling 'isintr'. If there is,
        ; the output gets flashed (see isintr) and interrupt action is
        ; taken by a branch to 'intract'. If there is no interrupt from
        ; the user, a rti is made.
        ;
        ; Calling sequence:
        ;       Fall through a 'bne' in 'sysret' & ?
        ; Arguments:
        ;       -
        ; ...............................................................
        ;
        ; 23/02/2014 (swapret)
        ; 22/09/2013
sysrel0: ;1:
        cmp     byte [u.quant], 0 ; 16/05/2013
                ; tstb uquant / is the time quantum 0?
         ja      short swapret
                ; bne 1f / no, don't swap it out
sysrelease: ; 07/12/2013 (jump from 'clock')
        call    tswap
                ; jsr r0,tswap / yes, swap it out
;
; Retro Unix 8086 v1 feature: return from 'swap' to 'swapret' address.
swapret: ;1:
        ; 10/09/2015
        ; 01/09/2015
        ; 14/05/2015
        ; 16/04/2015 (Retro UNIX 386 v1 - 32 bit, pm modifications)
        ; 26/05/2013 (Retro UNIX 8086 v1)
        ; cli
        ; 24/07/2015
        ;
        ;; 'esp' must be already equal to '[u.usp]' here !
        ;; mov  esp, [u.usp]

        ; 22/09/2013
        call    isintr
        ; 20/10/2013
        jz      short sysrel1
        call    intract
                ; jsr r0,isintr / is there an interrupt from the user
                ;     br intract / yes, output gets flushed, take interrupt
                               ; / action
sysrel1:
        cli ; 14/10/2015
        dec     byte [sysflg]
                ; decb sysflg / turn system flag off
        mov      eax, [u.pgdir]
        mov     cr3, eax  ; 1st PDE points to Kernel Page Table 0 (1st 4 MB)
                         ; (others are different than kernel page tables)
        ; 10/09/2015
        popad ; edi, esi, ebp, temp (icrement esp by 4), ebx, edx, ecx, eax
                ; mov (sp)+,sc / restore user registers
                ; mov (sp)+,mq
                ; mov (sp)+,ac
                ; mov (sp)+,r5
                ; mov (sp)+,r4
                ; mov (sp)+,r3
                ; mov (sp)+,r2
        ;
        mov     eax, [u.r0]  ; ((return value in EAX))
        pop     gs
        pop     fs
        pop     es
        pop     ds
        iretd
                ; rti / no, return from interrupt
```

```
badsys:
        ; 16/04/2015 (Retro UNIX 386 v1 - Beginning)
        ; (Major Modification: 'core' dumping procedure in
        ;        original UNIX v1 and Retro UNIX 8086 v1
        ;       has been changed to print 'Invalid System Call !'
        ;       message on the user's console tty.)
        ; (EIP, EAX values will be shown on screen with error message)
        ; (EIP = Return address just after the system call -INT 30h-)
        ; (EAX = Function number)
        ;
        inc     byte [u.bsys]
        ;
        mov     ebx, [u.sp] ; esp at the beginning of 'sysent'
        mov     eax, [ebx] ; EIP (return address, not 'INT 30h' address)
        call    dwordtohex
        mov     [bsys_msg_eip], edx
        mov     [bsys_msg_eip+4], eax
        mov     eax, [u.r0]
        call    dwordtohex
        mov     [bsys_msg_eax], edx
        mov     [bsys_msg_eax+4], eax
        xor     eax, eax
         mov     dword [u.base], badsys_msg ; "Invalid System call !"
        mov     ebx, [u.fofp]
        mov     [ebx], eax
        ;mov    eax, 1 ; inode number of console tty (for user)
        inc     eax
        mov     dword [u.count], BSYS_M_SIZE
                ; writei
                ; INPUTS ->
                ;    r1 - inode number
                ;    u.count - byte count to be written
                ;    u.base - points to user buffer
                ;    u.fofp - points to word with current file offset
                ; OUTPUTS ->
                ;    u.count - cleared
                ;    u.nread - accumulates total bytes passed back
                ;
                ; ((Modified registers: EDX, EBX, ECX, ESI, EDI, EBP))
        call    writei
        ;mov    eax, 1
        jmp     sysexit

                ; incb u.bsys / turn on the user's bad-system flag
                ; mov $3f,u.namep / point u.namep to "core\0\0"
                ; jsr r0,namei / get the i-number for the core image file
                ; br 1f / error
                ; neg r1 / negate the i-number to open the core image file
                       ; / for writing
                ; jsr r0,iopen / open the core image file
                ; jsr r0,itrunc / free all associated blocks
                ; br 2f
;1:
                ; mov $17,r1 / put i-node mode (17) in r1
                ; jsr r0,maknod / make an i-node
                ; mov u.dirbuf,r1 / put i-node number in r1
;2:
                ; mov $core,u.base / move address core to u.base
                ; mov $ecore-core,u.count / put the byte count in u.count
                ; mov $u.off,u.fofp / more user offset to u.fofp
                ; clr u.off / clear user offset
                ; jsr r0,writei / write out the core image to the user
                ; mov $user,u.base / pt. u.base to user
                ; mov $64.,u.count / u.count = 64
                ; jsr r0,writei / write out all the user parameters
                ; neg r1 / make i-number positive
                ; jsr r0,iclose / close the core image file
                ; br sysexit /
;3:
                ; <core\0\0>
```

```
intract: ; / interrupt action
        ; 14/10/2015
        ; 16/04/2015 (Retro UNIX 386 v1 - Beginning)
        ; 09/05/2013 - 07/12/2013 (Retro UNIX 8086 v1)
        ;
        ; Retro UNIX 8086 v1 modification !
        ; (Process/task switching and quit routine by using
        ; Retro UNIX 8086 v1 keyboard interrupt output.))
        ;
        ; input -> 'u.quit'  (also value of 'u.intr' > 0)
        ; output -> If value of 'u.quit' = FFFFh ('ctrl+brk' sign)
        ;              'intract' will jump to 'sysexit'.
        ;          Intract will return to the caller
        ;              if value of 'u.quit' <> FFFFh.
        ; 14/10/2015
        sti
        ; 07/12/2013
        inc     word [u.quit]
        jz      short intrct0 ; FFFFh -> 0
        dec     word [u.quit]
        ; 16/04/2015
        retn
intrct0:
        pop     eax ; call intract -> retn
        ;
        xor     eax, eax
        inc     al  ; mov ax, 1
;;;
        ; UNIX v1 original 'intract' routine...
        ; / interrupt action
                ;cmp *(sp),$rti / are you in a clock interrupt?
                ; bne 1f / no, 1f
                ; cmp (sp)+,(sp)+ / pop clock pointer
        ; 1: / now in user area
                ; mov r1,-(sp) / save r1
                ; mov u.ttyp,r1
                        ; / pointer to tty buffer in control-to r1
                ; cmpb 6(r1),$177
                        ; / is the interrupt char equal to "del"
                ; beq 1f / yes, 1f
                ; clrb 6(r1)
                        ; / no, clear the byte
                        ; / (must be a quit character)
                ; mov (sp)+,r1 / restore r1
                ; clr u.quit / clear quit flag
                ; bis $20,2(sp)
                        ; / set trace for quit (sets t bit of
                        ; / ps-trace trap)
                ; rti   ;  / return from interrupt
        ; 1: / interrupt char = del
                ; clrb 6(r1) / clear the interrupt byte
                           ; / in the buffer
                ; mov (sp)+,r1 / restore r1
                ; cmp u.intr,$core / should control be
                               ; / transferred to loc core?
                ; blo 1f
                ; jmp *u.intr / user to do rti yes,
                               ; / transfer to loc core
        ; 1:
                ; sys 1 / exit
```

```
sysexit: ; <terminate process>
        ; 01/09/2015
        ; 31/08/2015
        ; 14/05/2015
        ; 16/04/2015 (Retro UNIX 386 v1 - Beginning)
        ; 19/04/2013 - 14/02/2014 (Retro UNIX 8086 v1)
        ;
        ; 'sysexit' terminates a process. First each file that
        ; the process has opened is closed by 'flose'. The process
        ; status is then set to unused. The 'p.pid' table is then
        ; searched to find children of the dying process. If any of
        ; children are zombies (died by not waited for), they are
        ; set free. The 'p.pid' table is then searched to find the
        ; dying process's parent. When the parent is found, it is
        ; checked to see if it is free or it is a zombie. If it is
        ; one of these, the dying process just dies. If it is waiting
        ; for a child process to die, it notified that it doesn't
        ; have to wait anymore by setting it's status from 2 to 1
        ; (waiting to active). It is awakened and put on runq by
        ; 'putlu'. The dying process enters a zombie state in which
        ; it will never be run again but stays around until a 'wait'
        ; is completed by it's parent process. If the parent is not
        ; found, process just dies. This means 'swap' is called with
        ; 'u.uno=0'. What this does is the 'wswap' is not called
        ; to write out the process and 'rswap' reads the new process
        ; over the one that dies..i.e., the dying process is
        ; overwritten and destroyed.
        ;
        ; Calling sequence:
        ;       sysexit or conditional branch.
        ; Arguments:
        ;       -
        ; ...............................................................
        ;
        ; Retro UNIX 8086 v1 modification:
        ;       System call number (=1) is in EAX register.
        ;
        ;       Other parameters are in EDX, EBX, ECX, ESI, EDI, EBP
        ;       registers depending of function details.
        ;
        ; ('swap' procedure is mostly different than original UNIX v1.)
        ;
; / terminate process
        ; AX = 1
        dec     ax ; 0
        mov     [u.intr], ax ; 0
                ; clr u.intr / clear interrupt control word
                ; clr r1 / clear r1
        ; AX = 0
sysexit_1: ; 1:
        ; AX = File descriptor
                ; / r1 has file descriptor (index to u.fp list)
                ; / Search the whole list
        call    fclose
                ; jsr r0,fclose / close all files the process opened
        ;; ignore error return
                ; br .+2 / ignore error return
        ;inc    ax
        inc     al
                ; inc r1 / increment file descriptor
        ;cmp    ax, 10
        cmp     al, 10
                ; cmp r1,$10. / end of u.fp list?
        jb      short sysexit_1
                ; blt 1b / no, go back
        movzx   ebx, byte [u.uno] ; 01/09/2015
                ; movb u.uno,r1 / yes, move dying process's number to r1
        mov     [ebx+p.stat-1], ah ; 0, SFREE, 05/02/2014
                ; clrb p.stat-1(r1) / free the process
        ;shl    bx, 1
        shl     bl, 1
                ; asl r1 / use r1 for index into the below tables
        mov     cx, [ebx+p.pid-2]
                ; mov p.pid-2(r1),r3 / move dying process's name to r3
        mov     dx, [ebx+p.ppid-2]
                ; mov p.ppid-2(r1),r4 / move its parents name to r4
        ; xor   bx, bx ; 0
        xor     bl, bl ; 0
                ; clr r2
```

```
        xor     esi, esi ; 0
                ; clr r5 / initialize reg
sysexit_2: ; 1:
                 ; / find children of this dying process,
                ; / if they are zombies, free them
        ;add    bx, 2
        add     bl, 2
                ; add $2,r2 / search parent process table
                          ; / for dying process's name
        cmp     [ebx+p.ppid-2], cx
                ; cmp p.ppid-2(r2),r3 / found it?
        jne     short sysexit_4
                ; bne 3f / no
        ;shr    bx, 1
        shr     bl, 1
                ; asr r2 / yes, it is a parent
        cmp     byte [ebx+p.stat-1], 3 ; SZOMB, 05/02/2014
                ; cmpb p.stat-1(r2),$3 / is the child of this
                                    ; / dying process a zombie
        jne     short sysexit_3
                ; bne 2f / no
        mov     [ebx+p.stat-1], ah ; 0, SFREE, 05/02/2014
                ; clrb p.stat-1(r2) / yes, free the child process
sysexit_3: ; 2:
        ;shr    bx, 1
        shl     bl, 1
                ; asl r2
sysexit_4: ; 3:
                ; / search the process name table
                ; / for the dying process's parent
        cmp     [ebx+p.pid-2], dx ; 17/09/2013
                ; cmp p.pid-2(r2),r4 / found it?
        jne     short sysexit_5
                ; bne 3f / no
        mov     esi, ebx
                ; mov r2,r5 / yes, put index to p.pid table (parents
                          ; / process # x2) in r5
sysexit_5: ; 3:
        ;cmp    bx, nproc + nproc
        cmp     bl, nproc + nproc
                ; cmp r2,$nproc+nproc / has whole table been searched?
        jb      short sysexit_2
                ; blt 1b / no, go back
                ; mov r5,r1 / yes, r1 now has parents process # x2
        and     esi, esi ; r5=r1
        jz      short sysexit_6
                ; beq 2f / no parent has been found.
                       ; / The process just dies
        shr     si, 1
                ; asr r1 / set up index to p.stat
        mov     al, [esi+p.stat-1]
                ; movb p.stat-1(r1),r2 / move status of parent to r2
        and     al, al
        jz      short sysexit_6
                ; beq 2f / if its been freed, 2f
        cmp     al, 3
                ; cmp r2,$3 / is parent a zombie?
        je      short sysexit_6
                ; beq 2f / yes, 2f
        ; BH = 0
        mov     bl, [u.uno]
                ; movb u.uno,r3 / move dying process's number to r3
        mov     byte [ebx+p.stat-1], 3  ; SZOMB, 05/02/2014
                ; movb $3,p.stat-1(r3) / make the process a zombie
        ; 05/02/2014
        cmp     al, 1 ; SRUN
        je      short sysexit_6
        ;cmp    al, 2
                ; cmp r2,$2 / is the parent waiting for
                         ; / this child to die
        ;jne    short sysexit_6
                ; bne 2f / yes, notify parent not to wait any more
        ; 05/02/2014
        ; p.stat = 2 --> waiting
        ; p.stat = 4 --> sleeping
        mov     byte [esi+p.stat-1], 1 ; SRUN ; 05/02/2014
        ;dec    byte [esi+p.stat-1]
                ; decb  p.stat-1(r1) / awaken it by putting it (parent)
        mov     ax, si ; r1  (process number in AL)
```

```
		;mov	ebx, runq + 4
			; mov $runq+4,r2 / on the runq
		call	putlu
			; jsr r0, putlu
sysexit_6: ; 2:
		; 31/08/2015
			; / the process dies
		mov	byte [u.uno], 0
			; clrb u.uno / put zero as the process number,
			    ; / so "swap" will
		call	swap
			; jsr r0,swap / overwrite process with another process
hlt_sys:
		;sti ; 18/01/2014
hlts0:
		hlt
		jmp	short hlts0
			; 0 / and thereby kill it; halt?

syswait: ; < wait for a processs to die >
		; 17/09/2015
		; 02/09/2015
		; 01/09/2015
		; 16/04/2015 (Retro UNIX 386 v1 - Beginning)
		; 24/05/2013 - 05/02/2014 (Retro UNIX 8086 v1)
		;
		; 'syswait' waits for a process die.
		; It works in following way:
		;    1) From the parent process number, the parent's
		;	process name is found. The p.ppid table of parent
		;	names is then searched for this process name.
		;	If a match occurs, r2 contains child's process
		;	number. The child status is checked to see if it is
		;	a zombie, i.e; dead but not waited for (p.stat=3)
		;	If it is, the child process is freed and it's name
		;	is put in (u.r0). A return is then made via 'sysret'.
		;	If the child is not a zombie, nothing happens and
		;	the search goes on through the p.ppid table until
		;	all processes are checked or a zombie is found.
		;    2) If no zombies are found, a check is made to see if
		;	there are any children at all. If there are none,
		;	an error return is made. If there are, the parent's
		;	status is set to 2 (waiting for child to die),
		;	the parent is swapped out, and a branch to 'syswait'
		;	is made to wait on the next process.
		;
		; Calling sequence:
		;	?
		; Arguments:
		;	-
		; Inputs: -
		; Outputs: if zombie found, it's name put in u.r0.
		; ...............................................................
		;

; / wait for a process to die

syswait_0:
		movzx	ebx, byte [u.uno] ; 01/09/2015
			; movb u.uno,r1 / put parents process number in r1
		shl	bl, 1
		;shl	bx, 1
			; asl r1 / x2 to get index into p.pid table
		mov	ax, [ebx+p.pid-2]
			; mov p.pid-2(r1),r1 / get the name of this process
		xor	esi, esi
			; clr r2
		xor	ecx, ecx ; 30/10/2013
		;xor	cl, cl
			; clr r3 / initialize reg 3
syswait_1: ; 1:
		add	si, 2
			; add $2,r2 / use r2 for index into p.ppid table
				; / search table of parent processes
				; / for this process name
		cmp	ax, [esi+p.ppid-2]
			; cmp p.ppid-2(r2),r1 / r2 will contain the childs
					; / process number
		jne	short syswait_3
```

```
                ;bne 3f / branch if no match of parent process name
        ;inc    cx
        inc     cl
                ;inc r3 / yes, a match, r3 indicates number of children
        shr     si, 1
                ; asr r2 / r2/2 to get index to p.stat table
        ; The possible states ('p.stat' values) of a process are:
        ;       0 = free or unused
        ;       1 = active
        ;       2 = waiting for a child process to die
        ;       3 = terminated, but not yet waited for (zombie).
        cmp     byte [esi+p.stat-1], 3 ; SZOMB, 05/02/2014
                ; cmpb p.stat-1(r2),$3 / is the child process a zombie?
        jne     short syswait_2
                ; bne 2f / no, skip it
        mov     [esi+p.stat-1], bh ; 0
                ; clrb p.stat-1(r2) / yes, free it
        shl     si, 1
                ; asl r2 / r2x2 to get index into p.pid table
        movzx   eax, word [esi+p.pid-2]
        mov     [u.r0], eax
                ; mov p.pid-2(r2),*u.r0
                              ; / put childs process name in (u.r0)
        ;
        ; Retro UNIX 386 v1 modification ! (17/09/2015)
        ;
        ; Parent process ID -p.ppid- field (of the child process)
        ; must be cleared in order to prevent infinitive 'syswait'
        ; system call loop from the application/program if it calls
        ; 'syswait' again (mistakenly) while there is not a zombie
        ; or running child process to wait. ('forktest.s', 17/09/2015)
        ;
        ; Note: syswait will return with error if there is not a
        ;       zombie or running process to wait.
        ;
        sub     ax, ax
        mov     [esi+p.ppid-2], ax ; 0 ; 17/09/2015
        jmp     sysret0 ; ax = 0
        ;
        ;jmp    sysret
                ; br sysret1 / return cause child is dead
syswait_2: ; 2:
        shl     si, 1
                ; asl r2 / r2x2 to get index into p.ppid table
syswait_3: ; 3:
        cmp     si, nproc+nproc
                ; cmp r2,$nproc+nproc / have all processes been checked?
        jb      short syswait_1
                ; blt 1b / no, continue search
        ;and    cx, cx
        and     cl, cl
                ; tst r3 / one gets here if there are no children
                       ; / or children that are still active
        ; 30/10/2013
        jnz     short syswait_4
        ;jz     error
                ; beq error1 / there are no children, error
        mov     [u.r0], ecx ; 0
        jmp     error
syswait_4:
        mov     bl, [u.uno]
                ; movb u.uno,r1 / there are children so put
                              ; / parent process number in r1
        inc     byte [ebx+p.stat-1] ; 2, SWAIT, 05/02/2014
                ; incb p.stat-1(r1) / it is waiting for
                                 ; / other children to die
        ; 04/11/2013
        call    swap
                ; jsr r0,swap / swap it out, because it's waiting
        jmp     syswait_0
                ; br syswait / wait on next process
```

```
sysfork: ; < create a new process >
        ; 18/09/2015
        ; 04/09/2015
        ; 02/09/2015
        ; 01/09/2015
        ; 28/08/2015
        ; 14/05/2015
        ; 10/05/2015
        ; 09/05/2015
        ; 06/05/2015 (Retro UNIX 386 v1 - Beginning)
        ; 24/05/2013 - 14/02/2014 (Retro UNIX 8086 v1)
        ;
        ; 'sysfork' creates a new process. This process is referred
        ; to as the child process. This new process core image is
        ; a copy of that of the caller of 'sysfork'. The only
        ; distinction is the return location and the fact that (u.r0)
        ; in the old process (parent) contains the process id (p.pid)
        ; of the new process (child). This id is used by 'syswait'.
        ; 'sysfork' works in the following manner:
        ;    1) The process status table (p.stat) is searched to find
        ;       a process number that is unused. If none are found
        ;       an error occurs.
        ;    2) when one is found, it becomes the child process number
        ;       and it's status (p.stat) is set to active.
        ;    3) If the parent had a control tty, the interrupt
        ;       character in that tty buffer is cleared.
        ;    4) The child process is put on the lowest priority run
        ;       queue via 'putlu'.
        ;    5) A new process name is gotten from 'mpid' (actually
        ;       it is a unique number) and is put in the child's unique
        ;       identifier; process id (p.pid).
        ;    6) The process name of the parent is then obtained and
        ;       placed in the unique identifier of the parent process
        ;       name is then put in 'u.r0'.
        ;    7) The child process is then written out on disk by
        ;       'wswap',i.e., the parent process is copied onto disk
        ;       and the child is born. (The child process is written
        ;       out on disk/drum with 'u.uno' being the child process
        ;       number.)
        ;    8) The parent process number is then restored to 'u.uno'.
        ;    9) The child process name is put in 'u.r0'.
        ;   10) The pc on the stack sp + 18 is incremented by 2 to
        ;       create the return address for the parent process.
        ;   11) The 'u.fp' list as then searched to see what files
        ;       the parent has opened. For each file the parent has
        ;       opened, the corresponding 'fsp' entry must be updated
        ;       to indicate that the child process also has opened
        ;       the file. A branch to 'sysret' is then made.

        ;
        ; Calling sequence:
        ;       from shell ?
        ; Arguments:
        ;       -
        ; Inputs: -
        ; Outputs: *u.r0 - child process name
        ; ...............................................................
        ;
        ; Retro UNIX 8086 v1 modification:
        ;       AX = r0 = PID (>0) (at the return of 'sysfork')
        ;       = process id of child a parent process returns
        ;       = process id of parent when a child process returns
        ;
        ;        In original UNIX v1, sysfork is called and returns as
        ;       in following manner: (with an example: c library, fork)
        ;
        ;       1:
        ;               sys     fork
        ;                       br 1f   / child process returns here
        ;               bes     2f      / parent process returns here
        ;               / pid of new process in r0
        ;               rts     pc
        ;       2: / parent process condionally branches here
        ;               mov     $-1,r0 / pid = -1 means error return
        ;               rts     pc
        ;
        ;       1: / child process brances here
        ;               clr     r0    / pid = 0 in child process
        ;               rts     pc
```

```
	;	In UNIX v7x86 (386) by Robert Nordier (1999)
	;		// pid = fork();
	;		//
	;		// pid == 0 in child process;
	;		// pid == -1 means error return
	;		// in child,
	;		//	parents id is in par_uid if needed
	;
	;		_fork:
	;			mov	$.fork,eax
	;			int	$0x30
	;			jmp	1f
	;			jnc	2f
	;			jmp	cerror
	;		1:
	;			mov	eax,_par_uid
	;			xor	eax,eax
	;		2:
	;			ret
	;
	;	In Retro UNIX 8086 v1,
	;	'sysfork' returns in following manner:
	;
	;		mov	ax, sys_fork
	;		mov	bx, offset @f ; routine for child
	;		int	20h
	;		jc	error
	;
	;	; Routine for parent process here (just after 'jc')
	;		mov	word ptr [pid_of_child], ax
	;		jmp	next_routine_for_parent
	;
	;	@@: ; routine for child process here
	;		....
	;	NOTE: 'sysfork' returns to specified offset
	;	       for child process by using BX input.
	;	      (at first, parent process will return then
	;	      child process will return -after swapped in-
	;	      'syswait' is needed in parent process
	;	      if return from child process will be waited for.)

; / create a new process
	; EBX = return address for child process
	     ; (Retro UNIX 8086 v1 modification !)
	xor	esi, esi
		; clr r1
sysfork_1: ; 1: / search p.stat table for unused process number
	inc	esi
		; inc r1
	cmp	byte [esi+p.stat-1], 0 ; SFREE, 05/02/2014
		; tstb p.stat-1(r1) / is process active, unused, dead
	jna	short sysfork_2
		; beq 1f / it's unused so branch
	cmp	si, nproc
		; cmp r1,$nproc / all processes checked
	jb	short sysfork_1
		; blt 1b / no, branch back
	;
	; Retro UNIX 8086 v1. modification:
	;	Parent process returns from 'sysfork' to address
	;	which is just after 'sysfork' system call in parent
	;	process. Child process returns to address which is put
	;	in BX register by parent process for 'sysfork'.
	;
		;add $2,18.(sp) / add 2 to pc when trap occured, points
			     ; / to old process return
		; br error1 / no room for a new process
	jmp	error
sysfork_2: ; 1:
	call	allocate_page
	jc	error
	push	eax	; UPAGE (user structure page) address
	; Retro UNIX 386 v1 modification!
	call	duplicate_page_dir
		; EAX = New page directory
	jnc	short sysfork_3
	pop	eax	; UPAGE (user structure page) address
	call	deallocate_page
	jmp	error
```

```
sysfork_3:
        ; Retro UNIX 386 v1 modification !
        push    esi
        call    wswap ; save current user (u) structure, user registers
                      ; and interrupt return components (for IRET)
        xchg    eax, [u.pgdir] ; page directory of the child process
        mov     [u.ppgdir], eax ; page directory of the parent process
        pop     esi
        pop     eax   ; UPAGE (user structure page) address
                ; [u.usp] = esp
        mov     edi, esi
        shl     di, 2
        mov     [edi+p.upage-4], eax ; memory page for 'user' struct
        mov     [u.upage], eax ; memory page for 'user' struct (child)
        ; 28/08/2015
        movzx   eax, byte [u.uno] ; parent process number
                ; movb u.uno,-(sp) / save parent process number
        mov     edi, eax
         push   eax ; **
        mov      al, [edi+p.ttyc-1] ; console tty (parent)
        ; 18/09/2015
        ;mov      [esi+p.ttyc-1], al ; set child's console tty
        ;mov      [esi+p.waitc-1], ah ; 0 ; reset child's wait channel
        mov      [esi+p.ttyc-1], ax ; al - set child's console tty
                                 ; ah - reset child's wait channel
        mov     eax, esi
        mov     [u.uno], al ; child process number
                ;movb r1,u.uno / set child process number to r1
         inc      byte [esi+p.stat-1] ; 1, SRUN, 05/02/2014
                ; incb p.stat-1(r1) / set p.stat entry for child
                               ; / process to active status
                ; mov u.ttyp,r2 / put pointer to parent process'
                              ; / control tty buffer in r2
                 ; beq 2f / branch, if no such tty assigned
                ; clrb 6(r2) / clear interrupt character in tty buffer
        ; 2:
        push    ebx   ; * return address for the child process
                      ; * Retro UNIX 8086 v1 feature only !
        ; (Retro UNIX 8086 v1 modification!)
                ; mov $runq+4,r2
        call    putlu
                ; jsr r0,putlu / put child process on lowest priority
                          ; / run queue
        shl     si, 1
                ; asl r1 / multiply r1 by 2 to get index
                       ; / into p.pid table
        inc     word [mpid]
                ; inc mpid / increment m.pid; get a new process name
        mov     ax, [mpid]
        mov     [esi+p.pid-2], ax
                ;mov mpid,p.pid-2(r1) / put new process name
                                   ; / in child process' name slot
        pop     edx   ; * return address for the child process
                      ; * Retro UNIX 8086 v1 feature only !
        pop     ebx   ; **
        ;mov    ebx, [esp] ; ** parent process number
                ; movb (sp),r2 / put parent process number in r2
        shl     bx, 1
                ;asl r2 / multiply by 2 to get index into below tables
        ;movzx eax, word [ebx+p.pid-2]
        mov     ax, [ebx+p.pid-2]
                ; mov p.pid-2(r2),r2 / get process name of parent
                                 ; / process
        mov     [esi+p.ppid-2], ax
                ; mov r2,p.ppid-2(r1) / put parent process name
                         ; / in parent process slot for child
        mov     [u.r0], eax
                ; mov r2,*u.r0 / put parent process name on stack
                             ; / at location where r0 was saved
        mov     ebp, [u.sp] ; points to return address (EIP for IRET)
        mov     [ebp], edx ; *, CS:EIP -> EIP
                          ; * return address for the child process
                ; mov $sysret1,-(sp) /
                ; mov sp,u.usp / contents of sp at the time when
                              ; / user is swapped out
                ; mov $sstack,sp / point sp to swapping stack space
        ; 04/09/2015 - 01/09/2015
        ; [u.usp] = esp
        push    sysret ; ***
```

```
        mov     [u.usp], esp ; points to 'sysret' address (***)
                             ; (for child process)
        xor     eax, eax
        mov     [u.ttyp], ax ; 0
        call    wswap ; Retro UNIX 8086 v1 modification !
                ;jsr r0,wswap / put child process out on drum
                ;jsr r0,unpack / unpack user stack
                ;mov u.usp,sp / restore user stack pointer
                ; tst (sp)+ / bump stack pointer
        ; Retro UNIX 386 v1 modification !
        pop     eax ; ***
        shl     bx, 1
        mov      eax, [ebx+p.upage-4] ; UPAGE address ; 14/05/2015
        call    rswap ; restore parent process 'u' structure,
                      ; registers and return address (for IRET)
                ;movb (sp)+,u.uno / put parent process number in u.uno
         movzx   eax, word [mpid]
        mov     [u.r0], eax
                ; mov mpid,*u.r0 / put child process name on stack
                              ; / where r0 was saved
                ; add $2,18.(sp) / add 2 to pc on stack; gives parent
                                ; / process return
        ;xor    ebx, ebx
        xor      esi, esi
                ;clr r1
sysfork_4: ; 1: / search u.fp list to find the files
              ; / opened by the parent process
        ; 01/09/2015
        ;xor    bh, bh
        ;mov    bl, [esi+u.fp]
        mov     al, [esi+u.fp]
                ; movb u.fp(r1),r2 / get an open file for this process
         ;or      bl, bl
        or      al, al
        jz      short sysfork_5
                ; beq 2f / file has not been opened by parent,
                        ; / so branch
        mov     ah, 10 ; Retro UNIX 386 v1 fsp structure size = 10 bytes
        mul     ah
        ;movzx ebx, ax
        mov     bx, ax
        ;shl      bx, 3
                ; asl r2 / multiply by 8
                ; asl r2 / to get index into fsp table
                ; asl r2
        inc      byte [ebx+fsp-2]
                ; incb fsp-2(r2) / increment number of processes
                             ; / using file, because child will now be
                             ; / using this file
sysfork_5: ; 2:
         inc      esi
                ; inc r1 / get next open file
         cmp      si, 10
                ; cmp r1,$10. / 10. files is the maximum number which
                          ; / can be opened
        jb      short sysfork_4
                ; blt 1b / check next entry
        jmp     sysret
                ; br sysret1

sysread: ; < read from file >
        ; 13/05/2015
        ; 11/05/2015 (Retro UNIX 386 v1 - Beginning)
        ; 23/05/2013 (Retro UNIX 8086 v1)
        ;
        ; 'sysread' is given a buffer to read into and the number of
        ; characters to be read. If finds the file from the file
        ; descriptor located in *u.r0 (r0). This file descriptor
        ; is returned from a successful open call (sysopen).
        ; The i-number of file is obtained via 'rw1' and the data
        ; is read into core via 'readi'.
        ;
        ; Calling sequence:
        ;       sysread; buffer; nchars
        ; Arguments:
        ;       buffer - location of contiguous bytes where
        ;               input will be placed.
        ;       nchars - number of bytes or characters to be read.
        ; Inputs: *u.r0 - file descriptor (& arguments)
```

```
        ; Outputs: *u.r0 - number of bytes read.
        ; ...............................................................
        ;
        ; Retro UNIX 8086 v1 modification:
        ;        'sysread' system call has three arguments; so,
        ;       * 1st argument, file descriptor is in BX register
        ;       * 2nd argument, buffer address/offset in CX register
        ;       * 3rd argument, number of bytes is in DX register
        ;
        ;       AX register (will be restored via 'u.r0') will return
        ;       to the user with number of bytes read.
        ;
        call    rw1
        jc      error ; 13/05/2015, ax < 1
                ; jsr r0,rw1 / get i-number of file to be read into r1
        test    ah, 80h
                ; tst r1 / negative i-number?
        jnz     error
                ; ble error1 / yes, error 1 to read
                          ; / it should be positive
        call    readi
                ; jsr r0,readi / read data into core
        jmp     short rw0
                ; br 1f
syswrite: ; < write to file >
        ; 13/05/2015
        ; 11/05/2015 (Retro UNIX 386 v1 - Beginning)
        ; 23/05/2013 (Retro UNIX 8086 v1)
        ;
        ; 'syswrite' is given a buffer to write onto an output file
        ; and the number of characters to write. If finds the file
        ; from the file descriptor located in *u.r0 (r0). This file
        ; descriptor is returned from a successful open or create call
        ; (sysopen or syscreat). The i-number of file is obtained via
        ; 'rw1' and buffer is written on the output file via 'write'.
        ;
        ; Calling sequence:
        ;       syswrite; buffer; nchars
        ; Arguments:
        ;       buffer - location of contiguous bytes to be writtten.
        ;       nchars - number of characters to be written.
        ; Inputs: *u.r0 - file descriptor (& arguments)
        ; Outputs: *u.r0 - number of bytes written.
        ; ...............................................................
        ;
        ; Retro UNIX 8086 v1 modification:
        ;        'syswrite' system call has three arguments; so,
        ;       * 1st argument, file descriptor is in BX register
        ;       * 2nd argument, buffer address/offset in CX register
        ;       * 3rd argument, number of bytes is in DX register
        ;
        ;       AX register (will be restored via 'u.r0') will return
        ;       to the user with number of bytes written.
        ;
        call    rw1
        jc      error ; 13/05/2015, ax < 1
                ; jsr r0,rw1 / get i-number in r1 of file to write
         test   ah, 80h
                ; tst r1 / positive i-number ?
         jz     short rw3 ; 13/05/2015
        ;jz     error
                ; bge error1 / yes, error 1
                          ; / negative i-number means write
         neg    ax
                ; neg r1 / make it positive
        call    writei
                ; jsr r0,writei / write data
rw0: ; 1:
         mov    eax, [u.nread]
        mov     [u.r0], eax
                ; mov u.nread,*u.r0 / put no. of bytes transferred
                                ; / into (u.r0)
        jmp     sysret
                ; br sysret1
```

```
rw1:
        ; 14/05/2015
        ; 13/05/2015
        ; 11/05/2015 (Retro UNIX 386 v1 - Beginning)
        ; 23/05/2013 - 24/05/2013 (Retro UNIX 8086 v1)
        ; System call registers: bx, cx, dx (through 'sysenter')
        ;
        ;mov    [u.base], ecx  ; buffer address/offset
                               ;(in the user's virtual memory space)
        ;mov    [u.count], edx
                ; jsr r0,arg; u.base / get buffer pointer
                ; jsr r0,arg; u.count / get no. of characters
        ;;mov   eax, ebx ; file descriptor
                ; mov *u.r0,r1 / put file descriptor
                             ; / (index to u.fp table) in r1
        ; 13/05/2015
        mov     dword [u.r0], 0 ; r/w transfer count = 0 (reset)
        ;
        ;; call getf
         ; eBX = File descriptor
        call    getf1 ; calling point in 'getf' from 'rw1'
                ; jsr r0,getf / get i-number of the file in r1
        ; AX = I-number of the file ; negative i-number means write
        ; 13/05/2015
        cmp     ax, 1
        jb      short rw2
        ;
        mov     [u.base], ecx  ; buffer address/offset
                               ;(in the user's virtual memory space)
        mov     [u.count], edx
        ; 14/05/2015
         mov     dword [u.error], 0 ; reset the last error code
        retn
                ; rts r0
rw2:
        ; 13/05/2015
        mov     dword [u.error], ERR_FILE_NOT_OPEN ; file not open !
        retn
rw3:
        ; 13/05/2015
        mov     dword [u.error], ERR_FILE_ACCESS ; permission denied !
        stc
        retn

sysopen: ;<open file>
        ; 14/05/2015 (Retro UNIX 386 v1 - Beginning)
        ; 22/05/2013 - 27/05/2013 (Retro UNIX 8086 v1)
        ;
        ; 'sysopen' opens a file in following manner:
        ;    1) The second argument in a sysopen says whether to
        ;       open the file ro read (0) or write (>0).
        ;    2) I-node of the particular file is obtained via 'namei'.
        ;    3) The file is opened by 'iopen'.
        ;    4) Next housekeeping is performed on the fsp table
        ;       and the user's open file list - u.fp.
        ;       a) u.fp and fsp are scanned for the next available slot.
        ;       b) An entry for the file is created in the fsp table.
        ;       c) The number of this entry is put on u.fp list.
        ;       d) The file descriptor index to u.fp list is pointed
        ;          to by u.r0.
        ;
        ; Calling sequence:
        ;       sysopen; name; mode
        ; Arguments:
        ;       name - file name or path name
        ;       mode - 0 to open for reading
        ;              1 to open for writing
        ; Inputs: (arguments)
        ; Outputs: *u.r0 - index to u.fp list (the file descriptor)
        ;                is put into r0's location on the stack.
        ; ...............................................................
        ;
        ; Retro UNIX 8086 v1 modification:
        ;        'sysopen' system call has two arguments; so,
        ;       * 1st argument, name is pointed to by BX register
        ;       * 2nd argument, mode is in CX register
        ;
        ;       AX register (will be restored via 'u.r0') will return
        ;       to the user with the file descriptor/number
```

```
        ;       (index to u.fp list).
        ;
        ;call   arg2
        ; * name - 'u.namep' points to address of file/path name
        ;          in the user's program segment ('u.segmnt')
        ;          with offset in BX register (as sysopen argument 1).
        ; * mode - sysopen argument 2 is in CX register
        ;          which is on top of stack.
        ;
        ; jsr r0,arg2 / get sys args into u.namep and on stack
        ;
        ; system call registers: ebx, ecx (through 'sysenter')

        mov     [u.namep], ebx
        push    cx
        call    namei
                ; jsr r0,namei / i-number of file in r1
        ;and    ax, ax
        ;jz     error ; File not found
        jc      short fnotfound ; 14/05/2015
        ;jc     error ; 27/05/2013
                ; br  error2 / file not found
        pop     dx ; mode
        push    dx
        ;or     dx, dx
        or      dl, dl
                ; tst (sp) / is mode = 0 (2nd arg of call;
                         ; / 0 means, open for read)
        jz      short sysopen_0
                ; beq 1f / yes, leave i-number positive
        neg     ax
                ; neg r1 / open for writing so make i-number negative
sysopen_0: ;1:
        call    iopen
                ;jsr r0,iopen / open file whose i-number is in r1
        pop     dx
        ;and    dx, dx
        and     dl, dl
                ; tst (sp)+ / pop the stack and test the mode
        jz      short sysopen_2
                ; beq op1 / is open for read op1
sysopen_1: ;op0:
        neg     ax
                ; neg r1
                     ;/ make i-number positive if open for writing [???]
        ;; NOTE: iopen always make i-number positive.
        ;; Here i-number becomes negative again. [22/05/2013]
sysopen_2: ;op1:
         xor     esi, esi
                ; clr r2 / clear registers
         xor     ebx, ebx
                ; clr r3
sysopen_3: ;1: / scan the list of entries in fsp table
         cmp     [esi+u.fp], bl ; 0
                ; tstb u.fp(r2) / test the entry in the u.fp list
         jna      short sysopen_4
                ; beq 1f / if byte in list is 0 branch
         inc     esi
                ; inc r2 / bump r2 so next byte can be checked
         cmp     si, 10
                ; cmp r2,$10. / reached end of list?
        jb      short sysopen_3
                ; blt 1b / no, go back
toomanyf:
        ; 14/05/2015
        mov     dword [u.error], ERR_TOO_MANY_FILES ; too many open files !
        jmp     error
                ; br error2 / yes, error (no files open)
fnotfound:
        ; 14/05/2015
        mov     dword [u.error], ERR_FILE_NOT_FOUND ; file not found !
        jmp     error

sysopen_4: ; 1:
         cmp     word [ebx+fsp], 0
                ; tst fsp(r3) / scan fsp entries
         jna     short sysopen_5
                ; beq 1f / if 0 branch
```

```
        ; 14/05/2015 - Retro UNIX 386 v1 modification !
         add     bx, 10 ; fsp structure size = 10 bytes/entry
                ; add $8.,r3 / add 8 to r3
                        ; / to bump it to next entry mfsp table
         cmp     bx, nfiles*10
                ; cmp r3,$[nfiles*8.] / done scanning
        jb      short sysopen_4
                ; blt 1b / no, back
        jmp     error
                ; br error2 / yes, error
sysopen_5: ; 1: / r2 has index to u.fp list; r3, has index to fsp table
         mov     [ebx+fsp], ax
                ; mov r1,fsp(r3) / put i-number of open file
                        ; / into next available entry in fsp table,
        mov     di, [cdev] ; word ? byte ?
         mov     [ebx+fsp+2], di ; device number
                ; mov cdev,fsp+2(r3) / put # of device in next word
         xor    edi, edi
         mov     [ebx+fsp+4], edi ; offset pointer (0)
                ; clr fsp+4(r3)
         mov     [ebx+fsp+8], di ; open count (0), deleted flag (0)
                ; clr fsp+6(r3) / clear the next two words
        mov     eax, ebx
        mov     bl, 10
        div     bl
                ; asr r3
                ; asr r3 / divide by 8
                ; asr r3 ; / to get number of the fsp entry-1
        inc     al
                ; inc r3 / add 1 to get fsp entry number
         mov     [esi+u.fp], al
                ; movb r3,u.fp(r2) / move entry number into
                        ; / next available slot in u.fp list
         mov     [u.r0], esi
                ; mov r2,*u.r0 / move index to u.fp list
                             ; / into r0 loc on stack
         jmp    sysret
                ; br sysret2

        ; 'fsp' table (10 bytes/entry)
        ; bit 15                                bit 0
        ; ---|-------------------------------------------
        ; r/w|         i-number of open file
        ; ---|-------------------------------------------
        ;                device number
        ; -----------------------------------------------
        ; offset pointer, r/w pointer to file (bit 0-15)
        ; -----------------------------------------------
        ; offset pointer, r/w pointer to file (bit 16-31)
        ; ----------------------|------------------------
        ;  flag that says file       | number of processes
        ;   has been deleted   | that have file open
        ; ----------------------|------------------------

syscreat: ; < create file >
        ; 14/05/2015 (Retro UNIX 386 v1 - Beginning)
        ; 27/05/2013 (Retro UNIX 8086 v1)
        ;
        ; 'syscreat' called with two arguments; name and mode.
        ; u.namep points to name of the file and mode is put
        ; on the stack. 'namei' is called to get i-number of the file.
        ; If the file aready exists, it's mode and owner remain
        ; unchanged, but it is truncated to zero length. If the file
        ; did not exist, an i-node is created with the new mode via
        ; 'maknod' whether or not the file already existed, it is
        ; open for writing. The fsp table is then searched for a free
        ; entry. When a free entry is found, proper data is placed
        ; in it and the number of this entry is put in the u.fp list.
        ; The index to the u.fp (also know as the file descriptor)
        ; is put in the user's r0.
        ;
        ; Calling sequence:
        ;       syscreate; name; mode
        ; Arguments:
        ;       name - name of the file to be created
        ;       mode - mode of the file to be created
        ; Inputs: (arguments)
        ; Outputs: *u.r0 - index to u.fp list
        ;                (the file descriptor of new file)
```

```
        ; ..............................................................
        ;
        ; Retro UNIX 8086 v1 modification:
        ;       'syscreate' system call has two arguments; so,
        ;       * 1st argument, name is pointed to by BX register
        ;       * 2nd argument, mode is in CX register
        ;
        ;       AX register (will be restored via 'u.r0') will return
        ;       to the user with the file descriptor/number
        ;       (index to u.fp list).
        ;
        ;call   arg2
        ; * name - 'u.namep' points to address of file/path name
        ;          in the user's program segment ('u.segmnt')
        ;          with offset in BX register (as sysopen argument 1).
        ; * mode - sysopen argument 2 is in CX register
        ;          which is on top of stack.
        ;
                ; jsr r0,arg2 / put file name in u.namep put mode
                           ; / on stack
        mov     [u.namep], ebx ; file name address
        push    cx ; mode
        call    namei
                ; jsr r0,namei / get the i-number
         ;and   ax, ax
        ;jz     short syscreat_1
        jc      short syscreat_1
                ; br  2f / if file doesn't exist 2f
        neg     ax
                ; neg r1 / if file already exists make i-number
                       ; / negative (open for writing)
        call    iopen
                ; jsr r0,iopen /
        call    itrunc
                ; jsr r0,itrunc / truncate to 0 length
        pop     cx ; pop mode (did not exist in original Unix v1 !?)
         jmp     sysopen_1
                ; br op0
syscreat_1: ; 2: / file doesn't exist
        pop     ax
                ; mov (sp)+,r1 / put the mode in r1
        xor     ah, ah
                ; bic $!377,r1 / clear upper byte
        call    maknod
                ; jsr r0,maknod / make an i-node for this file
        mov     ax, [u.dirbuf]
                ; mov u.dirbuf,r1 / put i-number
                               ; / for this new file in r1
         jmp     sysopen_1
                ; br op0 / open the file

sysmkdir: ; < make directory >
        ; 14/05/2015 (Retro UNIX 386 v1 - Beginning)
        ; 27/05/2013 - 02/08/2013 (Retro UNIX 8086 v1)
        ;
        ; 'sysmkdir' creates an empty directory whose name is
        ; pointed to by arg 1. The mode of the directory is arg 2.
        ; The special entries '.' and '..' are not present.
        ; Errors are indicated if the directory already exists or
        ; user is not the super user.
        ;
        ; Calling sequence:
        ;       sysmkdir; name; mode
        ; Arguments:
        ;       name - points to the name of the directory
        ;       mode - mode of the directory
        ; Inputs: (arguments)
        ; Outputs: -
        ;    (sets 'directory' flag to 1;
        ;    'set user id on execution' and 'executable' flags to 0)
        ; ..............................................................
        ;
        ; Retro UNIX 8086 v1 modification:
        ;       'sysmkdir' system call has two arguments; so,
        ;       * 1st argument, name is pointed to by BX register
        ;       * 2nd argument, mode is in CX register
        ;
```

```
; / make a directory

        ;call   arg2
        ; * name - 'u.namep' points to address of file/path name
        ;          in the user's program segment ('u.segmnt')
        ;          with offset in BX register (as sysopen argument 1).
        ; * mode - sysopen argument 2 is in CX register
        ;          which is on top of stack.

               ; jsr r0,arg2 / put file name in u.namep put mode
                           ; / on stack
        mov    [u.namep], ebx
        push   cx ; mode
        call   namei
               ; jsr r0,namei / get the i-number
               ;     br .+4 / if file not found branch around error
         ;xor  ax, ax
        ;jnz   error
        jnc    short dir_exists ; 14/05/2015
        ;jnc   error
               ; br  error2 / directory already exists (error)
        cmp    byte [u.uid], 0 ; 02/08/2013
               ;tstb u.uid / is user the super user
        jna    short dir_access_err ; 14/05/2015
        ;jna   error
               ;bne error2 / no, not allowed
        pop    ax
               ;mov (sp)+,r1 / put the mode in r1
        and    ax, 0FFCFh ; 1111111111001111b
               ;bic $!317,r1 / all but su and ex
        ;or    ax , 4000h ; 1011111111111111b
        or     ah, 40h ; Set bit 14 to 1
               ;bis $40000,r1 / directory flag
        call   maknod
               ;jsr r0,maknod / make the i-node for the directory
        jmp    sysret
               ;br sysret2 /
dir_exists:
        ; 14/05/2015
        mov    dword [u.error], ERR_DIR_EXISTS ; dir. already exists !
        jmp    error
dir_access_err:
        ; 14/05/2015
        mov    dword [u.error], ERR_DIR_ACCESS ; permission denied !
        jmp    error

sysclose: ;<close file>
        ; 14/05/2015 (Retro UNIX 386 v1 - Beginning)
        ; 22/05/2013 - 26/05/2013 (Retro UNIX 8086 v1)
        ;
        ; 'sysclose', given a file descriptor in 'u.r0', closes the
        ; associated file. The file descriptor (index to 'u.fp' list)
        ; is put in r1 and 'fclose' is called.
        ;
        ; Calling sequence:
        ;      sysclose
        ; Arguments:
        ;      -
        ; Inputs: *u.r0 - file descriptor
        ; Outputs: -
        ; ...............................................................
        ;
        ; Retro UNIX 8086 v1 modification:
        ;       The user/application program puts file descriptor
        ;        in BX register as 'sysclose' system call argument.
        ;       (argument transfer method 1)

        ; / close the file

        mov    eax, ebx
        call   fclose
               ; mov *u.r0,r1 / move index to u.fp list into r1
               ; jsr r0,fclose / close the file
                       ; br error2 / unknown file descriptor
               ; br sysret2
        ; 14/05/2015
        jnc    sysret
        mov    dword [u.error], ERR_FILE_NOT_OPEN ; file not open !
        jmp    error
```

```
sysemt:
        ; 14/05/2015 (Retro UNIX 386 v1 - Beginning)
        ; 10/12/2013 - 20/04/2014 (Retro UNIX 8086 v1)
        ;
        ; Retro UNIX 8086 v1 modification:
        ;       'Enable Multi Tasking'  system call instead
        ;       of 'Emulator Trap' in original UNIX v1 for PDP-11.
        ;
        ; Retro UNIX 8086 v1 feature only!
        ;       Using purpose: Kernel will start without time-out
        ;       (internal clock/timer) functionality.
        ;       Then etc/init will enable clock/timer for
        ;       multi tasking. (Then it will not be disabled again
        ;       except hardware reset/restart.)
        ;

        cmp     byte [u.uid], 0 ; root ?
        ;ja     error
        ja      badsys ; 14/05/2015
emt_0:
        cli
        and     ebx, ebx
        jz      short emt_2
        ; Enable multi tasking -time sharing-
        mov     eax, clock
emt_1:
        mov     [x_timer], eax
        sti
        jmp     sysret
emt_2:
        ; Disable multi tasking -time sharing-
        mov     eax, u_timer
        jmp     short emt_1

        ; Original UNIX v1 'sysemt' routine
;sysemt:
         ;
        ;jsr     r0,arg; 30 / put the argument of the sysemt call
                        ; / in loc 30
         ;cmp    30,$core / was the argument a lower address
                       ; / than core
         ;blo    1f / yes, rtssym
         ;cmp    30,$ecore / no, was it higher than "core"
                       ; / and less than "ecore"
         ;blo    2f / yes, sysret2
;1:
         ;mov    $rtssym,30
;2:
         ;br     sysret2

sysilgins:
        ; 14/05/2015 (Retro UNIX 386 v1 - Beginning)
        ; 03/06/2013
        ; Retro UNIX 8086 v1 modification:
        ;       not a valid system call ! (not in use)
        ;
        jmp     badsys
        ;jmp    error
        ;;jmp   sysret

        ; Original UNIX v1 'sysemt' routine
;sysilgins: / calculate proper illegal instruction trap address
         ;jsr    r0,arg; 10 / take address from sysilgins call
                         ;/ put it in loc 8.,
         ;cmp    10,$core / making it the illegal instruction
                       ; / trap address
         ;blo    1f / is the address a user core address?
                ; / yes, go to 2f
         ;cmp    10,$ecore
         ;blo    2f
;1:
         ;mov    $fpsym,10 / no, make 'fpsum' the illegal
                    ; / instruction trap address for the system
;2:
         ;br     sysret2 / return to the caller via 'sysret'
```

```
sysmdate: ; < change the modification time of a file >
        ; 16/05/2015 (Retro UNIX 386 v1 - Beginning)
        ; 03/06/2013 - 02/08/2013 (Retro UNIX 8086 v1)
        ;
        ; 'sysmdate' is given a file name. It gets inode of this
        ; file into core. The user is checked if he is the owner
        ; or super user. If he is neither an error occurs.
        ; 'setimod' is then called to set the i-node modification
        ; byte and the modification time, but the modification time
        ; is overwritten by whatever get put on the stack during
        ; a 'systime' system call. This calls are restricted to
        ; the super user.
        ;
        ; Calling sequence:
        ;       sysmdate; name
        ; Arguments:
        ;       name - points to the name of file
        ; Inputs: (arguments)
        ; Outputs: -
        ; ...............................................................
        ; Retro UNIX 8086 v1 modification:
        ;        The user/application program puts address
        ;        of the file name in BX register
        ;        as 'sysmdate' system call argument.
        ;
; / change the modification time of a file
                ; jsr r0,arg; u.namep / point u.namep to the file name
        mov     [u.namep], ebx
        call    namei
                ; jsr r0,namei / get its i-number
        jc      fnotfound ; file not found !
        ;jc     error
                ; br error2 / no, such file
        call    iget
                ; jsr r0,iget / get i-node into core
        mov     al, [u.uid]
        cmp     al, [i.uid]
                ; cmpb u.uid,i.uid / is user same as owner
        je      short mdate_1
                ; beq 1f / yes
        and     al, al
                ; tstb u.uid / no, is user the super user
        ;jnz    error
                ; bne error2 / no, error
        jz      short mdate_1
        mov     dword [u.error], ERR_FILE_ACCESS ; permission denied !
        jmp     error
mdate_1: ;1:
        call    setimod
                ; jsr r0,setimod / fill in modification data,
                               ; / time etc.
        mov     esi, p_time
        mov     edi, i.mtim
        movsd
                ; mov 4(sp),i.mtim / move present time to
                ; mov 2(sp),i.mtim+2 / modification time
        jmp     sysret
                ; br sysret2

sysstty: ; < set tty status and mode >
        ; 17/11/2015
        ; 12/11/2015
        ; 29/10/2015
        ; 17/10/2015
        ; 13/10/2015
        ; 29/06/2015
        ; 27/06/2015 (Retro UNIX 386 v1 - Beginning)
        ; 02/06/2013 - 12/07/2014 (Retro UNIX 8086 v1)
        ;
        ; 'sysstty' sets the status and mode of the typewriter
        ; whose file descriptor is in (u.r0).
        ;
        ; Calling sequence:
        ;       sysstty; arg
        ; Arguments:
        ;       arg - address of 3 consequitive words that contain
        ;             the source of status data
        ; Inputs: ((*u.r0 - file descriptor & argument))
        ; Outputs: ((status in address which is pointed to by arg))
```

```
        ; ...............................................................
        ; Retro UNIX 8086 v1 modification:
        ;       'sysstty' system call will set the tty
        ;       (clear keyboard buffer and set cursor position)
        ;        in following manner:
        ;   NOTE: All of tty setting functions are here (16/01/2014)
        ;
        ; Inputs:
        ;       BX = 0 --> means
        ;          If CL = FFh
        ;             set cursor position for console tty, only
        ;             CH will be ignored (char. will not be written)
        ;          If CH = 0 (CL < FFh)
        ;             set console tty for (current) process
        ;             CL = tty number (0 to 9)
        ;             (If CH = 0, character will not be written)
        ;           If CH > 0 (CL < FFh)
        ;              CL = tty number (0 to 9)
        ;             CH = character will be written
        ;               at requested cursor position (in DX)
        ;          DX = cursor position for tty number 0 to 7.
        ;               (only tty number 0 to 7)
        ;           DL = communication parameters (for serial ports)
        ;                (only for COM1 and COM2 serial ports)
        ;          DH < 0FFh -> DL is valid, initialize serial port
        ;                       or set cursor position
        ;          DH = 0FFh -> DL is not valid
        ;               do not set serial port parameters
        ;               or do not set cursor position
        ;
        ;       BX > 0 --> points to name of tty
        ;          CH > 0 -->
        ;               CH = character will be written in current
        ;               cursor position (for tty number from 0 to 7)
        ;               or character will be sent to serial port
        ;               (for tty number 8 or 9)
        ;               CL = color of the character if tty number < 8.
        ;          CH = 0 --> Do not write a character,
        ;               set mode (tty 8 to 9) or
        ;               set current cursor positions (tty 0 to 7) only.
        ;          DX = cursor position for tty number 0 to 7.
        ;          DH = FFh --> Do not set cursor pos (or comm. params.)
        ;               (DL is not valid)
        ;          DL = communication parameters
        ;               for tty number 8 or 9 (COM1 or COM2).
        ; Outputs:
        ;       cf = 0 -> OK
        ;            AL = tty number (0 to 9)
        ;            AH = line status if tty number is 8 or 9
        ;            AH = process number (of the caller)
        ;       cf = 1 means error (requested tty is not ready)
        ;            AH = FFh if the tty is locked
        ;                 (owned by another process)
        ;               = process number (of the caller)
        ;                 (if < FFh and tty number < 8)
        ;            AL = tty number (0FFh if it does not exist)
        ;            AH = line status if tty number is 8 or 9
        ;       NOTE: Video page will be cleared if cf = 0.
        ;
        ; 27/06/2015 (32 bit modifications)
        ; 14/01/2014
        xor     eax, eax
        dec     ax ; 17/10/2015
        mov     [u.r0], eax ; 0FFFFh
        and     ebx, ebx
         jnz     sysstty_6
; set console tty
        ; 29/10/2015
        ; 17/01/2014
        cmp     cl, 9
        jna     short sysstty_0
        ; 17/11/2015
        cmp     cl, 0FFh
        jb      short sysstty_13
        mov     ch, cl ; force CH value to FFh
sysstty_13:
        mov     bl, [u.uno] ; process number
        mov     cl, [ebx+p.ttyc-1] ; current/console tty
```

```
sysstty_0:
        ; 29/06/2015
        push    dx
        push    cx
        xor     dl, dl ; sysstty call sign
        mov     al, cl
        mov     [u.r0], al ; tyy number (0 to 9)
        call    ottyp
        pop     cx
        pop     dx
        ;
        jc      short sysstty_pd_err
        ;
        cmp     cl, 8
        jb      short sysstty_2
        ;
        cmp     dh, 0FFh
        je      short sysstty_2
                ; set communication parameters for serial ports
        ; 29/10/2015
        mov     ah, dl ; communication parameters
                ; ah = 0E3h = 11100011b = 115200 baud,
                ;                       THRE int + RDA int
                ; ah = 23h = 00100011b = 9600 baud,
                ;                       THRE int + RDA int
        sub     al, al ; 0
        ; 12/07/2014
        cmp     cl, 9
        jb      short sysstty_1
        inc     al
sysstty_1:
        push    cx
        ; 29/06/2015
        call    sp_setp ; Set serial port communication parameters
        mov     [u.r0+1], cx ; Line status (ah)
                             ; Modem status (EAX bits 16 to 23)
        pop     cx
         jc       short sysstty_tmout_err ; 29/10/2015
sysstty_2:
        ; 17/01/2014
        and     ch, ch  ; set cursor position
                       ; or comm. parameters ONLY
        jnz     short sysstty_3
        movzx   ebx, byte [u.uno] ; process number
        mov     [ebx+p.ttyc-1], cl ; console tty
sysstty_3:
        ; 16/01/2014
        mov     al, ch ; character  ; 0 to FFh
        ; 17/11/2015
        mov     ch, 7  ; Default color (light gray)
        cmp     cl, ch ; 7 (tty number)
         jna      sysstty_9
sysstty_12:
        ;; BX = 0, CL = 8 or CL = 9
        ; (Set specified serial port as console tty port)
        ; CH = character to be written
        ; 15/04/2014
        ; CH = 0 --> initialization only
        ; AL = character
        ; 26/06/2014
        mov     [u.ttyn], cl
        ; 12/07/2014
        mov     ah, cl ; tty number (8 or 9)
        and     al, al
        jz      short sysstty_4 ; al = ch = 0
        ; 04/07/2014
        call    sndc
        ; 12/07/2014
        jmp     short sysstty_5
sysstty_pd_err: ; 29/06/2015
        ; 'permission denied !' error
        mov     dword [u.error], ERR_NOT_OWNER
        jmp     error
sysstty_4:
        ; 12/07/2014
        ;xchg   ah, al ; al = 0 -> al = ah, ah = 0
        mov     al, ah ; 29/06/2015
        sub     al, 8
```

```
        ; 27/06/2015
        call    sp_status ; get serial port status
        ; AL = Line status, AH = Modem status
        ; 12/11/2015
        cmp     al, 80h
        cmc
sysstty_5:
        mov     [u.r0+1], ax ; ah = line status
                     ; EAX bits 16-23 = modem status
        pushf
        xor     dl, dl ; sysstty call sign
        mov     al, [u.ttyn] ; 26/06/2014
        call    cttyp
        popf
        jnc     sysret  ; time out error

sysstty_tmout_err:
        mov     dword [u.error], ERR_TIME_OUT
        jmp     error
sysstty_6:
        push    dx
        push    cx
        mov     [u.namep], ebx
        call    namei
        pop     cx
        pop     dx
        jc      short sysstty_inv_dn
        ;
        cmp     ax, 19  ; inode number of /dev/COM2
        ja      short sysstty_inv_dn ; 27/06/2015
        ;
        cmp     al, 10 ; /dev/tty0 .. /dev/tty7
                       ; /dev/COM1, /dev/COM2
        jb      short sysstty_7
        sub     al, 10
        jmp     short sysstty_8
sysstty_inv_dn:
        ; 27/06/2015
        ; Invalid device name (not a tty) ! error
        ; (Device is not a tty or device name not found)
        mov     dword [u.error], ERR_INV_DEV_NAME
        jmp     error
sysstty_7:
        cmp     al, 1 ; /dev/tty
        jne     short sysstty_inv_dn ; 27/06/2015
        movzx   ebx, byte [u.uno] ; process number
        mov     al, [ebx+p.ttyc-1] ; console tty
sysstty_8:
        mov     [u.r0], al
        push    dx
        push    ax
        push    cx
        call    ottyp
        pop     cx
        pop     ax
        pop     dx
        jc      sysstty_pd_err ; 'permission denied !'
        ; 29/10/2015
        xchg    ch, cl
                ; cl = character, ch = color code
        xchg    al, cl
                ; al = character, cl = tty number
        cmp     cl, 7
        ja      sysstty_12
        ;
        ; 16/01/2014
        xor     bh, bh
        ;
sysstty_9:      ; tty 0 to tty 7
        ; al = character
        cmp     dh, 0FFh ; Do not set cursor position
        je      short sysstty_10
        push    cx
        push    ax
        ; movzx, ebx, cl
        mov     bl, cl ; (tty number = video page number)
        call    set_cpos
        pop     ax
        pop     cx
```

```
sysstty_10:
        ; 29/10/2015
        or      al, al ; character
        jz       short sysstty_11 ; al = 0
        ; 17/11/2015
        cmp     al, 0FFh
        jnb     short sysstty_11
                ; ch > 0 and ch < FFh
        ; write a character at current cursor position
        mov     ah, ch ; color/attribute
        ; 12/07/2014
        push    cx
        call    write_c_current
        pop     cx
sysstty_11:
        ; 14/01/2014
        xor     dl, dl ; sysstty call sign
        ; 18/01/2014
        ;movzx eax, cl ; 27/06/2015
        mov     al, cl
        call    cttyp
        jmp     sysret

; Original UNIX v1 'sysstty' routine:
; gtty:
;sysstty: / set mode of typewriter; 3 consequtive word arguments
        ;jsr    r0,gtty / r1 will have offset to tty block,
        ;              / r2 has source
        ;mov    r2,-(sp)
        ;mov    r1,-(sp) / put r1 and r2 on the stack
;1: / flush the clist wait till typewriter is quiescent
        ;mov    (sp),r1 / restore r1 to tty block offset
        ;movb   tty+3(r1),0f / put cc offset into getc argument
        ;mov    $240,*$ps / set processor priority to 5
        ;jsr    r0,getc; 0:../ put character from clist in r1
        ;       br .+4 / list empty, skip branch
        ;br     1b / get another character until list is empty
        ;mov    0b,r1 / move cc offset to r1
        ;inc    r1 / bump it for output clist
        ;tstb   cc(r1) / is it 0
        ;beq    1f / yes, no characters to output
        ;mov    r1,0f / no, put offset in sleep arg
        ;jsr    r0,sleep; 0:.. / put tty output process to sleep
        ;br     1b / try to calm it down again
;1:
        ;mov    (sp)+,r1
        ;mov    (sp)+,r2 / restore registers
        ;mov    (r2)+,r3 / put reader control status in r3
        ;beq    1f / if 0, 1f
        ;mov    r3,rcsr(r1) / move r.c. status to reader
        ;                   / control status register
;1:
        ;mov    (r2)+,r3 / move pointer control status to r3
        ;beq    1f / if 0 1f
        ;mov    r3,tcsr(r1) / move p.c. status to printer
        ;                 / control status reg
;1:
        ;mov    (r2)+,tty+4(r1) / move to flag byte of tty block
        ;jmp     sysret2 / return to user

sysgtty: ; < get tty status >
        ; 23/11/2015
        ; 29/10/2015
        ; 17/10/2015
        ; 28/06/2015 (Retro UNIX 386 v1 - Beginning)
        ; 30/05/2013 - 12/07/2014 (Retro UNIX 8086 v1)
        ;
        ; 'sysgtty' gets the status of tty in question.
        ; It stores in the three words addressed by it's argument
        ; the status of the typewriter whose file descriptor
        ; in (u.r0).
        ;
        ; Calling sequence:
        ;       sysgtty; arg
        ; Arguments:
        ;       arg - address of 3 words destination of the status
        ; Inputs: ((*u.r0 - file descriptor))
        ; Outputs: ((status in address which is pointed to by arg))
        ; ...............................................................
```

```
        ; Retro UNIX 8086 v1 modification:
        ;       'sysgtty' system call will return status of tty
        ;       (keyboard, serial port and video page status)
        ;        in following manner:
        ;
        ; Inputs:
        ;       BX = 0 --> means
        ;            CH = 0 -->       'return status of the console tty'
        ;                        for (current) process
        ;            CL = 0 --> return keyboard status (tty 0 to 9)
        ;            CL = 1 --> return video page status (tty 0 to 7)
        ;            CL = 1 --> return serial port status (tty 8 & 9)
        ;            CH > 0 -->       tty number + 1
        ;
        ;       BX > 0 --> points to name of tty
        ;            CL = 0 --> return keyboard status
        ;            CL = 1 --> return video page status
        ;            CH = undefined
        ;
        ; Outputs:
        ;       cf = 0 ->
        ;
        ;            AL = tty number from 0 to 9
        ;                 (0 to 7 is also the video page of the tty)
        ;            AH = 0 if the tty is free/unused
        ;            AH = the process number of the caller
        ;            AH = FFh if the tty is locked by another process
        ;
        ;         (if calling is for serial port status)
        ;            BX = serial port status if tty number is 8 or 9
        ;                 (BH = modem status, BL = Line status)
        ;            CX = 0FFFFh (if data is ready)
        ;            CX = 0 (if data is not ready or undefined)
        ;
        ;         (if calling is for keyboard status)
        ;            BX = current character in tty/keyboard buffer
        ;                 (BH = scan code, BL = ascii code)
        ;                 (BX=0 if there is not a waiting character)
        ;            CX  is undefined
        ;
        ;         (if calling is for video page status)
        ;            BX = cursor position on the video page
        ;                 if tty number < 8
        ;                 (BH = row, BL = column)
        ;            CX = current character (in cursor position)
        ;                 on the video page of the tty
        ;                 if tty number < 8
        ;                 (CH = color, CL = character)
        ;
        ;       cf = 1 means error (requested tty is not ready)
        ;
        ;            AH = FFh if the caller is not owner of
        ;                 specified tty or console tty
        ;            AL = tty number (0FFh if it does not exist)
        ;            BX, CX are undefined if cf = 1
        ;
        ;         (If tty number is 8 or 9)
        ;            AL = tty number
        ;            AH = the process number of the caller
        ;            BX = serial port status
        ;                (BH = modem status, BL = Line status)
        ;            CX = 0
        ;

gtty:    ; get (requested) tty number
        ; 17/10/2015
        ; 28/06/2015 (Retro UNIX 386 v1 - 32 bit modifications)
        ; 30/05/2013 - 12/07/2014
        ; Retro UNIX 8086 v1 modification !
        ;
        ; ((Modified regs: eAX, eBX, eCX, eDX, eSI, eDI, eBP))
        ;
        ; 28/06/2015 (32 bit modifications)
        ; 16/01/2014
        xor     eax, eax
        dec     ax ; 17/10/2015
        mov     [u.r0], eax ; 0FFFFh
        cmp     cl, 1
        jna     short sysgtty_0
```

```
sysgtty_invp:
        ; 28/06/2015
         mov     dword [u.error], ERR_INV_PARAMETER ; 'invalid parameter !'
        jmp     error
sysgtty_0:
        and     ebx, ebx
        jz      short sysgtty_1
        ;
        mov     [u.namep], ebx
        push    cx ; 23/11/2015
        call    namei
        pop     cx ; 23/11/2015
        jc      short sysgtty_inv_dn ; 28/06/2015
        ;
        cmp     ax, 1
        jna     short sysgtty_2
        sub     ax, 10
        cmp     ax, 9
        ;ja     short sysgtty_inv_dn
        ;mov    ch, al
        ;jmp    short sysgtty_4
        ; 23/11/2015
        jna     short sysgtty_4
sysgtty_inv_dn:
        ; 28/06/2015
        ; Invalid device name (not a tty) ! error
        ; (Device is not a tty or device name not found)
        mov     dword [u.error], ERR_INV_DEV_NAME
        jmp     error
sysgtty_1:
        ; 16/01/2014
        cmp     ch, 10
        ja      short sysgtty_invp ; 28/06/2015
        dec     ch ; 0 -> FFh (negative)
        jns     short sysgtty_3 ; not negative
        ;
sysgtty_2:
        ; get tty number of console tty
        mov     ah, [u.uno]
        ; 28/06/2015
        movzx   ebx, ah
        mov     ch, [ebx+p.ttyc-1]
sysgtty_3:
        mov     al, ch
sysgtty_4:
        mov     [u.r0], al
        ; 28/06/2015
        ;cmp    al, 9
        ;ja     short sysgtty_invp
        mov     ebp, [u.usp]
        ; 23/11/2015
        and     cl, cl
        jz      short sysgtty_6 ; keyboard status
        cmp     al, 8 ; cmp ch, 8
        jb      short sysgtty_6 ; video page status
        ; serial port status
        ; 12/07/2014
        ;mov    dx, 0
        ;je     short sysgtty_5
        ;inc    dl
;sysgtty_5:
        ; 28/06/2015
        sub     al, 8
        call    sp_status ; serial (COM) port (line) status
        ; AL = Line status, AH = Modem status
        mov     [ebp+16], ax ; serial port status (in EBX)
        mov     ah, [u.uno]
         mov     [u.r0+1], ah
        mov     word [ebp+24], 0 ; data status (0 = not ready)
                                  ; (in ECX)
        test    al, 80h
        jnz     short sysgtty_dnr_err ; 29/06/2015
        test    al, 1
        jz      sysret
        dec     word [ebp+24] ; data status (FFFFh = ready)
        jmp     sysret
```

```
sysgtty_6:
        mov     [u.ttyn], al ; tty number
        ;movzx  ebx, al
        mov     bl, al ; tty number (0 to 9)
        shl     bl, 1  ; aligned to word
        ; 22/04/2014 - 29/06/2015
         add     ebx, ttyl
        mov     ah, [ebx]
        cmp     ah, [u.uno]
        je      short sysgtty_7
        and     ah, ah
        ;jz     short sysgtty_7
        jnz     short sysgtty_8
        ;mov    ah, 0FFh
sysgtty_7:
         mov     [u.r0+1], ah
sysgtty_8:
        or      cl, cl
        jnz     short sysgtty_9
        mov     al, 1  ; test a key is available
        call    getc
        mov     [ebp+16], ax ; bx, character
        jmp     sysret
sysgtty_9:
        mov     bl, [u.ttyn]
        ; bl = video page number
        call    get_cpos
        ; dx = cursor position
        mov     [ebp+16], dx ; bx
        ;mov    bl, [u.ttyn]
        ; bl = video page number
        call    read_ac_current
        ; ax = character and attribute/color
        mov     [ebp+24], ax ; cx
        jmp     sysret
sysgtty_dnr_err:
        ; 'device not responding !' error
        ;mov    dword [u.error], ERR_TIME_OUT ; 25
        mov     dword [u.error], ERR_DEV_NOT_RESP ;  25
        jmp     error

; Original UNIX v1 'sysgtty' routine:
; sysgtty:
        ;jsr    r0,gtty / r1 will have offset to tty block,
        ;             / r2 has destination
        ;mov    rcsr(r1),(r2)+ / put reader control status
        ;                     / in 1st word of dest
        ;mov    tcsr(r1),(r2)+ / put printer control status
        ;                     / in 2nd word of dest
        ;mov    tty+4(r1),(r2)+ / put mode in 3rd word
        ;jmp    sysret2 / return to user

; Original UNIX v1 'gtty' routine:
; gtty:
        ;jsr    r0,arg; u.off / put first arg in u.off
        ;mov    *u.r0,r1 / put file descriptor in r1
        ;jsr    r0,getf / get the i-number of the file
        ;tst    r1 / is it open for reading
        ;bgt    1f / yes
        ;neg    r1 / no, i-number is negative,
        ;          / so make it positive
;1:
        ;sub    $14.,r1 / get i-number of tty0
        ;cmp    r1,$ntty-1 / is there such a typewriter
        ;bhis   error9 / no, error
        ;asl    r1 / 0%2
        ;asl    r1 / 0%4 / yes
        ;asl    r1 / 0%8 / multiply by 8 so r1 points to
        ;             ; / tty block
        ;mov    u.off,r2 / put argument in r2
        ;rts    r0 / return
```

```
; Retro UNIX 386 v1 Kernel (v0.2) - SYS2.INC
; Last Modification: 19/10/2015
; ----------------------------------------------------------------------------
; Derived from 'Retro UNIX 8086 v1' source code by Erdogan Tan
; (v0.1 - Beginning: 11/07/2012)
;
; Derived 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>
;
; Retro UNIX 8086 v1 - U2.ASM (24/03/2014) //// UNIX v1 -> u2.s
;
; ****************************************************************************

syslink:
	; 23/06/2015 (Retro UNIX 386 v1 - Beginning)
	; 19/06/2013 (Retro UNIX 8086 v1)
	;
	; 'syslink' is given two arguments, name 1 and name 2.
	; name 1 is a file that already exists. name 2 is the name
	; given to the entry that will go in the current directory.
	; name2 will then be a link to the name 1 file. The i-number
	; in the name 2 entry of current directory is the same
	; i-number for the name 1 file.
	;
	; Calling sequence:
	;	syslink; name 1; name 2
	; Arguments:
	;	name 1 - file name to which link will be created.
	;	name 2 - name of entry in current directory that
	;		links to name 1.
	; Inputs: -
	; Outputs: -
	; ...............................................................
	;
	; Retro UNIX 8086 v1 modification:
	;	 'syslink' system call has two arguments; so,
	;	* 1st argument, name 1 is pointed to by BX register
	;	* 2nd argument, name 2 is pointed to by CX register
	;
		; / name1, name2
		;jsr r0,arg2 / u.namep has 1st arg u.off has 2nd
	mov	[u.namep], ebx
	push	ecx
	call	namei
		; jsr r0,namei / find the i-number associated with
				; / the 1st path name
	;;and	ax, ax
	;;jz	error ; File not found
	;jc	error
		; br error9 / cannot be found
	jnc	short syslink0
	;pop	ecx
	; 'file not found !' error
	mov	dword [u.error], ERR_FILE_NOT_FOUND ; 12
	jmp	error
syslink0:
	call	iget
		; jsr r0,iget / get the i-node into core
	pop	dword [u.namep] ; ecx
		; mov (sp)+,u.namep / u.namep points to 2nd name
	push	ax
		; mov r1,-(sp) / put i-number of name1 on the stack
				; / (a link to this file is to be created)
	push	word [cdev]
		; mov cdev,-(sp) / put i-nodes device on the stack
	call	isdir
		; jsr r0,isdir / is it a directory
	call	namei
		; jsr r0,namei / no, get i-number of name2
	;jnc	error
		; br .+4   / not found
			; / so r1 = i-number of current directory
			; / ii = i-number of current directory
		; br error9 / file already exists., error
	jc	short syslink1
	; pop ax
	; pop ax
```

```
        ; 'file exists !' error
        mov     dword [u.error], ERR_FILE_EXISTS ; 14
        jmp     error
syslink1:
        pop     cx
        ;cmp    cx, [cdev]
        cmp     cl, [cdev]
        ;jne    error
                ; cmp (sp)+,cdev / u.dirp now points to
                               ; / end of current directory
                 ; bne error9
        je      short syslink2
        ; 'not same drive !' error
        mov     dword [u.error],  ERR_DRV_NOT_SAME ; 21
        jmp     error
syslink2:
        pop     ax
        push    ax
        mov     [u.dirbuf], ax
                ; mov (sp),u.dirbuf / i-number of name1 into u.dirbuf
        call    mkdir
                ; jsr r0,mkdir / make directory entry for name2
                             ; / in current directory
        pop     ax
                ; mov (sp)+,r1 / r1 has i-number of name1
        call    iget
                ; jsr r0,iget / get i-node into core
        inc     byte [i.nlks]
                ; incb i.nlks / add 1 to its number of links
        call    setimod
                ; jsr r0,setimod / set the i-node modified flag
        jmp     sysret

isdir:
        ; 22/06/2015 (Retro UNIX 386 v1 - Beginning)
        ; 04/05/2013 - 02/08/2013 (Retro UNIX 8086 v1)
        ;
        ; 'isdir' check to see if the i-node whose i-number is in r1
        ;  is a directory. If it is, an error occurs, because 'isdir'
        ;  called by syslink and sysunlink to make sure directories
        ;  are not linked. If the user is the super user (u.uid=0),
        ; 'isdir' does not bother checking. The current i-node
        ;  is not disturbed.
        ;
        ; INPUTS ->
        ;    r1 - contains the i-number whose i-node is being checked.
        ;    u.uid - user id
        ; OUTPUTS ->
        ;    r1 - contains current i-number upon exit
        ;        (current i-node back in core)
        ;
        ; ((AX = R1))
        ;
       ; ((Modified registers: eAX, eDX, eBX, eCX, eSI, eDI, eBP))

        ; / if the i-node whose i-number is in r1 is a directory
        ; / there is an error unless super user made the call

        cmp     byte [u.uid], 0
                ; tstb u.uid / super user
        jna     short isdir1
                ; beq 1f / yes, don't care
        push    word [ii]
                ; mov ii,-(sp) / put current i-number on stack
        call    iget
                ; jsr r0,iget / get i-node into core (i-number in r1)
        test    word [i.flgs], 4000h ; Bit 14 : Directory flag
                ; bit $40000,i.flgs / is it a directory
        ;jnz    error
                ; bne error9 / yes, error
        jz      short isdir0
        mov     dword [u.error], ERR_NOT_FILE  ; 11 ; ERR_DIR_ACCESS
                               ; 'permission denied !' error
        ; pop   ax
        jmp     error
isdir0:
        pop     ax
                ; mov (sp)+,r1 / no, put current i-number in r1 (ii)
        call    iget
```

```
		; jsr r0,iget / get it back in
isdir1: ; 1:
	retn
		; rts r0

sysunlink:
	; 23/06/2015 (Retro UNIX 386 v1 - Beginning)
	; 19/06/2013 (Retro UNIX 8086 v1)
	;
	; 'sysunlink' removes the entry for the file pointed to by
	; name from its directory. If this entry was the last link
	; to the file, the contents of the file are freed and the
	; file is destroyed. If, however, the file was open in any
	; process, the actual destruction is delayed until it is
	; closed, even though the directory entry has disappeared.
	;
	; The error bit (e-bit) is set to indicate that the file
	; does not exist or that its directory can not be written.
	; Write permission is not required on the file itself.
	; It is also illegal to unlink a directory (except for
	; the superuser).
	;
	; Calling sequence:
	;	sysunlink; name
	; Arguments:
	;	name - name of directory entry to be removed
	; Inputs: -
	; Outputs: -
	; ...............................................................
	;
	; Retro UNIX 8086 v1 modification:
	;	 The user/application program puts address of the name
	;	  in BX register as 'sysunlink' system call argument.

	; / name - remove link name
	mov	[u.namep], ebx
		;jsr r0,arg; u.namep / u.namep points to name
	call	namei
		; jsr r0,namei / find the i-number associated
			     ; / with the path name
	;jc	error
		; br error9 / not found
	jnc	short sysunlink1
	; 'file not found !' error
	mov	dword [u.error], ERR_FILE_NOT_FOUND ; 12
	jmp	error
sysunlink1:
	push	ax
		; mov r1,-(sp) / put its i-number on the stack
	call	isdir
		; jsr r0,isdir / is it a directory
	xor	ax, ax
	mov	[u.dirbuf], ax ; 0
		; clr u.dirbuf / no, clear the location that will
			   ; / get written into the i-number portion
			 ; / of the entry
	sub	dword [u.off], 10
		; sub $10.,u.off / move u.off back 1 directory entry
	call	wdir
		; jsr r0,wdir / free the directory entry
	pop	ax
		; mov (sp)+,r1 / get i-number back
	call	iget
		; jsr r0,iget / get i-node
	call	setimod
		; jsr r0,setimod / set modified flag
	dec	byte [i.nlks]
		; decb i.nlks / decrement the number of links
	jnz	sysret
		; bgt sysret9 / if this was not the last link
			    ; / to file return
	; AX = r1 = i-number
	call	anyi
		; jsr r0,anyi / if it was, see if anyone has it open.
			 ; / Then free contents of file and destroy it.
	jmp	sysret
		; br sysret9
```

```
mkdir:
        ; 12/10/2015
        ; 17/06/2015 (Retro UNIX 386 v1 - Beginning)
        ; 29/04/2013 - 01/08/2013 (Retro UNIX 8086 v1)
        ;
        ; 'mkdir' makes a directory entry from the name pointed to
        ; by u.namep into the current directory.
        ;
        ; INPUTS ->
        ;    u.namep - points to a file name
        ;                 that is about to be a directory entry.
        ;    ii - current directory's i-number.
        ; OUTPUTS ->
        ;    u.dirbuf+2 - u.dirbuf+10 - contains file name.
        ;    u.off - points to entry to be filled
        ;           in the current directory
        ;    u.base - points to start of u.dirbuf.
        ;    r1 - contains i-number of current directory
        ;
        ; ((AX = R1)) output
        ;
        ;    (Retro UNIX Prototype : 11/11/2012, UNIXCOPY.ASM)
        ;    ((Modified registers: eAX, eDX, eBX, eCX, eSI, eDI, eBP))

        ; 17/06/2015 - 32 bit modifications (Retro UNIX 386 v1)
        xor     eax, eax
         mov     edi, u.dirbuf+2
        mov     esi, edi
        stosd
        stosd
                ; jsr r0,copyz; u.dirbuf+2; u.dirbuf+10. / clear this
        mov     edi, esi ; offset to u.dirbuf
        ; 12/10/2015 ([u.namep] -> ebp)
        ;mov    ebp, [u.namep]
        call    trans_addr_nmbp ; convert virtual address to physical
                ; esi = physical address (page start + offset)
                ; ecx = byte count in the page (1 - 4096)
        ; edi = offset to u.dirbuf (edi is not modified in trans_addr_nm)
                ; mov u.namep,r2 / r2 points to name of directory entry
                ; mov $u.dirbuf+2,r3 / r3 points to u.dirbuf+2
mkdir_1: ; 1:
        inc     ebp ; 12/10/2015
        ; / put characters in the directory name in u.dirbuf+2 - u.dirbuf+10
         ; 01/08/2013
        lodsb
                ; movb (r2)+,r1 / move character in name to r1
        and     al, al
        jz      short mkdir_3
                ; beq 1f / if null, done
        cmp     al, '/'
                ; cmp r1,$'/ / is it a "/"?
        je      short mkdir_err
        ;je     error
                ; beq error9 / yes, error
        ; 12/10/2015
        dec     cx
        jnz     short mkdir_2
        ; 12/10/2015 ([u.namep] -> ebp)
        call    trans_addr_nm ; convert virtual address to physical
                ; esi = physical address (page start + offset)
                ; ecx = byte count in the page
        ; edi = offset to u.dirbuf (edi is not modified in trans_addr_nm)
mkdir_2:
        cmp      edi, u.dirbuf+10
                ; cmp r3,$u.dirbuf+10. / have we reached the last slot for
                                    ; / a char?
        je      short mkdir_1
                ; beq 1b / yes, go back
        stosb
                ; movb r1,(r3)+ / no, put the char in the u.dirbuf
        jmp     short mkdir_1
                ; br 1b / get next char
mkdir_err:
        ; 17/06/2015
        mov     dword [u.error], ERR_NOT_DIR ; 'not a valid directory !'
        jmp     error
```

```
mkdir_3: ; 1:
        mov     eax, [u.dirp]
        mov     [u.off], eax
                ; mov u.dirp,u.off / pointer to empty current directory
                                ; / slot to u.off
wdir: ; 29/04/2013
         mov     dword [u.base], u.dirbuf
                ; mov $u.dirbuf,u.base / u.base points to created file name
         mov     dword [u.count], 10
                ; mov $10.,u.count / u.count = 10
        mov     ax, [ii]
                ; mov ii,r1 / r1 has i-number of current directory
        mov     dl, 1 ; owner flag mask ; RETRO UNIX 8086 v1 modification !
        call    access
                ; jsr r0,access; 1 / get i-node and set its file up
                                ; / for writing
        ; AX = i-number of current directory
        ; 01/08/2013
        inc      byte [u.kcall] ; the caller is 'mkdir' sign
        call    writei
                ; jsr r0,writei / write into directory
        retn
                ; rts r0

sysexec:
        ; 19/10/2015
        ; 18/10/2015
        ; 10/10/2015
        ; 26/08/2015
        ; 05/08/2015
        ; 29/07/2015
        ; 25/07/2015
        ; 24/07/2015
        ; 21/07/2015
        ; 20/07/2015
        ; 02/07/2015
        ; 01/07/2015
        ; 25/06/2015
        ; 24/06/2015
        ; 23/06/2015 (Retro UNIX 386 v1 - Beginning)
        ; 03/06/2013 - 06/12/2013 (Retro UNIX 8086 v1)
        ;
        ; 'sysexec' initiates execution of a file whose path name if
        ; pointed to by 'name' in the sysexec call.
        ; 'sysexec' performs the following operations:
        ;    1. obtains i-number of file to be executed via 'namei'.
        ;    2. obtains i-node of file to be exceuted via 'iget'.
        ;    3. sets trap vectors to system routines.
        ;    4. loads arguments to be passed to executing file into
        ;       highest locations of user's core
        ;    5. puts pointers to arguments in locations immediately
        ;       following arguments.
        ;    6. saves number of arguments in next location.
        ;    7. intializes user's stack area so that all registers
        ;       will be zeroed and the PS is cleared and the PC set
        ;       to core when 'sysret' restores registers
        ;       and does an rti.
        ;    8. inializes u.r0 and u.sp
        ;    9. zeros user's core down to u.r0
        ;   10. reads executable file from storage device into core
        ;       starting at location 'core'.
        ;   11. sets u.break to point to end of user's code with
        ;       data area appended.
        ;   12. calls 'sysret' which returns control at location
        ;       'core' via 'rti' instruction.
        ;
        ; Calling sequence:
        ;       sysexec; namep; argp
        ; Arguments:
        ;       namep - points to pathname of file to be executed
        ;       argp  - address of table of argument pointers
        ;       argp1... argpn - table of argument pointers
        ;       argp1:<...0> ... argpn:<...0> - argument strings
        ; Inputs: (arguments)
        ; Outputs: -
        ; ...............................................................
        ;
```

```
	; Retro UNIX 386 v1 modification:
	;	User application runs in it's own virtual space
	;	which is izolated from kernel memory (and other
	;	memory pages) via 80386	paging in ring 3
	;	privilige mode. Virtual start address is always 0.
	;	User's core memory starts at linear address 400000h
	;	(the end of the 1st 4MB).
	;
	; Retro UNIX 8086 v1 modification:
	;	user/application segment and system/kernel segment
	;	are different and sysenter/sysret/sysrele routines
	;	are different (user's registers are saved to
	;	and then restored from system's stack.)
	;
	;	NOTE: Retro UNIX 8086 v1 'arg2' routine gets these
	;	      arguments which were in these registers;
	;	      but, it returns by putting the 1st argument
	;	      in 'u.namep' and the 2nd argument
	;	      on top of stack. (1st argument is offset of the
	;	      file/path name in the user's program segment.)

	;call	arg2
	; * name - 'u.namep' points to address of file/path name
	;          in the user's program segment ('u.segmnt')
	;          with offset in BX register (as sysopen argument 1).
	; * argp - sysexec argument 2 is in CX register
	;          which is on top of stack.
	;
		; jsr r0,arg2 / arg0 in u.namep,arg1 on top of stack

	; 23/06/2015 (32 bit modifications)

	mov	[u.namep], ebx ; argument 1
	 ; 18/10/2015
	mov	 [argv], ecx  ; * ; argument 2
	call	namei
		; jsr r0,namei / namei returns i-number of file
				; / named in sysexec call in r1
	;jc	error
		; br error9
	jnc	short sysexec_0
	;
	; 'file not found !' error
	mov	dword [u.error], ERR_FILE_NOT_FOUND
	jmp	error
sysexec_not_exf:
	; 'not executable file !' error
	mov	dword [u.error], ERR_NOT_EXECUTABLE
	jmp	error
sysexec_0:
	call	iget
		; jsr r0,iget / get i-node for file to be executed
	 test	 word [i.flgs], 10h
		; bit $20,i.flgs / is file executable
	jz	short sysexec_not_exf
	;jz	error
		; beq error9
	;;
	call	iopen
		; jsr r0,iopen / gets i-node for file with i-number
				; / given in r1 (opens file)
	; AX = i-number of the file
	test	word [i.flgs], 20h
		; bit $40,i.flgs / test user id on execution bit
	jz	short sysexec_1
		; beq 1f
	cmp	byte [u.uid], 0 ; 02/08/2013
		; tstb u.uid / test user id
	jna	short sysexec_1
		; beq 1f / super user
	mov	cl, [i.uid]
	mov	[u.uid], cl ; 02/08/2013
		; movb i.uid,u.uid / put user id of owner of file
					; / as process user id
```

```
sysexec_1:
        ; 18/10/2215
        ; 10/10/2015
        ; 24/07/2015
        ; 21/07/2015
        ; 25/06/2015
        ; 24/06/2015
         ; Moving arguments to the end of [u.upage]
        ; (by regarding page borders in user's memory space)
        ;
        ; 10/10/2015
        ; 21/07/2015
        mov     ebp, esp ; (**)
        ; 18/10/2015
        mov     edi, ebp
        mov     ecx, MAX_ARG_LEN ; 256
        ;sub    edi, MAX_ARG_LEN ; 256
        sub     edi, ecx
        mov     esp, edi
        xor     eax, eax
        mov     [u.nread], eax ; 0
        mov     [u.pcount], ax ; 0
        dec     ecx ; 256 - 1
        mov     [u.count], ecx ; MAX_ARG_LEN - 1 ; 255
        ;mov    dword [u.count], MAX_ARG_LEN - 1 ; 255
sysexec_2:
        mov     esi, [argv] ; 18/10/2015
        call    get_argp
        mov     ecx, 4 ; mov ecx, 4
sysexec_3:
        and     eax, eax
        jz      short sysexec_6
        ; 18/10/2015
        add     [argv], ecx ; 4
        inc     word [argc]
        ;
        mov     [u.base], eax
sysexec_4:
        call    cpass ; get a character from user's core memory
         jnz      short sysexec_5
                ; (max. 255 chars + null)
        ; 18/10/2015
        sub     al, al
        stosb
        inc     dword [u.nread]
        jmp     short sysexec_6
sysexec_5:
        stosb
        and     al, al
        jnz     short sysexec_4
        mov     ecx, 4
        cmp     [ncount], ecx ; 4
        jb      short sysexec_2
        mov     esi, [nbase]
        add     [nbase], ecx ; 4
        sub     [ncount], cx
        mov     eax, [esi]
        jmp     short sysexec_3
sysexec_6:
        ; 18/10/2015
        ; argument list transfer from user's core memory to
        ; kernel stack frame is OK here.
        ; [u.nread] = ; argument list length
        ;mov    [argv], esp ; start address of argument list
        ;
        ; 18/10/2015
        ; 24/07/2015
         ; 21/07/2015
        ; 02/07/2015
        ; 25/06/2015
        ; 24/06/2015
        ; 23/06/2015
        ;
        mov     ebx, [u.ppgdir] ; parent's page directory
        and     ebx, ebx  ; /etc/init ? (u.ppgdir = 0)
        jz      short sysexec_7
        mov     eax, [u.pgdir] ; physical address of page directory
        call    deallocate_page_dir
```

```
sysexec_7:
        call    make_page_dir
        ;jc     short sysexec_14
        jc      panic  ; allocation error
                       ; after a deallocation would be nonsence !?
        ; 24/07/2015
        ; map kernel pages (1st 4MB) to PDE 0
        ;     of the user's page directory
        ;     (It is needed for interrupts!)
        ; 18/10/2015
        mov     edx, [k_page_dir] ; Kernel's page directory
        mov     eax, [edx] ; physical address of
                          ; kernel's first page table (1st 4 MB)
                          ; (PDE 0 of kernel's page directory)
        mov     edx, [u.pgdir]
        mov     [edx], eax ; PDE 0 (1st 4MB)
        ;
        ; 20/07/2015
        mov     ebx, CORE ; start address = 0 (virtual) + CORE
        ; 18/10/2015
        mov     esi, pcore ; physical start address
sysexec_8:
        mov     ecx, PDE_A_USER + PDE_A_WRITE + PDE_A_PRESENT
        call    make_page_table
        jc      panic
        ;mov    ecx, PTE_A_USER + PTE_A_WRITE + PTE_A_PRESENT
        call    make_page ; make new page, clear and set the pte
        jc      panic
        ;
        mov     [esi], eax ; 24/06/2015
        ; ebx = virtual address (24/07/2015)
        call    add_to_swap_queue
        ; 18/10/2015
        cmp     esi, ecore ; user's stack (last) page ?
        je      short sysexec_9 ; yes
        mov     esi, ecore  ; physical address of the last page
        ; 20/07/2015
        mov     ebx, (ECORE - PAGE_SIZE) + CORE
        ; ebx = virtual end address + segment base address - 4K
         jmp     short sysexec_8

sysexec_9:
        ; 18/10/2015
        ; 26/08/2015
        ; 25/06/2015
        ; move arguments from kernel stack to [ecore]
        ; (argument list/line will be copied from kernel stack
        ; frame to the last (stack) page of user's core memory)
        ; 18/10/2015
        mov     edi, [ecore]
        add     edi, PAGE_SIZE
        movzx   eax, word [argc]
        or      eax, eax
        jnz     short sysexec_10
        mov     ebx, edi
        sub     ebx, 4
        mov     [ebx], eax ; 0
        jmp     short sysexec_13
sysexec_10:
        mov     ecx, [u.nread]
        ;mov    esi, [argv}
        mov     esi, esp ; start address of argument list
        sub     edi, ecx ; page end address - argument list length
        mov     edx, eax
        inc     dl ; argument count + 1 for argc value
        shl     dl, 2  ; 4 * (argument count + 1)
        mov     ebx, edi
        and     bl, 0FCh ; 32 bit (dword) alignment
        sub     ebx, edx
        mov     edx, edi
        rep     movsb
        mov     esi, edx
        mov     edi, ebx
        mov     edx, ECORE - PAGE_SIZE ; virtual addr. of the last page
        sub     edx, [ecore] ; difference (virtual - physical)
        stosd   ; eax = argument count
```

```
sysexec_11:
        mov     eax, esi
        add     eax, edx
        stosd   ; eax = virtual address
        dec     byte [argc]
        jz      short sysexec_13
sysexec_12:
        lodsb
        and     al, al
        jnz     short sysexec_12
        jmp     short sysexec_11
        ;
        ; 1:
                ; mov (sp)+,r5 / r5 now contains address of list of
                            ; / pointers to arguments to be passed
                ; mov $1,u.quit / u.quit determines handling of quits;
                             ; / u.quit = 1 take quit
                ; mov $1,u.intr / u.intr determines handling of
                            ; / interrupts; u.intr = 1 take interrupt
                ; mov $rtssym,30 / emt trap vector set to take
                              ; / system routine
                ; mov $fpsym,*10 / reserved instruction trap vector
                              ; / set to take system routine
                ; mov $sstack,sp / stack space used during swapping
                ; mov r5,-(sp) / save arguments pointer on stack
                ; mov $ecore,r5 / r5 has end of core
                ; mov $core,r4 / r4 has start of users core
                ; mov r4,u.base / u.base has start of users core
                ; mov (sp),r2 / move arguments list pointer into r2
        ; 1:
                ; tst (r2)+ / argument char = "nul"
                ; bne 1b
                ; tst -(r2) / decrement r2 by 2; r2 has addr of
                         ; / end of argument pointer list
        ; 1:
              ; / move arguments to bottom of users core
                ; mov -(r2),r3 / (r3) last non zero argument ptr
                ; cmp r2,(sp) / is r2 = beginning of argument
                           ; / ptr list
                ; blo 1f / branch to 1f when all arguments
                       ; / are moved
                ; mov -(r2),r3 / (r3) last non zero argument ptr
        ; 2:
                ; tstb (r3)+
                ; bne 2b / scan argument for \0 (nul)

        ; 2:
                ; movb -(r3),-(r5) / move argument char
                                ; / by char starting at "ecore"
                ; cmp r3,(r2) / moved all characters in
                           ; / this argument
                ; bhi 2b / branch 2b if not
                ; mov r5,(r4)+ / move r5 into top of users core;
                            ; / r5 has pointer to nth arg
                ; br 1b / string
        ; 1:
                ; clrb -(r5)
                ; bic $1,r5 / make r5 even, r5 points to
                       ; / last word of argument strings
                ; mov $core,r2

        ; 1: / move argument pointers into core following
               ; / argument strings
                ; cmp r2,r4
                ; bhis 1f / branch to 1f when all pointers
                       ; / are moved
                ; mov (r2)+,-(r5)
                ; br 1b
        ; 1:
                ; sub $core,r4 / gives number of arguments *2
                ; asr r4 / divide r4 by 2 to calculate
                       ; / the number of args stored
                ; mov r4,-(r5) / save number of arguments ahead
                            ; / of the argument pointers
```

```
sysexec_13:
        ; 19/10/2015
        ; 18/10/2015
        ; 29/07/2015
        ; 25/07/2015
        ; 24/07/2015
        ; 20/07/2015
        ; 25/06/2015
        ; 24/06/2015
        ; 23/06/2015
        ;
        ; moving arguments to [ecore] is OK here..
        ; 18/10/2015
        mov     esp, ebp ; (**) restore kernel stack pointer
        ; ebx = beginning addres of argument list pointers
        ;       in user's stack
        ; 19/10/2015
        sub     ebx, [ecore]
        add     ebx, (ECORE - PAGE_SIZE)
                        ; end of core - 4096 (last page)
                        ; (virtual address)
        mov     [argv], ebx
        mov     [u.break], ebx ; available user memory
        ;
        sub     eax, eax
        mov     dword [u.count], 32 ; Executable file header size
                ; mov $14,u.count
        mov     dword [u.fofp], u.off
                ; mov $u.off,u.fofp
        mov     [u.off], eax ; 0
                ; clr u.off / set offset in file to be read to zero
        ; 25/07/2015
        mov     [u.base], eax ; 0, start of user's core (virtual)
        ; 25/06/2015
        mov     ax, [ii]
        ; AX = i-number of the executable file
        call    readi
                ; jsr r0,readi / read in first six words of
                        ; / user's file, starting at $core
                ; mov sp,r5 / put users stack address in r5
                ; sub $core+40.,r5 / subtract $core +40,
                                ; / from r5 (leaves number of words
                                ; / less 26 available for
                                ; / program in user core
                ; mov r5,u.count /
        ; 25/06/2015
        mov     ecx, [u.break] ; top of user's stack (physical addr.)
        mov     [u.count], ecx ; save for overrun check
        ;
        mov     ecx, [u.nread]
        mov     [u.break], ecx ; virtual address (offset from start)
        cmp     cl, 32
        jne     short sysexec_15
        ;:
        ; 25/06/2015
        ; Retro UNIX 386 v1 (32 bit) executable file header format
        ; 18/10/2015
        mov     esi, [pcore] ; start address of user's core memory
                            ; (phys. start addr. of the exec. file)
        lodsd
        cmp     ax, 1EEBh ; EBH, 1Eh -> jump to +32
        jne     short sysexec_15
                ; cmp core,$405 / br .+14 is first instruction
                            ; / if file is standard a.out format
                ; bne 1f / branch, if not standard format
        lodsd
        mov     ecx, eax ; text (code) section size
        lodsd
        add     ecx, eax ; + data section size (initialized data)
                ; mov core+2,r5 / put 2nd word of users program in r5;
                            ; / number of bytes in program text
                ; sub $14,r5 / subtract 12
        mov     ebx, ecx
```

```
        ; 25/06/2015
        ; NOTE: These are for next versions of Retro UNIX 386
        ;       and SINGLIX operating systems (as code template).
        ;       Current Retro UNIX 386 v1 files can be max. 64KB
        ;       due to RUFS (floppy disk file system) restriction...
        ;       Overrun is not possible for current version.
        ;
        lodsd
        add     ebx, eax ; + bss section size (for overrun checking)
        cmp     ebx, [u.count]
        ja      short sysexec_14  ; program overruns stack !
        ;
        ; 24/07/2015
        ; add bss section size to [u.break]
        add     [u.break], eax
        ;
        sub     ecx, 32  ; header size (already loaded)
        ;cmp    ecx, [u.count]
        ;jnb    short sysexec_16
                ; cmp r5,u.count /
                ; bgt 1f / branch if r5 greater than u.count
        mov     [u.count], ecx ; required read count
                ; mov r5,u.count
        jmp     short sysexec_16
sysexec_14:
        ; 23/06/2015
        ; insufficient (out of) memory
        mov     dword [u.error], ERR_MINOR_IM ; 1
        jmp     error
sysexec_15:
        ; 25/06/2015
         movzx   edx, word [i.size] ; file size
        sub     edx, ecx ; file size - loaded bytes
        jna     short sysexec_17 ; no need to next read
        add     ecx, edx ; [i.size]
        cmp     ecx, [u.count] ; overrun check (!)
        ja      short sysexec_14
        mov     [u.count], edx
sysexec_16:
        mov     ax, [ii] ; i-number
        call    readi
                ; add core+10,u.nread / add size of user data area
                                    ; / to u.nread
                ; br 2f
        ; 1:
                ; jsr r0,readi / read in rest of file
        ; 2:
        mov     ecx, [u.nread]
        add     [u.break], ecx
                ; mov u.nread,u.break / set users program break to end of
                                  ; / user code
                ; add $core+14,u.break / plus data area
sysexec_17: ; 20/07/2015
        ;mov    ax, [ii] ;rgc i-number
        call    iclose
                ; jsr r0,iclose / does nothing
         xor     eax, eax
        inc     al
        mov     [u.intr], ax ; 1 (interrupt/time-out is enabled)
        mov     [u.quit], ax ; 1 ('crtl+brk' signal is enabled)
        ; 02/07/2015
         cmp    dword [u.ppgdir], 0  ; is the caller sys_init (kernel) ?
        ja      short sysexec_18 ; no, the caller is user process
        ; If the caller is kernel (sys_init), 'sysexec' will come here
        mov     edx, [k_page_dir] ; kernel's page directory
        mov     [u.ppgdir], edx ; next time 'sysexec' must not come here
sysexec_18:
        ; 18/10/2015
        ; 05/08/2015
        ; 29/07/2015
        mov     ebp, [argv] ; user's stack pointer must point to argument
                           ; list pointers (argument count)
        cli
         mov     esp, [tss.esp0]  ; ring 0 (kernel) stack pointer
        ;mov    esp, [u.sp] ; Restore Kernel stack
                           ; for this process
        ;add    esp, 20 ; --> EIP, CS, EFLAGS, ESP, SS
        ;xor    eax, eax ; 0
        dec     al ; eax = 0
```

```
mov     dx, UDATA
push    dx  ; user's stack segment
push    ebp ; user's stack pointer
            ; (points to number of arguments)
sti
pushfd  ; EFLAGS
        ; Set IF for enabling interrupts in user mode
;or     dword [esp], 200h
;
;mov    bx, UCODE
;push   bx ; user's code segment
push    UCODE
;push   0
push    eax ; EIP (=0) - start address -
        ; clr -(r5) / popped into ps when rti in
                  ; / sysrele is executed
        ; mov $core,-(r5) / popped into pc when rti
                         ; / in sysrele is executed
        ;mov r5,0f / load second copyz argument
        ;tst -(r5) / decrement r5
mov     [u.sp], esp ; 29/07/2015
; 05/08/2015
; Remedy of a General Protection Fault during 'iretd' is here !
; ('push dx' would cause to general protection fault,
; after 'pop ds' etc.)
;
;; push dx ; ds (UDATA)
;; push dx ; es (UDATA)
;; push dx ; fs (UDATA)
;; push dx ; gs (UDATA)
;
; This is a trick to prevent general protection fault
; during 'iretd' intruction at the end of 'sysrele' (in u1.s):
mov     es, dx ; UDATA
push    es ; ds (UDATA)
push    es ; es (UDATA)
push    es ; fs (UDATA)
push    es ; gs (UDATA)
mov     dx, KDATA
mov     es, dx
;
;; pushad simulation
mov     ebp, esp ; esp before pushad
push    eax ; eax (0)
push    eax ; ecx (0)
push    eax ; edx (0)
push    eax ; ebx (0)
push    ebp ; esp before pushad
push    eax ; ebp (0)
push    eax ; esi (0)
push    eax ; edi (0)
;
mov     [u.r0], eax ; eax = 0
mov     [u.usp], esp
        ; mov r5,u.r0 /
        ; sub $16.,r5 / skip 8 words
        ; mov r5,u.sp / assign user stack pointer value,
        ;              / effectively zeroes all regs
                    ; / when sysrele is executed
        ; jsr r0,copyz; core; 0:0 / zero user's core
        ; clr u.break
        ; mov r5,sp / point sp to user's stack
;
jmp     sysret0
;jmp    sysret
        ; br sysret3 / return to core image at $core
```

```
get_argp:
	; 18/10/2015 (nbase, ncount)
	; 21/07/2015
	; 24/06/2015 (Retro UNIX 386 v1)
	; Get (virtual) address of argument from user's core memory
	;
	; INPUT:
	;	esi = virtual address of argument pointer
	; OUTPUT:
	;	eax = virtual address of argument
	;
	; Modified registers: EAX, EBX, ECX, EDX, ESI
	;
	cmp	dword [u.ppgdir], 0 ; /etc/init ?
				; (the caller is kernel)
	jna	short get_argpk
	;
	mov	ebx, esi
	call	get_physical_addr ; get physical address
	jc	get_argp_err
	mov	[nbase], eax ; physical address
	mov	[ncount], cx ; remain byte count in page (1-4096)
	mov	eax, 4 ; 21/07/2015
	cmp	cx, ax ; 4
	jnb	short get_argp2
	mov	ebx, esi
	add	ebx, ecx
	call	get_physical_addr ; get physical address
	jc	short get_argp_err
	;push	esi
	mov	esi, eax
	xchg	cx, [ncount]
	xchg	esi, [nbase]
	mov	ch, 4
	sub	ch, cl
get_argp0:
	lodsb
	push	ax
	dec	cl
	jnz	short get_argp0
	mov	esi, [nbase]
	; 21/07/2015
	movzx	eax, ch
	add	[nbase], eax
	sub	[ncount], ax
get_argp1:
	lodsb
	dec	ch
	jz	short get_argp3
	push	ax
	jmp	short get_argp1
get_argpk:
	; Argument is in kernel's memory space
	mov	word [ncount], PAGE_SIZE ; 4096
	mov	[nbase], esi
	add	dword [nbase], 4
	mov	eax, [esi] ; virtual addr. = physcal addr.
	retn
get_argp2:
	; 21/07/2015
	;mov	eax, 4
	mov	edx, [nbase] ; 18/10/2015
	add	[nbase], eax
	sub	[ncount], ax
	;
	mov	eax, [edx]
	retn
get_argp_err:
	mov	[u.error], eax
	jmp	error
get_argp3:
	mov	cl, 3
get_argp4:
	shl	eax, 8
	pop	dx
	mov	al, dl
	loop	get_argp4
	;pop	esi
	retn
```

```
sysfstat:
        ; 23/06/2015 (Retro UNIX 386 v1 - Beginning)
        ; 19/06/2013 (Retro UNIX 8086 v1)
        ;
        ; 'sysfstat' is identical to 'sysstat' except that it operates
        ; on open files instead of files given by name. It puts the
        ; buffer address on the stack, gets the i-number and
        ; checks to see if the file is open for reading or writing.
        ; If the file is open for writing (i-number is negative)
        ; the i-number is set positive and a branch into 'sysstat'
        ; is made.
        ;
        ; Calling sequence:
        ;       sysfstat; buf
        ; Arguments:
        ;       buf - buffer address
        ;
        ; Inputs: *u.r0 - file descriptor
        ; Outputs: buffer is loaded with file information
        ; ...............................................................
        ;
        ; Retro UNIX 8086 v1 modification:
        ;        'sysfstat' system call has two arguments; so,
        ;       * 1st argument, file descriptor is in BX register
        ;       * 2nd argument, buf is pointed to by CX register

        ; / set status of open file
                ; jsr r0,arg; u.off / put buffer address in u.off
        push    ecx
                ; mov u.off,-(sp) / put buffer address on the stack
                ; mov *u.r0,r1 / put file descriptor in r1
                ; jsr r0,getf / get the files i-number
        ; BX = file descriptor (file number)
        call    getf1
        and     ax, ax ; i-number of the file
                ; tst   r1 / is it 0?
        ;jz     error
                ; beq error3 / yes, error
        jnz     short sysfstat1
        mov     dword [u.error], ERR_FILE_NOT_OPEN  ; 'file not open !'
        jmp     error
sysfstat1:
        cmp     ah, 80h
         jb      short sysstat1
                ; bgt 1f / if i-number is negative (open for writing)
        neg     ax
                ; neg r1 / make it positive, then branch
        jmp     short sysstat1
                ; br 1f / to 1f
sysstat:
        ; 18/10/2015
        ; 07/10/2015
        ; 02/09/2015
        ; 23/06/2015 (Retro UNIX 386 v1 - Beginning)
        ; 19/06/2013 (Retro UNIX 8086 v1)
        ;
        ; 'sysstat' gets the status of a file. Its arguments are the
        ; name of the file and buffer address. The buffer is 34 bytes
        ; long and information about the file placed in it.
        ; sysstat calls 'namei' to get the i-number of the file.
        ; Then 'iget' is called to get i-node in core. The buffer
        ; is then loaded and the results are given in the UNIX
        ; Programmers Manual sysstat (II).
        ;
        ; Calling sequence:
        ;       sysstat; name; buf
        ; Arguments:
        ;       name - points to the name of the file
        ;       buf - address of a 34 bytes buffer
        ; Inputs: -
        ; Outputs: buffer is loaded with file information
        ; ...............................................................
        ;
```

```
	; Retro UNIX 8086 v1 modification:
	;	 'sysstat' system call has two arguments; so,
	;	Retro UNIX 8086 v1 argument transfer method 2 is used
	;	to get sysstat system call arguments from the user;
	;	* 1st argument, name is pointed to by BX register
	;	* 2nd argument, buf is pointed to by CX register
	;
	;	NOTE: Retro UNIX 8086 v1 'arg2' routine gets these
	;	      arguments which were in these registers;
	;	      but, it returns by putting the 1st argument
	;	      in 'u.namep' and the 2nd argument
	;	      on top of stack. (1st argument is offset of the
	;	      file/path name in the user's program segment.)

	; / ; name of file; buffer - get files status
		; jsr r0,arg2 / get the 2 arguments
	mov	[u.namep], ebx
	push	ecx
	call	namei
		; jsr r0,namei / get the i-number for the file
	;jc	error
		; br error3 / no such file, error
	jnc	short sysstat1
	; pop	ecx
sysstat_err0:
	; 'file not found !' error
	mov	dword [u.error], ERR_FILE_NOT_FOUND ; 12
	jmp	error

statx: db 0

sysstat1: ; 1:
	call	iget
		; jsr r0,iget / get the i-node into core
	; 07/10/2015 (ax = [ii], inode number)
	; 02/09/2015
	pop	dword [u.base]
		; mov (sp)+,r3 / move u.off to r3 (points to buffer)
	call	sysstat_gpa ; get physical address
	jnc	short sysstat2
sysstat_err1:
	mov	dword [u.error], eax ; error code
	jmp	error
sysstat2:
	mov	al, [ii] ; 07/10/2015 (result of 'iget' call, above)
	stosb
	inc	dword [u.base]
	dec	cx
	jnz	short sysstat3
	call	sysstat_gpa
	;jc	short sysstat_err1
sysstat3:
	mov	al, [ii+1] ; 07/10/2015 (result of 'iget' call, above)
	stosb
		; mov r1,(r3)+ / put i-number in 1st word of buffer
	inc	dword [u.base]
	;dec	word [u.pcount]
	dec	cx
	jnz	short sysstat4
	call	sysstat_gpa
	;jc	short sysstat_err1
sysstat4:
	mov	esi, inode
		; mov $inode,r2 / r2 points to i-node
sysstat5: ; 1:
	movsb
		; mov (r2)+,(r3)+ / move rest of i-node to buffer
	inc	dword [u.base]
	;dec	word [u.pcount]
	dec	cx
	jnz	short sysstat6
	call	sysstat_gpa
	;jc	short sysstat_err1
sysstat6:
	cmp	esi, inode + 32
		; cmp r2,$inode+32 / done?
	jne	short sysstat5
		; bne 1b / no, go back
	jmp	sysret
```

```
		; br sysret3 / return through sysret
	;
sysstat_gpa: ; get physical address of file status buffer
	; 02/09/2015
	mov	ebx, [u.base]
	; 07/10/2015
	call	get_physical_addr ; get physical address
	;jc	short sysstat_gpa1
	jc	short sysstat_err1
	; 18/10/2015
	mov	edi, eax ; physical address
	;mov	[u.pcount], cx ; remain bytes in page
;sysstat_gpa1:
	retn

fclose:
	; 18/06/2015 (Retro UNIX 386 v1 - Beginning)
	;	     (32 bit offset pointer modification)
	; 19/04/2013 - 12/01/2014 (Retro UNIX 8086 v1)
	;
	; Given the file descriptor (index to the u.fp list)
	; 'fclose' first gets the i-number of the file via 'getf'.
	; If i-node is active (i-number > 0) the entry in
	; u.fp list is cleared. If all the processes that opened
	; that file close it, then fsp etry is freed and the file
	; is closed. If not a return is taken.
	; If the file has been deleted while open, 'anyi' is called
	; to see anyone else has it open, i.e., see if it is appears
	; in another entry in the fsp table. Upon return from 'anyi'
	; a check is made to see if the file is special.
	;
	; INPUTS ->
	;    r1 - contains the file descriptor (value=0,1,2...)
	;    u.fp - list of entries in the fsp table
	;    fsp - table of entries (4 words/entry) of open files.
	; OUTPUTS ->
	;    r1 - contains the same file descriptor
	;    r2 - contains i-number
	;
	; ((AX = R1))
	; ((Modified registers: eDX, eBX, eCX, eSI, eDI, eBP))
	;
	; Retro UNIX 8086 v1 modification : CF = 1
	;		if i-number of the file is 0. (error)
	;
	movzx	edx, ax ; **
	push	ax ; ***
		; mov r1,-(sp) / put r1 on the stack (it contains
			    ; / the index to u.fp list)
	call	getf
		; jsr r0,getf / r1 contains i-number,
			   ; / cdev has device =, u.fofp
			   ; / points to 3rd word of fsp entry
	cmp	ax, 1 ; r1
		; tst r1 / is i-number 0?
	jb	short fclose_2
		; beq 1f / yes, i-node not active so return
		; tst (r0)+ / no, jump over error return
	mov	ebx, edx ; **
	mov	dx, ax ; *
		; mov r1,r2 / move i-number to r2 ;*
		; mov (sp),r1 / restore value of r1 from the stack
			   ; / which is index to u.fp ; **
	mov	byte [ebx+u.fp], 0
		; clrb u.fp(r1) / clear that entry in the u.fp list
	mov	ebx, [u.fofp]
		; mov u.fofp,r1 / r1 points to 3rd word in fsp entry
fclose_0:
	dec	byte [ebx+4] ; 18/06/2015
		; decb 2(r1) / decrement the number of processes
			  ; / that have opened the file
	jns	short fclose_2 ; jump if not negative (jump if bit 7 is 0)
		; bge 1f / if all processes haven't closed the file, return
	;
	push	dx ;*
		; mov r2,-(sp) / put r2 on the stack (i-number)
	xor	ax, ax ; 0
	mov	[ebx-4], ax ; 0
		; clr -4(r1) / clear 1st word of fsp entry
```

```
        mov     al, [ebx+5] ; 18/06/2015
                ; tstb 3(r1) / has this file been deleted
        and     al, al
        jz      short fclose_1
                ; beq 2f / no, branch
        mov     ax, dx ; *
                ; mov r2,r1 / yes, put i-number back into r1
        ; AX = inode number
        call    anyi
                ; jsr r0,anyi / free all blocks related to i-number
                           ; / check if file appears in fsp again
fclose_1: ; 2:
        pop     ax ; *
                ; mov (sp)+,r1 / put i-number back into r1
        call    iclose ; close if it is special file
                ; jsr r0,iclose / check to see if its a special file
fclose_2: ; 1:
        pop     ax ; ***
                ; mov (sp)+,r1 / put index to u.fp back into r1
        retn
                ; rts r0

getf:   ; / get the device number and the i-number of an open file
        ; 13/05/2015
        ; 11/05/2015 (Retro UNIX 386 v1 - Beginning)
        ; 19/04/2013 - 18/11/2013 (Retro UNIX 8086 v1)
        ;
        mov     ebx, eax
getf1: ;; Calling point from 'rw1' (23/05/2013)
        cmp     ebx, 10
                ; cmp r1,$10. / user limited to 10 open files
         jnb    short getf2 ; 13/05/2015
        ;jnb      error
                ; bhis error3 / u.fp is table of users open files,
                           ; / index in fsp table
        mov     bl, [ebx+u.fp]
                ; movb u.fp(r1),r1 / r1 contains number of entry
                                  ; / in fsp table
        or      bl, bl
        jnz     short getf3
        ;jz     short getf4
                ; beq 1f / if its zero return
getf2:
        ; 'File not open !' error (ax=0)
        sub     eax, eax
        retn
getf3:
        ; Retro UNIX 386 v1 modification ! (11/05/2015)
        ;
        ; 'fsp' table (10 bytes/entry)
        ; bit 15                                bit 0
        ; ---|-------------------------------------------
        ; r/w|         i-number of open file
        ; ---|-------------------------------------------
        ;                device number
        ; -----------------------------------------------
        ; offset pointer, r/w pointer to file (bit 0-15)
        ; -----------------------------------------------
        ; offset pointer, r/w pointer to file (bit 16-31)
        ; ----------------------|-------------------------
        ;  flag that says file       | number of processes
        ;   has been deleted   | that have file open
        ; ----------------------|-------------------------
        ;
        mov     eax, 10
        mul     bl
        mov     ebx, fsp - 6 ; the 3rd word in the fsp entry
        add     ebx, eax
                ; asl r1
                ; asl r1 / multiply by 8 to get index into
                       ; / fsp table entry
                ; asl r1
                ; add $fsp-4,r1 / r1 is pointing at the 3rd word
                             ; / in the fsp entry
        mov     [u.fofp], ebx
                ; mov r1,u.fofp / save address of 3rd word
                             ; / in fsp entry in u.fofp
        dec     ebx
        dec     ebx
```

```
        mov     ax, [ebx]
        ;mov    [cdev], al ; ;;Retro UNIX 8086 v1 !
        mov     [cdev], ax ; ;;in fact (!)
                            ;;dev number is in 1 byte
                ; mov -(r1),cdev / remove the device number  cdev
        dec     ebx
        dec     ebx
        mov     ax, [ebx]
                ; mov -(r1),r1 / and the i-number  r1
getf4:  ; 1:
        retn
                ; rts r0

namei:
        ; 18/10/2015 (nbase, ncount)
        ; 12/10/2015
        ; 21/08/2015
        ; 18/07/2015
        ; 02/07/2015
        ; 17/06/2015
        ; 16/06/2015 (Retro UNIX 386 v1 - Beginning)
        ; 24/04/2013 - 31/07/2013 (Retro UNIX 8086 v1)
        ;
        ; 'namei' takes a file path name and returns i-number of
        ; the file in the current directory or the root directory
        ; (if the first character of the pathname is '/').
        ;
        ; INPUTS ->
        ;    u.namep - points to a file path name
        ;    u.cdir - i-number of users directory
        ;    u.cdev - device number on which user directory resides
        ; OUTPUTS ->
        ;    r1 - i-number of file
        ;    cdev
        ;    u.dirbuf - points to directory entry where a match
        ;               occurs in the search for file path name.
        ;              If no match u.dirb points to the end of
        ;               the directory and r1 = i-number of the current
        ;              directory.
        ; ((AX = R1))
        ;
        ; (Retro UNIX Prototype : 07/10/2012 - 05/01/2013, UNIXCOPY.ASM)
         ; ((Modified registers: eDX, eBX, eCX, eSI, eDI, eBP))
        ;

        mov     ax, [u.cdir]
                ; mov u.cdir,r1 / put the i-number of current directory
                             ; / in r1
        mov     dx, [u.cdrv]
        mov     [cdev], dx          ; NOTE: Retro UNIX 8086 v1
                                    ; device/drive number is in 1 byte,
                                    ; not in 1 word!
                ; mov u.cdev,cdev / device number for users directory
                              ; / into cdev
        ; 12/10/2015
        ; 16/06/2015 - 32 bit modifications (Retro UNIX 386 v1)
         ; convert virtual (pathname) addr to physical address
        call     trans_addr_nmbp ; 12/10/2015
                ; esi = physical address of [u.namep]
                ; ecx = byte count in the page
        cmp     byte [esi], '/'
                ; cmpb *u.namep,$'/ / is first char in file name a /
        jne     short namei_1
                ; bne 1f
        inc     dword [u.namep]
                ; inc u.namep / go to next char
        dec     cx ; remain byte count in the page
        jnz     short namei_0
        ; 12/10/2015
        call    trans_addr_nmbp ; convert virtual address to physical
                ; esi = physical address (page start + offset)
                ; ecx = byte count in the page
        dec     esi
namei_0:
        inc     esi  ; go to next char
        mov     ax, [rootdir] ; 09/07/2013
                ; mov rootdir,r1 / put i-number of rootdirectory in r1
        mov     byte [cdev], 0
                ; clr cdev / clear device number
```

```
namei_1: ; 1:
        test    byte [esi], 0FFh
        jz      short getf4
        ;jz       nig
                ; tstb *u.namep / is the character in file name a nul
                ; beq nig / yes, end of file name reached;
                        ; / branch to "nig"
namei_2: ; 1:
        ; 18/10/2015
        mov     [nbase], esi
        mov     [ncount], cx
        ;
        ;mov    dx, 2
        mov     dl, 2 ; user flag (read, non-owner)
        call    access
                ; jsr r0,access; 2 / get i-node with i-number r1
        ; 'access' will not return here if user has not "r" permission !
        test    word [i.flgs], 4000h
                ; bit $40000,i.flgs / directory i-node?
         jz       short namei_err
                ; beq error3 / no, got an error
        ; 16/06/2015 - 32 bit modifications (Retro UNIX 386 v1)
        xor     eax, eax
        mov     [u.off], eax ; 0
        mov     ax, [i.size]
        mov     [u.dirp], eax
                ; mov i.size,u.dirp / put size of directory in u.dirp
                ; clr u.off / u.off is file offset used by user
        mov     dword [u.fofp], u.off
                ; mov $u.off,u.fofp / u.fofp is a pointer to
                                 ; / the offset portion of fsp entry
namei_3: ; 2:
        mov     dword [u.base], u.dirbuf
                ; mov $u.dirbuf,u.base / u.dirbuf holds a file name
                                   ; / copied from a directory
        mov     dword [u.count], 10
                ; mov $10.,u.count / u.count is byte count
                                ; / for reads and writes
        mov     ax, [ii]
        ; 31/07/2013 ('namei_r') - 16/06/2015 ('u.kcall')
        inc      byte [u.kcall] ; the caller is 'namei' sign
        call    readi
                ; jsr r0,readi / read 10. bytes of file
                      ; with i-number (r1); i.e. read a directory entry
        mov     ecx, [u.nread]
        or      ecx, ecx
                ; tst u.nread
        jz      short nib
                ; ble nib / gives error return
        ;
        mov     bx, [u.dirbuf]
        and     bx, bx
                ; tst u.dirbuf /
        jnz     short namei_4
                ; bne 3f / branch when active directory entry
                       ; / (i-node word in entry non zero)
        mov     eax, [u.off]
        sub     eax, 10
        mov     [u.dirp], eax
                ; mov u.off,u.dirp
                ; sub $10.,u.dirp
        jmp     short namei_3
                ; br 2b

        ; 18/07/2013
nib:
        xor     eax, eax  ; xor ax, ax ; ax = 0 -> file not found
        stc
nig:
        retn

namei_err:
        ; 16/06/2015
        mov     dword [u.error], ERR_NOT_DIR ; 'not a directory !' error
        jmp     error
```

```
namei_4: ; 3:
        ; 18/10/2015
        ; 12/10/2015
        ; 21/08/2015
        ; 18/07/2015
        mov     ebp, [u.namep]
                ; mov u.namep,r2 / u.namep points into a file name string
        mov     edi, u.dirbuf + 2
                ; mov $u.dirbuf+2,r3 / points to file name of directory entry
        ; 18/10/2015
        mov     esi, [nbase]
        mov     cx, [ncount]
        ;
        and     cx, cx
        jnz     short namei_5
        ;
        call    trans_addr_nm ; convert virtual address to physical
                ; esi = physical address (page start + offset)
                ; ecx = byte count in the page
namei_5: ; 3:
        inc     ebp ; 18/07/2015
        lodsb    ; mov al, [esi] ; inc esi (al = r4)
                ; movb (r2)+,r4 / move a character from u.namep string into r4
        or      al, al
        jz      short namei_7
                ; beq 3f / if char is nul, then the last char in string
                        ; / has been moved
        cmp     al, '/'
                ; cmp r4,$'/ / is char a </>
        je      short namei_7
                ; beq 3f
        ; 12/10/2015
        dec     cx ; remain byte count in the page
        jnz     short namei_6
        call    trans_addr_nm ; convert virtual address to physical
                ; esi = physical address (page start + offset)
                ; ecx = byte count in the page
namei_6:
         cmp     edi, u.dirbuf + 10
                ; cmp r3,$u.dirbuf+10. / have I checked
                                    ; / all 8 bytes of file name
        je      short namei_5
                ; beq 3b
        scasb
                ; cmpb (r3)+,r4 / compare char in u.namep string to file name
                             ; / char read from directory
        je      short namei_5
                ; beq 3b / branch if chars match

         jmp     namei_3 ; 2b
                ; br 2b / file names do not match go to next directory entry
namei_7: ; 3:
        cmp     edi, u.dirbuf + 10
                ; cmp r3,$u.dirbuf+10. / if equal all 8 bytes were matched
        je      short namei_8
                ; beq 3f
        mov     ah, [edi]
        ;inc    edi
        and     ah, ah
                ; tstb (r3)+ /
         jnz     namei_3
                ; bne 2b
namei_8: ; 3
        mov     [u.namep], ebp ; 18/07/2015
                ; mov r2,u.namep / u.namep points to char
                              ; / following a / or nul
        ;mov    bx, [u.dirbuf]
                ; mov u.dirbuf,r1 / move i-node number in directory
                              ; / entry to r1
        and     al, al
                ; tst r4 / if r4 = 0 the end of file name reached,
                        ;  / if r4 = </> then go to next directory
        ; mov   ax, bx
        mov     ax, [u.dirbuf] ; 17/06/2015
         jnz     namei_2
                ; bne 1b
        ; AX = i-number of the file
;;nig:
        retn
```

```
                ; tst (r0)+ / gives non-error return
;;nib:
;;      xor     ax, ax ; Retro UNIX 8086 v1 modification !
                       ; ax = 0 -> file not found
;;      stc     ; 27/05/2013
;;      retn
                ; rts r0

trans_addr_nmbp:
        ; 18/10/2015
        ; 12/10/2015
        mov     ebp, [u.namep]
trans_addr_nm:
        ; Convert virtual (pathname) address to physical address
        ; (Retro UNIX 386 v1 feature only !)
        ; 18/10/2015
        ; 12/10/2015 (u.pnbase & u.pncount has been removed from code)
        ; 02/07/2015
        ; 17/06/2015
        ; 16/06/2015
        ;
        ; INPUTS:
        ;       ebp = pathname address (virtual) ; [u.namep]
        ;       [u.pgdir] = user's page directory
        ; OUTPUT:
        ;        esi = physical address of the pathname
        ;       ecx = remain byte count in the page
        ;
        ; (Modified registers: EAX, EBX, ECX, EDX, ESI)
        ;
         cmp     dword [u.ppgdir], 0  ; /etc/init ? (sysexec)
        jna     short trans_addr_nmk ; the caller is os kernel;
                                     ; it is already physical address
        push    eax
        mov     ebx, ebp ; [u.namep] ; pathname address (virtual)
        call    get_physical_addr ; get physical address
        jc      short tr_addr_nm_err
        ; 18/10/2015
        ; eax = physical address
        ; cx = remain byte count in page (1-4096)
                ; 12/10/2015 (cx = [u.pncount])
        mov     esi, eax ; 12/10/2015 (esi=[u.pnbase])
        pop     eax
        retn
tr_addr_nm_err:
        mov     [u.error], eax
        ;pop    eax
        jmp     error
trans_addr_nmk:
        ; 12/10/2015
        ; 02/07/2015
        mov     esi, [u.namep]  ; [u.pnbase]
        mov     cx, PAGE_SIZE ; 4096 ; [u.pncount]
        retn

syschdir:
        ; / makes the directory specified in the argument
        ; / the current directory
        ;
        ; 23/06/2015 (Retro UNIX 386 v1 - Beginning)
        ; 19/06/2013 (Retro UNIX 8086 v1)
        ;
        ; 'syschdir' makes the directory specified in its argument
        ; the current working directory.
        ;
        ; Calling sequence:
        ;       syschdir; name
        ; Arguments:
        ;       name - address of the path name of a directory
        ;              terminated by nul byte.
        ; Inputs: -
        ; Outputs: -
        ; ...............................................................
        ;
        ; Retro UNIX 8086 v1 modification:
        ;        The user/application program puts address of
        ;        the path name in BX register as 'syschdir'
        ;        system call argument.
```

```
        mov     [u.namep], ebx
                ;jsr r0,arg; u.namep / u.namep points to path name
        call    namei
                ; jsr r0,namei / find its i-number
        ;jc     error
                ; br error3
        jnc     short syschdir0
        ; 'directory not found !' error
        mov     dword [u.error], ERR_DIR_NOT_FOUND ; 12
        jmp     error
syschdir0:
        call    access
                ; jsr r0,access; 2 / get i-node into core
        test    word [i.flgs], 4000h
                ; bit $40000,i.flgs / is it a directory?
        ;jz     error
                ; beq error3 / no error
        jnz     short syschdir1
        mov     dword [u.error], ERR_NOT_DIR ; 'not a valid directory !'
        jmp     error
syschdir1:
        mov     [u.cdir], ax
                ; mov r1,u.cdir / move i-number to users
                              ; / current directory
        mov     ax, [cdev]
        mov     [u.cdrv], ax
                ; mov cdev,u.cdev / move its device to users
                                ; / current device
        jmp     sysret
                ; br sysret3

syschmod: ; < change mode of file >
        ; 23/06/2015 (Retro UNIX 386 v1 - Beginning)
        ; 20/06/2013 - 07/07/2013 (Retro UNIX 8086 v1)
        ;
        ; 'syschmod' changes mode of the file whose name is given as
        ; null terminated string pointed to by 'name' has it's mode
        ; changed to 'mode'.
        ;
        ; Calling sequence:
        ;       syschmod; name; mode
        ; Arguments:
        ;       name - address of the file name
        ;              terminated by null byte.
        ;       mode - (new) mode/flags < attributes >
        ;
        ; Inputs: -
        ; Outputs: -
        ; ...............................................................
        ;
        ; Retro UNIX 8086 v1 modification:
        ;        'syschmod' system call has two arguments; so,
        ;       * 1st argument, name is pointed to by BX register
        ;       * 2nd argument, mode is in CX register
        ;
        ; Mode bits (Flags):
        ;       bit 0 - write permission for non-owner (1)
        ;       bit 1 - read permission for non-owner (2)
        ;       bit 2 - write permission for owner (4)
        ;       bit 3 - read permission for owner (8)
        ;       bit 4 - executable flag (16)
        ;       bit 5 - set user ID on execution flag (32)
        ;       bit 6,7,8,9,10,11 are not used (undefined)
        ;       bit 12 - large file flag (4096)
        ;       bit 13 - file has modified flag (always on) (8192)
        ;       bit 14 - directory flag (16384)
        ;       bit 15 - 'i-node is allocated' flag (32768)

        ; / name; mode
        call    isown
                ;jsr r0,isown / get the i-node and check user status
        test    word [i.flgs], 4000h
                ; bit  $40000,i.flgs / directory?
        jz      short syschmod1
                ; beq 2f / no
        ; AL = (new) mode
        and     al, 0CFh ; 11001111b (clears bit 4 & 5)
                ; bic $60,r2 / su & ex / yes, clear set user id and
                           ; / executable modes
```

```
syschmod1: ; 2:
        mov     [i.flgs], al
                ; movb r2,i.flgs / move remaining mode to i.flgs
        jmp     short isown1
                ; br 1f

isown:
        ; 22/06/2015 (Retro UNIX 386 v1 - Beginning)
        ; 04/05/2013 - 07/07/2013 (Retro UNIX 8086 v1)
        ;
        ; 'isown' is given a file name (the 1st argument).
        ;  It find the i-number of that file via 'namei'
        ;  then gets the i-node into core via 'iget'.
        ;  It then tests to see if the user is super user.
        ;  If not, it cheks to see if the user is owner of
        ;  the file. If he is not an error occurs.
        ;  If user is the owner 'setimod' is called to indicate
        ;  the inode has been modificed and the 2nd argument of
        ;  the call is put in r2.
        ;
        ; INPUTS ->
        ;    arguments of syschmod and syschown calls
        ; OUTPUTS ->
        ;    u.uid - id of user
        ;    imod - set to a 1
        ;    r2 - contains second argument of the system call

        ;
        ;   ((AX=R2) output as 2nd argument)
        ;
         ; ((Modified registers: eAX, eDX, eBX, eCX, eSI, eDI, eBP))
        ;
                ; jsr r0,arg2 / u.namep points to file name
        ;; ! 2nd argument on top of stack !
        ;; 22/06/2015 - 32 bit modifications
        ;; 07/07/2013
        mov     [u.namep], ebx ;; 1st argument
        push    ecx ;; 2nd argument
        ;;
        call    namei
                ; jsr r0,namei / get its i-number
       ; Retro UNIX 8086 v1 modification !
       ; ax = 0 -> file not found
        ;and    ax, ax
        ;jz     error
        ;jc     error ; 27/05/2013
                ; br error3
        jnc     short isown0
        ; 'file not found !' error
        mov     dword [u.error], ERR_FILE_NOT_FOUND ; 12
        jmp     error
isown0:
        call    iget
                ; jsr r0,iget / get i-node into core
        mov     al, [u.uid] ; 02/08/2013
        or      al, al
                ; tstb u.uid / super user?
        jz      short isown1
                ; beq 1f / yes, branch
        cmp     al, [i.uid]
                ; cmpb i.uid,u.uid / no, is this the owner of
                                ; / the file
        ;jne    error
                ; beq 1f / yes
                ; jmp error3 / no, error
        je      short isown1

        mov     dword [u.error], ERR_NOT_OWNER  ; 11
                       ;  'permission denied !' error
        jmp     error
isown1: ; 1:
        call    setimod
                ; jsr r0,setimod / indicates
                ;              ; / i-node has been modified
        pop     eax ; 2nd argument
                ; mov (sp)+,r2 / mode is put in r2
                       ; / (u.off put on stack with 2nd arg)
        retn
                ; rts r0
```

```
;;arg:  ; < get system call arguments >
        ; 'arg' extracts an argument for a routine whose call is
        ; of form:
        ;       sys 'routine' ; arg1
        ;               or
        ;       sys 'routine' ; arg1 ; arg2
        ;               or
        ;       sys 'routine' ; arg1;...;arg10 (sys exec)
        ;
        ; INPUTS ->
        ;    u.sp+18 - contains a pointer to one of arg1..argn
        ;       This pointers's value is actually the value of
        ;       update pc at the the trap to sysent (unkni) is
        ;       made to process the sys instruction
        ;    r0 - contains the return address for the routine
        ;       that called arg. The data in the word pointer
        ;       to by the return address is used as address
        ;       in which the extracted argument is stored
        ;
        ; OUTPUTS ->
        ;    'address' - contains the extracted argument
        ;    u.sp+18 - is incremented by 2
        ;    r1 - contains the extracted argument
        ;    r0 - points to the next instruction to be
        ;        executed in the calling routine.
        ;

        ; mov u.sp,r1
        ; mov *18.(r1),*(r0)+ / put argument of system call
                       ; / into argument of arg2
        ; add $2,18.(r1) / point pc on stack
                             ; / to next system argument
        ; rts r0

;;arg2: ; < get system calls arguments - with file name pointer>
        ; 'arg2' takes first argument in system call
        ;  (pointer to name of the file) and puts it in location
        ;  u.namep; takes second argument and puts it in u.off
        ;  and on top of the stack
        ;
        ; INPUTS ->
        ;    u.sp, r0
        ;
        ; OUTPUTS ->
        ;    u.namep
        ;    u.off
        ;    u.off pushed on stack
        ;    r1
        ;

        ; jsr   r0,arg; u.namep / u.namep contains value of
                              ; / first arg in sys call
        ; jsr r0,arg; u.off / u.off contains value of
                              ; / second arg in sys call
        ; mov r0,r1 / r0 points to calling routine
        ; mov (sp),r0 / put operation code back in r0
        ; mov u.off,(sp) / put pointer to second argument
                       ; / on stack
        ; jmp (r1) / return to calling routine

syschown: ; < change owner of file >
        ; 23/06/2015 (Retro UNIX 386 v1 - Beginning)
        ; 20/06/2013 - 02/08/2013 (Retro UNIX 8086 v1)
        ;
        ; 'syschown' changes the owner of the file whose name is given
        ; as null terminated string pointed to by 'name' has it's owner
        ; changed to 'owner'
        ;
        ; Calling sequence:
        ;       syschown; name; owner
        ; Arguments:
        ;       name - address of the file name
        ;              terminated by null byte.
        ;       owner - (new) owner (number/ID)
        ;
        ; Inputs: -
        ; Outputs: -
        ; ...............................................................
```

```
        ; Retro UNIX 8086 v1 modification:
        ;        'syschown' system call has two arguments; so,
        ;       * 1st argument, name is pointed to by BX register
        ;       * 2nd argument, owner number is in CX register
        ;
        ; / name; owner
        call    isown
                ; jsr r0,isown / get the i-node and check user status
        cmp     byte [u.uid], 0 ; 02/08/2013
                ; tstb u.uid / super user
        jz      short syschown1
                ; beq 2f / yes, 2f
         test     byte [i.flgs], 20h ; 32
                ; bit $40,i.flgs / no, set userid on execution?
        ;jnz    error
                ; bne 3f / yes error, could create Trojan Horses
        jz      short syschown1
        ; 'permission denied !'
        mov     dword [u.error], ERR_FILE_ACCESS  ; 11
        jmp     error
syschown1: ; 2:
        ; AL = owner (number/ID)
        mov     [i.uid], al ; 23/06/2015
                ;  movb r2,i.uid / no, put the new owners id
                              ; / in the i-node
        jmp     sysret
        ; 1:
                ; jmp sysret4
        ; 3:
                ; jmp   error

systime: ; / get time of year
        ; 23/06/2015 (Retro UNIX 386 v1 - Beginning)
        ; 20/06/2013 (Retro UNIX 8086 v1)
        ;
        ; 20/06/2013
        ; 'systime' gets the time of the year.
        ; The present time is put on the stack.
        ;
        ; Calling sequence:
        ;       systime
        ; Arguments: -
        ;
        ; Inputs: -
        ; Outputs: sp+2, sp+4 - present time
        ; ...............................................................
        ;
        ; Retro UNIX 8086 v1 modification:
        ;        'systime' system call will return to the user
        ;       with unix time (epoch) in DX:AX register pair
        ;
        ;       !! Major modification on original Unix v1 'systime'
        ;       system call for PC compatibility !!

        call    epoch
        mov     [u.r0], eax
                ; mov s.time,4(sp)
                ; mov s.time+2,2(sp) / put the present time
                                 ; / on the stack
                ; br sysret4
        jmp     sysret

sysstime: ; / set time
        ; 23/06/2015 (Retro UNIX 386 v1 - Beginning)
        ; 20/06/2013 - 02/08/2013 (Retro UNIX 8086 v1)
        ;
        ; 'sysstime' sets the time. Only super user can use this call.
        ;
        ; Calling sequence:
        ;       sysstime
        ; Arguments: -
        ;
        ; Inputs: sp+2, sp+4 - time system is to be set to.
        ; Outputs: -
        ; ...............................................................
        ; Retro UNIX 8086 v1 modification:
        ;       the user calls 'sysstime' with unix (epoch) time
        ;       (to be set) is in CX:BX register pair as two arguments.
        ;
```

```
        ;       Retro UNIX 8086 v1 argument transfer method 2 is used
        ;       to get sysstime system call arguments from the user;
        ;       * 1st argument, lowword of unix time is in BX register
        ;       * 2nd argument, highword of unix time is in CX register
        ;
        ;       !! Major modification on original Unix v1 'sysstime'
        ;       system call for PC compatibility !!

        cmp     byte [u.uid], 0
                ; tstb u.uid / is user the super user
        ;ja     error
                ; bne error4 / no, error
        jna     short systime1
        ; 'permission denied !'
        mov     dword [u.error], ERR_NOT_SUPERUSER  ; 11
        jmp     error
systime1:
        ; 23/06/2015 (Retro UNIX 386 v1 - 32 bit version)
        ; EBX = unix (epoch) time (from user)
        mov     eax, ebx
        call    set_date_time
                ; mov 4(sp),s.time
                ; mov 2(sp),s.time+2 / set the system time
        jmp     sysret
                ; br sysret4

sysbreak:
        ; 18/10/2015
        ; 07/10/2015
        ; 23/06/2015 (Retro UNIX 386 v1 - Beginning)
        ; 20/06/2013 - 24/03/2014 (Retro UNIX 8086 v1)
        ;
        ; 'sysbreak' sets the programs break points.
        ; It checks the current break point (u.break) to see if it is
        ; between "core" and the stack (sp). If it is, it is made an
        ; even address (if it was odd) and the area between u.break
        ; and the stack is cleared. The new breakpoint is then put
        ; in u.break and control is passed to 'sysret'.
        ;
        ; Calling sequence:
        ;       sysbreak; addr
        ; Arguments: -
        ;
        ; Inputs: u.break - current breakpoint
        ; Outputs: u.break - new breakpoint
        ;       area between old u.break and the stack (sp) is cleared.
        ; ...............................................................
        ; Retro UNIX 8086 v1 modification:
        ;       The user/application program puts breakpoint address
        ;        in BX register as 'sysbreak' system call argument.
        ;       (argument transfer method 1)
        ;
        ;  NOTE: Beginning of core is 0 in Retro UNIX 8086 v1 !
        ;       ((!'sysbreak' is not needed in Retro UNIX 8086 v1!))
        ;  NOTE:
        ;       'sysbreak' clears extended part (beyond of previous
        ;       'u.break' address) of user's memory for original unix's
        ;       'bss' compatibility with Retro UNIX 8086 v1 (19/11/2013)

                ; mov u.break,r1 / move users break point to r1
                ; cmp r1,$core / is it the same or lower than core?
                ; blos 1f / yes, 1f
        ; 23/06/2015
        mov     ebp, [u.break] ; virtual address (offset)
        ;and    ebp, ebp
        ;jz     short sysbreak_3
        ; Retro UNIX 386 v1 NOTE: u.break points to virtual address !!!
        ; (Even break point address is not needed for Retro UNIX 386 v1)
        mov     edx, [u.sp] ; kernel stack at the beginning of sys call
        add     edx, 12 ; EIP -4-> CS -4-> EFLAGS -4-> ESP (user)
        ; 07/10/2015
        mov     [u.break], ebx ; virtual address !!!
        ;
        cmp     ebx, [edx] ; compare new break point with
                          ; with top of user's stack (virtual!)
        jnb     short sysbreak_3
                ; cmp r1,sp / is it the same or higher
                         ; / than the stack?
                ; bhis 1f / yes, 1f
```

```
        mov     esi, ebx
        sub     esi, ebp ; new break point - old break point
        jna     short sysbreak_3
        ;push   ebx
sysbreak_1:
        mov     ebx, ebp
        call    get_physical_addr ; get physical address
        jc      tr_addr_nm_err
        ; 18/10/2015
        mov     edi, eax
        sub     eax, eax ; 0
                 ; ECX = remain byte count in page (1-4096)
        cmp     esi, ecx
        jnb     short sysbreak_2
        mov     ecx, esi
sysbreak_2:
        sub     esi, ecx
        add     ebp, ecx
        rep     stosb
        or      esi, esi
        jnz     short sysbreak_1
        ;
                ; bit $1,r1 / is it an odd address
                ; beq 2f / no, its even
                ; clrb (r1)+ / yes, make it even
        ; 2: / clear area between the break point and the stack
                ; cmp r1,sp / is it higher or same than the stack
                ; bhis 1f / yes, quit
                ; clr (r1)+ / clear word
                ; br 2b / go back
        ;pop    ebx
sysbreak_3: ; 1:
        ;mov    [u.break], ebx ; virtual address !!!
                ; jsr r0,arg; u.break / put the "address"
                        ; / in u.break (set new break point)
                ; br sysret4 / br sysret
        jmp     sysret

maknod:
        ; 22/06/2015 (Retro UNIX 386 v1 - Beginning)
        ; 02/05/2013 - 02/08/2013 (Retro UNIX 8086 v1)
        ;
        ; 'maknod' creates an i-node and makes a directory entry
        ; for this i-node in the current directory.
        ;
        ; INPUTS ->
        ;    r1 - contains mode
        ;    ii - current directory's i-number
        ;
        ; OUTPUTS ->
        ;    u.dirbuf - contains i-number of free i-node
        ;    i.flgs - flags in new i-node
        ;    i.uid - filled with u.uid
        ;    i.nlks - 1 is put in the number of links
        ;    i.ctim - creation time
        ;    i.ctim+2 - modification time
        ;    imod - set via call to setimod
        ;
        ; ((AX = R1)) input
        ;
        ; (Retro UNIX Prototype :
        ;       30/10/2012 - 01/03/2013, UNIXCOPY.ASM)
         ; ((Modified registers: eAX, eDX, eBX, eCX, eSI, eDI, eBP))

        ; / r1 contains the mode
        or      ah, 80h  ; 10000000b
                ; bis   $100000,r1 / allocate flag set
        push    ax
                ; mov r1,-(sp) / put mode on stack
        ; 31/07/2013
        mov     ax, [ii] ; move current i-number to AX/r1
                ; mov ii,r1 / move current i-number to r1
        mov     dl, 1 ; owner flag mask
        call    access
                ; jsr r0,access; 1 / get its i-node into core
        push    ax
                ; mov r1,-(sp) / put i-number on stack
        mov     ax, 40
                ; mov $40.,r1 / r1 = 40
```

```
maknod1: ; 1: / scan for a free i-node (next 4 instructions)
        inc     ax
                ; inc r1 / r1 = r1 + 1
        call    imap
                ; jsr r0,imap / get byte address and bit position in
                            ; /inode map in r2 & m
           ; DX (MQ) has a 1 in the calculated bit position
           ; eBX (R2) has byte address of the byte with allocation bit
        ; 22/06/2015 - NOTE for next Retro UNIX version:
        ;             Inode count must be checked here
        ; (Original UNIX v1 did not check inode count here !?)
        test    [ebx], dl
                ; bitb mq,(r2) / is the i-node active
        jnz     short maknod1
                ; bne 1b / yes, try the next one
        or      [ebx], dl
                ; bisb mq,(r2) / no, make it active
                             ; / (put a 1 in the bit map)
        call    iget
                ; jsr r0,iget / get i-node into core
        test    word [i.flgs], 8000h
                ; tst i.flgs / is i-node already allocated
        jnz     short maknod1
                ; blt 1b / yes, look for another one
        mov     [u.dirbuf], ax
                ; mov r1,u.dirbuf / no, put i-number in u.dirbuf
        pop     ax
                ; mov (sp)+,r1 / get current i-number back
        call    iget
                ; jsr r0,iget / get i-node in core
        call    mkdir
                ; jsr r0,mkdir / make a directory entry
                             ; / in current directory
        mov     ax, [u.dirbuf]
                ; mov u.dirbuf,r1 / r1 = new inode number
        call    iget
                ; jsr r0,iget / get it into core
                ; jsr r0,copyz; inode; inode+32. / 0 it out
        mov     ecx, 8
        xor     eax, eax ; 0
        mov     edi, inode
        rep     stosd
        ;
        pop     word [i.flgs]
                ; mov (sp)+,i.flgs / fill flags
        mov     cl, [u.uid] ; 02/08/2013
        mov     [i.uid], cl
                ; movb u.uid,i.uid / user id
        mov      byte [i.nlks], 1
                ; movb $1,i.nlks / 1 link
        ;call   epoch ; Retro UNIX 8086 v1 modification !
        ;mov    eax, [s.time]
        ;mov    [i.ctim], eax
                ; mov s.time,i.ctim / time created
                ; mov s.time+2,i.ctim+2 / time modified
        ; Retro UNIX 8086 v1 modification !
        ; i.ctime=0, i.ctime+2=0 and
         ; 'setimod' will set ctime of file via 'epoch'
        call setimod
                ; jsr r0,setimod / set modified flag
        retn
                ; rts r0 / return

sysseek: ; / moves read write pointer in an fsp entry
        ; 22/06/2015 (Retro UNIX 386 v1 - Beginning)
        ; 07/07/2013 - 05/08/2013 (Retro UNIX 8086 v1)
        ;
        ; 'sysseek' changes the r/w pointer of (3rd word of in an
        ; fsp entry) of an open file whose file descriptor is in u.r0.
        ; The file descriptor refers to a file open for reading or
        ; writing. The read (or write) pointer is set as follows:
        ;       * if 'ptrname' is 0, the pointer is set to offset.
        ;       * if 'ptrname' is 1, the pointer is set to its
        ;         current location plus offset.
        ;       * if 'ptrname' is 2, the pointer is set to the
        ;         size of file plus offset.
        ; The error bit (e-bit) is set for an undefined descriptor.
        ;
```

```
        ; Calling sequence:
        ;       sysseek; offset; ptrname
        ; Arguments:
        ;       offset - number of bytes desired to move
        ;               the r/w pointer
        ;       ptrname - a switch indicated above
        ;
        ; Inputs: r0 - file descriptor
        ; Outputs: -
        ; ...............................................................
        ;
        ; Retro UNIX 8086 v1 modification:
        ;        'sysseek' system call has three arguments; so,
        ;       * 1st argument, file descriptor is in BX (BL) register
        ;       * 2nd argument, offset is in CX register
        ;       * 3rd argument, ptrname/switch is in DX (DL) register
        ;

        call    seektell
        ; AX = u.count
        ; BX = *u.fofp
                ; jsr r0,seektell / get proper value in u.count
                ; add u.base,u.count / add u.base to it
        add     eax, [u.base] ; add offset (u.base) to base
        mov     [ebx], eax
                ; mov u.count,*u.fofp / put result into r/w pointer
        jmp     sysret
                ; br sysret4

systell: ; / get the r/w pointer
        ; 22/06/2015 (Retro UNIX 386 v1 - Beginning)
        ; 07/07/2013 - 05/08/2013 (Retro UNIX 8086 v1)
        ;
        ; Retro UNIX 8086 v1 modification:
        ; ! 'systell' does not work in original UNIX v1,
        ;           it returns with error !
        ; Inputs: r0 - file descriptor
        ; Outputs: r0 - file r/w pointer

        ;xor    ecx, ecx ; 0
        mov     edx, 1 ; 05/08/2013
        ;call   seektell
        call    seektell0 ; 05/08/2013
        ;mov    ebx, [u.fofp]
        mov     eax, [ebx]
        mov     [u.r0], eax
        jmp     sysret

; Original unix v1 'systell' system call:
                ; jsr r0,seektell
                ; br error4

seektell:
        ; 22/06/2015 (Retro UNIX 386 v1 - Beginning)
        ; 07/07/2013 - 05/08/2013 (Retro UNIX 8086 v1)
        ;
        ; 'seektell' puts the arguments from sysseek and systell
        ; call in u.base and u.count. It then gets the i-number of
        ; the file from the file descriptor in u.r0 and by calling
        ; getf. The i-node is brought into core and then u.count
        ; is checked to see it is a 0, 1, or 2.
        ; If it is 0 - u.count stays the same
        ;          1 - u.count = offset (u.fofp)
        ;          2 - u.count = i.size (size of file)
        ;
        ; !! Retro UNIX 8086 v1 modification:
        ;       Argument 1, file descriptor is in BX;
        ;       Argument 2, offset is in CX;
        ;       Argument 3, ptrname/switch is in DX register.
        ;
        ; mov   ax, 3 ; Argument transfer method 3 (three arguments)
        ; call  arg
        ;
        ; ((Return -> ax = base for offset (position= base+offset))
        ;
        mov     [u.base], ecx ; offset
                ; jsr r0,arg; u.base / puts offset in u.base
```

```
seektell0:
        mov     [u.count], edx
                ; jsr r0,arg; u.count / put ptr name in u.count
        ; mov   ax, bx
                ; mov *u.r0,r1 / file descriptor in r1
                             ; / (index in u.fp list)
        ; call getf
                ; jsr r0,getf / u.fofp points to 3rd word in fsp entry
        ; BX = file descriptor (file number)
        call    getf1
        or      ax, ax ; i-number of the file
                ; mov r1,-(sp) / r1 has i-number of file,
                             ; / put it on the stack
        ;jz     error
                ; beq error4 / if i-number is 0, not active so error
        jnz     short seektell1
        mov     dword [u.error], ERR_FILE_NOT_OPEN  ; 'file not open !'
        jmp     error
seektell1:
        ;push   eax
        cmp     ah, 80h
        jb      short seektell2
                ; bgt .+4 / if its positive jump
        neg     ax
                ; neg r1 / if not make it positive
seektell2:
        call    iget
                ; jsr r0,iget / get its i-node into core
         mov     ebx, [u.fofp] ; 05/08/2013
        cmp     byte [u.count], 1
                ; cmp u.count,$1 / is ptr name =1
        ja      short seektell3
                ; blt 2f / no its zero
        je      short seektell_4
                ; beq 1f / yes its 1
        xor     eax, eax
        ;jmp    short seektell_5
        retn
seektell3:
         mov     eax, [i.size]
                 ; mov i.size,u.count /  put number of bytes
                                   ; / in file in u.count
        ;jmp    short seektell_5
                ; br 2f
        retn
seektell_4: ; 1: / ptrname =1
        ;mov    ebx, [u.fofp]
        mov     eax, [ebx]
                ; mov *u.fofp,u.count / put offset in u.count
;seektell_5: ; 2: / ptrname =0
        ;mov    [u.count], eax
        ;pop    eax
                ; mov (sp)+,r1 / i-number on stack  r1
        retn
                ; rts r0

sysintr: ; / set interrupt handling
        ; 22/06/2015 (Retro UNIX 386 v1 - Beginning)
        ; 07/07/2013 (Retro UNIX 8086 v1)
        ;
        ; 'sysintr' sets the interrupt handling value. It puts
        ; argument of its call in u.intr then branches into 'sysquit'
        ; routine. u.tty is checked if to see if a control tty exists.
        ; If one does the interrupt character in the tty buffer is
        ; cleared and 'sysret'is called. If one does not exits
        ; 'sysret' is just called.
        ;
        ; Calling sequence:
        ;       sysintr; arg
        ; Argument:
        ;       arg - if 0, interrupts (ASCII DELETE) are ignored.
        ;           - if 1, intterupts cause their normal result
        ;               i.e force an exit.
        ;           - if arg is a location within the program,
        ;              control is passed to that location when
        ;              an interrupt occurs.
        ; Inputs: -
        ; Outputs: -
```

```
        ; ..............................................................
        ; Retro UNIX 8086 v1 modification:
        ;       'sysintr' system call sets u.intr to value of BX
        ;       then branches into sysquit.
        ;
        mov     [u.intr], bx
                ; jsr r0,arg; u.intr / put the argument in u.intr
                ; br 1f / go into quit routine
        jmp     sysret

sysquit:
        ; 22/06/2015 (Retro UNIX 386 v1 - Beginning)
        ; 07/07/2013 (Retro UNIX 8086 v1)
        ;
        ; 'sysquit' turns off the quit signal. it puts the argument of
        ; the call in u.quit. u.tty is checked if to see if a control
        ; tty exists. If one does the interrupt character in the tty
        ; buffer is cleared and 'sysret'is called. If one does not exits
        ; 'sysret' is just called.
        ;
        ; Calling sequence:
        ;       sysquit; arg
        ; Argument:
        ;       arg - if 0, this call diables quit signals from the
        ;               typewriter (ASCII FS)
        ;           - if 1, quits are re-enabled and cause execution to
        ;               cease and a core image to be produced.
        ;                i.e force an exit.
        ;           - if arg is an addres in the program,
        ;               a quit causes control to sent to that
        ;               location.
        ; Inputs: -
        ; Outputs: -
        ; ..............................................................
        ; Retro UNIX 8086 v1 modification:
        ;       'sysquit' system call sets u.quit to value of BX
        ;       then branches into 'sysret'.
        ;
        mov     [u.quit], bx
        jmp     sysret
                ; jsr r0,arg; u.quit / put argument in u.quit
        ;1:
                ; mov u.ttyp,r1 / move pointer to control tty buffer
                              ; / to r1
                ; beq sysret4 / return to user
                ; clrb 6(r1) / clear the interrupt character
                          ; / in the tty buffer
                ; br sysret4 / return to user

syssetuid: ; / set process id
        ; 22/06/2015 (Retro UNIX 386 v1 - Beginning)
        ; 07/07/2013 - 02/08/2013 (Retro UNIX 8086 v1)
        ;
        ; 'syssetuid' sets the user id (u.uid) of the current process
        ; to the process id in (u.r0). Both the effective user and
        ; u.uid and the real user u.ruid are set to this.
        ; Only the super user can make this call.
        ;
        ; Calling sequence:
        ;       syssetuid
        ; Arguments: -
        ;
        ; Inputs: (u.r0) - contains the process id.
        ; Outputs: -
        ; ..............................................................
        ; Retro UNIX 8086 v1 modification:
        ;       BL contains the (new) user ID of the current process

                ; movb *u.r0,r1 / move process id (number) to r1
        cmp     bl, [u.ruid]
                ; cmpb r1,u.ruid / is it equal to the real user
                              ; / id number
        je      short setuid1
                ; beq 1f / yes
        cmp     byte [u.uid], 0 ; 02/08/2013
                ; tstb u.uid / no, is current user the super user?
        ;ja     error
                ; bne error4 / no, error
        jna     short setuid0
```

```
        mov     dword [u.error], ERR_NOT_SUPERUSER  ; 11
                                ;  'permission denied !' error
        jmp     error
setuid0:
        mov     [u.ruid], bl
setuid1: ; 1:
        mov     [u.uid], bl ; 02/08/2013
                ; movb r1,u.uid / put process id in u.uid
                ; movb r1,u.ruid / put process id in u.ruid
        jmp     sysret
                ; br sysret4 / system return

sysgetuid: ; < get user id >
        ; 22/06/2015 (Retro UNIX 386 v1 - Beginning)
        ; 07/07/2013 (Retro UNIX 8086 v1)
        ;
        ; 'sysgetuid' returns the real user ID of the current process.
        ; The real user ID identifies the person who is logged in,
        ; in contradistinction to the effective user ID, which
        ; determines his access permission at each moment. It is thus
        ; useful to programs which operate using the 'set user ID'
        ; mode, to find out who invoked them.
        ;
        ; Calling sequence:
        ;       syssetuid
        ; Arguments: -
        ;
        ; Inputs: -
        ; Outputs: (u.r0) - contains the real user's id.
        ; ...............................................................
        ; Retro UNIX 8086 v1 modification:
        ;       AL contains the real user ID at return.
        ;
        movzx   eax, byte [u.ruid]
        mov     [u.r0], eax
                ; movb  u.ruid,*u.r0 / move the real user id to (u.r0)
        jmp     sysret
                ; br sysret4 / systerm return, sysret

anyi:
        ; 22/06/2015 (Retro UNIX 386 v1 - Beginning)
        ; 25/04/2013 (Retro UNIX 8086 v1)
        ;
        ; 'anyi' is called if a file deleted while open.
        ; "anyi" checks to see if someone else has opened this file.
        ;
        ; INPUTS ->
        ;    r1 - contains an i-number
        ;    fsp - start of table containing open files
        ;
        ; OUTPUTS ->
        ;    "deleted" flag set in fsp entry of another occurrence of
        ;          this file and r2 points 1st word of this fsp entry.
        ;    if file not found - bit in i-node map is cleared
        ;                      (i-node is freed)
        ;               all blocks related to i-node are freed
        ;              all flags in i-node are cleared
        ; ((AX = R1)) input
        ;
        ;    (Retro UNIX Prototype : 02/12/2012, UNIXCOPY.ASM)
        ;    ((Modified registers: eDX, eCX, eBX, eSI, eDI, eBP))
        ;
                ; / r1 contains an i-number
        mov     ebx, fsp
                ; mov $fsp,r2 / move start of fsp table to r2
anyi_1: ; 1:
        cmp     ax, [ebx]
                ; cmp r1,(r2) / do i-numbers match?
        je      short anyi_3
                ; beq 1f / yes, 1f
        neg     ax
                ; neg r1 / no complement r1
        cmp     ax, [ebx]
                ; cmp r1,(r2) / do they match now?
        je      short anyi_3
                ; beq 1f / yes, transfer
                ; / i-numbers do not match
        add     ebx, 10 ; fsp table size is 10 bytes
                        ; in Retro UNIX 386 v1 (22/06/2015)
```

```
                ; add $8,r2 / no, bump to next entry in fsp table
        cmp     ebx, fsp + (nfiles*10) ; 22/06/2015
                ; cmp r2,$fsp+[nfiles*8]
                               ; / are we at last entry in the table
        jb      short anyi_1
                ; blt 1b / no, check next entries i-number
        ;cmp    ax, 32768
        cmp     ah, 80h ; negative number check
                ; tst r1 / yes, no match
                ; bge .+4
        jb      short anyi_2
        neg     ax
                ; neg r1 / make i-number positive
anyi_2:
        call    imap
                ; jsr r0,imap / get address of allocation bit
                            ; / in the i-map in r2
        ;; DL/DX (MQ) has a 1 in the calculated bit position
         ;; eBX (R2) has address of the byte with allocation bit
        ; not   dx
        not     dl ;; 0 at calculated bit position, other bits are 1
         ;and   [ebx], dx
        and     [ebx], dl
                ; bicb mq,(r2) / clear bit for i-node in the imap
        call    itrunc
                ; jsr r0,itrunc / free all blocks related to i-node
        mov     word [i.flgs], 0
                ; clr i.flgs / clear all flags in the i-node
        retn
                ;rts    r0 / return
anyi_3: ; 1: / i-numbers match
        inc     byte [ebx+9] ; 22/06/2015
                ;incb 7(r2) / increment upper byte of the 4th word
                   ; / in that fsp entry (deleted flag of fsp entry)
        retn
                ; rts r0
```

```
; Retro UNIX 386 v1 Kernel (v0.2) - SYS3.INC
; Last Modification: 15/09/2015
; ----------------------------------------------------------------------------
; Derived from 'Retro UNIX 8086 v1' source code by Erdogan Tan
; (v0.1 - Beginning: 11/07/2012)
;
; Derived 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>
;
; Retro UNIX 8086 v1 - U3.ASM (08/03/2014) //// UNIX v1 -> u3.s
;
; ****************************************************************************

tswitch: ; Retro UNIX 386 v1
tswap:
       ; 01/09/2015
       ; 10/05/2015 (Retro UNIX 386 v1 - Beginning)
       ; 14/04/2013 - 14/02/2014 (Retro UNIX 8086 v1)
       ; time out swap, called when a user times out.
       ; the user is put on the low priority queue.
       ; This is done by making a link from the last user
       ; on the low priority queue to him via a call to 'putlu'.
       ; then he is swapped out.
       ;
       ; Retro UNIX 386 v1 modification ->
       ;       swap (software task switch) is performed by changing
       ;       user's page directory (u.pgdir) instead of segment change
       ;       as in Retro UNIX 8086 v1.
       ;
       ; RETRO UNIX 8086 v1 modification ->
       ;       'swap to disk' is replaced with 'change running segment'
       ;       according to 8086 cpu (x86 real mode) architecture.
       ;       pdp-11 was using 64KB uniform memory while IBM PC
       ;       compatibles was using 1MB segmented memory
       ;       in 8086/8088 times.
       ;
       ; INPUTS ->
       ;    u.uno - users process number
       ;    runq+4 - lowest priority queue
       ; OUTPUTS ->
       ;    r0 - users process number
       ;    r2 - lowest priority queue address
       ;
       ; ((AX = R0, BX = R2)) output
       ; ((Modified registers: EDX, EBX, ECX, ESI, EDI))
       ;
       mov     al, [u.uno]
               ; movb u.uno,r1 / move users process number to r1
               ; mov  $runq+4,r2
                       ; / move lowest priority queue address to r2
        call   putlu
               ; jsr r0,putlu / create link from last user on Q to
                              ; / u.uno's user

switch: ; Retro UNIX 386 v1
swap:
       ; 02/09/2015
       ; 01/09/2015
       ; 31/08/2015
       ; 10/05/2015 (Retro UNIX 386 v1 - Beginning)
       ; 14/04/2013 - 08/03/2014 (Retro UNIX 8086 v1)
       ; 'swap' is routine that controls the swapping of processes
       ; in and out of core.
       ;
       ; Retro UNIX 386 v1 modification ->
       ;       swap (software task switch) is performed by changing
       ;       user's page directory (u.pgdir) instead of segment change
       ;       as in Retro UNIX 8086 v1.
       ;
       ; RETRO UNIX 8086 v1 modification ->
       ;       'swap to disk' is replaced with 'change running segment'
       ;       according to 8086 cpu (x86 real mode) architecture.
       ;       pdp-11 was using 64KB uniform memory while IBM PC
       ;       compatibles was using 1MB segmented memory
       ;       in 8086/8088 times.
       ;
```

```
        ; INPUTS ->
        ;    runq table - contains processes to run.
        ;    p.link - contains next process in line to be run.
        ;    u.uno - process number of process in core
        ;    s.stack - swap stack used as an internal stack for swapping.
        ; OUTPUTS ->
        ;    (original unix v1 -> present process to its disk block)
        ;    (original unix v1 -> new process into core ->
        ;         Retro Unix 8086 v1 -> segment registers changed
        ;         for new process)
        ;    u.quant = 3 (Time quantum for a process)
        ;       ((INT 1Ch count down speed -> 18.2 times per second)
        ;    RETRO UNIX 8086 v1 will use INT 1Ch (18.2 times per second)
        ;        for now, it will swap the process if there is not
        ;        a keyboard event (keystroke) (Int 15h, function 4Fh)
        ;        or will count down from 3 to 0 even if there is a
        ;         keyboard event locking due to repetitive key strokes.
        ;        u.quant will be reset to 3 for RETRO UNIX 8086 v1.
        ;
        ;    u.pri -points to highest priority run Q.
        ;    r2 - points to the run queue.
        ;    r1 - contains new process number
        ;    r0 - points to place in routine or process that called
        ;         swap all user parameters
        ;
        ; ((Modified registers: EAX, EDX, EBX, ECX, ESI, EDI))
        ;
swap_0:
                ;mov $300,*$ps / processor priority = 6
        mov     esi, runq
                ; mov $runq,r2 / r2 points to runq table
swap_1: ; 1: / search runq table for highest priority process
        mov     ax, [esi]
        and     ax, ax
                ; tst (r2)+ / are there any processes to run
                          ; / in this Q entry
        jnz     short swap_2
                ; bne 1f / yes, process 1f
                ; cmp r2,$runq+6 / if zero compare address
                               ; / to end of table
                ; bne 1b / if not at end, go back
        call    idle
                ; jsr r0,idle; s.idlet+2 / wait for interrupt;
                                      ; / all queues are empty
        jmp     short swap_1
                ; br swap
swap_2: ; 1:
        movzx   ebx, al ; 02/09/2015
                ; tst -(r2) / restore pointer to right Q entry
                ; mov r2,u.pri / set present user to this run queue
                 ; movb (r2)+,r1 / move 1st process in queue to r1
        cmp     al, ah
                ; cmpb r1,(r2)+ / is there only 1 process
                             ; / in this Q to be run
        je      short swap_3
                ; beq 1f / yes
                ; tst -(r2) / no, pt r2 back to this Q entry
        ;movzx  ebx, al
        mov     ah, [ebx+p.link-1]
        mov     [esi], ah
                ; movb p.link-1(r1),(r2) / move next process
                                      ; / in line into run queue
        jmp     short swap_4
                ; br 2f
swap_3: ; 1:
        xor     dx, dx
        mov     [esi], dx
                ; clr -(r2) / zero the entry; no processes on the Q
swap_4: ; / write out core to appropriate disk area and read
      ; / in new process if required
                ; clr *$ps / clear processor status
        mov     ah, [u.uno]
        cmp     ah, al
                ; cmpb r1,u.uno / is this process the same as
                             ; / the process in core?
        je      short swap_8
                ; beq 2f / yes, don't have to swap
                ; mov r0,-(sp) / no, write out core; save r0
                          ; / (address in routine that called swap)
```

```
		; mov r1,-(sp) / put r1 (new process #) on the stack
	; 01/09/2015
	;mov	[u.usp], esp
		; mov sp,u.usp / save stack pointer
		; mov $sstack,sp / move swap stack pointer
			       ; / to the stack pointer
	or	ah, ah
		; tstb u.uno / is the process # = 0
	jz	short swap_6 ; 'sysexit'
		; beq  1f / yes, kill process by overwriting
	; 02/09/2015
	mov	[u.usp], esp ; return  address for 'syswait' & 'sleep'
	;
	call	wswap
		;jsr r0,wswap / write out core to disk
	 ; 31/08/2015
	;movzx	ebx, al ; New (running) process number
	jmp	short swap_7
swap_6:
	; 31/08/2015
	; Deallocate memory pages belong to the process
	; which is being terminated
	; 14/05/2015 ('sysexit')
	; Deallocate memory pages of the process
	; (Retro UNIX 386 v1 modification !)
	;
	; movzx ebx, al
	push	ebx
	mov	eax, [u.pgdir]  ; page directory of the process
	mov	ebx, [u.ppgdir] ; page directory of the parent process
	call	deallocate_page_dir
	mov	eax, [u.upage] ; 'user' structure page of the process
	call	deallocate_page
	pop	ebx
swap_7: ;1:
	; 02/09/2015
	; 31/08/2015
	; 14/05/2015
	shl	bl, 2 ; * 4
	mov	eax, [ebx+p.upage-4] ; the 'u' page of the new process
	;cli
	call	rswap
		; mov (sp)+,r1 / restore r1 to new process number
		; jsr r0,rswap / read new process into core
		; jsr r0,unpack / unpack the users stack from next
			      ; / to his program to its normal
	; 01/09/2015
	;mov	esp, [u.usp]
		; mov u.usp,sp / location; restore stack pointer to
			     ; / new process stack
		; mov (sp)+,r0 / put address of where the process
			     ; / that just got swapped in, left off.,
			     ; / i.e., transfer control to new process
	;sti
swap_8: ;2:
	; RETRO UNIX 8086 v1 modification !
	mov	byte [u.quant], time_count
		; movb	$30.,uquant / initialize process time quantum
	retn
		; rts r0 / return

wswap:  ; < swap out, swap to disk >
	; 09/05/2015 (Retro UNIX 386 v1 - Beginning)
	; 26/05/2013 - 08/03/2014 (Retro UNIX 8086 v1)
	; 'wswap' writes out the process that is in core onto its
	; appropriate disk area.
	;
	; Retro UNIX 386 v1 modification ->
	;	 User (u) structure content and the user's register content
	;	will be copied to the process's/user's UPAGE (a page for
	;	saving 'u' structure and user registers for task switching).
	;	u.usp - points to kernel stack address which contains
	;		user's registers while entering system call.
	;	u.sp  - points to kernel stack address
	;		to return from system call -for IRET-.
	;	[u.usp]+32+16 = [u.sp]
	;	[u.usp] -> edi, esi, ebp, esp (= [u.usp]+32), ebx,
	;		edx, ecx, eax, gs, fs, es, ds, -> [u.sp].
	;
```

```
; Retro UNIX 8086 v1 modification ->
;        'swap to disk' is replaced with 'change running segment'
;       according to 8086 cpu (x86 real mode) architecture.
;       pdp-11 was using 64KB uniform memory while IBM PC
;       compatibles was using 1MB segmented memory
;       in 8086/8088 times.
;
; INPUTS ->
;    u.break - points to end of program
;    u.usp - stack pointer at the moment of swap
;    core - beginning of process program
;    ecore - end of core
;    user - start of user parameter area
;    u.uno - user process number
;    p.dska - holds block number of process
; OUTPUTS ->
;    swp I/O queue
;    p.break - negative word count of process
;    r1 - process disk address
;    r2 - negative word count
;
; RETRO UNIX 8086 v1 input/output:
;
; INPUTS ->
;    u.uno - process number (to be swapped out)
; OUTPUTS ->
;    none
;
;   ((Modified registers: ECX, ESI, EDI))
;
mov     edi, [u.upage] ; process's user (u) structure page addr
mov     ecx, (U_SIZE + 3) / 4
mov     esi, user ; active user (u) structure
rep     movsd
;
mov     esi, [u.usp] ; esp (system stack pointer,
                    ;      points to user registers)
mov     ecx, [u.sp]  ; return address from the system call
                    ; (for IRET)
                    ; [u.sp] -> EIP (user)
                    ; [u.sp+4]-> CS (user)
                    ; [u.sp+8] -> EFLAGS (user)
                    ; [u.sp+12] -> ESP (user)
                    ; [u.sp+16] -> SS (user)
sub     ecx, esi     ; required space for user registers
add     ecx, 20     ; +5 dwords to return from system call
                    ; (for IRET)
shr     ecx, 2
rep     movsd
retn

; Original UNIX v1 'wswap' routine:
; wswap:
        ; mov *$30,u.emt / determines handling of emts
        ; mov *$10,u.ilgins / determines handling of
                       ; / illegal instructions
        ; mov u.break,r2 / put process program break address in r2
        ; inc r2 / add 1 to it
        ; bic $1,r2 / make it even
        ; mov r2,u.break / set break to an even location
        ; mov u.usp,r3 / put users stack pointer
                    ; / at moment of swap in r3
        ; cmp r2,$core / is u.break less than $core
        ; blos 2f / yes
        ; cmp r2,r3 / no, is (u.break) greater than stack ptr.
        ; bhis 2f / yes
; 1:
        ; mov (r3)+,(r2)+ / no, pack stack next to users program
        ; cmp r3,$ecore / has stack reached end of core
        ; bne 1b / no, keep packing
        ; br 1f / yes
; 2:
        ; mov $ecore,r2 / put end of core in r2
; 1:
        ; sub  $user,r2 / get number of bytes to write out
                  ; / (user up to end of stack gets written out)
        ; neg r2 / make it negative
        ; asr r2 / change bytes to words (divide by 2)
        ; mov r2,swp+4 / word count
```

```
        ; movb u.uno,r1 / move user process number to r1
        ; asl r1 / x2 for index
        ; mov r2,p.break-2(r1) / put negative of word count
                             ; / into the p.break table
        ; mov p.dska-2(r1),r1 / move disk address of swap area
                            ; / for process to r1
        ; mov r1,swp+2 / put processes dska address in swp+2
                      ; / (block number)
        ; bis $1000,swp / set it up to write (set bit 9)
        ; jsr r0,ppoke / write process out on swap area of disk
; 1:
        ; tstb swp+1 / is lt done writing?
        ; bne 1b / no, wait
        ; rts r0 / yes, return to swap

rswap:  ; < swap in, swap from disk >
        ; 15/09/2015
        ; 28/08/2015
        ; 14/05/2015
        ; 09/05/2015 (Retro UNIX 386 v1 - Beginning)
        ; 26/05/2013 - 08/03/2014 (Retro UNIX 8086 v1)
        ; 'rswap' reads a process whose number is in r1,
        ; from disk into core.
        ;
        ; Retro UNIX 386 v1 modification ->
        ;       User (u) structure content and the user's register content
        ;       will be restored from process's/user's UPAGE (a page for
        ;       saving 'u' structure and user registers for task switching).
        ;       u.usp - points to kernel stack address which contains
        ;               user's registers while entering system call.
        ;       u.sp  - points to kernel stack address
        ;               to return from system call -for IRET-.
        ;       [u.usp]+32+16 = [u.sp]
        ;       [u.usp] -> edi, esi, ebp, esp (= [u.usp]+32), ebx,
        ;               edx, ecx, eax, gs, fs, es, ds, -> [u.sp].
        ;
        ; RETRO UNIX 8086 v1 modification ->
        ;       'swap to disk' is replaced with 'change running segment'
        ;       according to 8086 cpu (x86 real mode) architecture.
        ;       pdp-11 was using 64KB uniform memory while IBM PC
        ;       compatibles was using 1MB segmented memory
        ;       in 8086/8088 times.
        ;
        ; INPUTS ->
        ;    r1 - process number of process to be read in
        ;    p.break - negative of word count of process
        ;    p.dska - disk address of the process
        ;    u.emt - determines handling of emt's
        ;    u.ilgins - determines handling of illegal instructions
        ; OUTPUTS ->
        ;    8 = (u.ilgins)
        ;    24 = (u.emt)
        ;    swp - bit 10 is set to indicate read
        ;               (bit 15=0 when reading is done)
        ;    swp+2 - disk block address
        ;    swp+4 - negative word count
        ;      ((swp+6 - address of user structure))
        ;
        ; RETRO UNIX 8086 v1 input/output:
        ;
        ; INPUTS ->
        ;    AL - new process number (to be swapped in)
        ; OUTPUTS ->
        ;    none
        ;
        ;   ((Modified registers: EAX, ECX, ESI, EDI, ESP))
        ;
        ; Retro UNIX 386 v1 - modification ! 14/05/2015
        mov     esi, eax  ; process's user (u) structure page addr
        mov     ecx, (U_SIZE + 3) / 4
        mov     edi, user ; active user (u) structure
        rep     movsd
        pop     eax ; 15/09/2015, 'rswap' return address
        mov     edi, [u.usp] ; esp (system stack pointer,
                      ;       points to user registers)
        mov     ecx, [u.sp]  ; return address from the system call
                      ; (for IRET)
                      ; [u.sp] -> EIP (user)
                      ; [u.sp+4]-> CS (user)
```

```
                        ; [u.sp+8] -> EFLAGS (user)
                        ; [u.sp+12] -> ESP (user)
                        ; [u.sp+16] -> SS (user)
        ; 28/08/2015
        sub     ecx, edi     ; required space for user registers
        add     ecx, 20     ; +5 dwords to return from system call
                        ; (for IRET)
        shr     ecx, 2
        rep     movsd
        mov     esp, [u.usp] ; 15/09/2015
        push    eax ; 15/09/2015 'rswap' return address
        retn

        ; Original UNIX v1 'rswap'  and 'unpack' routines:
        ;rswap:
               ; asl r1 / process number x2 for index
               ; mov p.break-2(r1), swp+4 / word count
               ; mov p.dska-2(r1),swp+2 / disk address
               ; bis $2000,swp / read
               ; jsr r0,ppoke / read it in
        ; 1:
               ; tstb swp+1 / done
               ; bne 1b / no, wait for bit 15 to clear (inhibit bit)
               ; mov u.emt,*$30 / yes move these
               ; mov u.ilgins,*$10 / back
               ; rts r0 / return

        ;unpack: ; / move stack back to its normal place
               ; mov u.break,r2 / r2 points to end of user program
               ; cmp r2,$core / at beginning of user program yet?
               ; blos 2f / yes, return
               ; cmp r2,u.usp / is break_above the stack pointer
                        ; / before swapping
               ; bhis 2f / yes, return
               ; mov $ecore,r3 / r3 points to end of core
               ; add r3,r2
               ; sub u.usp,r2 / end of users stack is in r2
        ; 1:
               ; mov -(r2),-(r3) / move stack back to its normal place
               ; cmp r2,u.break / in core
               ; bne 1b
        ; 2:
               ; rts r0

putlu:
        ; 12/09/2015
        ; 02/09/2015
        ; 10/05/2015 (Retro UNIX 386 v1 - Beginning)
        ; 15/04/2013 - 23/02/2014 (Retro UNIX 8086 v1)
        ; 'putlu' is called with a process number in r1 and a pointer
        ; to lowest priority Q (runq+4) in r2. A link is created from
        ; the last process on the queue to process in r1 by putting
        ; the process number in r1 into the last process's link.
        ;
        ; INPUTS ->
        ;    r1 - user process number
        ;    r2 - points to lowest priority queue
        ;    p.dska - disk address of the process
        ;    u.emt - determines handling of emt's
        ;    u.ilgins - determines handling of illegal instructions
        ; OUTPUTS ->
        ;    r3 - process number of last process on the queue upon
        ;        entering putlu
        ;    p.link-1 + r3 - process number in r1
        ;    r2 - points to lowest priority queue
        ;
        ; ((Modified registers: EDX, EBX))
        ;
        ; / r1 = user process no.; r2 points to lowest priority queue

        ; eBX = r2
        ; eAX = r1 (AL=r1b)

        mov     ebx, runq
        movzx   edx, byte [ebx]
        inc     ebx
        and     dl, dl
               ; tstb (r2)+ / is queue empty?
        jz      short putlu_1
```

```
                ; beq 1f / yes, branch
        mov     dl, [ebx] ; 12/09/2015
                ; movb (r2),r3 / no, save the "last user" process number
                             ; / in r3
        mov     [edx+p.link-1], al
                ; movb r1,p.link-1(r3) / put pointer to user on
                             ; / "last users" link
        jmp     short putlu_2
                ; br 2f /
putlu_1: ; 1:
        mov     [ebx-1], al
                ; movb r1,-1(r2) / user is only user;
                            ; / put process no. at beginning and at end
putlu_2: ; 2:
        mov     [ebx], al
                ; movb r1,(r2) / user process in r1 is now the last entry
                             ; / on the queue
        mov     dl, al
        mov      [edx+p.link-1], dh ; 0
                ; dec r2 / restore r2
        retn
                ; rts r0

;copyz:
;       mov      r1,-(sp) / put r1 on stack
;       mov      r2,-(sp) / put r2 on stack
;       mov      (r0)+,r1
;       mov      (r0)+,r2
;1:
;       clr      (r1)+ / clear all locations between r1 and r2
;       cmp      r1,r2
;       blo      1b
;       mov      (sp)+,r2 / restore r2
;       mov      (sp)+,r1 / restore r1
;       rts      r0

idle:
        ; 01/09/2015
        ; 10/05/2015 (Retro UNIX 386 v1 - Beginning)
        ; 10/04/2013 - 23/10/2013 (Retro UNIX 8086 v1)
        ; (idle & wait loop)
        ; Retro Unix 8086 v1 modification on original UNIX v1
        ; idle procedure!
        ;
        ; 01/09/2015
        sti
        ; 29/07/2013
        hlt
        nop ; 10/10/2013
        nop
        nop
        ; 23/10/2013
        nop
        nop
        nop
        nop
        retn

        ;mov *$ps,-(sp) / save ps on stack
        ;clr *$ps / clear ps
        ;mov clockp,-(sp) / save clockp on stack
        ;mov (r0)+,clockp / arg to idle in clockp
        ;1 / wait for interrupt
        ;mov (sp)+,clockp / restore clockp, ps
        ;mov (sp)+,*$ps
        ;rts r0
```

```
clear:
        ; 10/05/2015 (Retro UNIX 386 v1 - Beginning)
        ; 09/04/2013 - 03/08/2013 (Retro UNIX 8086 v1)
        ; 'clear' zero's out of a block (whose block number is in r1)
        ; on the current device (cdev)
        ;
        ; INPUTS ->
        ;    r1 - block number of block to be zeroed
        ;    cdev - current device number
        ; OUTPUTS ->
        ;    a zeroed I/O buffer onto the current device
        ;    r1 - points to last entry in the I/O buffer
        ;
        ; ((AX = R1)) input/output
        ;    (Retro UNIX Prototype : 18/11/2012 - 14/11/2012, UNIXCOPY.ASM)
         ;    ((Modified registers: EDX, ECX, EBX, ESI, EDI, EBP))

        call    wslot
                ; jsr r0,wslot / get an I/O buffer set bits 9 and 15 in first
                    ; / word of I/O queue r5 points to first data word in buffer
        mov     edi, ebx ; r5
        mov     edx, eax
        mov     ecx, 128
                ; mov $256.,r3
        xor     eax, eax
        rep     stosd
        mov     eax, edx
; 1:
                ; clr (r5)+ / zero data word in buffer
                ; dec r3
                ; bgt 1b / branch until all data words in buffer are zero
        call    dskwr
                ; jsr r0,dskwr / write zeroed buffer area out onto physical
                              ; / block specified in r1
        ; eAX (r1) = block number
        retn
                ; rts r0
```

```
; Retro UNIX 386 v1 Kernel (v0.2) - SYS4.INC
; Last Modification: 14/10/2015
; ----------------------------------------------------------------------------
; Derived from 'Retro UNIX 8086 v1' source code by Erdogan Tan
; (v0.1 - Beginning: 11/07/2012)
;
; Derived 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>
;
; Retro UNIX 8086 v1 - U4.ASM (04/07/2014) //// UNIX v1 -> u4.s
;
; ****************************************************************************

;setisp:
       ;mov     r1,-(sp)
       ;mov     r2,-(sp)
       ;mov     r3,-(sp)
       ;mov     clockp,-(sp)
       ;mov     $s.syst+2,clockp
       ;jmp     (r0)

clock: ; / interrupt from 60 cycle clock

        ; 14/10/2015
        ; 14/05/2015 (Retro UNIX 386 v1 - Beginning)
        ; 07/12/2013 - 10/04/2014 (Retro UNIX 8086 v1)

       ;mov     r0,-(sp) / save r0
       ;tst     *$lks / restart clock?
       ;mov     $s.time+2,r0 / increment the time of day
       ;inc     (r0)
       ;bne     1f
       ;inc     -(r0)
;1:
       ;mov     clockp,r0 / increment appropriate time category
       ;inc     (r0)
       ;bne     1f
       ;inc     -(r0)
;1:
        cmp     byte [u.quant], 0
        ja      short clk_1
        ;
         cmp     byte [sysflg], 0FFh ; user or system space ?
        jne     short clk_2 ; system space (sysflg <> 0FFh)
        cmp      byte [u.uno], 1 ; /etc/init ?
        jna     short clk_1 ; yes, do not swap out
        cmp     word [u.intr], 0
        jna     short clk_2
clk_0:
        ; 14/10/2015
        inc     byte [sysflg]  ; Now, we are in system space
        pop     eax ; return address to the timer interrupt
        ;
        MOV     AL,EOI                  ; GET END OF INTERRUPT MASK
        ;CLI                            ; DISABLE INTERRUPTS TILL STACK CLEARED
        OUT     INTA00,AL               ; END OF INTERRUPT TO 8259 - 1
        ;
        jmp      sysrelease ; 'sys release' by clock/timer
clk_1:
        dec     byte [u.quant]
clk_2:
        retn   ; return to (hardware) timer interrupt routine

       ;mov     $uquant,r0 / decrement user time quantum
       ;decb    (r0)
       ;bge     1f / if less than 0
       ;clrb    (r0) / make it 0
;1: / decrement time out counts return now if priority was not 0
       ;cmp     4(sp),$200 / ps greater than or equal to 200
       ;bge     2f / yes, check time outs
       ;tstb    (r0) / no, user timed out?
       ;bne     1f / no
       ;cmpb    sysflg,$-1 / yes, are we outside the system?
       ;bne     1f / no, 1f
       ;mov     (sp)+,r0 / yes, put users r0 in r0
       ;sys     0 / sysrele
       ;rti
```

```
;2: / priority is high so just decrement time out counts
        ;mov     $toutt,r0 / r0 points to beginning of time out table
;2:
        ;tstb    (r0) / is the time out?
        ;beq     3f / yes, 3f (get next entry)
        ;decb    (r0) / no, decrement the time
        ;bne     3f / isit zero now?
        ;incb    (r0) / yes, increment the time
;3:
        ;inc     r0 / next entry
        ;cmp     r0,$touts / end of toutt table?
        ;blo     2b / no, check this entry
        ;mov     (sp)+,r0 / yes, restore r0
        ;rti / return from interrupt
;1: / decrement time out counts; if 0 call subroutine
        ;mov     (sp)+,r0 / restore r0
        ;mov     $240,*$ps / set processor priority to 5
        ;jsr     r0,setisp / save registers
        ;mov     $touts-toutt-1,r0 / set up r0 as index to decrement thru
                                ;  / the table
;1:
        ;tstb    toutt(r0) / is the time out for this entry
        ;beq     2f / yes
        ;decb    toutt(r0) / no, decrement the time
        ;bne     2f / is the time 0, now
        ;asl     r0 / yes, 2 x r0 to get word index for tout entry
        ;jsr     r0,*touts(r0) / go to appropriate routine specified in this
        ;asr     r0 / touts entry; set r0 back to toutt index
;2:
        ;dec     r0 / set up r0 for next entry
        ;bge     1b / finished? , no, go back
        ;br      retisp / yes, restore registers and do a rti

;retisp:
        ;mov     (sp)+,clockp / pop values before interrupt off the stack
        ;mov     (sp)+,r3
        ;mov     (sp)+,r2
        ;mov     (sp)+,r1
        ;mov     (sp)+,r0
        ;rti     / return from interrupt


wakeup: ; / wakeup processes waiting for an event
        ; / by linking them to the queue
        ;
        ; 15/09/2015
        ; 29/06/2015
        ; 15/04/2015 (Retro UNIX 386 v1 - Beginning)
        ;
        ; 15/05/2013 - 02/06/2014
        ; Retro UNIX 8086 v1 modification !
        ; (Process/task switching routine by using
        ; Retro UNIX 8086 v1 keyboard interrupt output.)
        ;
        ; In original UNIX v1, 'wakeup' is called to wake the process
        ; sleeping in the specified wait channel by creating a link
        ; to it from the last user process on the run queue.
        ; If there is no process to wake up, nothing happens.
        ;
        ; In Retro UNIX 8086 v1, Int 09h keyboard interrupt will set
        ; 'switching' status of the current process (owns current tty)
        ; (via alt + function keys) to a process which has highest
        ; priority (on run queue) on the requested tty (0 to 7, except
        ; 8 and 9 which are tty identifiers of COM1, COM2 serial ports)
        ; as it's console tty. (NOTE: 'p.ttyc' is used to set console
        ; tty for tty switching by keyboard.)
        ;
        ; INPUT ->
        ;         AL = wait channel (r3) ('tty number' for now)
        ;         ;;EBX = Run queue (r2) offset
        ;
        ; ((modified registers: EAX, EBX))
        ;
        movzx   ebx, al ; 29/06/2015
        add     ebx, wlist
        mov     al, [ebx] ; waiting list (waiting process number)
        and     al, al
        jz      short wa0 ; nothing to wakeup
        ;
```

```
        xor     ah, ah
        mov     [u.quant], ah ; 0 ; time quantum = 0
        mov     [ebx], ah ; 0 ; zero wait channel entry
        ; 15/09/2015
        movzx   ebx, al
        mov     [ebx+p.waitc-1], ah ; 0
        inc     ah
        mov     byte [ebx+p.stat-1], ah ; 1 ; SRUN
        ;
        push    edi
        push    edx
        call    putlu
        pop     edx
        pop     edi
wa0:
        retn

sleep:
        ; 15/09/2015
        ; 30/06/2015 (Retro UNIX 386 v1 - Beginning)
        ;
        ; 09/05/2013 - 20/03/2014
        ;
        ; Retro UNIX 8086 v1 modification !
        ; (Process/task switching and quit routine by using
        ; Retro UNIX 8086 v1 keyboard interrupt output.))
        ;
        ; In original UNIX v1, 'sleep' is called to wait for
        ; tty and tape output or input becomes available
        ; and process is put on waiting channel and swapped out,
        ; then -when the tty or tape is ready to write or read-
        ; 'wakeup' gets process back to active swapped-in status.)
        ;
        ; In Retro UNIX 8086 v1, Int 1Bh ctrl+brk interrupt and
        ; Int 09h keyboard interrupt will set 'quit' or 'switching'
        ; status of the current process also INT 1Ch will count down
        ; 'uquant' value and INT 09h will redirect scancode of keystroke
        ; to tty buffer of the current process and kernel will get
        ; user input by using tty buffer of the current process
        ; (instead of standard INT 16h interrupt).
        ; TTY output will be redirected to related video page of text mode
        ; (INT 10h will be called with different video page depending
        ; on tty assignment of the active process: 0 to 7 for
        ; pseudo screens.)
        ;
        ; In Retro UNIX 8086 v1, 'sleep' will be called to wait for
        ; a keystroke from keyboard or wait for reading or writing
        ; characters/data on serial port(s).
        ;
        ; Character/Terminal input/output through COM1 and COM2 will be
        ; performed by related routines in addition to pseudo TTY routines.
        ;
        ; R1 = AH = wait channel (0-9 for TTYs) ; 05/10/2013 (22/09/2013)
        ;
        ;; 05/10/2013
        ;10/12/2013
        ;cmp    byte [u.uno], 1
        ;ja     short sleep0
        ;retn

        ; 20/03/2014
        ;mov    bx, [runq]
        ;cmp    bl, bh
        ;jne    short sleep0
        ; 25/02/2014
        ;cmp word ptr [runq], 0
        ;ja short sleep0
        ;retn
sleep0:
        ;
        call    isintr
        jnz     sysret
                ; / wait for event
                ; jsr r0,isintr / check to see if interrupt
                              ; / or quit from user
                               ; br 2f / something happened
                              ; / yes, his interrupt so return
                              ;/ to user
```

```
        ; 30/06/2015
        movzx   ebx, ah ; 30/06/2015
        add     ebx, wlist
        mov     al, [ebx]
        and     al, al
        jz      short sleep1
        push    ebx
        call    putlu
        pop     ebx
sleep1:
        mov     al, [u.uno]
        mov     [ebx], al       ; put the process number
                                ; in the wait channel
                ; mov (r0)+,r1 / put number of wait channel in r1
                ; movb wlist(r1),-(sp) / put old process number in there,
                                    ; / on the stack
                ; movb u.uno,wlist(r1) / put process number of process
                                    ; / to put to sleep in there
         ; 15/09/2015
        movzx   ebx, al
         mov     byte [ebx+p.stat-1], 4 ; SSLEEP
        inc     ah
        mov     [ebx+p.waitc-1], ah ; wait channel + 1
        ;
        push     word [cdev]
                ; mov cdev,-(sp) / nothing happened in isintr so
        call    swap
                ; jsr r0,swap / swap out process that needs to sleep
         pop     word [cdev]
                ; mov (sp)+,cdev / restore device
        call    isintr
        ; 22/09/2013
        jnz     sysret
                ; jsr r0,isintr / check for interrupt of new process
                              ; br 2f / yes, return to new user
                ; movb (sp)+,r1 / no, r1 = old process number that was
                              ; / originally on the wait channel
                ; beq 1f / if 0 branch
                ; mov $runq+4,r2 / r2 points to lowest priority queue
                ; mov $300,*$ps / processor priority = 6
                ; jsr r0,putlu / create link to old process number
                ; clr *$ps / clear the status; process priority = 0
     ;1:
        retn
                ; rts r0 / return
     ;2:
         ;;jmp  sysret
                ; jmp sysret / return to user

isintr:
        ; 30/06/2015 (Retro UNIX 386 v1 - Beginning)
        ;
        ; 09/05/2013 - 30/05/2014
        ;
        ; Retro UNIX 8086 v1 modification !
        ; (Process/task switching and quit routine by using
        ; Retro UNIX 8086 v1 keyboard interrupt output.))
        ;
        ; Retro UNIX 8086 v1 modification:
        ; 'isintr' checks if user interrupt request is enabled
        ;  and there is a 'quit' request by user;
        ;  otherwise, 'isintr' will return with zf=1 that means
        ;  "nothing to do". (20/10/2013)
        ;
        ; 20/10/2013
        cmp     word [u.ttyp], 0 ; has process got a tty ?
        jna     short isintr2 ; retn
        ; 03/09/2013
        ; (nothing to do)
        ;retn
        ; 22/09/2013
        cmp     word [u.intr], 0
        jna     short isintr2 ; retn
        ; 30/05/2014
        push    ax
        mov     ax, [u.quit]
        or      ax, ax ; 0 ?
        jz      short isintr1 ; zf = 1
```

```
        cmp     ax, 0FFFEh  ; 'ctrl + brk' check
        ja      short isintr1 ; 0FFFFh, zf = 0
        xor     ax, ax ; zf = 1
isintr1:
        pop     ax
isintr2: ; 22/09/2013
        ; zf=1 -> nothing to do
        retn

        ; UNIX v1 original 'isintr' routine...
        ;mov     r1,-(sp) / put number of wait channel on the stack
        ;mov     r2,-(sp) / save r2
        ;mov     u.ttyp,r1 / r1 = pointer to buffer of process control
         ;                 / typewriter
        ;beq     1f / if 0, do nothing except skip return
        ;movb    6(r1),r1 / put interrupt char in the tty buffer in r1
        ;beq     1f / if its 0 do nothing except skip return
        ;cmp     r1,$177 / is interrupt char = delete?
        ;bne     3f / no, so it must be a quit (fs)
        ;tst     u.intr / yes, value of u.intr determines handling
         ;            / of interrupts
        ;bne     2f / if not 0, 2f. If zero do nothing.
      ;1:
        ;tst     (r0)+ / bump r0 past system return (skip)
      ;4:
        ;mov     (sp)+,r2 / restore r1 and r2
        ;mov     (sp)+,r1
        ;rts     r0
      ;3: / interrupt char = quit (fs)
        ;tst     u.quit / value of u.quit determines handling of quits
        ;beq     1b / u.quit = 0 means do nothing
      ;2: / get here because either u.intr <> 0 or u.qult <> O
        ;mov     $tty+6,r1 / move pointer to tty block into r1
      ;1: / find process control tty entry in tty block
        ;cmp     (r1),u.ttyp / is this the process control tty buffer?
        ;beq     1f / block found go to 1f
        ;add     $8,r1 / look at next tty block
        ;cmp     r1,$tty+[ntty*8]+6 / are we at end of tty blocks
        ;blo     1b / no
        ;br      4b / no process control tty found so go to 4b
      ;1:
        ;mov     $240,*$ps / set processor priority to 5
        ;movb    -3(r1),0f / load getc call argument; character llst
         ;                 / identifier
        ;inc     0f / increment
      ;1:
        ;jsr     r0,getc; 0:.. / erase output char list for control
         ;        br 4b / process tty. This prevents a line of stuff
         ;             / being typed out after you hit the interrupt
         ;             / key
        ;br      1b
```

```
; Retro UNIX 386 v1 Kernel (v0.2) - SYS5.INC
; Last Modification: 14/11/2015
; ----------------------------------------------------------------------------
; Derived from 'Retro UNIX 8086 v1' source code by Erdogan Tan
; (v0.1 - Beginning: 11/07/2012)
;
; Derived 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>
;
; Retro UNIX 8086 v1 - U5.ASM (07/08/2013) //// UNIX v1 -> u5.s
; ****************************************************************************

mget:
        ; 03/06/2015 (Retro UNIX 386 v1 - Beginning)
        ; 22/03/2013 - 31/07/2013 (Retro UNIX 8086 v1)
        ;
        ; Get existing or (allocate) a new disk block for file
        ;
        ; INPUTS ->
        ;    u.fofp (file offset pointer)
        ;    inode
        ;    u.off (file offset)
        ; OUTPUTS ->
        ;    r1 (physical block number)
        ;    r2, r3, r5 (internal)
        ;
        ; ((AX = R1)) output
        ;    (Retro UNIX Prototype : 05/03/2013 - 14/11/2012, UNIXCOPY.ASM)
        ;    ((Modified registers: eDX, eBX, eCX, eSI, eDI, eBP))

                ; mov *u.fofp,mq / file offset in mq
                ; clr ac / later to be high sig
                ; mov $-8,lsh   / divide ac/mq by 256.
                ; mov mq,r2
                ; bit $10000,i.flgs / lg/sm is this a large or small file
                ; bne 4f / branch for large file
mget_0:
        mov     esi, [u.fofp]
        movzx   ebx, byte [esi+1]
        ; BX = r2
        test   word [i.flgs], 4096 ; 1000h
                                   ; is this a large or small file
        jnz     short mget_5 ; 4f ; large file

        test    bl, 0F0h ; !0Fh
                ; bit $!17,r2
        jnz     short mget_2
                ; bne 3f / branch if r2 greater than or equal to 16
        and     bl, 0Eh
                ; bic $!16,r2 / clear all bits but bits 1,2,3
        movzx   eax, word [ebx+i.dskp] ; AX = R1, physical block number
                ; mov i.dskp(r2),r1 / r1 has physical block number
        or      ax, ax
        jnz     short mget_1
                ; bne 2f / if physical block num is zero then need a new block
                        ; / for file
        call    alloc
                ; jsr r0,alloc / allocate a new block
          ; eAX (r1) = Physical block number
        mov     [ebx+i.dskp], ax
                ; mov r1,i.dskp(r2) / physical block number stored in i-node
        call    setimod
                ; jsr r0,setimod / set inode modified byte (imod)
        call    clear
                ; jsr r0,clear / zero out disk/drum block just allocated
mget_1: ; 2:
        ; eAX (r1) = Physical block number
        retn
                ; rts r0
mget_2: ; 3: / adding on block which changes small file to a large file
        call    alloc
                ; jsr r0,alloc / allocate a new block for this file;
                              ; / block number in r1
        ; eAX (r1) = Physical block number
        call    wslot
                ; jsr r0,wslot / set up I/O buffer for write, r5 points to
                            ; / first data word in buffer
```

```
         ; eAX (r1) = Physical block number
        mov     ecx, 8  ; R3, transfer old physical block pointers
                   ; into new indirect block area for the new
                   ; large file
        mov     edi, ebx ; r5
        mov     esi, i.dskp
                ; mov $8.,r3 / next 6 instructions transfer old physical
                          ; / block pointers
                ; mov $i.dskp,r2 / into new indirect block for the new
                         ; / large file
        xor     ax, ax ; mov ax, 0
mget_3: ;1:
        movsw
                ; mov (r2),(r5)+
        mov     [esi-2], ax
                ; clr (r2)+
        loop    mget_3 ; 1b
                ; dec r3
                ; bgt 1b

        mov     cl, 256-8
                ; mov $256.-8.,r3 / clear rest of data buffer
mget_4: ; 1
        rep     stosw
                ; clr (r5)+
                ; dec r3
                ; bgt 1b
        ; 24/03/2013
         ; AX (r1) = Physical block number
        call    dskwr
                ; jsr r0,dskwr / write new indirect block on disk
         ; eAX (r1) = Physical block number
        mov     [i.dskp], ax
                ; mov r1,i.dskp / put pointer to indirect block in i-node
        or      word [i.flgs], 4096 ; 1000h
                ; bis $10000,i.flgs / set large file bit
                                ; / in i.flgs word of i-node
        call    setimod
                ; jsr r0,setimod / set i-node modified flag
         jmp     mget_0
                ; br mget

mget_5:  ; 4 ; large file
                ; mov $-8,lsh / divide byte number by 256.
                ; bic $!776,r2 / zero all bits but 1,2,3,4,5,6,7,8; gives offset
                           ; / in indirect block
                ; mov r2,-(sp) / save on stack (*)
                ; mov mq,r2 / calculate offset in i-node for pointer to proper
                          ; / indirect block
                ; bic $!16,r2
         and     bl, 0FEh ; bh = 0
         push    ebx  ; i-node pointer offset in indirect block  (*)
         ; 01/03/2013 Max. possible BX (offset) value is 127 (65535/512)
        ;            for this file system (offset 128 to 255 not in use)
        ; There is always 1 indirect block for this file system
        movzx   eax, word [i.dskp] ; i.dskp[0]
                ; mov i.dskp(r2),r1
        or      ax, ax ; R1
        jnz     short mget_6 ; 2f
                ; bne 2f / if no indirect block exists
        call    alloc
                ; jsr r0,alloc / allocate a new block
        mov     [i.dskp], ax  ; 03/03/2013
                ; mov r1,i.dskp(r2) / put block number of new block in i-node
        call    setimod
                ; jsr r0,setimod / set i-node modified byte
        ; eAX = new block number
        call    clear
                ; jsr r0,clear / clear new block
mget_6: ;2
        ; 05/03/2013
        ; eAX = r1, physical block number (of indirect block)
        call    dskrd ; read indirect block
                ; jsr r0,dskrd / read in indirect block
        pop     edx  ; R2, get offset (*)
                ; mov (sp)+,r2 / get offset
        ; eAX = r1, physical block number (of indirect block)
        push    eax ; ** ; 24/03/2013
                ; mov r1,-(sp) / save block number of indirect block on stack
```

```
        ; eBX (r5) = pointer to buffer (indirect block)
        add     ebx, edx ; / r5 points to first word in indirect block, r2
                ; add r5,r2 / r5 points to first word in indirect block, r2
                           ; / points to location of inter
        movzx   eax, word [ebx] ; put physical block no of block
                              ; in file sought in R1 (AX)
                ; mov (r2),r1 / put physical block no of block in file
                                   ; / sought in r1
        or      ax, ax
         jnz    short mget_7 ; 2f
                ; bne 2f / if no block exists
        call    alloc
                ; jsr r0,alloc / allocate a new block
        mov     [ebx], ax ; R1
                ; mov r1,(r2) / put new block number into proper location in
                             ; / indirect block
        pop     edx ; ** ; 24/03/2013
                ; mov (sp)+,r1 / get block number of indirect block
        push    edx ; ** ; 31/07/2013
        push    eax ; * ; 24/03/2013, 31/07/2013 (new block number)
        mov     eax, edx ; 24/03/2013
                ; mov (r2),-(sp) / save block number of new block
        ; eAX (r1) = physical block number (of indirect block)
        call    wslot
                ; jsr r0,wslot
         ; eAX (r1) = physical block number
        ; eBX (r5) = pointer to buffer (indirect block)
        call    dskwr
        ; eAX = r1 = physical block number (of indirect block)
                ; jsr r0,dskwr / write newly modified indirect block
                             ; / back out on disk
        pop     eax ; *  ; 31/07/2013
                ; mov (sp),r1 / restore block number of new block
        ; eAX (r1) = physical block number of new block
        call    clear
                ; jsr r0,clear / clear new block
mget_7: ; 2
        pop     edx ; **
                ; tst (sp)+ / bump stack pointer
        ; eAX (r1) = Block number of new block
        retn
                ; rts r0

alloc:
        ; 03/06/2015 (Retro UNIX 386 v1 - Beginning)
        ; 01/04/2013 - 01/08/2013 (Retro UNIX 8086 v1)
        ;
        ; get a free block and
        ; set the corresponding bit in the free storage map
        ;
        ; INPUTS ->
        ;    cdev (current device)
        ;    r2
        ;    r3
        ; OUTPUTS ->
        ;    r1 (physical block number of block assigned)
        ;    smod, mmod, systm (super block), mount (mountable super block)
        ;
        ; ((AX = R1)) output
        ;    (Retro UNIX Prototype : 14/11/2012 - 21/07/2012, UNIXCOPY.ASM)
         ;    ((Modified registers: DX, CX))

                ;mov r2,-(sp) / save r2, r3 on stack
                ;mov r3,-(sp)
        ;push   ecx
        push    ebx ; R2
        ;push   edx ; R3
        mov     ebx, systm ; SuperBlock
                ; mov $systm,r2 / start of inode and free storage map for drum
        cmp     byte [cdev], 0
                ; tst cdev
        jna     short alloc_1
                ; beq 1f / drum is device
        mov     ebx, mount
                ; mov $mount,r2 / disk or tape is device, start of inode and
                              ; / free storage map
```

```
alloc_1: ; 1
        mov     cx, [ebx]
                ; mov (r2)+,r1 / first word contains number of bytes in free
                            ; / storage map
        shl     cx, 3
                ; asl r1 / multiply r1 by eight gives
                ; number of blocks in device
                ; asl r1
                ; asl r1
        ;; push cx ;; 01/08/2013
                ; mov r1,-(sp) / save # of blocks in device on stack
        xor     eax, eax ; 0
                ; clr r1 / r1 contains bit count of free storage map
alloc_2: ; 1
        inc     ebx ; 18/8/2012
        inc     ebx ;
        mov     dx, [ebx]
                ; mov (r2)+,r3 / word of free storage map in r3
        or      dx, dx
        jnz     short alloc_3 ; 1f
                ; bne 1f / branch if any free blocks in this word
        add     ax, 16
                ; add $16.,r1
        cmp     ax, cx
                ; cmp r1 ,(sp) / have we examined all free storage bytes
        jb      short alloc_2
                ; blo 1b
        ; 14/11/2015
        ; Note: If the super block buffer has wrong content (zero bytes)
        ;       because of a (DMA or another) r/w error,
        ;       we will be here, at 'jmp panic' code address,
        ;       even if the (disk) file system space is not full !!!
        ;       (cx = 0)
        ;
        jmp      panic
                ; jmp panic / found no free storage
alloc_3: ; 1
        shr     dx, 1
                ; asr r3 / find a free block
        jc      short alloc_4 ; 1f
                ; bcs 1f / branch when free block found; bit for block k
                       ; / is in byte k/8 / in bit k (mod 8)
        inc     ax
                ; inc r1 / increment bit count in bit k (mod8)
        jmp     short alloc_3
                ; br 1b
alloc_4: ; 1:
        ;; pop cx ;; 01/08/2013
                ; tst (sp)+ / bump sp
        ; 02/04/2013
        call    free3
                ; jsr r0,3f / have found a free block
        ; 21/8/2012
        not     dx ; masking bit is '0' and others are '1'
        and     [ebx], dx   ;; 0 -> allocated
                ; bic r3,(r2) / set bit for this block
                            ; / i.e. assign block
                ; br 2f
        jmp     short alloc_5

free:
        ; 03/06/2015 (Retro UNIX 386 v1 - Beginning)
        ; 07/04/2013 - 01/08/2013 (Retro UNIX 8086 v1)
        ;
        ; calculates byte address and bit position for given block number
        ; then sets the corresponding bit in the free storage map
        ;
        ; INPUTS ->
        ;    r1 - block number for a block structured device
        ;    cdev - current device
        ; OUTPUTS ->
        ;    free storage map is updated
        ;    smod is incremented if cdev is root device (fixed disk)
        ;    mmod is incremented if cdev is a removable disk
        ;
        ;  (Retro UNIX Prototype : 01/12/2012, UNIXCOPY.ASM)
        ;  ((Modified registers: DX, CX))

                ;mov r2,-(sp) / save r2, r3
```

```
                ;mov r3,-(sp)
        ;push   ecx
        push    ebx ; R2
        ;push   edx ; R3

         call    free3
                ; jsr r0,3f  / set up bit mask and word no.
                                ; / in free storage map for block
        or      [ebx], dx
                ; bis r3, (r2) / set free storage block bit;
                           ;  / indicates free block
        ; 0 -> allocated, 1 -> free

alloc_5:
        ; 07/04/2013
free_1: ; 2:
        ; pop   edx
                ; mov (sp)+,r3 / restore r2, r3
        pop     ebx
                ; mov (sp)+,r2
        ; pop   ecx
        cmp     byte [cdev], 0
                ; tst cdev / cdev = 0, block structured, drum;
                        ; / cdev = 1, mountable device
        ja      short alloc_6 ; 1f
                ; bne 1f
        ;mov    byte [smod], 1
        inc     byte [smod]
                ; incb smod / set super block modified for drum
        ; eAX (r1) = block number
        retn
                ; rts r0
free_2:
alloc_6: ; 1:
        ;mov    byte [mmod], 1
        inc     byte [mmod]
                ; incb  mmod
                  ; / set super block modified for mountable device
        ; eAX (r1) = block number
        retn
                ; rts r0
free3:
        ; 03/06/2015 (Retro UNIX 386 v1 - Beginning)
        ; 02/04/2013 - 01/08/2013 (Retro UNIX 8086 v1)
        ;
        ; free3 is called from 'alloc' and 'free' procedures
        ;
alloc_free_3: ; 3
        mov     dx, 1
        mov     cl, al
                ; mov r1,r2 / block number, k, = 1
        and     cl, 0Fh  ; 0Fh <-- (k) mod 16
                ; bic $!7,r2 / clear all bits but 0,1,2; r2 = (k) mod (8)
        jz      short free4
                ; bisb 2f(r2),r3 / use mask to set bit in r3 corresponding to
                              ; / (k) mod 8
        shl     dx, cl
free4:
        movzx   ebx, ax
                ; mov r1,r2 / divide block number by 16
        shr     bx, 4
                ; asr r2
                ; asr r2
                ; asr r2
                ; asr r2
                ; bcc 1f / branch if bit 3 in r1 was 0 i.e.,
                       ; / bit for block is in lower half of word
                ; swab r3 / swap bytes in r3; bit in upper half of word in free
                        ; / storage map
alloc_free_4: ; 1
        shl     bx, 1
                ; asl r2 / multiply block number by 2; r2 = k/8
        add     ebx, systm+2 ; SuperBlock+2
                ; add $systm+2,r2 / address of word of free storage map for drum
                               ; / with block bit in it
        cmp     byte [cdev], 0
                ; tst cdev
        jna     short alloc_free_5
                ; beq 1f / cdev = 0 indicates device is drum
```

```
        add     ebx, mount - systm
                ; add $mount-systm,r2 / address of word of free storage map for
                                   ; / mountable device with bit of block to be
                                   ; / freed
alloc_free_5: ; 1
        retn
                ; rts r0 / return to 'free'
              ; 2
                ; .byte       1,2,4,10,20,40,100,200 / masks for bits 0,...,7

iget:
        ; 03/06/2015 (Retro UNIX 386 v1 - Beginning)
        ; 07/04/2013 - 07/08/2013 (Retro UNIX 8086 v1)
        ;
        ; get a new i-node whose i-number in r1 and whose device is in cdev
        ;
        ; ('iget' returns current i-number in r1, if input value of r1 is 0)
        ;
        ; INPUTS ->
        ;    ii - current i-number, rootdir
        ;    cdev - new i-node device
        ;    idev - current i-node device
        ;    imod - current i-node modified flag
        ;    mnti - cross device file i-number
        ;    r1 - i-numbe rof new i-node
        ;    mntd - mountable device number
        ;
        ; OUTPUTS ->
        ;    cdev, idev, imod, ii, r1
        ;
        ; ((AX = R1)) input/output
        ;
        ;  (Retro UNIX Prototype : 14/07/2012 - 18/11/2012, UNIXCOPY.ASM)
         ;  ((Modified registers: eDX, eCX, eBX, eSI, eDI, eBP))

        mov     dl, [cdev] ; 18/07/2013
        mov     dh, [idev] ; 07/08/2013
        ;
        cmp     ax, [ii]
                ; cmp r1,ii / r1 = i-number of current file
        jne     short iget_1
                ; bne 1f
        cmp     dl, dh
                ; cmp idev,cdev
                          ; / is device number of i-node = current device
         je      short iget_5
                ; beq 2f
iget_1: ; 1:
        xor     bl, bl
        cmp     [imod], bl ; 0
                ; tstb imod / has i-node of current file
                          ; / been modified i.e., imod set
        jna     short iget_2
                ; beq 1f
        mov     [imod], bl ; 0
                ;  clrb imod / if it has,
                           ; / we must write the new i-node out on disk
        push    ax
                ; mov r1,-(sp)
        ;mov    dl, [cdev]
        push    dx
                ; mov cdev,-(sp)
        mov     ax, [ii]
                ; mov ii,r1
        ;mov    dh, [idev]
        mov     [cdev], dh
                ; mov idev,cdev
        inc     bl ; 1
        ; 31/07/2013
        mov      [rw], bl ; 1 == write
        ;;28/07/2013 rw -> u.rw
         ;;mov   [u.rw], bl ; 1 == write
        call    icalc
                ; jsr r0,icalc; 1
        pop     dx
        mov     [cdev], dl
                ; mov (sp)+,cdev
        pop     ax
                ; mov (sp)+,r1
```

```
iget_2: ; 1:
        and     ax, ax
                ; tst r1 / is new i-number non zero
        jz      short iget_4 ; 2f
                ; beq 2f / branch if r1=0

        ; mov   dl, [cdev]
        or      dl, dl
                ; tst cdev / is the current device number non zero
                        ; / (i.e., device =/ drum)
        jnz     short iget_3 ;  1f
                ; bne 1f / branch 1f cdev =/ 0  ;; (cdev != 0)
        cmp     ax, [mnti]
                ; cmp r1,mnti / mnti is the i-number of the cross device
                           ; / file (root directory of mounted device)
        jne     short iget_3 ; 1f
                ; bne 1f
         ;mov    bl, [mntd]
        inc     dl ; mov dl, 1 ; 17/07/2013
         mov    [cdev], dl ; 17/07/2013 - 09/07/2013
                ; mov mntd,cdev / make mounted device the current device
        mov     ax, [rootdir]
                ; mov rootdir,r1
iget_3: ; 1:
        mov     [ii], ax
                ; mov r1,ii
        mov     [idev], dl ; cdev
                ; mov cdev,idev
        xor     bl, bl
         ; 31/07/2013
        mov      [rw], bl ; 0 == read
        ;;28/07/2013 rw -> u.rw
         ;;mov    [u.rw], bl ; 0 = read
        call    icalc
                ; jsr r0,icalc; 0 / read in i-node ii
iget_4: ; 2:
        mov     ax, [ii]
                ; mov ii,r1
iget_5:
        retn
                ; rts r0

icalc:
        ; 02/07/2015
        ; 03/06/2015 (Retro UNIX 386 v1 - Beginning)
        ; 07/04/2013 - 31/07/2013 (Retro UNIX 8086 v1)
        ;
        ; calculate physical block number from i-number then
        ; read or write that block
        ;
        ; 'icalc' is called from 'iget'
        ;
        ; for original unix v1:
        ; / i-node i is located in block (i+31.)/16. and begins 32.*
        ; / (i+31.) mod 16. bytes from its start
        ;
        ; for retro unix 8086 v1:
        ;  i-node is located in block (i+47)/16 and
        ;  begins 32*(i+47) mod 16 bytes from its start
        ;
        ; INPUTS ->
        ;    r1 - i-number of i-node
        ;
        ; OUTPUTS ->
        ;    inode r/w
        ;
        ; ((AX = R1)) input
        ;
        ;  (Retro UNIX Prototype : 14/07/2012 - 18/11/2012, UNIXCOPY.ASM)
         ;  ((Modified registers: eAX, eDX, eCX, eBX, eSI, eDI, eBP))
        ;
        movzx   edx, ax
        add     dx, 47
        mov     eax, edx
        ;add    ax, 47  ; add 47 to inode number
                ;  add $31.,r1 / add 31. to i-number
        push    eax
                ; mov r1,-(sp) / save i+31. on stack
        shr     ax, 4
```

```
                ; asr r1 / divide by 16.
                ; asr r1
                ; asr r1
                ; asr r1 / r1 contains block number of block
                       ; / in which i-node exists
        call    dskrd
                ; jsr r0,dskrd / read in block containing i-node i.
        ; 31/07/2013
         cmp     byte [rw], 0 ; Retro Unix 8086 v1 feature !
        ;; 28/07/2013 rw -> u.rw
         ;;cmp     byte [u.rw], 0 ; Retro Unix 8086 v1 feature !
                ; tst (r0)
        jna     short icalc_1
                ; beq 1f / branch to wslot when argument
                       ; / in icalc call = 1
        ; eAX = r1  = block number
        call    wslot
                ; jsr r0,wslot / set up data buffer for write
                            ; / (will be same buffer as dskrd got)
        ; eBX = r5 points to first word in data area for this block
icalc_1: ; 1:
        pop     edx
        and     edx, 0Fh ; (i+47) mod 16
                ; bic $!17,(sp) / zero all but last 4 bits;
                             ; / gives (i+31.) mod 16
        shl     edx, 5
        ; eDX = 32 * ((i+47) mod 16)
        mov     esi, ebx  ; ebx points 1st word of the buffer
        add     esi, edx  ; edx is inode offset in the buffer
                ; eSI (r5) points to first word in i-node i.
                ; mov (sp)+,mq / calculate offset in data buffer;
                            ; / 32.*(i+31.)mod16
                ; mov $5,lsh / for i-node i.
                ; add mq,r5 / r5 points to first word in i-node i.
        mov     edi, inode
                ; mov $inode,r1 / inode is address of first word
                             ; / of current i-node
        mov     ecx, 8 ; 02/07/2015(32 bit modification)
                ; mov $16.,r3
        ; 31/07/2013
        cmp      [rw], ch ; 0  ;; Retro Unix 8086 v1 feature !
        ;;28/07/2013 rw -> u.rw
        ;;cmp     [u.rw], ch ; 0  ;; Retro Unix 8086 v1 feature !
                ; tst (r0)+ / branch to 2f when argument in icalc call = 0
        jna     short icalc_3
                ; beq 2f / r0 now contains proper return address
                       ; / for rts r0
icalc_2: ; 1:
        xchg    esi, edi
        ; overwrite old i-node (in buffer to be written)
        rep     movsd
                ; mov (r1)+,(r5)+ / over write old i-node
                ; dec r3
                ; bgt 1b
        call    dskwr
                ; jsr r0,dskwr / write inode out on device
        retn
                ; rts r0
icalc_3: ; 2:
        ; copy new i-node into inode area of (core) memory
        rep     movsd
                ; mov (r5)+,(r1)+ / read new i-node into
                                ; / "inode" area of core
                ; dec r3
                ; bgt 2b
        retn
                ; rts r0
```

```
access:
        ; 03/06/2015 (Retro UNIX 386 v1 - Beginning)
        ; 24/04/2013 - 29/04/2013 (Retro UNIX 8086 v1)
        ;
        ; check whether user is owner of file or user has read or write
        ; permission (based on i.flgs).
        ;
        ; INPUTS ->
        ;    r1 - i-number of file
        ;    u.uid
        ; arg0 -> (owner flag mask)
        ;     Retro UNIX 8086 v1 feature -> owner flag mask in DL (DX)
        ; OUTPUTS ->
        ;    inode (or jump to error)
        ;
        ; ((AX = R1)) input/output
        ;
         ;  ((Modified registers: eCX, eBX, eDX, eSI, eDI, eBP))
        ;
        push    dx  ; save flags (DL)
        call    iget
                ; jsr r0,iget / read in i-node for current directory
                            ; / (i-number passed in r1)
        mov     cl, [i.flgs]
                ; mov i.flgs,r2
        pop     dx  ; restore flags (DL)
        mov     dh, [u.uid]
        cmp     dh, [i.uid]
                ; cmpb i.uid,u.uid / is user same as owner of file
        jne     short access_1
                ; bne 1f / no, then branch
        shr     cl, 2
                ; asrb r2 / shift owner read write bits into non owner
                        ; / read/write bits
                ; asrb r2
access_1: ; 1:
        and     cl, dl
                ; bit r2,(r0)+ / test read-write flags against argument
                             ; / in access call
        jnz     short access_2
                ; bne 1f
        or      dh, dh ; super user  (root) ?
                ; tstb u.uid
        jz      short access_2 ; yes, super user
        ;jnz    error
                ; beq 1f
                ; jmp error
        mov     dword [u.error], ERR_FILE_ACCESS
                       ; 'permission denied !' error
        jmp     error

access_2: ; 1:
        ; DL = flags
        retn
                ; rts r0
```

```
setimod:
        ; 03/06/2015 (Retro UNIX 386 v1 - Beginning)
        ; 09/04/2013 - 31/07/2013 (Retro UNIX 8086 v1)
        ;
        ; 'setimod' sets byte at location 'imod' to 1; thus indicating that
        ; the inode has been modified. Also puts the time of modification
        ; into the inode.
        ;
        ;  (Retro UNIX Prototype : 14/07/2012 - 23/02/2013, UNIXCOPY.ASM)
         ;  ((Modified registers: eDX, eCX, eBX))

        ; push  edx
        push    eax

        mov     byte [imod], 1
                ; movb $1,imod / set current i-node modified bytes
        ; Erdogan Tan 14-7-2012
        call    epoch
                 ; mov s.time,i.mtim
                            ; / put present time into file modified time
                 ; mov s.time+2,i.mtim+2

        mov     [i.mtim], eax

        ; Retro UNIX 386 v1 modification ! (cmp)
        ; Retro UNIX 8086 v1 modification ! (test)
        cmp     dword [i.ctim], 0
        jnz     short setimod_ok

        mov     [i.ctim], eax

setimod_ok: ; 31/07/2013
        pop     eax
        ;pop    edx

        retn
                ; rts r0

itrunc:
        ; 03/06/2015 (Retro UNIX 386 v1 - Beginning)
        ; 23/04/2013 - 01/08/2013 (Retro UNIX 8086 v1)
        ;
        ; 'itrunc' truncates a file whose i-number is given in r1
        ;  to zero length.
        ;
        ; INPUTS ->
        ;    r1 - i-number of i-node
        ;    i.dskp - pointer to contents or indirect block in an i-node
        ;    i.flgs - large file flag
        ;    i.size - size of file
        ;
        ; OUTPUTS ->
        ;    i.flgs - large file flag is cleared
        ;    i.size - set to 0
        ;    i.dskp .. i.dskp+16 - entire list is cleared
        ;    setimod - set to indicate i-node has been modified
        ;    r1 - i-number of i-node
        ;
        ; ((AX = R1)) input/output
        ;
        ;  (Retro UNIX Prototype : 01/12/2012 - 10/03/2013, UNIXCOPY.ASM)
         ;  ((Modified registers: eDX, eCX, eBX, eSI, eDI, eBP))

        call    iget
                ; jsr r0,iget
        mov     esi, i.dskp
                ; mov $i.dskp,r2 / address of block pointers in r2
        xor     eax, eax
itrunc_1: ; 1:
        lodsw
                ; mov (r2)+,r1 / move physical block number into r1
        or      ax, ax
        jz      short itrunc_5
                ; beq 5f
        push    esi
                ; mov r2,-(sp)
        test     word [i.flgs], 1000h
                ; bit $10000,i.flgs / test large file bit?
        jz      short itrunc_4
```

```
                ; beq 4f / if clear, branch
        push    eax
                ; mov r1,-(sp) / save block number of indirect block
        call    dskrd
                ; jsr r0,dskrd / read in block, 1st data word
                            ; / pointed to by r5
        ; eBX = r5 = Buffer data address (the 1st word)
        mov     ecx, 256
                ; mov $256.,r3 / move word count into r3
        mov     esi, ebx
itrunc_2: ; 2:
        lodsw
                ; mov (r5)+,r1 / put 1st data word in r1;
                            ; / physical block number
        and     ax, ax
        jz      short itrunc_3
                ; beq 3f / branch if zero
        ;push   ecx
        push    cx
                ; mov r3,-(sp) / save r3, r5 on stack
        ;push   esi
                ; mov r5,-(sp)
        call    free
                ; jsr r0,free / free block in free storage map
        ;pop    esi
                ; mov(sp)+,r5
        pop     cx
        ;pop    ecx
                ; mov (sp)+,r3
itrunc_3: ; 3:
        loop    itrunc_2
                ; dec r3 / decrement word count
                ; bgt 2b / branch if positive
        pop     eax
                ; mov (sp)+,r1 / put physical block number of
                            ; / indirect block
        ; 01/08/2013
         and     word [i.flgs], 0EFFFh ; 1110111111111111b
itrunc_4: ; 4:
        call    free
                ; jsr r0,free / free indirect block
        pop     esi
                ; mov (sp)+,r2
itrunc_5: ; 5:
        cmp     esi, i.dskp+16
                ; cmp r2,$i.dskp+16.
        jb      short itrunc_1
                ; bne 1b / branch until all i.dskp entries check
        ; 01/08/2013
        ;and     word [i.flgs], 0EFFFh ; 1110111111111111b
                ; bic $10000,i.flgs / clear large file bit
        mov     edi, i.dskp
        mov     cx, 8
        xor     ax, ax
        mov     [i.size], ax ; 0
                ; clr i.size / zero file size
        rep     stosw
                ; jsr r0,copyz; i.dskp; i.dskp+16.
                          ; / zero block pointers
        call    setimod
                ; jsr r0,setimod / set i-node modified flag
        mov     ax, [ii]
                ; mov ii,r1
        retn
                ; rts r0
```

```
imap:
        ; 03/06/2015 (Retro UNIX 386 v1 - Beginning)
        ; 26/04/2013 (Retro UNIX 8086 v1)
        ;
        ; 'imap' finds the byte in core (superblock) containing
        ; allocation bit for an i-node whose number in r1.
        ;
        ; INPUTS ->
        ;    r1 - contains an i-number
        ;    fsp - start of table containing open files
        ;
        ; OUTPUTS ->
        ;    r2 - byte address of byte with the allocation bit
        ;    mq - a mask to locate the bit position.
        ;         (a 1 is in calculated bit posisiton)
        ;
        ; ((AX = R1)) input/output
        ; ((DL/DX = MQ)) output
        ; ((BX = R2)) output
        ;
        ;    (Retro UNIX Prototype : 02/12/2012, UNIXCOPY.ASM)
        ;     ((Modified registers: eDX, eCX, eBX, eSI))
        ;
                ; / get the byte that has the allocation bit for
                ; / the i-number contained in r1
        ;mov    dx, 1
        mov     dl, 1
                ; mov $1,mq / put 1 in the mq
        movzx   ebx, ax
                ; mov r1,r2 / r2 now has i-number whose byte
                          ; / in the map we must find
        sub     bx, 41
                ; sub $41.,r2 / r2 has i-41
        mov     cl, bl
                ; mov r2,r3 / r3 has i-41
        and     cl, 7
                ; bic $!7,r3 / r3 has (i-41) mod 8 to get
                          ; / the bit position
        jz      short imap1
        ;shl    dx, cl
        shl     dl, cl
                ; mov r3,lsh / move the 1 over (i-41) mod 8 positions
imap1:                    ; / to the left to mask the correct bit
        shr     bx, 3
                ; asr r2
                ; asr r2
                ; asr r2 / r2 has (i-41) base 8 of the byte number
                       ; / from the start of the map
                ; mov r2,-(sp) / put (i-41) base 8 on the stack
        mov     esi, systm
                ; mov $systm,r2 / r2 points to the in-core image of
                               ; / the super block for drum
        ;cmp    word [cdev], 0
        cmp     byte [cdev], 0
                ; tst cdev / is the device the disk
        jna     short imap2
                ; beq 1f / yes
        add     esi, mount - systm
                ; add $mount-systm,r2 / for mounted device,
                       ; / r2 points to 1st word of its super block
imap2: ; 1:
        add     bx, [esi] ;; add free map size to si
                ; add (r2)+,(sp) / get byte address of allocation bit
        add     bx, 4
        add     ebx, esi
                ; add (sp)+,r2 / ?
        ;add    ebx, 4 ;; inode map offset in superblock
                      ;; (2 + free map size + 2)
                ; add $2,r2 / ?
        ; DL/DX (MQ) has a 1 in the calculated bit position
        ; BX (R2) has byte address of the byte with allocation bit
        retn
                ; rts r0
```

```
; Retro UNIX 386 v1 Kernel (v0.2) - SYS6.INC
; Last Modification: 18/11/2015
; ----------------------------------------------------------------------------
; Derived from 'Retro UNIX 8086 v1' source code by Erdogan Tan
; (v0.1 - Beginning: 11/07/2012)
;
; Derived 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>
;
; Retro UNIX 8086 v1 - U6.ASM (23/07/2014) //// UNIX v1 -> u6.s
; ****************************************************************************

readi:
        ; 20/05/2015
        ; 19/05/2015 (Retro UNIX 386 v1 - Beginning)
        ; 11/03/2013 - 31/07/2013 (Retro UNIX 8086 v1)
        ;
        ; Reads from an inode whose number in R1
        ;
        ; INPUTS ->
        ;    r1 - inode number
        ;    u.count - byte count user desires
        ;    u.base - points to user buffer
        ;    u.fofp - points to word with current file offset
        ; OUTPUTS ->
        ;    u.count - cleared
        ;    u.nread - accumulates total bytes passed back
        ;
        ; ((AX = R1)) input/output
        ;    (Retro UNIX Prototype : 01/03/2013 - 14/12/2012, UNIXCOPY.ASM)
         ;    ((Modified registers: edx, ebx, ecx, esi, esi, ebp))

        xor     edx, edx ; 0
        mov     [u.nread], edx ; 0
                 ; clr u.nread / accumulates number of bytes transmitted
        mov     [u.pcount], dx ; 19/05/2015
        cmp     [u.count], edx ; 0
                 ; tst u.count / is number of bytes to be read greater than 0
        ja      short readi_1 ; 1f
                 ; bgt 1f / yes, branch
        retn
                 ; rts r0 / no, nothing to read; return to caller
readi_1: ; 1:
                 ; mov r1,-(sp) / save i-number on stack
        cmp     ax, 40
                 ; cmp r1,$40. / want to read a special file
                 ;              / (i-nodes 1,...,40 are for special files)
         ja      dskr
                 ; ble 1f / yes, branch
                 ; jmp dskr / no, jmp to dskr;
                 ;          / read file with i-node number (r1)
                 ;     / starting at byte ((u.fofp)), read in u.count bytes
        ; (20/05/2015)
        push    eax ; because subroutines will jump to 'ret_'
        ; 1:
        movzx   ebx, al
        shl     bx, 2
                 ; asl r1 / multiply inode number by 2
        add     ebx, readi_2 - 4
        jmp     dword [ebx]
                 ; jmp *1f-2(r1)
readi_2: ; 1:
        dd      rtty ; tty, AX = 1 (runix)
                 ;rtty / tty; r1=2
                 ;rppt / ppt; r1=4
        dd      rmem ; mem, AX = 2 (runix)
                 ;rmem / mem; r1=6
                 ;rrf0 / rf0
                 ;rrk0 / rk0
                 ;rtap / tap0
                 ;rtap / tap1
                 ;rtap / tap2
                 ;rtap / tap3
                 ;rtap / tap4
                 ;rtap / tap5
                 ;rtap / tap6
                 ;rtap / tap7
```

```
        dd      rfd ; fd0, AX = 3 (runix only)
        dd      rfd ; fd1, AX = 4 (runix only)
        dd      rhd ; hd0, AX = 5 (runix only)
        dd      rhd ; hd1, AX = 6 (runix only)
        dd      rhd ; hd2, AX = 7 (runix only)
        dd      rhd ; hd3, AX = 8 (runix only)
        dd      rlpr ; lpr, AX = 9 (invalid, write only device !?)
        dd      rcvt ; tty0, AX = 10 (runix)
                 ;rcvt / tty0
        dd      rcvt ; tty1, AX = 11 (runix)
                 ;rcvt / tty1
        dd      rcvt ; tty2, AX = 12 (runix)
                 ;rcvt / tty2
        dd      rcvt ; tty3, AX = 13 (runix)
                 ;rcvt / tty3
        dd      rcvt ; tty4, AX = 14 (runix)
                 ;rcvt / tty4
        dd      rcvt ; tty5, AX = 15 (runix)
                 ;rcvt / tty5
        dd      rcvt ; tty6, AX = 16 (runix)
                 ;rcvt / tty6
        dd      rcvt ; tty7, AX = 17 (runix)
                 ;rcvt / tty7
        dd      rcvt ; COM1, AX = 18 (runix only)
                 ;rcrd / crd
        dd      rcvt ; COM2, AX = 19 (runix only)

rtty: ; / read from console tty
        ; 17/10/2015 - 16/07/2015 (Retro UNIX 8086 v1)
        ;           (Only 1 byte is read, by ignoring byte count!)
        ;           WHAT FOR: Every character from Keyboard input
        ;           must be written immediate on video page (screen)
        ;           when it is required.
        ; 19/05/2015 (Retro UNIX 386 v1 - Beginning)
        ; 11/03/2013 - 19/06/2014 (Retro UNIX 8086 v1)
        ;
        ; Console tty buffer is PC keyboard buffer
        ; and keyboard-keystroke handling is different than original
        ; unix (PDP-11) here. TTY/Keyboard procedures here are changed
        ; according to IBM PC compatible ROM BIOS keyboard functions.
        ;
        ; 06/12/2013
        movzx   ebx, byte [u.uno] ; process number
        mov     al, [ebx+p.ttyc-1] ; current/console tty
rttys:
                ; mov tty+[8*ntty]-8+6,r5 / r5 is the address of the 4th word of
                        ; / of the control and status block
                ; tst 2(r5) / for the console tty; this word points to the console
                       ; / tty buffer
        ; 28/07/2013
        mov     [u.ttyn], al
        ; 13/01/2014
        inc     al
        mov     [u.ttyp], al ; tty number + 1
rtty_nc: ; 01/02/2014
        ; 29/09/2013
        mov     ecx, 10
rtty_1:         ; 01/02/2014
        push    cx ; 29/09/2013
        ; byte [u.ttyn] = tty number (0 to 9)
        mov     al, 1
        call    getc
        pop     cx ; 29/09/2013
        jnz     short rtty_2
                ; bne 1f / 2nd word of console tty buffer contains number
                        ; / of chars. Is this number non-zero?
        loop    rtty_idle ; 01/02/2014
        ; 05/10/2013
        mov     ah, [u.ttyn]
        ; 29/09/2013
        call    sleep
                ; jsr r0,canon; ttych / if 0, call 'canon' to get a line
                 ;            / (120 chars.)
        ;byte [u.ttyn] = tty number (0 to 9)
        jmp     short rtty_nc ; 01/02/2014
```

```
rtty_idle:
        ; 29/07/2013
        call    idle
        jmp     short rtty_1 ; 01/02/2014
        ;1:
                ; tst 2(r5) / is the number of characters zero
                ; beq ret1 / yes, return to caller via 'ret1'
                ; movb *4(r5),r1 / no, put character in r1
                ; inc 4(r5) / 3rd word of console tty buffer points to byte which
                          ; / contains the next char.
                ; dec 2(r5) / decrement the character count
rtty_2:
        xor     al, al
        call    getc
        call    passc
                ; jsr r0,passc / move the character to core (user)
        ;; 17/10/2015 - 16/07/2015
        ; 19/06/2014
        ;;jnz   short rtty_nc
        pop     eax  ; (20/05/2015)
        retn
;ret1:
                ; jmp ret / return to caller via 'ret'

rcvt:   ; < receive/read character from tty >
        ; 19/05/2015 (Retro UNIX 386 v1 - Beginning)
        ; 15/05/2013 - 06/12/2013 (Retro UNIX 8086 v1)
        ;
        ; Retro UNIX 8086 v1 modification !
        ;
        ; In original UNIX v1, 'rcvt' routine
        ;               (exactly different than this one)
        ;       was in 'u9.s' file.
        ;
        sub     al, 10
        ; AL = tty number (0 to 9), (COM1=8, COM2=9)
        ; 16/07/2013
        ; 21/05/2013
         jmp     short rttys

;rppt: / read paper tape
;       jsr     r0,pptic / gets next character in clist for ppt input and
;                       / places
;               br ret / it in r1; if there 1s no problem with reader, it
;                      / also enables read bit in prs
;       jsr     r0,passc / place character in users buffer area
;       br      rppt

rmem: ; / transfer characters from memory to a user area of core
        ; 17/10/2015
        ; 11/06/2015
        ; 24/05/2015
        ; 19/05/2015 (Retro UNIX 386 v1 - Beginning)
        ;
        mov      esi, [u.fofp]
rmem_1:
         mov     ebx, [esi]
                ; mov *u.fofp,r1 / save file offset which points to the char
                              ; / to be transferred to user
         inc     dword [esi] ; 17/10/2015
                ; inc *u.fofp / increment file offset to point to 'next'
                          ; / char in memory file
        mov     al, [ebx]
                ; movb (r1),r1 / get character from memory file,
                             ; / put it in r1
        call    passc         ; jsr r0,passc / move this character to
                             ;  / the next byte of the users core area
                ; br rmem / continue
        jnz     short rmem_1
ret_:
        pop     eax ; 09/06/2015
        retn

rlpr:
;1:
;rcrd:
         mov     dword [u.error], ERR_DEV_NOT_RDY ; 19/05/2015
        jmp     error
                ;jmp    error / see 'error' routine
```

```
dskr:
        ; 12/10/2015
        ; 21/08/2015
        ; 25/07/2015
        ; 10/07/2015
        ; 16/06/2015
        ; 31/05/2015
        ; 24/05/2015 (Retro UNIX 386 v1 - Beginning)
        ; 26/04/2013 - 03/08/2013 (Retro UNIX 8086 v1)
dskr_0:
        push    eax
                ; mov (sp),r1 / i-number in r1
        ; AX = i-number
        call    iget
                ; jsr r0,iget / get i-node (r1) into i-node section of core
         movzx   edx, word [i.size] ; 16/06/2015
                ; mov i.size,r2 / file size in bytes in r2
        mov     ebx, [u.fofp]
        sub     edx, [ebx]
                ; sub *u.fofp,r2 / subtract file offset
         ; 12/10/2015
        ; jna     short ret_
                ; blos ret
        ja      short dskr_1
dskr_retn: ; 12/10/2015
        pop     eax
        mov     byte [u.kcall], 0
        retn
dskr_1:
        cmp      edx, [u.count]
                ; cmp r2,u.count / are enough bytes left in file
                              ; / to carry out read
        jnb     short dskr_2
                ; bhis 1f
        mov     [u.count], edx
                ; mov r2,u.count / no, just read to end of file
dskr_2: ; 1:
        ; AX = i-number
        call    mget
                ; jsr r0,mget / returns physical block number of block
                           ; / in file where offset points
        ; eAX = physical block number
        call    dskrd
                ; jsr r0,dskrd / read in block, r5 points to
                            ; / 1st word of data in buffer
        ; 09/06/2015
        cmp     byte [u.kcall], 0 ; the caller is 'namei' sign (=1)
        ja      short dskr_4     ; zf=0 -> the caller is 'namei'
        cmp     word [u.pcount], 0
        ja      short dskr_4
dskr_3:
        ; [u.base] = virtual address to transfer (as destination address)
        call    trans_addr_w ; translate virtual address to physical (w)
dskr_4:
        ; eBX (r5) = system (I/O) buffer address -physical-
        call    sioreg
                ; jsr r0,sioreg
        xchg    esi, edi
        ; eDI = file (user data) offset
        ; eSI = sector (I/O) buffer offset
        ; eCX = byte count
        rep     movsb
                ; movb (r2)+,(r1)+ / move data from buffer into working core
                                ; / starting at u.base
                ; dec r3
                ; bne 2b / branch until proper number of bytes are transferred
        ; 25/07/2015
        ; eax = remain bytes in buffer
         ;        (check if remain bytes in the buffer > [u.pcount])
        or      eax, eax
        jnz     short dskr_3 ; (page end before system buffer end!)
        ; 03/08/2013
        ;pop    eax
        cmp     [u.count], ecx ; 0
                ; tst u.count / all bytes read off disk
                ; bne dskr
                ; br ret
         ;ja      short dskr_0
```

```
        ;mov    [u.kcall], cl ; 0 ; 09/06/2015
        ;retn
        ; 12/10/2015
        jna     short dskr_retn
        pop     eax  ; (i-node number)
        jmp     short dskr_0

passc:
        ; 18/10/2015
        ; 10/07/2015
        ; 01/07/2015
        ; 08/06/2015
        ; 04/06/2015
        ; 20/05/2015
        ; 19/05/2015 (Retro UNIX 386 v1 - Beginning)
        ;
        ;(Retro UNIX 386 v1 - translation from user's virtual address
        ;                    to physical address
        cmp     word [u.pcount], 0 ; byte count in page = 0 (initial value)
                            ; 1-4095 --> use previous physical base address
                            ; in [u.pbase]
        ja      short passc_3
        ; 08/06/2015 - 10/07/2015
        call    trans_addr_w
passc_3:
        ; 19/05/2015
        dec     word [u.pcount]
        ;
        mov     ebx, [u.pbase]
        mov     [ebx], al
                ; movb r1,*u.base / move a character to the next byte of the
                               ; / users buffer
        inc     dword [u.base]
                ; inc u.base / increment the pointer to point to
                         ; / the next byte in users buffer
        inc     dword [u.pbase] ; 04/06/2015
        inc     dword [u.nread]
                ; inc u.nread / increment the number of bytes read
        dec     dword [u.count]
                ; dec u.count / decrement the number of bytes to be read
                ; bne 1f / any more bytes to read?; yes, branch
        retn
                ; mov (sp)+,r0 / no, do a non-local return to the caller of
                             ; / 'readi' by:
                ;/ (1) pop the return address off the stack into r0
                ; mov (sp)+,r1 / (2) pop the i-number off the stack into r1
        ;1:
                ; clr   *$ps / clear processor status
                ; rts r0 / return to address currently on top of stack

trans_addr_r:
        ; Translate virtual address to physical address
        ; for reading from user's memory space
        ; (Retro UNIX 386 v1 feature only !)
        ; 18/10/2015
        ; 10/07/2015
        ; 09/06/2015
        ; 08/06/2015
        ; 04/06/2015
        ;
        ; 18/10/2015
        xor     edx, edx ; 0 (read access sign)
        jmp     short trans_addr_rw

        ;push   eax
        ;push   ebx
        ;mov    ebx, [u.base]
        ;call   get_physical_addr ; get physical address
        ;;jnc   short cpass_0
        ;jnc    short passc_1
        ;mov    [u.error], eax
        ;;pop   ebx
        ;;pop   eax
        ;jmp    error
;cpass_0:
        ; 18/10/2015
        ; 20/05/2015
        ;mov    [u.pbase], eax ; physical address
        ;mov    [u.pcount], cx ; remain byte count in page (1-4096)
```

```
        ;pop    ebx
        ;pop    eax
        ;retn   ; 08/06/2015

trans_addr_w:
        ; Translate virtual address to physical address
        ; for writing to user's memory space
        ; (Retro UNIX 386 v1 feature only !)
        ; 18/10/2015
        ; 29/07/2015
        ; 10/07/2015
        ; 09/06/2015
        ; 08/06/2015
        ; 04/06/2015 (passc)
        ;
        ; 18/10/2015
        sub     edx, edx
        inc     dl ; 1 (write access sign)
trans_addr_rw:
        push    eax
        push    ebx
        ; 18/10/2015
        push    edx ; r/w sign (in DL)
        ;
        mov     ebx, [u.base]
        call    get_physical_addr ; get physical address
        jnc     short passc_0
        mov     [u.error], eax
        ;pop    edx
        ;pop    ebx
        ;pop    eax
        jmp     error
passc_0:
        test    dl, PTE_A_WRITE ; writable page ; 18/10/2015
        pop     edx ; 18/10/2015
        jnz     short passc_1
        ; 18/10/2015
        and     dl, dl
        jz      short passc_1
        ; 20/05/2015
        ; read only (duplicated) page -must be copied to a new page-
        ; EBX = linear address
        push    ecx
        call    copy_page
        pop     ecx
        jc      short passc_2
        push    eax ; physical address of the new/allocated page
        call    add_to_swap_queue
        pop     eax
        ; 18/10/2015
        and     ebx, PAGE_OFF ; 0FFFh
        ;mov    ecx, PAGE_SIZE
        ;sub    ecx, ebx
        add     eax, ebx
passc_1:
        ; 18/10/2015
        ; 20/05/2015
        mov     [u.pbase], eax ; physical address
        mov     [u.pcount], cx ; remain byte count in page (1-4096)
        pop     ebx
        pop     eax
        retn    ; 08/06/2015
passc_2:
        mov     dword [u.error], ERR_MINOR_IM ; "Insufficient memory !" error
        ;pop    ebx
        ;pop    eax
        jmp     error
```

```
writei:
        ; 20/05/2015
        ; 19/05/2015 (Retro UNIX 386 v1 - Beginning)
        ; 12/03/2013 - 31/07/2013 (Retro UNIX 8086 v1)
        ;
        ; Write data to file with inode number in R1
        ;
        ; INPUTS ->
        ;    r1 - inode number
        ;    u.count - byte count to be written
        ;    u.base - points to user buffer
        ;    u.fofp - points to word with current file offset
        ; OUTPUTS ->
        ;    u.count - cleared
        ;    u.nread - accumulates total bytes passed back
        ; ((AX = R1))
        ;    (Retro UNIX Prototype : 18/11/2012 - 11/11/2012, UNIXCOPY.ASM)
        ;    ((Modified registers: DX, BX, CX, SI, DI, BP))

        xor     ecx, ecx
        mov     [u.nread], ecx  ; 0
                ; clr u.nread / clear the number of bytes transmitted during
                            ; / read or write calls
        mov     [u.pcount], cx ; 19/05/2015
        cmp     [u.count], ecx
        ;       ; tst u.count / test the byte count specified by the user
        ja      short writei_1 ; 1f
                ; bgt 1f / any bytes to output; yes, branch
        retn
        ;       ; rts r0 / no, return - no writing to do
writei_1: ;1:
                ; mov r1 ,-(sp) / save the i-node number on the stack
        cmp     ax, 40
                ; cmp r1,$40.
                ; / does the i-node number indicate a special file?
         ja      dskw
                ; bgt dskw / no, branch to standard file output
        ; (20/05/2015)
        push    eax ; because subroutines will jump to 'ret_'
        movzx   ebx, al
        shl     bx, 2
                ; asl r1 / yes, calculate the index into the special file
        add     ebx, writei_2 - 4
        jmp     dword [ebx]
                ; jmp *1f-2(r1)
                ; / jump table and jump to the appropriate routine
writei_2: ;1:
        dd      wtty ; tty, AX = 1 (runix)
                 ;wtty / tty; r1=2
                 ;wppt / ppt; r1=4
        dd      wmem ; mem, AX = 2 (runix)
                 ;wmem / mem; r1=6
                 ;wrf0 / rf0
                 ;wrk0 / rk0
                 ;wtap / tap0
                 ;wtap / tap1
                 ;wtap / tap2
                 ;wtap / tap3
                 ;wtap / tap4
                 ;wtap / tap5
                 ;wtap / tap6
                 ;wtap / tap7
        dd      wfd ; fd0, AX = 3 (runix only)
        dd      wfd ; fd1, AX = 4 (runix only)
        dd      whd ; hd0, AX = 5 (runix only)
        dd      whd ; hd1, AX = 6 (runix only)
        dd      whd ; hd2, AX = 7 (runix only)
        dd      whd ; hd3, AX = 8 (runix only)
        dd      wlpr ; lpr, AX = 9   (runix)
        dd      xmtt ; tty0, AX = 10 (runix)
                 ;xmtt / tty0
        dd      xmtt ; tty1, AX = 11 (runix)
                 ;xmtt / tty1
        dd      xmtt ; tty2, AX = 12 (runix)
                 ;xmtt / tty2
        dd      xmtt ; tty3, AX = 13 (runix)
                 ;xmtt / tty3
        dd      xmtt ; tty4, AX = 14 (runix)
                 ;xmtt / tty4
```

```
        dd      xmtt ; tty5, AX = 15 (runix)
                 ;xmtt / tty5
        dd      xmtt ; tty6, AX = 16 (runix)
                 ;xmtt / tty6
        dd      xmtt ; tty7, AX = 17 (runix)
                 ;xmtt / tty7
        dd      xmtt ; COM1, AX = 18 (runix only)
                ; / wlpr / lpr
        dd      xmtt ; COM2, AX = 19 (runix only)

wtty: ; write to console tty (write to screen)
        ; 18/11/2015
        ; 19/05/2015 (Retro UNIX 386 v1 - Beginning)
        ; 12/03/2013 - 07/07/2014 (Retro UNIX 8086 v1)
        ;
        ; Console tty output is on current video page
        ; Console tty character output procedure is changed here
        ; acording to IBM PC compatible ROM BIOS video (text mode) functions.
        ;
        movzx   ebx, byte [u.uno] ; process number
        mov     ah, [ebx+p.ttyc-1] ; current/console tty
        mov     al, ah ; 07/07/2014
wttys:
        ; 10/10/2013
        mov     [u.ttyn], ah
        ; 13/01/2014
        inc     al
        mov     [u.ttyp+1], al ; tty number + 1
wtty_nc: ; 15/05/2013
        ; AH = [u.ttyn] = tty number ; 28/07/2013
        call    cpass
                ; jsr r0,cpass / get next character from user buffer area; if
                              ; / none go to return address in syswrite
                ; tst r1 / is character = null
                ; beq wtty / yes, get next character
        ; 10/10/2013
        jz      short wret
        ;1 :
                ;mov    $240,*$ps / no, set processor priority to five
                ;cmpb   cc+1,$20. / is character count for console tty greater
                ;                 / than 20
                ;bhis   2f / yes; branch to put process to sleep
        ; 27/06/2014
wtty_1:
        ; AH = tty number
        ; AL = ASCII code of the character
        ; 15/04/2014
        push    ax
        call    putc ; 14/05/2013
        jnc     short wtty_2
        ; 18/11/2015
        call    idle
        mov     ax, [esp]
        call    putc
        jnc     short wtty_2
        ; 02/06/2014
        mov     ah, [u.ttyn]
        call    sleep
        pop     ax
        jmp     short wtty_1
                ; jc    error ; 15/05/2013 (COM1 or COM2 serial port error)
                ; jsr   r0,putc; 1 / find place in freelist to assign to
                              ; / console tty and
                ; br    2f / place character in list; if none available
                          ; / branch to put process to sleep
                ; jsr   r0,startty / attempt to output character on tty
wtty_2:
        ; 15/04/2014
        pop     ax
        jmp     short wtty_nc
                ; br wtty
wret:   ; 10/10/2013 (20/05/2015)
        pop     eax
        retn
        ;2:
                ;mov    r1,-(sp) / place character on stack
                ;jsr    r0,sleep; 1 / put process to sleep
                ;mov    (sp)+,r1 / remove character from stack
                ;br     1b / try again to place character in clist and output
```

```
xmtt:   ; < send/write character to tty >
        ; 19/05/2015 (Retro UNIX 386 v1 - Beginning)
        ; 15/05/2013 - 06/12/2013 (Retro UNIX 8086 v1)
        ;
        ; Retro UNIX 8086 v1 modification !
        ;
        ; In original UNIX v1, 'xmtt' routine
        ;               (exactly different than this one)
        ;       was in 'u9.s' file.
        ;
        sub     al, 10
        ; AL = tty number (0 to 9), (COM1=8, COM2=9)
         ; 10/10/2013
        mov     ah, al
        ; 28/07/2013
        jmp     short wttys

;wppt:
;       jsr     r0,cpass / get next character from user buffer area,
;                        / if none return to writei's calling routine
;       jsr     r0,pptoc / output character on ppt
;       br      wppt
wlpr:
         mov     dword [u.error], ERR_DEV_NOT_RDY ; 19/05/2015
        jmp     error   ; ... Printing procedure will be located here ...
                ;/      jsr     r0,cpass
                ;/      cmp     r0,$'a
                ;/      blo     1f
                ;/      cmp     r1,$'z
                ;/      bhi     1f
                ;/      sub     $40,r1
                ;/1:
                ;/      jsr     r0,lptoc
                ;/      br      wlpr
                ; br rmem / continue

wmem: ; / transfer characters from a user area of core to memory file
        ; 17/10/2015
        ; 11/06/2015
        ; 24/05/2015
        ; 19/05/2015 (Retro UNIX 386 v1 - Beginning)
        ;
        cmp     dword [x_timer], clock ; multi tasking clock/timer
         je      short wmem_acc_err
        ;
         mov     esi, [u.fofp]
wmem_1:
        call    cpass
                ; jsr r0,cpass / get next character from users area of
                             ; / core and put it in r1
                ; mov r1,-(sp) / put character on the stack
        ; 20/09/2013
        jz      short wret ; wmem_2
         mov     ebx, [esi]
                ; mov *u.fofp,r1 / save file offset in r1
         inc     dword [esi] ; 17/10/2015
                ; inc *u.fofp / increment file offset to point to next
                           ; / available location in file
        mov     [ebx], al
                ; movb (sp)+,(r1) / pop char off stack, put in memory loc
                               ; / assigned to it
        jmp     short wmem_1
                ; br wmem / continue
        ;1:
        ;jmp    error / ?
;wmem_2:
;       ; 20/09/2013
;       pop     ax
;       retn

wmem_acc_err:
        mov     dword [u.error], ERR_FILE_ACCESS ; permission denied !
        jmp     error
```

```
dskw: ; / write routine for non-special files
        ;
        ; 25/07/2015
        ; 16/06/2015
        ; 09/06/2015
        ; 31/05/2015 (Retro UNIX 386 v1 - Beginning)
        ; 26/04/2013 - 20/09/2013 (Retro UNIX 8086 v1)
        ;
        ; 01/08/2013 (mkdir_w check)
        push    ax ; 26/04/2013
                ; mov (sp),r1 / get an i-node number from the stack into r1
        ; AX = inode number
        call    iget
                ; jsr r0,iget / write i-node out (if modified),
                            ; / read i-node 'r1' into i-node area of core
         mov     ebx, [u.fofp]
        mov     edx, [ebx]
                ; mov *u.fofp,r2 / put the file offset [(u.off) or the offset
                              ; / in the fsp entry for this file] in r2
        add     edx, [u.count]
                ; add u.count,r2 / no. of bytes to be written
                              ; / + file offset is put in r2
        ; 16/06/2015
        cmp     edx, 65535 ; file size limit (for UNIX v1 file system)
        jna     short dskw_0
        mov     dword [u.error], ERR_FILE_SIZE ; 'file size error !'
        jmp     error
dskw_0:
        cmp      dx, [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      [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
        ; eAX = Block number
                ; jsr r0,mget / get the block no. in which to write
                            ; / the next data byte
        ; eax = block number
        mov      ebx, [u.fofp]
        mov     edx, [ebx]
        and     edx, 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     dword [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_3
                ; 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).
        call    dskrd
                ; jsr r0,dskrd / no, must retain old info..
                            ; / Hence, read block 'r1' into an I/O buffer
dskw_3: ; 3:
        ; eAX (r1) = block/sector number
        call    wslot
                ; jsr r0,wslot / set write and inhibit bits in I/O queue,
                           ; / proc. status=0, r5 points to 1st word of data
        cmp     byte [u.kcall], 0
        ja      short dskw_5 ; zf=0 -> the caller is 'mkdir'
        ;
        cmp     word [u.pcount], 0
        ja      short dskw_5
dskw_4:
        ; [u.base] = virtual address to transfer (as source address)
        call    trans_addr_r ; translate virtual address to physical (r)
```

```
dskw_5:
        ; eBX (r5) = system (I/O) buffer address
        call   sioreg
               ; jsr r0,sioreg / r3 = no. of bytes of data,
                            ; / r1 = address of data, r2 points to location
                            ; / in buffer in which to start writing data
        ; eSI = file (user data) offset
        ; eDI = sector (I/O) buffer offset
        ; eCX = byte count
        ;
        rep     movsb
               ; movb (r1 )+,(r2)+
                         ; / transfer a byte of data to the I/O buffer
               ; dec r3 / decrement no. of bytes to be written
               ; bne 2b / have all bytes been transferred? No, branch
        ; 25/07/2015
        ; eax = remain bytes in buffer
         ;       (check if remain bytes in the buffer > [u.pcount])
        or      eax, eax
        jnz     short dskw_4 ; (page end before system buffer end!)
dskw_6:
        call   dskwr
               ; jsr r0,dskwr / yes, write the block and the i-node
         cmp     dword [u.count], 0
               ; tst u.count / any more data to write?
        ja      short dskw_1
               ; bne 1b / yes, branch
        ; 03/08/2013
        mov     byte [u.kcall], 0
        ; 20/09/2013 (;;)
        pop     ax
        retn
        ;;jmp   short dskw_ret
                ; jmp ret / no, return to the caller via 'ret'

cpass: ; / get next character from user area of core and put it in r1
        ; 18/10/2015
        ; 10/10/2015
        ; 10/07/2015
        ; 02/07/2015
        ; 01/07/2015
        ; 24/06/2015
        ; 08/06/2015
        ; 04/06/2015
        ; 20/05/2015
        ; 19/05/2015 (Retro UNIX 386 v1 - Beginning)
        ;
        ; INPUTS ->
        ;     [u.base] = virtual address in user area
        ;     [u.count] = byte count (max.)
        ;     [u.pcount] = byte count in page (0 = reset)
        ; OUTPUTS ->
        ;     AL = the character which is pointed by [u.base]
        ;     zf = 1 -> transfer count has been completed
         ;
        ; ((Modified registers:  EAX, EDX, ECX))
        ;
        ;
        cmp     dword [u.count], 0  ; 14/08/2013
               ; tst u.count / have all the characters been transferred
                           ; / (i.e., u.count, # of chars. left
        jna     short cpass_3
               ; beq 1f / to be transferred = 0?) yes, branch
        dec     dword [u.count]
               ; dec u.count / no, decrement u.count
         ; 19/05/2015
        ;(Retro UNIX 386 v1 - translation from user's virtual address
        ;                      to physical address
        cmp     word [u.pcount], 0 ; byte count in page = 0 (initial value)
                            ; 1-4095 --> use previous physical base address
                            ; in [u.pbase]
        ja      short cpass_1
        ; 02/07/2015
         cmp     dword [u.ppgdir], 0  ; is the caller os kernel
         je      short cpass_k       ; (sysexec, '/etc/init') ?
        ; 08/06/2015 - 10/07/2015
        call   trans_addr_r
```

```
cpass_1:
        ; 02/07/2015
        ; 24/06/2015
        dec     word [u.pcount]
cpass_2:
         ;10/10/2015
        ; 02/07/2015
        mov     edx, [u.pbase]
        mov     al, [edx] ; 10/10/2015
                ; movb *u.base,r1 / take the character pointed to
                                ; / by u.base and put it in r1
        inc     dword [u.nread]
                ; inc u.nread / increment no. of bytes transferred
        inc     dword [u.base]
                ; inc u.base / increment the buffer address to point to the
                            ; / next byte
        inc     dword [u.pbase] ; 04/06/2015
cpass_3:
        retn
                ; rts   r0 / next byte
        ; 1:
                ; mov (sp)+,r0
                          ; / put return address of calling routine into r0
                ; mov (sp)+,r1 / i-number in r1
                ; rts r0 / non-local return
cpass_k:
        ; 02/07/2015
        ; The caller is os kernel
        ; (get sysexec arguments from kernel's memory space)
        ;
        mov     ebx, [u.base]
         mov     word [u.pcount], PAGE_SIZE ; 4096
        mov     [u.pbase], ebx
        jmp     short cpass_2

sioreg:
        ; 25/07/2015
        ; 18/07/2015
        ; 02/07/2015
        ; 17/06/2015
        ; 09/06/2015
        ; 19/05/2015 (Retro UNIX 386 v1 - Beginning)
        ; 12/03/2013 - 22/07/2013 (Retro UNIX 8086 v1)
        ;
        ; INPUTS ->
        ;     eBX = system buffer (data) address (r5)
        ;     [u.fofp] = pointer to file offset pointer
        ;     [u.base] = virtual address of the user buffer
        ;     [u.pbase] = physical address of the user buffer
        ;     [u.count] = byte count
        ;     [u.pcount] = byte count within page frame
        ; OUTPUTS ->
        ;     eSI = user data offset (r1)
        ;     eDI = system (I/O) buffer offset (r2)
        ;     eCX = byte count (r3)
        ;     EAX = remain bytes after byte count within page frame
        ;       (If EAX > 0, transfer will continue from the next page)
         ;
        ; ((Modified registers:  EDX))

         mov     esi, [u.fofp]
         mov     edi, [esi]
                ; mov *u.fofp,r2 / file offset (in bytes) is moved to r2
        mov     ecx, edi
                ; mov r2,r3 / and also to r3
        or      ecx, 0FFFFFE00h
                ; bis $177000,r3 / set bits 9,...,15 of file offset in r3
        and     edi, 1FFh
                ; bic $!777,r2 / calculate file offset mod 512.
        add     edi, ebx ; EBX = system buffer (data) address
                ; add r5,r2 / r2 now points to 1st byte in system buffer
                          ; / where data is to be placed
                 ; mov u.base,r1 / address of data is in r1
        neg     ecx
                ; neg r3 / 512 - file offset (mod512.) in r3
                      ; / (i.e., the no. of free bytes in the file block)
```

```
        cmp     ecx, [u.count]
                ; cmp r3,u.count / compare this with the no. of data bytes
                               ; / to be written to the file
        jna     short sioreg_0
                ; blos  2f / if less than branch. Use the no. of free bytes
                        ; / in the file block as the number to be written
        mov     ecx, [u.count]
                ; mov u.count,r3 / if greater than, use the no. of data
                               ; / bytes as the number to be written
sioreg_0:
        ; 17/06/2015
        cmp     byte [u.kcall], 0
        jna     short sioreg_1
        ; 25/07/2015
         ; the caller is 'mkdir' or 'namei'
        mov     eax, [u.base] ; 25/07/2015
        mov     [u.pbase], eax ; physical address = virtual address
        mov     word [u.pcount], cx ; remain bytes in buffer (1 sector)
        jmp     short sioreg_2
sioreg_1:
        ; 25/07/2015
        ; 18/07/2015
        ; 09/06/2015
        movzx   edx, word [u.pcount]
                ; ecx and [u.pcount] are always > 0, here
        cmp     ecx, edx
        ja      short sioreg_4 ; transfer count > [u.pcount]
sioreg_2: ; 2:
        xor     eax, eax ; 25/07/2015
sioreg_3:
        add     [u.nread], ecx
                ; add r3,u.nread / r3 + number of bytes xmitted
                                 ; / during write is put into u.nread
        sub     [u.count], ecx
                ; sub r3,u.count / u.count = no. of bytes that still
                               ; / must be written or read
        add     [u.base], ecx
                ; add r3,u.base / u.base points to the 1st of the remaining
                              ; / data bytes
         add    [esi], ecx
                ; add r3,*u.fofp / new file offset = number of bytes done
                               ; / + old file offset
        ; 25/07/2015
        mov     esi, [u.pbase]
        sub     [u.pcount], cx
        add     [u.pbase], ecx
         retn
                ; rts r0
                ; transfer count > [u.pcount]
sioreg_4:
        ; 25/07/2015
        ; transfer count > [u.pcount]
        ; (ecx > edx)
        mov     eax, ecx
        sub     eax, edx ; remain bytes for 1 sector (block) transfer
        mov     ecx, edx ; current transfer count = [u.pcount]
        jmp     short sioreg_3
```

```
; Retro UNIX 386 v1 Kernel (v0.2) - SYS7.INC
; Last Modification: 14/11/2015
; ----------------------------------------------------------------------------
; Derived from 'Retro UNIX 8086 v1' source code by Erdogan Tan
; (v0.1 - Beginning: 11/07/2012)
;
; Derived 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>
;
; Retro UNIX 8086 v1 - U7.ASM (13/07/2014) //// UNIX v1 -> u7.s
;
; ****************************************************************************

sysmount: ; / mount file system; args special; name
	; 14/11/2015
	; 24/10/2015
	; 13/10/2015
	; 10/07/2015
	; 16/05/2015 (Retro UNIX 386 v1 - Beginning)
	; 09/07/2013 - 04/11/2013 (Retro UNIX 8086 v1)
	;
	; 'sysmount' anounces to the system that a removable
	; file system has been mounted on a special file.
	; The device number of the special file is obtained via
	; a call to 'getspl'. It is put in the I/O queue entry for
	; dismountable file system (sb1) and the I/O queue entry is
	; set up to read (bit 10 is set). 'ppoke' is then called to
	; to read file system into core, i.e. the first block on the
	; mountable file system is read in. This block is super block
	; for the file system. This call is super user restricted.
	;
	; Calling sequence:
	;	sysmount; special; name
	; Arguments:
	;	special - pointer to name of special file (device)
	;	name -  pointer to name of the root directory of the
	;		newly mounted file system. 'name' should
	;		always be a directory.
	; Inputs: -
	; Outputs: -
	; ...............................................................
	;
	; Retro UNIX 8086 v1 modification:
	;	 'sysmount' system call has two arguments; so,
	;	* 1st argument, special is pointed to by BX register
	;	* 2nd argument, name is in CX register
	;
	;	NOTE: Device numbers, names and related procedures are
	;	       already modified for IBM PC compatibility and
	;	       Retro UNIX 8086 v1 device configuration.

	;call	arg2
		; jsr r0,arg2 / get arguments special and name
	mov	[u.namep], ebx
	push	ecx ; directory name
	cmp	word [mnti], 0
		; tst mnti / is the i-number of the cross device file
			; / zero?
	;ja	error
		; bne errora / no, error
	ja	sysmnt_err0
	call	getspl
		; jsr r0,getspl / get special files device number in r1
	; 13/10/2015
	movzx	ebx, ax ; ; Retro UNIX 8086 v1 device number (0 to 5)
	 test	 byte [ebx+drv.status], 80h ; 24/10/2015
	jnz	short sysmnt_1
sysmnt_err1:
	 mov	 dword [u.error], ERR_DRV_NOT_RDY ; drive not ready !
	jmp	error
sysmnt_1:
	pop	dword [u.namep]
		; mov (sp)+,u.namep / put the name of file to be placed
				; / on the device
	; 14/11/2015
	push	ebx ; 13/10/2015
		; mov r1,-(sp) / save the device number
```

```
        call    namei
        ;or     ax, ax ; Retro UNIX 8086 v1 modification !
                       ; ax = 0 -> file not found
        ;jz     error
        ;jc     error
                ; jsr r0,namei / get the i-number of the file
                       ; br errora
        jnc     short sysmnt_2
sysmnt_err2:
         mov     dword [u.error], ERR_FILE_NOT_FOUND ; drive not ready !
        jmp     error
sysmnt_2:
        mov     [mnti], ax
                ; mov r1,mnti / put it in mnti
;       mov     ebx, sb1 ; super block buffer (of mounted disk)
sysmnt_3: ;1:
         ;cmp   byte [ebx+1], 0
                ; tstb sb1+1 / is 15th bit of I/O queue entry for
                          ; / dismountable device set?
         ;jna   short sysmnt_4
                ; bne 1b / (inhibit bit) yes, skip writing
        ;call   idle    ; (wait for hardware interrupt)
        ;jmp    short sysmnt_3
sysmnt_4:
        pop     eax ; Retro UNIX 8086 v1 device number/ID (0 to 5)
        mov     [mdev], al
                ; mov  (sp),mntd / no, put the device number in mntd
        mov     [ebx], al
                ; movb (sp),sb1 / put the device number in the lower byte
                             ; / of the I/O queue entry
        ;mov    byte [cdev], 1 ; mounted device/drive
                ; mov (sp)+,cdev / put device number in cdev
         or     word [ebx], 400h ; Bit 10, 'read' flag/bit
                ; bis $2000,sb1 / set the read bit
        ; Retro UNIX 386 v1 modification :
        ;       32 bit block number at buffer header offset 4
        mov     dword [ebx+4], 1 ; physical block number = 1
        call    diskio
        jnc     short sysmnt_5
        xor     eax, eax
        mov     [mnti], ax ; 0
        mov     [mdev], al ; 0
        ;mov    [cdev], al ; 0
sysmnt_invd:
        ; 14/11/2015
        dec     al
        mov     [ebx], eax ; 000000FFh
        inc     al
        dec     eax
        mov     [ebx+4], eax ; 0FFFFFFFFh
        jmp     error
sysmnt_5:
        ; 14/11/2015 (Retro UNIX 386 v1 modification)
        ; (Following check is needed to prevent mounting an
        ; in valid valid file system (in valid super block).
        ;
        movzx   eax, byte [ebx] ; device number
        shl     al, 2 ; 4*index
        mov     ecx, [eax+drv.size] ; volume (fs) size
        shl     ecx, 3
        movzx   edx, word [sb1+4] ; the 1st data word
        cmp     ecx, edx ; compare free map bits and volume size
                         ; (in sectors), if they are not equal
                         ; the disk to be mounted is an...
        jne     short sysmnt_invd ; invalid disk !
                         ; (which has not got a valid super block)
        ;
        mov     byte [ebx+1], 0
                ; jsr r0,ppoke / read in entire file system
;sysmnt_6: ;1:
        ;;cmp   byte [sb1+1], 0
                ; tstb   sb1+1 / done reading?
        ;;jna   sysret
        ;;call idle ; (wait for hardware interrupt)
        ;;jmp   short sysmnt_6
                ;bne 1b / no, wait
                ;br sysreta / yes
        jmp     sysret
```

```
sysumount: ; / special dismount file system
        ; 16/05/2015 (Retro UNIX 386 v1 - Beginning)
        ; 09/07/2013 - 04/11/2013 (Retro UNIX 8086 v1)
        ;
        ; 04/11/2013
        ; 09/07/2013
        ; 'sysumount' anounces to the system that the special file,
        ; indicated as an argument is no longer contain a removable
        ; file system. 'getspl' gets the device number of the special
        ; file. If no file system was mounted on that device an error
        ; occurs. 'mntd' and 'mnti' are cleared and control is passed
        ; to 'sysret'.
        ;
        ; Calling sequence:
        ;       sysmount; special
        ; Arguments:
        ;       special - special file to dismount (device)
        ;
        ; Inputs: -
        ; Outputs: -
        ; ...............................................................
        ;
        ; Retro UNIX 8086 v1 modification:
        ;        'sysumount' system call has one argument; so,
        ;       * Single argument, special is pointed to by BX register
        ;

        ;mov    ax, 1 ; one/single argument, put argument in BX
        ;call   arg
                ; jsr r0,arg; u.namep / point u.namep to special
         mov    [u.namep], ebx
        call    getspl
                ; jsr r0,getspl / get the device number in r1
        cmp     al, [mdev]
                ; cmp r1,mntd / is it equal to the last device mounted?
        jne     short sysmnt_err0 ; 'permission denied !' error
        ;jne    error
                ; bne errora / no error
        xor     al, al ; ah = 0
sysumnt_0: ;1:
        cmp     [sb1+1], al ; 0
                ; tstb sb1+1 / yes, is the device still doing I/O
                          ; / (inhibit bit set)?
        jna     short sysumnt_1
                ; bne 1b / yes, wait
        call    idle ; (wait for hardware interrupt)
        jmp     short sysumnt_0
sysumnt_1:
        mov     [mdev], al
                ; clr mntd / no, clear these
        mov     [mnti], ax
                ; clr mnti
         jmp    sysret
                ; br sysreta / return

getspl: ; / get device number from a special file name
        call    namei
        ;or     ax, ax ; Retro UNIX 8086 v1 modification !
                       ; ax = 0 -> file not found
         jc      sysmnt_err2 ; 'file not found !' error
        ;jz     error
        ;jc     error
                ; jsr r0,namei / get the i-number of the special file
                 ; br errora / no such file
         sub    ax, 3 ; Retro UNIX 8086 v1 modification !
                      ;i-number-3, 0 = fd0, 5 = hd3
                ; sub $4,r1 / i-number-4 rk=1,tap=2+n
         jc     short sysmnt_err0 ; 'permission denied !' error
        ;jc     error
                ; ble errora / less than 0?  yes, error
         cmp    ax, 5 ;
                ; cmp  r1,$9. / greater than 9  tap 7
        ja      short sysmnt_err0 ; 'permission denied !' error
        ;ja     error
                ; bgt errora / yes, error
         ; AX = Retro UNIX 8086 v1 Device Number (0 to 5)
iopen_retn:
        retn
                ; rts    r0 / return with device number in r1
```

```
sysmnt_err0:
        mov     dword [u.error], ERR_FILE_ACCESS ; permission denied !
        jmp     error
iopen:
        ; 19/05/2015
        ; 18/05/2015 (Retro UNIX 386 v1 - Beginning)
        ; 21/05/2013 - 27/08/2013 (Retro UNIX 8086 v1)
        ;
        ; open file whose i-number is in r1
        ;
        ; INPUTS ->
        ;    r1 - inode number
        ; OUTPUTS ->
        ;    file's inode in core
        ;    r1 - inode number (positive)
        ;
        ; ((AX = R1))
         ; ((Modified registers: edx, ebx, ecx, esi, edi, ebp))
        ;
; / open file whose i-number is in r1
        test    ah, 80h ; Bit 15 of AX
                ;tst r1 / write or read access?
         jnz    short iopen_2
                ;blt 2f / write, go to 2f
        mov     dl, 2 ; read access
        call    access
                ; jsr r0,access; 2
        ; / get inode into core with read access
        ; DL=2
iopen_0:
         cmp    ax, 40
                ; cmp r1,$40. / is it a special file
         ja     short iopen_retn
                ;bgt  3f / no. 3f
        push    ax
                ; mov r1,-(sp) / yes, figure out
        movzx   ebx, al
        shl     bx, 2
                ; asl r1
         add     ebx, iopen_1 - 4
        jmp     dword [ebx]
                ; jmp *1f-2(r1) / which one and transfer to it
iopen_1: ; 1:
        dd      otty ; tty, AX = 1 (runix)
                 ;otty / tty ; r1=2
                 ;oppt / ppt ; r1=4
        dd      sret ; mem, AX = 2 (runix)
                 ;sret / mem ; r1=6
                 ;sret / rf0
                 ;sret / rk0
                 ;sret / tap0
                 ;sret / tap1
                 ;sret / tap2
                 ;sret / tap3
                 ;sret / tap4
                 ;sret / tap5
                 ;sret / tap6
                 ;sret / tap7
         dd     sret ; fd0, AX = 3 (runix only)
         dd     sret ; fd1, AX = 4 (runix only)
         dd     sret ; hd0, AX = 5 (runix only)
         dd     sret ; hd1, AX = 6 (runix only)
         dd     sret ; hd2, AX = 7 (runix only)
         dd     sret ; hd3, AX = 8 (runix only)
        ;dd     error ; lpr, AX = 9 (error !)
         dd      sret ; lpr, AX = 9 (runix)
        dd      ocvt ; tty0, AX = 10 (runix)
                 ;ocvt / tty0
        dd      ocvt ; tty1, AX = 11 (runix)
                 ;ocvt / tty1
        dd      ocvt ; tty2, AX = 12 (runix)
                 ;ocvt / tty2
        dd      ocvt ; tty3, AX = 13 (runix)
                 ;ocvt / tty3
        dd      ocvt ; tty4, AX = 14 (runix)
                 ;ocvt / tty4
        dd      ocvt ; tty5, AX = 15 (runix)
                 ;ocvt / tty5
        dd      ocvt ; tty6, AX = 16 (runix)
```

```
                ;ocvt / tty6
        dd      ocvt ; tty7, AX = 17 (runix)
                ;ocvt / tty7
        dd      ocvt ; COM1, AX = 18 (runix only)
                ;error / crd
        dd      ocvt ; COM2, AX = 19 (runix only)

iopen_2: ; 2: / check open write access
        neg     ax
                ;neg r1 / make inode number positive
        mov     dl, 1 ; write access
        call    access
                ;jsr r0,access; 1 / get inode in core
        ; DL=1
        test    word [i.flgs], 4000h ; Bit 14 : Directory flag
                ;bit $40000,i.flgs / is it a directory?
        jz      short iopen_0
        ;mov    [u.error], ERR_DIR_ACCESS
        ;jmp    error ; permission denied !
        jmp     sysmnt_err0
        ;;jnz   error
                ; bne 2f / yes, transfer (error)
         ;;jmp     short iopen_0
        ;cmp    ax, 40
                ; cmp r1,$40. / no, is it a special file?
         ;ja    short iopen_2
                ;bgt 3f / no, return
        ;push   ax
                ;mov r1,-(sp) / yes
        ;movzx ebx, al
        ;shl    bx, 1
                ; asl r1
        ;add    ebx, ipen_3 - 2
        ;jmp    dword [ebx]
                ; jmp *1f-2(r1) / figure out
                        ; / which special file it is and transfer
;iopen_3: ; 1:
;       dd      otty ; tty, AX = 1 (runix)
                ;otty / tty ; r1=2
                ;leadr / ppt ; r1=4
;       dd      sret ; mem, AX = 2 (runix)
                ;sret / mem ; r1=6
                ;sret / rf0
                ;sret / rk0
                ;sret / tap0
                ;sret / tap1
                ;sret / tap2
                ;sret / tap3
                ;sret / tap4
                ;sret / tap5
                ;sret / tap6
                ;sret / tap7
;       dd      sret ; fd0, AX = 3 (runix only)
;       dd      sret ; fd1, AX = 4 (runix only)
;       dd      sret ; hd0, AX = 5 (runix only)
;       dd      sret ; hd1, AX = 6 (runix only)
;       dd      sret ; hd2, AX = 7 (runix only)
;       dd      sret ; hd3, AX = 8 (runix only)
;       dd      sret ; lpr, AX = 9  (runix)
        ;dd     ejec ; lpr, AX = 9  (runix)
;       dd      sret ; tty0, AX = 10 (runix)
                ;ocvt / tty0
;       dd      sret ; tty1, AX = 11 (runix)
                ;ocvt / tty1
;       dd      sret ; tty2, AX = 12 (runix)
                ;ocvt / tty2
;       dd      sret ; tty3, AX = 13 (runix)
                ;ocvt / tty3
;       dd      sret ; tty4, AX = 14 (runix)
                ;ocvt / tty4
;       dd      sret ; tty5, AX = 15 (runix)
                ;ocvt / tty5
;       dd      sret ; tty6, AX = 16 (runix)
                ;ocvt / tty6
;       dd      sret ; tty7, AX = 17 (runix)
                ;ocvt / tty7
;       dd      ocvt ; COM1, AX = 18 (runix only)
                ;/ ejec / lpr
;       dd      ocvt ; COM2, AX = 19 (runix only)
```

```
otty: ;/ open console tty for reading or writing
        ; 16/11/2015
        ; 12/11/2015
        ; 18/05/2015 (Retro UNIX 386 v1 - Beginning)
        ; 21/05/2013 - 13/07/2014 (Retro UNIX 8086 v1)
        ; 16/07/2013
        ; Retro UNIX 8086 v1 modification:
        ;  If a tty is open for read or write by
        ;     a process (u.uno), only same process can open
        ;     same tty to write or read (R->R&W or W->W&R).
        ;
        ; (INPUT: DL=2 for Read, DL=1 for Write, DL=0 for sysstty)
        ;
        movzx   ebx, byte [u.uno] ; process number
        mov     al, [ebx+p.ttyc-1] ; current/console tty
        ; 13/01/2014
        jmp     short ottyp
ocvt:
        sub     al, 10
ottyp:
        ; 16/11/2015
        ; 12/11/2015
        ; 18/05/2015 (32 bit modifications)
        ; 06/12/2013 - 13/07/2014
        mov     dh, al ; tty number
        movzx   ebx, al ; AL = tty number (0 to 9), AH = 0
        shl     bl, 1  ; aligned to word
        ;26/01/2014
        add     ebx, ttyl
        mov     cx, [ebx]
                   ; CL = lock value (0 or process number)
                   ; CH = open count
        and     cl, cl
        ; 13/01/2014
        jz      short otty_ret
        ;
        ; 16/11/2015
        cmp     cl, [u.uno]
        je      short ottys_3
        ;
        movzx   ebx, cl ; the process which has locked the tty
        shl     bl, 1
        mov     ax, [ebx+p.pid-2]
        ;movzx ebx, byte [u.uno]
        mov     bl, [u.uno]
        shl     bl, 1
        cmp     ax, [ebx+p.ppid-2]
        je      short ottys_3   ; 16/11/2015
        ;
        ; the tty is locked by another process
        ; except the parent process (p.ppid)
         ;
        mov      dword [u.error], ERR_DEV_ACCESS
                       ; permission denied ! error
otty_err: ; 13/01/2014
        or      dl, dl ; DL = 0 -> called by sysstty
        jnz     error
        stc
        retn
otty_ret:
        ; 13/01/2014
        cmp     dh, 7
        jna     short ottys_2
        ; 16/11/2015
com_port_check:
        mov     esi, com1p
        cmp     dh, 8   ; COM1 (tty8) ?
        jna     short ottys_1 ; yes, it is COM1
        inc     esi     ; no, it is COM2 (tty9)
ottys_1:
        ; 12/11/2015
        cmp     byte [esi], 0 ; E3h (or 23h)
        ja      short com_port_ready
        ;
         mov     dword [u.error], ERR_DEV_NOT_RDY
                          ; device not ready ! error
        jmp     short otty_err
```

```
com_port_ready:
ottys_2:
        or      cl, cl  ; cl = lock/owner, ch = open count
        jnz     short ottys_3
        mov     cl, [u.uno]
ottys_3:
        inc     ch
        mov     [ebx], cx ; set tty lock again
        ; 06/12/2013
        inc     dh ; tty number + 1
        mov     ebx, u.ttyp
        ; 13/01/2014
        test    dl, 2 ; open for read sign
        jnz     short ottys_4
        inc     ebx
ottys_4:
        ; Set 'u.ttyp' ('the recent TTY') value
        mov     [ebx], dh ; tty number + 1
sret:
        or      dl, dl ; sysstty system call check (DL=0)
        jz      short iclose_retn
        pop     ax
iclose_retn:
        retn

        ;
        ; Original UNIX v1 'otty' routine:
        ;
        ;mov    $100,*$tks / set interrupt enable bit (zero others) in
         ;                 / reader status reg
         ;mov    $100,*$tps / set interrupt enable bit (zero others) in
         ;                 / punch status reg
         ;mov    tty+[ntty*8]-8+6,r5 / r5 points to the header of the
         ;                          / console tty buffer
         ;incb   (r5) / increment the count of processes that opened the
         ;            / console tty
         ;tst    u.ttyp / is there a process control tty (i.e., has a tty
         ;              / buffer header
         ;bne    sret / address been loaded into u.ttyp yet)?  yes, branch
         ;mov    r5,u.ttyp / no, make the console tty the process control
         ;                 / tty
         ;br     sret / ?
;sret:
                ;clr *$ps / set processor priority to zero
;       pop     ax
                ;mov (sp)+,r1 / pop stack to r1
;3:
;       retn
                ;rts r0

;ocvt:  ; < open tty >
        ; 13/01/2014
        ; 06/12/2013 (major modification: p.ttyc, u.ttyp)
        ; 24/09/2013 consistency check -> ok
        ; 16/09/2013
        ; 03/09/2013
        ; 27/08/2013
        ; 16/08/2013
        ; 16/07/2013
        ; 27/05/2013
        ; 21/05/2013
        ;
        ; Retro UNIX 8086 v1 modification !
        ;
        ; In original UNIX v1, 'ocvt' routine
        ;               (exactly different than this one)
        ;       was in 'u9.s' file.
        ;
        ; 16/07/2013
        ; Retro UNIX 8086 v1 modification:
        ;  If a tty is open for read or write by
        ;     a process (u.uno), only same process can open
        ;     same tty to write or read (R->R&W or W->W&R).
        ;
        ; INPUT: DL=2 for Read DL=1 for Write

        ; 16/09/2013
        ; sub   al, 10
```

```
        ; 06/12/2013
        ;cmp    al, 7
         ;jna     short ottyp
        ; 13/01/2014
        ;jmp    short ottyp


;oppt: / open paper tape for reading or writing
;         mov    $100,*$prs / set reader interrupt enable bit
;         tstb   pptiflg / is file already open
;         bne    2f / yes, branch
;1:
;         mov    $240,*$ps / no, set processor priority to 5
;         jsr    r0,getc; 2 / remove all entries in clist
;                br .+4 / for paper tape input and place in free list
;         br     1b
;         movb   $2,pptiflg / set pptiflg to indicate file just open
;         movb   $10.,toutt+1 / place 10 in paper tape input tout entry
;         br     sret
;2:
;         jmp    error / file already open

iclose:
        ; 19/05/2015
        ; 18/05/2015 (Retro UNIX 386 v1 - Beginning)
        ; 21/05/2013 - 13/01/2014 (Retro UNIX 8086 v1)
        ;
        ; close file whose i-number is in r1
        ;
        ; INPUTS ->
        ;    r1 - inode number
        ; OUTPUTS ->
        ;    file's inode in core
        ;    r1 - inode number (positive)
        ;
        ; ((AX = R1))
         ;    ((Modified registers: -ebx-, edx))
        ;
;/ close file whose i-number is in r1
        mov     dl, 2 ; 12/01/2014
        test    ah, 80h ; Bit 15 of AX
                ;tst r1 / test i-number
         ;jnz   short iclose_2
                ;blt 2f / if neg., branch
        jz      short iclose_0 ; 30/07/2013
        ; 16/07/2013
        neg     ax ; make it positive
        ; 12/01/2014
        dec     dl ; dl = 1 (open for write)
iclose_0:
        cmp     ax, 40
                ;cmp r1,$40. / is it a special file
         ja     short iclose_retn  ; 13/01/2014
                ;bgt 3b / no, return
        ; 12/01/2014
        ; DL=2 -> special file was opened for reading
        ; DL=1 -> special file was opened for writing
        push    ax
                ;mov r1,-(sp) / yes, save r1 on stack
        movzx   ebx, al
        shl     bx, 2
                ; asl r1
        add     ebx, iclose_1 - 4
        jmp     dword [ebx]
                ; jmp *1f-2(r1) / compute jump address and transfer
iclose_1 :
        dd      ctty ; tty, AX = 1 (runix)
        dd      cret ; mem, AX = 2 (runix)
        dd      cret ; fd0, AX = 3 (runix only)
        dd      cret ; fd1, AX = 4 (runix only)
        dd      cret ; hd0, AX = 5 (runix only)
        dd      cret ; hd1, AX = 6 (runix only)
        dd      cret ; hd2, AX = 7 (runix only)
        dd      cret ; hd3, AX = 8 (runix only)
        dd      cret ; lpr, AX = 9 (runix)
        ;dd     error; lpr, AX = 9 (error !)
        ;;dd    offset ejec ;;lpr, AX = 9
        dd      ccvt ; tty0, AX = 10 (runix)
        dd      ccvt ; tty1, AX = 11 (runix)
```

```
        dd      ccvt ; tty2, AX = 12 (runix)
        dd      ccvt ; tty3, AX = 13 (runix)
        dd      ccvt ; tty4, AX = 14 (runix)
        dd      ccvt ; tty5, AX = 15 (runix)
        dd      ccvt ; tty6, AX = 16 (runix)
        dd      ccvt ; tty7, AX = 17 (runix)
        dd      ccvt ; COM1, AX = 18 (runix only)
        dd      ccvt ; COM2, AX = 19 (runix only)

        ; 1:
        ;        ctty   / tty
        ;        cppt   / ppt
        ;        sret   / mem
        ;        sret   / rf0
        ;        sret   / rk0
        ;        sret   / tap0
        ;        sret   / tap1
        ;        sret   / tap2
        ;        sret   / tap3
        ;        sret   / tap4
        ;        sret   / tap5
        ;        sret   / tap6
        ;        sret   / tap7
        ;        ccvt   / tty0
        ;        ccvt   / tty1
        ;        ccvt   / tty2
        ;        ccvt   / tty3
        ;        ccvt   / tty4
        ;        ccvt   / tty5
        ;        ccvt   / tty6
        ;        ccvt   / tty7
        ;        error / crd

;iclose_2: ; 2: / negative i-number
        ;neg    ax
                ;neg r1 / make it positive
        ;cmp    ax, 40
                ;cmp r1,$40. / is it a special file?
         ;ja    short @b
                ;bgt    3b / no. return
        ;push   ax
                ;mov r1,-(sp)
        ;movzx  ebx, al
        ;shl    bx, 1
                ;asl r1 / yes. compute jump address and transfer
        ;add    ebx, iclose_3 - 2
        ;jmp    dword [ebx]
                ;jmp *1f-2(r1) / figure out
;iclose_3:
        ;dd     ctty ; tty, AX = 1 (runix)
        ;dd     sret ; mem, AX = 2 (runix)
        ;dd     sret ; fd0, AX = 3 (runix only)
        ;dd     sret ; fd1, AX = 4 (runix only)
        ;dd     sret ; hd0, AX = 5 (runix only)
        ;dd     sret ; hd1, AX = 6 (runix only)
        ;dd     sret ; hd2, AX = 7 (runix only)
        ;dd     sret ; hd3, AX = 8 (runix only)
         ;dd    sret ; lpr, AX = 9
        ;dd     ejec ; lpr, AX = 9  (runix)
        ;dd     ccvt ; tty0, AX = 10 (runix)
        ;dd     ccvt ; tty1, AX = 11 (runix)
        ;dd     ccvt ; tty2, AX = 12 (runix)
        ;dd     ccvt ; tty3, AX = 13 (runix)
        ;dd     ccvt ; tty4, AX = 14 (runix)
        ;dd     ccvt ; tty5, AX = 15 (runix)
        ;dd     ccvt ; tty6, AX = 16 (runix)
        ;dd     ccvt ; tty7, AX = 17 (runix)
        ;dd     ccvt ; COM1, AX = 18 (runix only)
        ;dd     ccvt ; COM2, AX = 19 (runix only)

        ;1:
        ;       ctty   / tty
        ;        leadr  / ppt
        ;        sret   / mem
        ;        sret   / rf0
        ;        sret   / rk0
        ;        sret   / tap0
        ;        sret   / tap1
        ;        sret   / tap2
```

```
	;	sret	/ tap3
	;	sret	/ tap4
	;	sret	/ tap5
	;	sret	/ tap6
	;	sret	/ tap7
	;	ccvt	/ tty0
	;	ccvt	/ tty1
	;	ccvt	/ tty2
	;	ccvt	/ tty3
	;	ccvt	/ tty4
	;	ccvt	/ tty5
	;	ccvt	/ tty6
	;	ccvt	/ tty7
	;/	 ejec / lpr

ctty: ; / close console tty
	; 18/05/2015 (Retro UNIX 386 v1 - Beginning)
	; 21/05/2013 - 26/01/2014 (Retro UNIX 8086 v1)
	;
	; Retro UNIX 8086 v1 modification !
	; (DL = 2 -> it is open for reading)
	; (DL = 1 -> it is open for writing)
	; (DL = 0 -> it is open for sysstty system call)
	;
	; 06/12/2013
	 movzx	 ebx, byte [u.uno] ; process number
	 mov	 al, [ebx+p.ttyc-1]
	; 13/01/2014
	jmp	short cttyp
ccvt:
	sub	al, 10
cttyp:
	; 18/05/2015 (32 bit modifications)
	; 16/08/2013 - 26/01/2014
	movzx	ebx, al ; tty number (0 to 9)
	shl	bl, 1  ; aligned to word
	; 26/01/2014
	add	ebx, ttyl
	mov	dh, al ; tty number
	mov	ax, [ebx]
		  ; AL = lock value (0 or process number)
		  ; AH = open count
	and	ah, ah
	jnz	short ctty_ret
	 mov	 dword [u.error], ERR_DEV_NOT_OPEN
			; device not open ! error
	;jmp	short ctty_err ; open count = 0, it is not open !
	jmp	error
	; 26/01/2014
ctty_ret:
	dec	ah ; decrease open count
	jnz	short ctty_1
	xor	al, al ; unlock/free tty
ctty_1:
	mov	[ebx], ax ; close tty instance
	;
	mov	ebx, u.ttyp
	test	dl, 1 ; open for write sign
	jz	short ctty_2
	inc	ebx
ctty_2:
	inc	dh ; tty number + 1
	cmp	dh, [ebx]
	jne	short cret
	; Reset/Clear 'u.ttyp' ('the recent TTY') value
	mov	byte [ebx], 0
cret:
	or	dl, dl ; sysstty system call check (DL=0)
	jz	short ctty_3
	pop	ax
ctty_3:
	retn

;ctty_err: ; 13/01/2014
;	or	dl, dl ; DL = 0 -> called by sysstty
;	jnz	error
;	stc
;	retn
```

```
        ; Original UNIX v1 'ctty' routine:
        ;
         ;mov    tty+[ntty*8]-8+6,r5
        ;               ;/ point r5 to the console tty buffer
         ;decb   (r5) / dec number of processes using console tty
         ;br     sret / return via sret

;ccvt:  ; < close tty >
        ; 21/05/2013 - 13/01/2014 (Retro UNIX 8086 v1)
        ;
        ; Retro UNIX 8086 v1 modification !
        ;
        ; In original UNIX v1, 'ccvt' routine
        ;               (exactly different than this one)
        ;       was in 'u9.s' file.
        ;
        ; DL = 2 -> it is open for reading
        ; DL = 1 -> it is open for writing
        ;
        ; 17/09/2013
        ;sub    al, 10
        ;cmp    al, 7
        ;jna    short cttyp
        ; 13/01/2014
        ;jmp    short cttyp

;cppt: / close paper tape
;        clrb   pptiflg / set pptiflg to indicate file not open
;1:
;        mov    $240,*$ps /set process or priority to 5
;        jsr    r0,getc; 2 / remove all ppt input entries from clist
;                          / and assign to free list
;               br sret
;        br     1b

;ejec:
;       jmp     error
;/ejec:
;/       mov    $100,*$lps / set line printer interrupt enable bit
;/       mov    $14,r1 / 'form feed' character in r1 (new page).
;/       jsr    r0,lptoc / space the printer to a new page
;/       br     sret / return to caller via 'sret'
```

```
; Retro UNIX 386 v1 Kernel (v0.2) - SYS8.INC
; Last Modification: 20/08/2015
; ----------------------------------------------------------------------------
; Derived from 'Retro UNIX 8086 v1' source code by Erdogan Tan
; (v0.1 - Beginning: 11/07/2012)
;
; Derived 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>
;
; Retro UNIX 8086 v1 - U8.ASM (18/01/2014) //// UNIX v1 -> u8.s
;
; ****************************************************************************

;; I/O Buffer - Retro UNIX 386 v1 modification
;;     (8+512 bytes, 8 bytes header, 512 bytes data)
;; Word 1, byte 0 = device id
;; Word 1, byte 1 = status bits (bits 8 to 15)
;;          bit 9 = write bit
;;          bit 10 = read bit
;;          bit 12 = waiting to write bit
;;          bit 13 = waiting to read bit
;;          bit 15 = inhibit bit
;; Word 2 (byte 2 & byte 3) = reserved (for now - 07/06/2015)
;; Word 3 + Word 4 (byte 4,5,6,7) = physical block number
;;                (In fact, it is 32 bit LBA for Retro UNIX 386 v1)
;;
;; I/O Buffer ((8+512 bytes in original Unix v1))
;;            ((4+512 bytes in Retro UNIX 8086 v1))
;;
;; I/O Queue Entry (of original UNIX operating system v1)
;; Word 1, Byte 0 = device id
;; Word 1, Byte 1 = (bits 8 to 15)
;;          bit 9 = write bit
;;          bit 10 = read bit
;;          bit 12 = waiting to write bit
;;          bit 13 = waiting to read bit
;;          bit 15 = inhibit bit
;; Word 2 = physical block number (In fact, it is LBA for Retro UNIX 8086 v1)
;;
;; Original UNIX v1 ->
;;             Word 3 = number of words in buffer (=256)
;; Original UNIX v1 ->
;;             Word 4 = bus address (addr of first word of data buffer)
;;
;; Retro UNIX 8086 v1 -> Buffer Header (I/O Queue Entry) size is 4 bytes !
;;
;; Device IDs (of Retro Unix 8086 v1)
;;          0 = fd0
;;          1 = fd1
;;          2 = hd0
;;          3 = hd1
;;          4 = hd2
;;          5 = hd3

; Retro UNIX 386 v1 - 32 bit modifications (rfd, wfd, rhd, whd) - 09/06/2015

rfd:    ; 09/06/2015 (Retro UNIX 386 v1 - Beginning)
        ; 26/04/2013
        ; 13/03/2013 Retro UNIX 8086 v1 device (not an original unix v1 device)
        ;sub    ax, 3 ; zero based device number (Floppy disk)
        ;jmp    short bread ; **** returns to routine that called readi

rhd:    ; 09/06/2015 (Retro UNIX 386 v1 - Beginning)
        ; 26/04/2013
        ; 14/03/2013 Retro UNIX 8086 v1 device (not an original unix v1 device)
        ;sub    ax, 3 ; zero based device number (Hard disk)
        ;jmp    short bread ; **** returns to routine that called readi
```

```
bread:
        ; 14/07/2015
        ; 10/07/2015
        ; 09/06/2015
        ; 07/06/2015 (Retro UNIX 386 v1 - Beginning)
        ; 13/03/2013 - 29/07/2013 (Retro UNIX 8086 v1)
        ;
        ; / read a block from a block structured device
        ;
        ; INPUTS ->
        ;    [u.fopf] points to the block number
        ;    CX = maximum block number allowed on device
        ;        ; that was an arg to bread, in original Unix v1, but
        ;        ; CX register is used instead of arg in Retro Unix 8086 v1
        ;    [u.count] number of bytes to read in
        ; OUTPUTS ->
        ;    [u.base] starting address of data block or blocks in user area
        ;    [u.fopf] points to next consecutive block to be read
        ;
        ; ((Modified registers: eAX, eDX, eCX, eBX, eSI, eDI, eBP))
        ;
        ; NOTE: Original UNIX v1 has/had a defect/bug here, even if read
        ;       byte count is less than 512, block number in *u.fofp (u.off)
        ;       is increased by 1. For example: If user/program request
        ;       to read 16 bytes in current block, 'sys read' increases
        ;       the next block number just as 512 byte reading is done.
        ;       This wrong is done in 'bread'. So, in Retro UNIX 8086 v1,
        ;       for user (u) structure compatibility (because 16 bit is not
        ;       enough to keep byte position/offset of the disk), this
        ;       defect will not be corrected, user/program must request
        ;       512 byte read per every 'sys read' call to block devices
        ;       for achieving correct result. In future version(s),
        ;       this defect will be corrected by using different
        ;       user (u) structure.  26/07/2013 - Erdogan Tan

                ; jsr r0,tstdeve / error on special file I/O
                                ; / (only works on tape)
                ; mov *u.fofp,r1 / move block number to r1
                ; mov $2.-cold,-(sp) / "2-cold" to stack
;1:
                ; cmp r1,(r0) / is this block # greater than or equal to
                            ; / maximum block # allowed on device
                ; jnb short @f
                ; bhis 1f / yes, 1f (error)
                ; mov r1,-(sp) / no, put block # on stack
                ; jsr r0,preread / read in the block into an I/O buffer
                ; mov (sp)+,r1 / return block # to r1
                ; inc r1 / bump block # to next consecutive block
                ; dec (sp) / "2-1-cold" on stack
                ; bgt 1b / 2-1-cold = 0?  No, go back and read in next block
;1:
                ; tst (sp)+ / yes, pop stack to clear off cold calculation
        ;push   ecx ; **
        ;26/04/2013
        ;sub    ax, 3 ; 3 to 8 -> 0 to 5
        sub     al, 3
                ; AL = Retro Unix 8086 v1 disk (block device) number
        mov     [u.brwdev], al
        ; 09/06/2015
        movzx   ebx, al
        mov     ecx, [ebx+drv.size] ; disk size (in sectors)
bread_0:
        push    ecx ; ** ; 09/06/2015
        ; 10/07/2015 (Retro UNIX 386 v1 modification!)
        ; [u.fopf] points to byte position in disk, not sector/block !
        mov     ebx, [u.fofp]
        mov     eax, [ebx]
        shr     eax, 9 ; convert byte position to block/sector number
                ; mov *u.fofp,r1 / restore r1 to initial value of the
                                ; / block #
        cmp     eax, ecx
                ; cmp r1,(r0)+ / block # greater than or equal to maximum
                                ; / block number allowed
        ;jnb    error        ; 18/04/2013
                ; bhis error10 / yes, error
        jb      short bread_1
        mov     dword [u.error], ERR_DEV_VOL_SIZE  ; 'out of volume' error
        jmp     error
```

```
bread_1:
        ; inc   dword [ebx] ; 10/07/2015 (Retro UNIX 386 v1 - modification!)
                ; inc *u.fofp / no, *u.fofp has next block number
        ; eAX = Block number (zero based)
                ;;jsr r0,preread / read in the block whose number is in r1
preread: ;; call preread
        mov     edi, u.brwdev ; block device number for direct I/O
        call    bufaloc_0 ; 26/04/2013
        ;; jc   error
        ; eBX = Buffer (Header) Address -Physical-
         ; eAX = Block/Sector number (r1)
                ; jsr r0,bufaloc / get a free I/O buffer (r1 has block number)
        ; 14/03/2013
         jz     short bread_2 ; Retro UNIX 8086 v1 modification
                ; br 1f / branch if block already in a I/O buffer
        or      word [ebx], 400h ; set read bit (10) in I/O Buffer
                ; bis $2000,(r5) / set read bit (bit 10 in I/O buffer)
        call    poke
                ; jsr r0,poke / perform the read
        ;;jc    error ;2 0/07/2013
; 1:
                ; clr *$ps / ps = 0
                ; rts r0
;; return from preread
bread_2:
        or      word [ebx], 4000h
                ; bis $40000,(r5)
                        ; / set bit 14 of the 1st word of the I/O buffer
bread_3: ; 1:
        test    word [ebx], 2400h
                ; bit $22000,(r5) / are 10th and 13th bits set (read bits)
        jz      short bread_4
                ; beq 1f / no
                ; cmp cdev,$1 / disk or drum?
                ; ble 2f / yes
                ; tstb uquant / is the time quantum = 0?
                ; bne 2f / no, 2f
                ; mov r5,-(sp) / yes, save r5 (buffer address)
                ; jsr r0,sleep; 31.
                        ; / put process to sleep in channel 31 (tape)
                ; mov (sp)+,r5 / restore r5
                ; br 1b / go back
; 2: / drum or disk
         ;; mov     cx, [s.wait_]+2 ;; 29/07/2013
        call    idle
                ; jsr r0,idle; s.wait+2 / wait
        jmp     short bread_3
                ; br 1b
bread_4: ; 1: / 10th and 13th bits not set
        and     word [ebx], 0BFFFh ; 1011111111111111b
                ; bic $40000,(r5) / clear bit 14
                ; jsr r0,tstdeve / test device for error (tape)
        add     ebx, 8
                ; add $8,r5 / r5 points to data in I/O buffer
        ; 09/06/2015
        cmp     word [u.pcount], 0
        ja      short bread_5
        call    trans_addr_w ; translate virtual address to physical (w)
bread_5:
        ; eBX = system (I/O) buffer address
        call    dioreg
                ; jsr r0,dioreg / do bookkeeping on u.count etc.
        ; esi =  start address of the transfer (in the buffer)
        ; edi =  [u.pbase], destination address in user's memory space
        ; ecx =  transfer count (in bytes)
        ;
;1: / r5 points to beginning of data in I/O buffer, r2 points to beginning
;   / of users data
        rep     movsb
                ; movb (r5)+,(r2)+ / move data from the I/O buffer
                ; dec r3 / to the user's area in core starting at u.base
                ; bne 1b
        pop     ecx ; **
        cmp     dword [u.count], 0
                ; tst u.count / done
        ja      short bread_0 ; 09/06/2015
                ; beq 1f / yes, return
                ; tst -(r0) / no, point r0 to the argument again
                ; br bread / read some more
```

```
; 1:
        pop     eax ; ****
                ; mov (sp)+,r0
         retn           ; 09/06/2015
        ;jmp     ret_
                ;jmp ret  / jump to routine that called readi

wfd:     ; 09/06/2015 (Retro UNIX 386 v1 - Beginning)
        ; 26/04/2013
        ; 14/03/2013 Retro UNIX 8086 v1 device (not an original unix v1 device)
        ;sub    ax, 3 ; zero based device number (Hard disk)
        ;jmp    short bwrite ; **** returns to routine that called writei

whd:    ; 09/06/2015 (Retro UNIX 386 v1 - Beginning)
        ; 14/03/2013 Retro UNIX 8086 v1 device (not an original unix v1 device)
        ;sub    ax, 3 ; zero based device number (Hard disk)
        ;jmp    short bwrite ; **** returns to routine that called writei ('jmp ret')

bwrite:
        ; 14/07/2015
        ; 10/07/2015
        ; 09/06/2015 (Retro UNIX 386 v1 - Beginning)
        ; 14/03/2013 - 20/07/2013 (Retro UNIX 8086 v1)
        ;
        ;; / write on block structured device
        ;
        ; INPUTS ->
        ;    [u.fopf] points to the block number
        ;    CX = maximum block number allowed on device
        ;        ; that was an arg to bwrite, in original Unix v1, but
        ;        ; CX register is used instead of arg in Retro Unix 8086 v1
        ;    [u.count] number of bytes to user desires to write
        ; OUTPUTS ->
        ;    [u.fopf] points to next consecutive block to be written into
        ;
        ; ((Modified registers: eDX, eCX, eBX, eSI, eDI, eBP))
        ;
        ; NOTE: Original UNIX v1 has/had a defect/bug here, even if write
        ;        byte count is less than 512, block number in *u.fofp (u.off)
        ;       is increased by 1. For example: If user/program request
        ;        to write 16 bytes in current block, 'sys write' increases
        ;       the next block number just as 512 byte writing is done.
        ;        This wrong is done in 'bwrite'. So, in Retro UNIX 8086 v1,
        ;        for user (u) structure compatibility (because 16 bit is not
        ;        enough to keep byte position/offset of the disk), this
        ;       defect will not be corrected, user/program must request
        ;       512 byte write per every 'sys write' call to block devices
        ;        for achieving correct result. In future version(s),
        ;       this defect will be corrected by using different
        ;        user (u) structure.  26/07/2013 - Erdogan Tan

                ; jsr r0,tstdeve / test the device for an error
        ;push   ecx ; **
        ;26/04/2013
        ;sub    ax, 3 ; 3 to 8 -> 0 to 5
        sub     al, 3
                ; AL = Retro Unix 8086 v1 disk (block device) number
        mov     [u.brwdev], al
        ; 09/06/2015
        movzx   ebx, al
        mov     ecx, [ebx+drv.size] ; disk size (in sectors)
bwrite_0:
        push    ecx ; ** ; 09/06/2015
        ; 10/07/2015 (Retro UNIX 386 v1 modification!)
        ; [u.fopf] points to byte position in disk, not sector/block !
        mov     ebx, [u.fofp]
        mov     eax, [ebx]
        shr     eax, 9 ; convert byte position to block/sector number
                ; mov *u.fofp,r1 / put the block number in r1
        cmp     eax, ecx
                ; cmp r1,(r0)+ / does block number exceed maximum allowable #
                              ; / block number allowed
        ;jnb    error       ; 18/04/2013
                ; bhis error10 / yes, error
        jb      short bwrite_1
        mov     dword [u.error], ERR_DEV_VOL_SIZE  ; 'out of volume' error
        jmp     error
```

```
bwrite_1:
        ; inc   dword [ebx] ; 10/07/2015 (Retro UNIX 386 v1 - modification!)
                ; inc *u.fofp / no, increment block number
        ; 09/06/2015 - 10/07/2015
        cmp     word [u.pcount], 0
        ja      short bwrite_2
        call    trans_addr_r ; translate virtual address to physical (r)
bwrite_2:
        mov     edi, u.brwdev  ; block device number for direct I/O
        call    bwslot ; 26/04/2013 (wslot -> bwslot)
                ; jsr r0,wslot / get an I/O buffer to write into
                ; add $8,r5 / r5 points to data in I/O buffer
         call   dioreg
                ; jsr r0,dioreg / do the necessary bookkeeping
        ; esi =  destination address (in the buffer)
        ; edi =  [u.pbase], start address of transfer in user's memory space
        ; ecx =  transfer count (in bytes)
; 1: / r2 points to the users data; r5 points to the I/O buffers data area
        xchg    esi, edi ; 14/07/2015
        rep     movsb
                ; movb (r2)+,(r5)+ / ; r3, has the byte count
                ; dec r3 / area to the I/O buffer
                ; bne 1b
        call    dskwr
                ; jsr r0,dskwr / write it out on the device
        pop     ecx ; **
         cmp     dword [u.count], 0
                ; tst u.count / done
        ja      short bwrite_0 ; 09/06/2015
                ; beq 1f / yes, 1f
                ; tst -(r0) / no, point r0 to the argument of the call
                ; br bwrite / go back and write next block
; 1:
        pop     eax ; ****
                ; mov (sp)+,r0
        retn            ; 09/06/2015
         ;jmp     ret_
                ; jmp ret / return to routine that called writei
;error10:
;        jmp     error  ; / see 'error' routine

dioreg:
        ; 14/07/2015
        ; 10/07/2015 (UNIX v1 bugfix - [u.fofp]: byte pos., not block)
        ; 09/06/2015 (Retro UNIX 386 v1 - Beginning)
        ; 14/03/2013 (Retro UNIX 8086 v1)
        ;
        ; bookkeeping on block transfers of data
        ;
        ; * returns value of u.pbase before it gets updated, in EDI
        ; * returns byte count (to transfer) in ECX (<=512)
        ; 10/07/2015
        ; * returns byte offset from beginning of current sector buffer
        ; (beginning of data) in ESI
        ;
        mov     ecx, [u.count]
                ; mov u.count,r3 / move char count to r3
        cmp     ecx, 512
                ; cmp r3,$512. / more than 512. char?
        jna     short dioreg_0
                ; blos 1f / no, branch
        mov     ecx, 512
                ; mov $512.,r3 / yes, just take 512.
dioreg_0:
        ; 09/06/2015
        cmp     cx, [u.pcount]
        jna     short dioreg_1
        mov     cx, [u.pcount]
dioreg_1:
; 1:
        mov     edx, [u.base] ; 09/06/2015 (eax -> edx)
                 ; mov u.base,r2 / put users base in r2
        add     [u.nread], ecx
                ; add r3,u.nread / add the number to be read to u.nread
        sub     [u.count], ecx
                ; sub r3,u.count / update count
        add     [u.base], ecx
                ; add r3,u.base / update base
```

```
        ; 10/07/2015
        ; Retro UNIX 386 v1 - modification !
        ; (File pointer points to byte position, not block/sector no.)
        ; (It will point to next byte position instead of next block no.)
        mov     esi, [u.fofp] ; u.fopf points to byte position pointer
        mov     eax, [esi] ; esi points to current byte pos. on the disk
        add     [esi], ecx ; ecx is added to set the next byte position
        and     eax, 1FFh  ; get offset from beginning of current block
        mov     esi, ebx   ; beginning of data in sector/block buffer
        add     esi, eax   ; esi contains start address of the transfer
        ; 09/06/2015 - 10/07/2015
        sub     [u.pcount], cx
        and     edx, PAGE_OFF ; 0FFFh
        mov     edi, [u.pbase]
        and     edi, ~PAGE_OFF
        add     edi, edx
        mov     [u.pbase], edi
        add     [u.pbase], ecx ; 14/07/2015
        retn
                ; rts r0 / return

dskrd:
        ; 18/08/2015
        ; 02/07/2015
        ; 09/06/2015 (Retro UNIX 386 v1 - Beginning)
        ; 14/03/2013 - 29/07/2013 (Retro UNIX 8086 v1)
        ;
        ; 'dskrd' acquires an I/O buffer, puts in the proper
        ; I/O queue entries (via bufaloc) then reads a block
        ; (number specified in r1) in the acquired buffer.)
        ; If the device is busy at the time dskrd is called,
        ; dskrd calls idle.
        ;
        ; INPUTS ->
        ;    r1 - block number
        ;    cdev - current device number
        ; OUTPUTS ->
        ;    r5 - points to first data word in I/O buffer
        ;
        ; ((AX = R1)) input/output
        ; ((BX = R5)) output
        ;
         ; ((Modified registers: eDX, eCX, eBX, eSI, eDI, eBP))
        ;
        call    bufaloc
                ; jsr r0,bufaloc / shuffle off to bufaloc;
                               ; / get a free I/O buffer
        ;;jc    error ; 20/07/2013
        jz      short dskrd_1 ; Retro UNIX 8086 v1 modification
                ; br 1f / branch if block already in a I/O buffer
dskrd_0: ; 10/07/2015 (wslot)
        or      word [ebx], 400h ; set read bit (10) in I/O Buffer
                ; bis $2000,(r5) / set bit 10 of word 1 of
                               ; / I/O queue entry for buffer
        call    poke
                ; jsr r0,poke / just assigned in bufaloc,
                           ; /bit 10=1 says read
        ; 09/06/2015
        jnc     short dskrd_1
        mov     dword [u.error], ERR_DRV_READ ; disk read error !
        jmp     error
dskrd_1: ; 1:
                ;clr *$ps
        test    word [ebx], 2400h
                ; bit $22000,(r5) / if either bits 10, or 13 are 1;
                               ; / jump to idle
        jz      short dskrd_2
                ; beq 1f
         ;;mov   ecx, [s.wait_]
        call    idle
                ; jsr r0,idle; s.wait+2
        jmp     short dskrd_1
                ; br 1b
dskrd_2: ; 1:
         add    ebx, 8
                ; add $8,r5 / r5 points to first word of data in block
                         ; / just read in
        retn
                ; rts r0
```

```
bwslot:
        ; 10/07/2015
        ;       If the block/sector is not placed in a buffer
        ;       before 'wslot', it must be read before
        ;       it is written! (Otherwise transfer counts less
        ;       than 512 bytes will be able to destroy existing
        ;       data on disk.)
        ;
        ; 11/06/2015 (Retro UNIX 386 v1 - Beginning)
        ; 26/04/2013(Retro UNIX 8086 v1)
        ; Retro UNIX 8086 v1 modification !
        ; ('bwslot' will be called from 'bwrite' only!)
        ; INPUT -> eDI - points to device id (in u.brwdev)
        ;       -> eAX = block number
        ;
        call    bufaloc_0
        jz      short wslot_0 ; block/sector already is in the buffer
bwslot_0:
        ; 10/07/2015
        mov     esi, [u.fofp]
        mov     eax, [esi]
        and     eax, 1FFh ; offset from beginning of the sector/block
        jnz     short bwslot_1 ; it is not a full sector write
                       ; recent disk data must be placed in the buffer
        cmp     dword [u.count], 512
        jnb     short wslot_0
bwslot_1:
        call    dskrd_0
        sub     ebx, 8 ; set ebx to the buffer header address again
        jmp     short wslot_0

wslot:
        ; 11/06/2015 (Retro UNIX 386 v1 - Beginning)
        ;            (32 bit modifications)
        ; 14/03/2013 - 29/07/2013(Retro UNIX 8086 v1)
        ;
        ; 'wslot' calls 'bufaloc' and obtains as a result, a pointer
        ; to the I/O queue of an I/O buffer for a block structured
        ; device. It then checks the first word of I/O queue entry.
        ; If bits 10 and/or 13 (read bit, waiting to read bit) are set,
        ; wslot calls 'idle'. When 'idle' returns, or if bits 10
        ; and/or 13 are not set, 'wslot' sets bits 9 and 15 of the first
        ; word of the I/O queue entry (write bit, inhibit bit).
        ;
        ; INPUTS ->
        ;    r1 - block number
        ;    cdev - current (block/disk) device number
        ;
        ; OUTPUTS ->
        ;    bufp - bits 9 and 15 are set,
        ;           the remainder of the word left unchanged
        ;    r5 - points to first data word in I/O buffer
        ;
        ; ((AX = R1)) input/output
        ; ((BX = R5)) output
        ;
         ; ((Modified registers: eDX, eCX, eBX, eSI, eDI, eBP))

        call    bufaloc
        ; 10/07/2015
                ; jsr r0,bufaloc / get a free I/O buffer; pointer to first
                ; br 1f / word in buffer in r5
        ; eBX = Buffer (Header) Address (r5) (ES=CS=DS, system/kernel segment)
         ; eAX = Block/Sector number (r1)
wslot_0: ;1:
        test    word [ebx], 2400h
                ; bit $22000,(r5) / check bits 10, 13 (read, waiting to read)
                               ; / of I/O queue entry
        jz      short wslot_1
                 ; beq 1f  / branch if 10, 13 zero (i.e., not reading,
                     ; / or not waiting to read)

         ;; mov     ecx, [s.wait_] ; 29/07/2013
        call    idle
                ; jsr r0,idle; / if buffer is reading or writing to read,
                              ; / idle
        jmp     short wslot_0
                ; br 1b / till finished
```

```
wslot_1: ;1:
         or      word [ebx], 8200h
                ; bis $101000,(r5) / set bits 9, 15 in 1st word of I/O queue
                                ; / (write, inhibit bits)
                ; clr     *$ps / clear processor status
         add    ebx, 8 ; 11/06/2015
                ; add $8,r5 / r5 points to first word in data area
                          ; / for this block
        retn
                ; rts r0
dskwr:
        ; 09/06/2015 (Retro UNIX 386 v1 - Beginning)
        ; 14/03/2013 - 03/08/2013 (Retro UNIX 8086 v1)
        ;
        ; 'dskwr' writes a block out on disk, via ppoke. The only
        ; thing dskwr does is clear bit 15 in the first word of I/O queue
        ; entry pointed by 'bufp'. 'wslot' which must have been called
        ; previously has supplied all the information required in the
        ; I/O queue entry.
        ;
        ; (Modified registers: eCX, eDX, eBX, eSI, eDI)
        ;
        ;
        mov     ebx, [bufp]
        and     word [ebx], 7FFFh ; 0111111111111111b
                ; bic $100000,*bufp / clear bit 15 of I/O queue entry at
                                    ; / bottom of queue
        call    poke
        ; 09/06/2015
        jnc     short dskwr_1
        mov     dword [u.error], ERR_DRV_WRITE ; disk write error !
        jmp     error
dskwr_1:
        retn

;ppoke:
                ; mov $340,*$ps
                ; jsr r0,poke
                ; clr *$ps
                ; rts r0
poke:
        ; 20/08/2015
        ; 18/08/2015
        ; 02/07/2015
        ; 09/06/2015 (Retro UNIX 386 v1 - Beginning)
        ; 15/03/2013 - 18/01/2014 (Retro UNIX 8086 v1)
        ;
        ; (NOTE: There are some disk I/O code modifications & extensions
        ; & exclusions on original 'poke' & other device I/O procedures of
        ; UNIX v1 OS for performing disk I/O functions by using IBM PC
        ; compatible rombios calls in Retro UNIX 8086 v1 kernel.)
        ;
        ; Basic I/O functions for all block structured devices
        ;
         ; (Modified registers: eCX, eDX, eSI, eDI)
        ;
        ; 20/07/2013 modifications
        ;            (Retro UNIX 8086 v1 features only !)
        ; INPUTS ->
        ;        (EBX = buffer header address)
        ; OUTPUTS ->
        ;        cf=0 -> successed r/w (at least, for the caller's buffer)
        ;        cf=1 -> error, word [eBX] = 0FFFFh
        ;               (drive not ready or r/w error!)
        ;        (dword [EBX+4] <> 0FFFFFFFFh indicates r/w success)
        ;        (dword [EBx+4] = 0FFFFFFFFh means RW/IO error)
        ;         (also it indicates invalid buffer data)
        ;
        push    ebx
                ; mov r1,-(sp)
                ; mov r2,-(sp)
                ; mov r3,-(sp)
        push    eax ; Physical Block Number (r1) (mget)
        ;
        ; 09/06/2015
        ; (permit read/write after a disk  R/W error)
        mov     cl, [ebx] ; device id (0 to 5)
        mov     al, 1
        shl     al, cl
```

```
        test    al, [active] ; busy ? (error)
        jz      short poke_0
        not     al
        and     [active], al ; reset busy bit for this device only
poke_0:
         mov     esi, bufp + (4*(nbuf+2))
                ; mov $bufp+nbuf+nbuf+6,r2 / r2 points to highest priority
                                        ; / I/O queue pointer
poke_1: ; 1:
         sub    esi, 4
        mov     ebx, [esi]
                ; mov -(r2),r1 / r1 points to an I/O queue entry
        mov     ax, [ebx] ; 17/07/2013
        test    ah, 06h
        ;test   word [ebx], 600h ; 0000011000000000b
                ; bit $3000,(r1) / test bits 9 and 10 of word 1 of I/O
                              ; / queue entry
         jz      short poke_5
                ; beq 2f / branch to 2f if both are clear
        ; 31/07/2013
        ;test   ah, 0B0h ; (*)
        ;;test  word [ebx], 0B000h ; 1011000000000000b
                ; bit $130000,(r1) / test bits 12, 13, and 15
         ;jnz     short poke_5 ; 31/07/2013 (*)
                ; bne 2f / branch if any are set
        ;movzx  ecx, byte [ebx] ; 09/06/2015 ; Device Id
                ; movb (r1),r3 / get device id
        movzx   ecx, al ; 18/08/2015
        ;mov    edi, ecx ; 26/04/2013
        xor     eax, eax ; 0
        ;cmp    [edi+drv.error], al ; 0
                ; tstb deverr(r3) / test for errors on this device
        ;jna    short poke_2
                ; beq 3f / branch if no errors
        ; 02/07/2015
        ;dec    eax
        ;mov    [ebx+4], ax ; 0FFFFFFFFh ; -1
                ; mov $-1,2(r1) / destroy associativity
        ;shr    eax, 24
        ;mov    [ebx], eax ; 000000FFh, reset
                ; clrb 1(r1) / do not do I/O
        ;jmp     short poke_5
         ;       ; br 2f
                 ; rts r0
poke_2: ; 3:
        ; 02/07/2015
        inc     cl ; 0FFh -> 0
        jz      short poke_5
        inc     al ; mov ax, 1
        dec     cl
        jz      short poke_3
        ; 26/04/2013 Modification
        ;inc    al ; mov ax, 1
        ;or     cl, cl ; Retro UNIX 8086 v1 device id.
        ;jz     short poke_3 ; cl = 0
        shl     al, cl ; shl ax, cl
poke_3:
        ;test   [active], ax
        test    [active], al
                ; bit $2,active / test disk busy bit
        jnz      short poke_5
                ; bne 2f / branch if bit is set
        ;or     [active], ax
        or      [active], al
                ; bis $2,active / set disk busy bit
        push    ax
        call    diskio ; Retro UNIX 8086 v1 Only !
        ;mov     [edi+drv.error], ah
        pop     ax
        jnc     short poke_4 ; 20/07/2013
        ;cmp    [edi+drv.error], al ; 0
        ;jna    short poke_4
                ; tstb deverr(r3) / test for errors on this device
                ; beq 3f / branch if no errors
        ; 02/07/2015 (32 bit modification)
        ; 20/07/2013
        mov     dword [ebx+4], 0FFFFFFFFh ; -1
                ; mov $-1,2(r1) / destroy associativity
        mov     word [ebx], 0FFh ; 20/08/2015
```

```
                ; clrb 1(r1) / do not do I/O
        jmp      short poke_5
poke_4: ; 20/07/2013
        ; 17/07/2013
        not     al
        and     [active], al ; reset, not busy
        ; eBX = system I/O buffer header (queue entry) address
seta: ; / I/O queue bookkeeping; set read/write waiting bits.
        mov     ax, [ebx]
                ; mov (r1),r3 / move word 1 of I/O queue entry into r3
         and    ax, 600h
                ; bic $!3000,r3 / clear all bits except 9 and 10
        and     word [ebx], 0F9FFh
                ; bic $3000,(r1) / clear only bits 9 and 10
        shl     ah, 3
                ; rol r3
                  ; rol r3
                  ; rol r3
        or      [ebx], ax
                ; bis r3,(r1) / or old value of bits 9 and 10 with
                                ; bits 12 and 13
        call    idle ; 18/01/2014
        ;; sti
        ;hlt    ; wait for a hardware interrupt
        ;; cli
        ; NOTE: In fact, disk controller's 'disk I/O completed'
         ; interrupt would be used to reset busy bits, but INT 13h
        ; returns when disk I/O is completed. So, here, as temporary
        ; method, this procedure will wait for a time according to
        ; multi tasking and time sharing concept.
        not     ax
        and     [ebx], ax ; clear bits 12 and 13
poke_5: ;2:
         cmp      esi, bufp
                ; cmp r2,$bufp / test to see if entire I/O queue
                                ; / has been scanned
        ja       short poke_1
                ; bhi 1b
        ; 24/03/2013
                ; mov (sp)+,r3
                ; mov (sp)+,r2
                ; mov (sp)+,r1
         pop    eax  ; Physical Block Number (r1) (mget)
        pop     ebx
        ; 02/07/2015 (32 bit modification)
        ; 20/07/2013
        ;cmp    dword [ebx+4], 0FFFFFFFFh
        cmp     byte [ebx], 0FFh ; 20/08/2015
        ;
        ; 'poke' returns with cf=0 if the requested buffer is read
        ; or written succesfully; even if an error occurs while
        ; reading to or writing from other buffers. 20/07/2013
        ;
        ; 09/06/2015
        cmc
        retn
                  ; rts r0
```

```
bufaloc:
        ; 20/08/2015
        ; 19/08/2015
        ; 02/07/2015
        ; 11/06/2015 (Retro UNIX 386 v1 - Beginning)
        ;            (32 bit modifications)
        ; 13/03/2013 - 29/07/2013 (Retro UNIX 8086 v1)
        ;
        ; bufaloc - Block device I/O buffer allocation
        ;
        ; INPUTS ->
        ;    r1 - block number
        ;    cdev - current (block/disk) device number
        ;    bufp+(2*n)-2 --- n = 1 ... nbuff
        ; OUTPUTS ->
        ;    r5 - pointer to buffer allocated
        ;    bufp ... bufp+12 --- (bufp), (bufp)+2
        ;
        ; ((AX = R1)) input/output
        ; ((BX = R5)) output
        ;    ((Modified registers: DX, CX, BX, SI, DI, BP))
        ;    zf=1 -> block already in a I/O buffer
        ;    zf=0 -> a new I/O buffer has been allocated
        ;    ((DL = Device ID))
        ;    (((DH = 0 or 1)))
        ;    (((CX = previous value of word ptr [bufp])))
        ;    ((CX and DH will not be used after return)))

        ;;push  esi ; ***
                ; mov r2,-(sp) / save r2 on stack
                ; mov $340,*$ps / set processor priority to 7
        ; 20/07/2013
        ; 26/04/2013
        movzx   ebx, byte [cdev] ; 0 or 1
        mov     edi, rdev  ; offset mdev = offset rdev + 1
        add     edi, ebx
bufaloc_0: ; 26/04/2013 !! here is called from bread or bwrite !!
                        ;; eDI points to device id.
        movzx   ebx, byte [edi] ; [EDI] -> rdev/mdev or brwdev
        ; 11/06/20215
        cmp     byte [ebx+drv.status], 0F0h ; Drive not ready !
        jb      short bufaloc_9
        mov     dword [u.error], ERR_DRV_NOT_RDY
        jmp     error
bufaloc_9:
        mov     edx, ebx ; dh = 0, dl = device number (0 to 5)
bufaloc_10: ; 02/07/2015
        xor     ebp, ebp ; 0
        push    ebp ; 0
         mov     ebp, esp
bufaloc_1: ;1:
                ; clr -(sp) / vacant buffer
         mov    esi, bufp
                ; mov $bufp,r2 / bufp contains pointers to I/O queue
                            ; / entrys in buffer area
bufaloc_2: ;2:
        mov     ebx, [esi]
                ; mov (r2)+,r5 / move pointer to word 1 of an I/O
                           ; queue entry into r5
        test    word [ebx], 0F600h
                ; bit $173000,(r5) / lock+keep+active+outstanding
         jnz    short bufaloc_3
                ; bne 3f / branch when
                       ; / any of bits 9,10,12,13,14,15 are set
                        ; / (i.e., buffer busy)
         mov     [ebp], esi ; pointer to I/0 queue entry
                  ; mov  r2,(sp) ;/ save pointer to last non-busy buffer
                        ; / found points to word 2 of I/O queue entry)
bufaloc_3: ;3:
        ;mov    dl, [edi] ; 26/04/2013
        cmp     [ebx], dl
                ; cmpb (r5),cdev / is device in I/O queue entry same
                               ; / as current device
        jne     short bufaloc_4
                ; bne 3f
        cmp     [ebx+4], eax
                ; cmp 2(r5),r1 / is block number in I/O queue entry,
                            ; / same as current block number
        jne     short bufaloc_4
```

```
                ; bne 3f
        ;add    esp, 4
        pop     ecx
                ; tst (sp)+ / bump stack pointer
        jmp     short bufaloc_7 ; Retro Unix 8086 v1 modification
                               ; jump to bufaloc_6 in original Unix v1
                ; br 1f / use this buffer
bufaloc_4: ;3:
        add     esi, 4 ; 20/08/2015
        ;
        cmp     esi, bufp + (nbuf*4)
                ; cmp r2,$bufp+nbuf+nbuf
        jb      short bufaloc_2
                ; blo 2b / go to 2b if r2 less than bufp+nbuf+nbuf (all
                        ; / buffers not checked)
         pop    esi
                ; mov (sp)+,r2 / once all bufs are examined move pointer
                            ; / to last free block
        or      esi, esi
        jnz     short bufaloc_5
                ; bne 2f / if (sp) is non zero, i.e.,
                 ; / if a free buffer is found branch to 2f
         ;; mov  ecx, [s.wait_]
        call    idle
                ; jsr r0,idle; s.wait+2 / idle if no free buffers
        jmp     short bufaloc_10 ; 02/07/2015
                ; br 1b
bufaloc_5: ;2:
                ; tst (r0)+ / skip if warmed over buffer
        inc     dh ; Retro UNIX 8086 v1 modification
bufaloc_6: ;1:
         mov            ebx, [esi]
                ; mov -(r2),r5 / put pointer to word 1 of I/O queue
                            ; / entry in r5
        ;; 26/04/2013
         ;mov   dl, [edi] ; byte [rdev] or byte [mdev]
        mov     [ebx], dl
                ; movb cdev,(r5) / put current device number
                                ; / in I/O queue entry
        mov     [ebx+4], eax
                ; mov r1,2(r5) / move block number into word 2
                            ; / of I/O queue entry
bufaloc_7: ;1:
         cmp    esi, bufp
                ; cmp r2,$bufp / bump all entrys in bufp
                            ; / and put latest assigned
        jna     short bufaloc_8
                ; blos 1f / buffer on the top
                       ; / (this makes if the lowest priority)
        sub     esi, 4
        mov     ecx, [esi]
        mov     [esi+4], ecx
                ; mov -(r2),2(r2) / job for a particular device
        jmp     short bufaloc_7
                ; br 1b
bufaloc_8: ;1:
         mov    [esi], ebx
                ; mov r5,(r2)
        ;;pop   esi ; ***
                ; mov (sp)+,r2 / restore r2
        or      dh, dh ; 0 or 1 ?
                ; Retro UNIX 8086 v1 modification
                ; zf=1 --> block already is in an I/O buffer
                ; zf=0 --> a new I/O buffer has been allocated
        retn
                ; rts r0
```

```
diskio:
        ; 10/07/2015
        ; 02/07/2015
        ; 16/06/2015
        ; 11/06/2015 (Retro UNIX 386 v1 - Beginning)
        ;            (80386 protected mode modifications)
        ; 15/03/2013 - 29/04/2013 (Retro UNIX 8086 v1)
        ;
        ; Retro UNIX 8086 v1 feature only !
        ;
        ; Derived from proc_chs_read procedure of TRDOS DISKIO.ASM (2011)
        ; 04/07/2009 - 20/07/2011
        ;
        ; NOTE: Reads only 1 block/sector (sector/block size is 512 bytes)
        ;
         ; INPUTS ->
        ;          eBX = System I/O Buffer header address
         ; OUTPUTS -> cf=0 --> done
        ;            cf=1 ---> error code in AH
        ;
        ; (Modified registers: eAX, eCX, eDX)

;rw_disk_sector:
        ; 10/07/2015
        ; 02/07/2015
        ; 11/06/2015 - Retro UNIX 386 v1 - 'u8.s'
        ; 21/02/2015 ('dsectpm.s', 'read_disk_sector')
        ; 16/02/2015 (Retro UNIX 386 v1 test - 'unix386.s')
        ; 01/12/2014 - 18/01/2015 ('dsectrm2.s')
        ;
        ;mov    dx, 0201h ; Read 1 sector/block
        mov     dh, 2
        mov     ax, [ebx]
        ;
        push    esi ; ****
        push    ebx ; ***
        ;
        movzx   ecx, al
        mov     esi, ecx
        ;
        cmp     cl, dh ; 2
        jb      short rwdsk0
        add     al, 7Eh  ; 80h, 81h, 82h, 83h
rwdsk0:
        mov     [drv], al
        add     esi, drv.status
        ; 11/06/2015
        cmp     byte [esi], 0F0h
        jb       short rwdsk1
        ; 'drive not ready' error
        mov     dword [u.error], ERR_DRV_NOT_RDY
        jmp     error
rwdsk1:
        test    ah, 2
        ;test   ax, 200h ; Bit 9 of word 0 (status word)
                          ; write bit
        jz      short rwdsk2
        ;test   ah, 4
        ;;test  ax, 400h ; Bit 10 of word 0 (status word)
        ;               ; read bit
        ;jz     short diskio_ret
        inc     dh ; 03h = write
rwdsk2:
        mov     dl, al
        add     ebx, 4 ; sector/block address/number pointer
        mov     eax, [ebx] ; sector/block number (LBA)
        shl     cl, 2
        add     ecx, drv.size ; disk size
        cmp     eax, [ecx] ; Last sector + 1 (number of secs.)
        jb       short rwdsk3
        ; 'out of volume' error
        mov     dword [u.error], ERR_DEV_VOL_SIZE
        jmp     error
rwdsk3:
        ; 11/06/2015
        add     ebx, 4 ; buffer address
        mov     byte [retry_count], 4
        test    byte [esi], 1 ; LBA ready ?
         jz      short rwdsk_chs
```

```
rwdsk_lba:
        ; LBA read/write (with private LBA function)
        ;((Retro UNIX 386 v1 - DISK I/O code by Erdogan Tan))
         add     esi, drv.error - drv.status ; 10/07/2015
        mov     ecx, eax ; sector number
        ; ebx = buffer (data) address
        ; dl = physical drive number (0,1, 80h, 81h, 82h, 83h)
rwdsk_lba_retry:
        ;mov    dl, [drv]
                ; Function 1Bh = LBA read, 1Ch = LBA write
        mov     ah, 1Ch - 3h ; LBA write function number - 3
        add     ah, dh
        mov     al, 1
        ;int    13h
        call    int13h
        mov     [esi], ah ; error code ; 10/07/2015
        jnc     short rwdsk_lba_ok
        cmp     ah, 80h ; time out ?
         je      short rwdsk_lba_fails
        dec     byte [retry_count]
         jnz     short rwdsk_lba_reset ; 10/07/2015
rwdsk_lba_fails:
        stc
rwdsk_lba_ok:
        pop     ebx ; ***
        pop     esi ; ****
        retn
rwdsk_lba_reset:
        mov     ah, 0Dh ; Alternate reset
        ;int    13h
         call   int13h
        jnc      short rwdsk_lba_retry
        mov     [esi], ah ; error code ; 10/07/2015
        jmp     short rwdsk_lba_ok
        ;
        ; CHS read (convert LBA address to CHS values)
rwdsk_chs:
        ; 10/07/2015
        sub     esi, drv.status
        mov     ecx, esi
        add     esi, drv.error
        ; 02/07/2015
        ; 16/06/2015
        ; 11/06/2015
        push    ebx ; ** ; buffer
        shl     ecx, 1
        push    ecx ; *
        ;
        mov     ebx, ecx
        mov     [rwdsk], dh ; 02/07/2015
        xor     edx, edx ; 0
        sub     ecx, ecx
         add     ebx, drv.spt
        mov     cx, [ebx] ; sector per track
                ; EDX:EAX = LBA
        div     ecx
        mov     cl, dl  ; sector number - 1
        inc     cl      ; sector number (1 based)
        pop     ebx ; * ; 11/06/2015
        push    cx
         add     ebx, drv.heads
        mov     cx, [ebx] ; heads
        xor     edx, edx
                ; EAX = cylinders * heads + head
        div     ecx
        pop     cx      ; sector number
        mov     dh, dl ; head number
        mov     dl, [drv]
        mov     ch, al ; cylinder (bits 0-7)
        shl     ah, 6
        or      cl, ah ; cylinder (bits 8-9)
                       ; sector (bits 0-7)
        pop     ebx ; ** ; buffer ; 11/06/2015
                ; CL = sector (bits 0-5)
                ;      cylinder (bits 8-9 -> bits 6-7)
                ; CH = cylinder (bits 0-7)
                ; DH = head
                ; DL = drive
        mov     byte [retry_count], 4
```

```
rwdsk_retry:
        mov     ah, [rwdsk] ; 02h = read, 03h = write
        mov     al, 1 ; sector count
        ;int    13h
        call    int13h
        mov     [esi], ah ; error code ; 10/07/2015
        jnc     short rwdsk_ok ; ah = 0
        cmp     ah, 80h ; time out ?
        je      short rwdsk_fails
        dec     byte [retry_count]
        jnz     short rwdsk_reset
rwdsk_fails:
        stc
rwdsk_ok:
        pop     ebx ; ***
        pop     esi ; ****
        retn
rwdsk_reset:
        ; 02/02/2015
        sub     ah, ah
        cmp     dl, 80h
        jb      short rwdsk_fd_reset
        mov     ah, 0Dh ; Alternate reset
rwdsk_fd_reset:
        ;int    13h
         call   int13h
        jnc     short rwdsk_retry
        mov     [esi], ah ; error code ; 10/07/2015
        jmp     short rwdsk_ok
```

```
; Original UNIX v1 - drum (& disk) interrupt routine
;       (Equivalent to IRQ 14 & IRQ 15 disk/hardware interrupts)
;
; This feature is not used in Retro UNIX 386 (& 8086) for now.
; Because, current Retro UNIX 386 disk I/O -INT13H- routine is
; derived from IBM PC AT -infact: XT286- BIOS source code, int 13h
; that uses hardware -transfer has been completed-  interrupt inside it.
; In a next Retro UNIX 386 version, these interrupts
; (fdc_int, hdc1_int, hdc2_int) will be handled by a separate routine
; as in original unix v1.
; I am not removing IBM BIOS source code derivatives -compatible code-
; for now, regarding the new/next 32 bit TRDOS project by me
; (to keep source code files easy adaptable to 32 bit TRDOS.)
;
; Erdogan tan (10/07/2015)

;drum: / interrupt handler
;       jsr     r0,setisp / save r1,r2,r3, and clockp on the stack
;       jsr     r0,trapt; dcs; rfap; 1 / check for stray interrupt or
;                                      / error
;               br 3f / no, error
;       br      2f / error
;
;disk:
;       jsr     r0,setisp / save r1,r2,r3, and clockp on the stack
;       jmp     *$0f
;0:
;       jsr     r0,trapt; rkcs; rkap; 2
;              br 3f / no, errors
;       mov     $115,(r2) / drive reset, errbit was set
;       mov     $1f,0b-2 / next time jmp *$0f is executed jmp will be
;                        / to 1f
;       br      4f
;1:
;       bit     $20000,rkcs
;       beq     4f / wait for seek complete
;       mov     $0b,0b-2
;       mov     rkap,r1
;2:
;       bit     $3000,(r1) / are bits 9 or 10 set in the 1st word of
;                          / the disk buffer
;       bne     3f / no, branch ignore error if outstanding
;       inc     r1
;       asr     (r1)
;       asr     (r1)
;       asr     (r1) / reissue request
;       dec     r1
;3:
;       bic     $30000,(r1) / clear bits 12 and 13 in 1st word of buffer
;       mov     ac,-(sp)
;       mov     mq,-(sp) / put these on the stack
;       mov     sc,-(sp)
;       jsr     r0,poke
;       mov     (sp)+,sc
;       mov     (sp)+,mq / pop them off stack
;       mov     (sp)+,ac
;4:
;       jmp     retisp / u4-3
;
;trapt:                  / r2 points to the
;       mov     (r0)+,r2 / device control register
;       mov     *(r0)+,r1 / transaction pointer points to buffer
;       tst     (sp)+
;       tstb    (r2) / is ready bit of dcs set?
;       bge     4b / device still active so branch
;       bit     (r0),active / was device busy?
;       beq     4b / no, stray interrupt
;       bic     (r0)+,active / yes, set active to zero
;       tst     (r2) / test the err(bit is) of dcs
;       bge     2f / if no error jump to 2f
;       tst     (r0)+ / skip on error
; 2:
;       jmp     (r0)
```

```
; Retro UNIX 386 v1 Kernel (v0.2) - SYS9.INC
; Last Modification: 09/12/2015
; ----------------------------------------------------------------------------
; Derived from 'Retro UNIX 8086 v1' source code by Erdogan Tan
; (v0.1 - Beginning: 11/07/2012)
;
; Derived 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>
;
; Retro UNIX 8086 v1 - U9.ASM (01/09/2014) //// UNIX v1 -> u9.s
;
; ****************************************************************************

getch:
       ; 30/06/2015
       ; 18/02/2015 - Retro UNIX 386 v1 - feature only!
       sub     al, al ; 0
getch_q: ; 06/08/2015
       mov     ah, [ptty] ; active (current) video page
        jmp     short getc_n

getc:
       ; 12/11/2015
       ; 15/09/2015
       ; 01/07/2015
       ; 30/06/2015
       ; 18/02/2015 (Retro UNIX 386 v1 - Beginning)
       ; 13/05/2013 - 04/07/2014 (Retro UNIX 8086 v1)
       ;
       ; Retro UNIX 8086 v1 modification !
       ;
       ; 'getc' gets (next) character
       ;       from requested TTY (keyboard) buffer
       ; INPUTS ->
       ;     [u.ttyn] = tty number (0 to 7) (8 is COM1, 9 is COM2)
       ;     AL=0 -> Get (next) character from requested TTY buffer
       ;       (Keyboard buffer will point to
       ;                      next character at next call)
       ;     AL=1 -> Test a key is available in requested TTY buffer
       ;       (Keyboard buffer will point to
       ;                      current character at next call)
       ; OUTPUTS ->
       ;     (If AL input is 1) ZF=1 -> 'empty buffer' (no chars)
       ;                        ZF=0 -> AX has (current) character
       ;      AL = ascii code
       ;      AH = scan code (AH = line status for COM1 or COM2)
       ;                       (cf=1 -> error code/flags in AH)
       ; Original UNIX V1 'getc':
       ;               get a character off character list
       ;
       ; ((Modified registers: eAX, eBX, eCX, eDX, eSI, eDI))
       ;
       ; 30/06/20045 (32 bit modifications)
       ; 16/07/2013
       ; mov   [getctty], ah
       ;

       mov     ah, [u.ttyn]   ; 28/07/2013
getc_n:
       ; 30/06/2015
       or      ah, ah
       jz      short getc0
       shl     ah, 1
       movzx   ebx, ah
       add     ebx, ttychr
       jmp     short getc1
getc0:
       mov     ebx, ttychr
getc1:
       mov     cx, [ebx]      ; ascii & scan code
                              ; (by kb_int)
       or      cx, cx
       jnz     short getc2
       and     al, al
       jz      short getc_s
       xor     ax, ax
       retn
```

```
getc2:
        and     al, al
        mov     ax, cx
        mov     cx, 0
        jnz     short getc3
getc_sn:
        mov     [ebx], cx ; 0, reset
        cmp     ax, cx   ; zf = 0
getc3:
        retn
getc_s:
        ; 12/11/2015
        ; 15/09/2015
        ; 01/07/2015
        ; 30/06/2015 (Retro UNIX 386 v1 - Beginning)
        ; 16/07/2013 - 14/02/2014 (Retro UNIX 8086 v1)
        ;
        ; tty  of the current process is not
        ; current tty (ptty); so, current process only
        ; can use keyboard input when its tty becomes
        ; current tty (ptty).
        ; 'sleep' is for preventing an endless lock
        ; during this tty input request.
        ; (Because, the user is not looking at the video page
        ; of the process to undersand there is a keyboard
        ; input request.)
        ;
        ;((Modified registers: eAX, eBX, eCX, eDX, eSI, eDI))
        ;
        ; 05/10/2013
        ; ah = byte ptr [u.ttyn] ; (tty number)
        ;
        ; 10/10/2013
gcw0:
        mov     cl, 10 ; ch = 0
gcw1:
        ; 12/11/2015
        call intract ; jumps to 'sysexit' if [u.quit] = FFFFh
        ; 10/10/2013
        call    idle
        mov     ax, [ebx]       ; ascii & scan code
                                ; (by kb_int)
        or      ax, ax
;       jnz     short gcw3
        jnz     short gcw2 ; 15/09/2015
        ; 30/06/2015
        dec     cl
        jnz     short gcw1
        ;
        mov     ah, [u.ttyn]    ; 20/10/2013
;       ; 10/12/2013
;       cmp     ah, [ptty]
;       jne     short gcw2
;       ; 14/02/2014
;       cmp     byte [u.uno], 1
;       jna     short gcw0
;gcw2:
        call    sleep
        ;
        ; 20/09/2013
        mov     ah, [u.ttyn]
        xor     al, al
        jmp     short getc_n
;gcw3:
gcw2:   ; 15/09/2015
        ; 10/10/2013
        xor     cl, cl
        jmp     short getc_sn
```

```
sndc:   ; <Send character>
        ;
        ; 16/11/2015
        ; 11/11/2015
        ; 10/11/2015
        ; 09/11/2015
        ; 08/11/2015
        ; 07/11/2015
        ; 06/11/2015 (serial4.asm, 'sendchr')
        ; 29/10/2015
        ; 30/06/2015 (Retro UNIX 386 v1 - Beginning)
        ; 14/05/2013 - 28/07/2014 (Retro UNIX 8086 v1)
        ;
        ; Retro UNIX 8086 v1 feature only !
        ;
        ; ah = [u.ttyn]
        ;
        ; 30/06/2015
        sub     ah, 8 ; ; 0 = tty8 or 1 = tty9
        ; 07/11/2015
        movzx   ebx, ah ; serial port index (0 or 1)
sndc0:
        ; 07/11/2015
        call    isintr ; quit (ctrl+break) check
        jz      short sndc1
        call    intract ; quit (ctrl+break) check
        ; CPU will jump to 'sysexit' if 'u.quit' = 0FFFFh (yes)
sndc1:
        ; 16/11/2015
        mov     cx, ax ; *** al = character (to be sent)
sndcx:
        mov     al, [ebx+schar] ; last sent character
        mov     ah, [ebx+rchar] ; last received character
        ;
        ; 16/11/2015
        or      ah, ah  ; 0 = query  (from terminal)
        jnz     short query
         ; check RDA interrupt occurence status
        xchg    ah, [ebx+rda_int] ; reset
        or      ah, ah ; 0
        jnz     short response
        sub     al, al ; force query
                       ; (request a response from terminal)
        jmp     short fquery
response:
        cmp     al, 0FFh ; response
        je      short sndc2 ; (already responded)
        inc     byte [comqr] ; query or response status
        xor     al, al
        mov     byte [ebx+rda_int], al ; 0
        dec     al ; 0FFh
        jmp     short sndc3
query:
        or      al, al  ; 0 = query (also end of text)
        jnz     short sndc2 ; normal character
        cmp     ah, 0FFh     ; is it responded by terminal ?
        je      short sndc2  ; yes, already responded
        ; 16/11/2015
        mov     [ebx+rchar], al ; 0 ; reset
fquery:
        ; query: request for response (again)
        inc     byte [comqr] ; query or response status
        jmp     short sndc3
sndc2:
        mov     al, cl  ; *** character (to be sent)
sndc3:
        mov     [ebx+schar], al ; current character (to be sent)
        mov     al, bl ; 0 or 1 (serial port index)
        ; 30/06/2015
        call    sp_status ; get serial port status
        ; AL = Line status, AH = Modem status
        ; 07/11/2015
        test    al, 80h
        jnz     short sndc4
        test    al, 20h ; Transmitter holding register empty ?
        jnz     short sndc5
```

```
sndc4:  ; Check line status again
        ; 16/11/2015
        push    cx
        mov     ecx, 6 ; 6*30 micro seconds (~5556 chars/second)
        call    WAITF
        pop     cx
        ;
        mov     al, bl ; 0 or 1 (serial port index)
        call    sp_status ; get serial port status
        ; 16/11/2015
        ; 09/11/2015
        ; 08/11/2015
        test    al, 80h ; time out error
         jnz     short sndc7
        test    al, 20h ; Transmitter holding register empty ?
         jz     short sndc7
sndc5:
        mov     al, [ebx+schar] ; character (to be sent)
        mov     dx, 3F8h   ; data port (COM2)
        sub     dh, bl
        out     dx, al     ; send on serial port
        ; 10/11/2015
        ; delay for 3*30 (3*(15..80)) micro seconds
        ; (to improve text flow to the terminal)
        ; ('diskette.inc': 'WAITF')
        ; Uses port 61h, bit 4 to have CPU speed independent waiting.
        ; (refresh periods = 1 per 30 microseconds on most machines)
        push    cx
        mov     ecx, 6 ; 6*30 micro seconds (~5556 chars/second)
        call    WAITF
        pop     cx
        ;
        ; 07/11/2015
        mov     al, bl ; al = 0 (tty8) or 1 (tty9)
        ;
        call    sp_status ; get serial port status
        ; AL = Line status, AH = Modem status
        ;
        call    isintr ; quit (ctrl+break) check
        jz      short sndc6
        call    intract ; quit (ctrl+break) check
        ; CPU will jump to 'sysexit' if 'u.quit' = 0FFFFh (yes)
sndc6:
        cmp     al, 80h
        jnb     short sndc7
        ;
        cmp     byte [comqr], 1 ; 'query or response' ?
        jb      short sndc8    ; no, normal character
        mov     byte [comqr], bh ; 0 ; reset
        ;
        cmp     [ebx+schar], bh ; 0 ; query ?
        ja      short sndc2 ; response (will be followed by
                            ; a normal character)
        ; Query request must be responded by the terminal
        ; before sending a normal character !
        push    ebx
        push    cx ; *** cl = character (to be sent)
        mov     ah, [u.ttyn]
        call    sleep ; this process will be awakened by
                      ; received data available interrupt
        pop     cx ; *** cl = character (to be sent)
        pop     ebx
         jmp    sndcx

        ;16/11/2015
        ;call   idle
        ;jmp    sndcx

sndc7:
         ; 16/11/2015
        cmp     byte [comqr], 1 ; 'query or response' ?
        jb      short sndc9    ; no
        ;
        mov     [ebx+rchar], bh ; 0 ; reset
        mov     [ebx+schar], bh ; 0 ; reset
        ;
        mov     byte [comqr], bh ; 0 ; reset
```

```
sndc8:
        cmc   ; jnc -> jc, jb -> jnb
sndc9:
        ; AL = Line status, AH = Modem status
        retn

putc:
        ; 13/08/2015
        ; 30/06/2015 (Retro UNIX 386 v1 - Beginning)
        ; 15/05/2013 - 27/07/2014 (Retro UNIX 8086 v1)
        ;
        ; Retro UNIX 8086 v1 modification !
        ;
        ; 'putc' puts a character
        ;       onto requested (tty) video page or
        ;       serial port
        ; INPUTS ->
        ;     AL = ascii code of the character
        ;     AH = video page (tty) number (0 to 7)
        ;                        (8 is COM1, 9 is COM2)
        ; OUTPUTS ->
        ;    (If AL input is 1) ZF=1 -> 'empty buffer' (no chars)
        ;                      ZF=0 -> AX has (current) character
        ;     cf=0 and AH = 0 -> no error
        ;     cf=1 and AH > 0 -> error (only for COM1 and COM2)

        ;
        ; Original UNIX V1 'putc':
        ;     put a character at the end of character list
        ;
        ; ((Modified registers: eAX, eBX, eCX, eDX, eSI, eDI))
        ;
        cmp     ah, 7
         ja      sndc
        ; 30/06/2015
        movzx   ebx, ah
        ; 13/08/2015
        mov     ah, 07h ; black background, light gray character color
        jmp     write_tty ; 'video.inc'

get_cpos:
        ; 29/06/2015 (Retro UNIX 386 v1)
        ; 04/12/2013 (Retro UNIX 8086 v1 - 'sysgtty')
        ;
        ; INPUT -> bl = video page number
        ; RETURN -> dx = cursor position

        push    ebx
        and     ebx, 0Fh ; 07h ; tty0 to tty7
        shl     bl, 1
        add     ebx, cursor_posn
        mov     dx, [ebx]
        pop     ebx
        retn

read_ac_current:
        ; 29/06/2015 (Retro UNIX 386 v1)
        ; 04/12/2013 (Retro UNIX 8086 v1 - 'sysgtty')
        ;
        ; INPUT -> bl = video page number
        ; RETURN -> ax = character (al) and attribute (ah)

        call    find_position ; 'video.inc'
        ; dx = status port
        ; esi = cursor location/address
        add     esi, 0B8000h   ; 30/08/2014 (Retro UNIX 386 v1)
        mov     ax, [esi]      ; get the character and attribute
        retn
```

```
syssleep:
        ; 29/06/2015 - (Retro UNIX 386 v1)
        ; 11/06/2014 - (Retro UNIX 8086 v1)
        ;
        ; Retro UNIX 8086 v1 feature only
        ; (INPUT -> none)
        ;
        movzx   ebx, byte [u.uno] ; process number
        mov     ah, [ebx+p.ttyc-1] ; current/console tty
        call    sleep
        jmp     sysret

vp_clr:
        ; Reset/Clear Video Page
        ;
        ; 30/06/2015 - (Retro UNIX 386 v1)
        ; 21/05/2013 - 30/10/2013(Retro UNIX 8086 v1) (U0.ASM)
        ;
        ; Retro UNIX 8086 v1 feature only !
        ;
        ; INPUTS ->
        ;   BL = video page number
        ;
        ; OUTPUT ->
        ;   none
        ; ((Modified registers: eAX, BH, eCX, eDX, eSI, eDI))
        ;
        ; 04/12/2013
        sub     al, al
        ; al = 0 (clear video page)
        ; bl = video page
        mov     ah, 07h
        ; ah = 7 (attribute/color)
        xor     cx, cx ; 0, left upper column (cl) & row (cl)
        mov     dx, 184Fh ; right lower column & row (dl=24, dh=79)
        call    scroll_up
        ; bl = video page
        xor     dx, dx ; 0 (cursor position)
        jmp     set_cpos

sysmsg:
        ; 11/11/2015
        ; 01/07/2015 - (Retro UNIX 386 v1 feature only!)
        ; Print user-application message on user's console tty
        ;
        ; Input -> EBX = Message address
        ;          ECX = Message length (max. 255)
        ;          DL = Color (IBM PC Rombios color attributes)
        ;
        cmp     ecx, MAX_MSG_LEN ; 255
        ja      sysret ; nothing to do with big message size
        or      cl, cl
        jz      sysret
        and     dl, dl
        jnz     short sysmsg0
        mov     dl, 07h ; default color
                ; (black background, light gray character)
sysmsg0:
        mov     [u.base], ebx
        mov     [ccolor], dl ; color attributes
        mov     ebp, esp
        xor     ebx, ebx ; 0
        mov     [u.nread], ebx ; 0
        ;
        cmp     [u.kcall], bl ; 0
        ja      short sysmsgk ; Temporary (01/07/2015)
        ;
        mov     [u.count], ecx
        inc     ecx ; + 00h ; ASCIZZ
        sub     esp, ecx
        mov     edi, esp
        mov     esi, esp
        mov     [u.pcount], bx ; reset page (phy. addr.) counter
        ; 11/11/2015
        mov     ah, [u.ttyp] ; recent open tty
        ; 0 = none
        dec     ah
        jns     short sysmsg1
        mov     bl, [u.uno] ; process number
```

```
        mov     ah, [ebx+p.ttyc-1] ; user's (process's) console tty
sysmsg1:
        mov     [u.ttyn], ah
sysmsg2:
        call    cpass
        jz      short sysmsg5
        stosb
        and     al, al
        jnz     short sysmsg2
sysmsg3:
        cmp     ah, 7 ; tty number
        ja      short sysmsg6 ; serial port
        call    print_cmsg
sysmsg4:
        mov     esp, ebp
        jmp     sysret
sysmsg5:
        mov     byte [edi], 0
        jmp     short sysmsg3
sysmsg6:
        mov     al, [esi]
        call    sndc
        jc      short sysmsg4
        cmp     byte [esi], 0  ; 0 is stop character
        jna     short sysmsg4
        inc     esi
        mov     ah, [u.ttyn]
        jmp     short sysmsg6

sysmsgk: ; Temporary (01/07/2015)
        ; The message has been sent by Kernel (ASCIIZ string)
        ; (ECX -character count- will not be considered)
        mov     esi, [u.base]
        mov     ah, [ptty] ; present/current screen (video page)
        mov     [u.ttyn], ah
        mov     byte [u.kcall], 0
        jmp     short sysmsg3

print_cmsg:
        ; 01/07/2015 (retro UNIX 386 v1 feature only !)
        ;
        ; print message (on user's console tty)
        ;       with requested color
        ;
        ; INPUTS:
        ;       esi = message address
        ;       [u.ttyn] = tty number (0 to 7)
        ;       [ccolor] = color attributes (IBM PC BIOS colors)
        ;
        lodsb
pcmsg1:
        push    esi
         movzx   ebx, byte [u.ttyn]
        mov     ah, [ccolor]
        call    write_tty
        pop     esi
        lodsb
        and     al, al  ; 0
        jnz     short pcmsg1
        retn
```

```
sysgeterr:
        ; 09/12/2015
        ; 21/09/2015 - (Retro UNIX 386 v1 feature only!)
        ; Get last error number or page fault count
        ; (for debugging)
        ;
        ; Input -> EBX = return type
        ;          0 = last error code (which is in 'u.error')
        ;          FFFFFFFFh = page fault count for running process
        ;          FFFFFFFEh = total page fault count
        ;          1 .. FFFFFFFDh = undefined
        ;
        ; Output -> EAX = last error number or page fault count
        ;          (depending on EBX input)
        ;
        and     ebx, ebx
        jnz     short glerr_2
glerr_0:
        mov     eax, [u.error]
glerr_1:
        mov     [u.r0], eax
        jmp     sysret
glerr_2:
        inc     ebx ; FFFFFFFFh -> 0, FFFFFFFEh -> FFFFFFFFh
        jz      short glerr_2 ; page fault count for process
        inc     ebx ; FFFFFFFFh -> 0
        jnz     short glerr_0
        mov     eax, [PF_Count] ; total page fault count
         jmp     short glerr_1
glerr_3:
        mov     eax, [u.pfcount]
        jmp     short glerr_1
```

```
; Retro UNIX 386 v1 Kernel - KYBDATA.INC
; Last Modification: 11/03/2015
;               (Data Section for 'KEYBOARD.INC')
;
; ///////// KEYBOARD DATA ///////////////

; 05/12/2014
; 04/12/2014 (derived from pc-xt-286 bios source code -1986-)
; 03/06/86  KEYBOARD BIOS

;-------------------------------------------------------------------------------
;       KEY IDENTIFICATION SCAN TABLES
;-------------------------------------------------------------------------------

;----- TABLES FOR ALT CASE ------------
;----- ALT-INPUT-TABLE
K30:    db      82,79,80,81,75
        db      76,77,71,72,73          ; 10 NUMBER ON KEYPAD
;----- SUPER-SHIFT-TABLE
        db      16,17,18,19,20,21       ; A-Z TYPEWRITER CHARS
        db      22,23,24,25,30,31
        db      32,33,34,35,36,37
        db      38,44,45,46,47,48
        db      49,50

;----- TABLE OF SHIFT KEYS AND MASK VALUES
;----- KEY_TABLE
_K6:     db      INS_KEY                  ; INSERT KEY
        db      CAPS_KEY,NUM_KEY,SCROLL_KEY,ALT_KEY,CTL_KEY
         db      LEFT_KEY,RIGHT_KEY
_K6L     equ     $-_K6

;----- MASK_TABLE
_K7:     db      INS_SHIFT                ; INSERT MODE SHIFT
        db      CAPS_SHIFT,NUM_SHIFT,SCROLL_SHIFT,ALT_SHIFT,CTL_SHIFT
        db      LEFT_SHIFT,RIGHT_SHIFT

;----- TABLES FOR CTRL CASE          ;---- CHARACTERS ------
_K8:    db      27,-1,0,-1,-1,-1        ; Esc, 1, 2, 3, 4, 5
        db      30,-1,-1,-1,-1,31       ; 6, 7, 8, 9, 0, -
        db      -1,127,-1,17,23,5       ; =, Bksp, Tab, Q, W, E
        db      18,20,25,21,9,15        ; R, T, Y, U, I, O
        db      16,27,29,10,-1,1        ; P, [, ], Enter, Ctrl, A
        db      19,4,6,7,8,10           ; S, D, F, G, H, J
        db      11,12,-1,-1,-1,-1       ; K, L, :, ', `, LShift
        db      28,26,24,3,22,2                ; Bkslash, Z, X, C, V, B
        db      14,13,-1,-1,-1,-1       ; N, M, ,, ., /, RShift
        db      150,-1,' ',-1           ; *, ALT, Spc, CL
        ;                               ;----- FUNCTIONS ------
        db      94,95,96,97,98,99       ; F1 - F6
        db      100,101,102,103,-1,-1 ; F7 - F10, NL, SL
        db      119,141,132,142,115,143        ; Home, Up, PgUp, -, Left, Pad5
        db      116,144,117,145,118,146 ; Right, +, End, Down, PgDn, Ins
        db      147,-1,-1,-1,137,138  ; Del, SysReq, Undef, WT, F11, F12

;----- TABLES FOR LOWER CASE ----------
K10:    db      27,'1234567890-=',8,9
        db      'qwertyuiop[]',13,-1,'asdfghjkl;',39
        db      96,-1,92,'zxcvbnm,./',-1,'*',-1,' ',-1
;----- LC TABLE SCAN
        db      59,60,61,62,63          ; BASE STATE OF F1 - F10
        db      64,65,66,67,68
        db      -1,-1                   ; NL, SL

;----- KEYPAD TABLE
K15:    db      71,72,73,-1,75,-1       ; BASE STATE OF KEYPAD KEYS
        db      77,-1,79,80,81,82,83
        db      -1,-1,92,133,134        ; SysRq, Undef, WT, F11, F12

;----- TABLES FOR UPPER CASE ----------
K11:    db      27,'!@#$%',94,'&*()_+',8,0
        db      'QWERTYUIOP{}',13,-1,'ASDFGHJKL:"'
        db      126,-1,'|ZXCVBNM<>?',-1,'*',-1,' ',-1
;----- UC TABLE SCAN
K12:    db      84,85,86,87,88          ; SHIFTED STATE OF F1 - F10
        db      89,90,91,92,93
        db      -1,-1                   ; NL, SL
```

```
;----- NUM STATE TABLE
K14:    db      '789-456+1230.'             ; NUMLOCK STATE OF KEYPAD KEYS
        ;
        db      -1,-1,124,135,136     ; SysRq, Undef, WT, F11, F12

Align   4
;-----------------------------------------
;       VIDEO DISPLAY DATA AREA            ;
;-----------------------------------------
CRT_MODE        db      3       ; CURRENT DISPLAY MODE (TYPE)
CRT_MODE_SET    db      29h     ; CURRENT SETTING OF THE 3X8 REGISTER
                                ; (29h default setting for video mode 3)
                                ; Mode Select register Bits
                                ;   BIT 0 - 80x25 (1), 40x25 (0)
                                ;   BIT 1 - ALPHA (0), 320x200 GRAPHICS (1)
                                ;   BIT 2 - COLOR (0), BW (1)
                                ;   BIT 3 - Video Sig. ENABLE (1), DISABLE (0)
                                ;   BIT 4 - 640x200 B&W Graphics Mode (1)
                                ;   BIT 5 - ALPHA mode BLINKING (1)
                                ;   BIT 6, 7 - Not Used

; Mode 0 - 2Ch = 101100b        ; 40x25 text, 16 gray colors
; Mode 1 - 28h = 101000b        ; 40x25 text, 16 fore colors, 8 back colors
; Mode 2 - 2Dh = 101101b        ; 80x25 text, 16 gray colors
; MODE 3 - 29h = 101001b        ; 80x25 text, 16 fore color, 8 back color
; Mode 4 - 2Ah = 101010b        ; 320x200 graphics, 4 colors
; Mode 5 - 2Eh = 101110b        ; 320x200 graphics, 4 gray colors
; Mode 6 - 1Eh = 011110b        ; 640x200 graphics, 2 colors
; Mode 7 - 29h = 101001b        ; 80x25 text, black & white colors
; Mode & 37h = Video signal OFF


; 26/08/2014
; Retro UNIX 8086 v1 - UNIX.ASM (03/03/2014)
; Derived from IBM "pc-at"
; rombios source code (06/10/1985)
; 'dseg.inc'

;----------------------------------------;
;       SYSTEM DATA AREA               ;
;-----------------------------------------
BIOS_BREAK      db      0               ; BIT 7=1 IF BREAK KEY HAS BEEN PRESSED

;-----------------------------------------
;       KEYBOARD DATA AREAS            ;
;-----------------------------------------

KB_FLAG         db      0               ; KEYBOARD SHIFT STATE AND STATUS FLAGS
KB_FLAG_1       db      0               ; SECOND BYTE OF KEYBOARD STATUS
KB_FLAG_2       db      0               ; KEYBOARD LED FLAGS
KB_FLAG_3       db      0               ; KEYBOARD MODE STATE AND TYPE FLAGS
ALT_INPUT       db      0               ; STORAGE FOR ALTERNATE KEY PAD ENTRY
BUFFER_START    dd      KB_BUFFER       ; OFFSET OF KEYBOARD BUFFER START
BUFFER_END      dd      KB_BUFFER + 32 ; OFFSET OF END OF BUFFER
BUFFER_HEAD     dd      KB_BUFFER       ; POINTER TO HEAD OF KEYBOARD BUFFER
BUFFER_TAIL     dd      KB_BUFFER       ; POINTER TO TAIL OF KEYBOARD BUFFER
; ------        HEAD = TAIL     INDICATES THAT THE BUFFER IS EMPTY
KB_BUFFER       times   16 dw 0         ; ROOM FOR 16 SCAN CODE ENTRIES

; /// End Of KEYBOARD DATA ///
```

```
; Retro UNIX 386 v1 Kernel - VIDATA.INC
; Last Modification: 11/03/2015
;                  (Data section for 'VIDEO.INC')
;
; ///////// VIDEO DATA ///////////////

video_params:
       ; 02/09/2014 (Retro UNIX 386 v1)
       ;ORGS.ASM ----- 06/10/85   COMPATIBILITY MODULE
       ; VIDEO MODE 3
       db      71h,50h,5Ah,0Ah,1Fh,6,19h     ; SET UP FOR 80X25
       db      1Ch,2,7,6,7    ; cursor start = 6, cursor stop = 7
       db      0,0,0,0

; /// End Of VIDEO DATA ///
```

```
; Retro UNIX 386 v1 Kernel - DISKDATA.INC
; Last Modification: 11/03/2015
;       (Initialized Disk Parameters Data section for 'DISKIO.INC')
;
; ****************************************************************************

;----------------------------------------
;       80286 INTERRUPT LOCATIONS       :
;       REFERENCED BY POST & BIOS       :
;----------------------------------------

DISK_POINTER:  dd      MD_TBL6        ; Pointer to Diskette Parameter Table

; IBM PC-XT Model 286 source code ORGS.ASM (06/10/85) - 14/12/2014
;----------------------------------------------------------------
; DISK_BASE                                                    :
;       THIS IS THE SET OF PARAMETERS REQUIRED FOR             :
;       DISKETTE OPERATION. THEY ARE POINTED AT BY THE              :
;       DATA VARIABLE @DISK_POINTER. TO MODIFY THE PARAMETERS,      :
;       BUILD ANOTHER PARAMETER BLOCK AND POINT AT IT               :
;----------------------------------------------------------------

;DISK_BASE:
;       DB      11011111B      ; SRT=D, HD UNLOAD=0F - 1ST SPECIFY BYTE
;       DB      2              ; HD LOAD=1, MODE=DMA - 2ND SPECIFY BYTE
;       DB      MOTOR_WAIT     ; WAIT TIME AFTER OPERATION TILL MOTOR OFF
;       DB      2              ; 512 BYTES/SECTOR
;       ;DB     15             ; EOT (LAST SECTOR ON TRACK)
;       db      18             ; (EOT for 1.44MB diskette)
;       DB      01BH           ; GAP LENGTH
;       DB      0FFH           ; DTL
;       ;DB     054H           ; GAP LENGTH FOR FORMAT
;       db      06ch           ; (for 1.44MB dsikette)
;       DB      0F6H           ; FILL BYTE FOR FORMAT
;       DB      15             ; HEAD SETTLE TIME (MILLISECONDS)
;       DB      8              ; MOTOR START TIME (1/8 SECONDS)

;----------------------------------------
;       ROM BIOS DATA AREAS             :
;----------------------------------------

;DATA           SEGMENT AT 40H        ; ADDRESS= 0040:0000

;@EQUIP_FLAG    DW      ?             ; INSTALLED HARDWARE FLAGS

;----------------------------------------
;       DISKETTE DATA AREAS             :
;----------------------------------------

;@SEEK_STATUS   DB      ?             ; DRIVE RECALIBRATION STATUS
;                                     ; BIT 3-0 = DRIVE 3-0 RECALIBRATION
;                                     ; BEFORE NEXT SEEK IF BIT IS = 0
;@MOTOR_STATUS DB       ?             ; MOTOR STATUS
;                                     ; BIT 3-0 = DRIVE 3-0 CURRENTLY RUNNING
;                                     ; BIT 7 = CURRENT OPERATION IS A WRITE
;@MOTOR_COUNT   DB      ?             ; TIME OUT COUNTER FOR MOTOR(S) TURN OFF
;@DSKETTE_STATUS DB     ?             ; RETURN CODE STATUS BYTE
;                                     ; CMD_BLOCK  IN STACK FOR DISK OPERATION
;@NEC_STATUS    DB      7 DUP(?)      ; STATUS BYTES FROM DISKETTE OPERATION

;----------------------------------------
;       POST AND BIOS WORK DATA AREA    :
;----------------------------------------

;@INTR_FLAG     DB      ?             ; FLAG INDICATING AN INTERRUPT HAPPENED

;----------------------------------------
;       TIMER DATA AREA                 :
;----------------------------------------

; 17/12/2014  (IRQ 0 - INT 08H)
;TIMER_LOW      equ     46Ch          ; Timer ticks (counter)  @ 40h:006Ch
;TIMER_HIGH     equ     46Eh          ; (18.2 timer ticks per second)
;TIMER_OFL      equ     470h          ; Timer - 24 hours flag  @ 40h:0070h
```

```
;----------------------------------------
;       ADDITIONAL MEDIA DATA          :
;----------------------------------------

;@LASTRATE      DB      ?              ; LAST DISKETTE DATA RATE SELECTED
;@DSK_STATE     DB      ?              ; DRIVE 0 MEDIA STATE
;               DB      ?              ; DRIVE 1 MEDIA STATE
;               DB      ?              ; DRIVE 0 OPERATION START STATE
;               DB      ?              ; DRIVE 1 OPERATION START STATE
;@DSK_TRK       DB      ?              ; DRIVE 0 PRESENT CYLINDER
;               DB      ?              ; DRIVE 1 PRESENT CYLINDER

;DATA           ENDS                   ; END OF BIOS DATA SEGMENT

;--------------------------------------------------------
;       DRIVE TYPE TABLE                               :
;--------------------------------------------------------
                ; 16/02/2015 (unix386.s, 32 bit modifications)
DR_TYPE:
                DB      01             ;DRIVE TYPE, MEDIA TABLE
                 ;DW       MD_TBL1
                dd      MD_TBL1
                DB      02+BIT7ON
                ;DW       MD_TBL2
                 dd       MD_TBL2
DR_DEFAULT:     DB      02
                 ;DW       MD_TBL3
                dd       MD_TBL3
                DB      03
                 ;DW       MD_TBL4
                dd       MD_TBL4
                DB      04+BIT7ON
                 ;DW       MD_TBL5
                dd       MD_TBL5
                DB      04
                 ;DW       MD_TBL6
                dd       MD_TBL6
DR_TYPE_E        equ $                    ; END OF TABLE
;DR_CNT         EQU     (DR_TYPE_E-DR_TYPE)/3
DR_CNT          equ     (DR_TYPE_E-DR_TYPE)/5
;--------------------------------------------------------
;       MEDIA/DRIVE PARAMETER TABLES                   :
;--------------------------------------------------------
;--------------------------------------------------------
;       360 KB MEDIA IN 360 KB DRIVE                   :
;--------------------------------------------------------
MD_TBL1:
        DB      11011111B       ; SRT=D, HD UNLOAD=0F - 1ST SPECIFY BYTE
        DB      2               ; HD LOAD=1, MODE=DMA - 2ND SPECIFY BYTE
        DB      MOTOR_WAIT      ; WAIT TIME AFTER OPERATION TILL MOTOR OFF
        DB      2               ; 512 BYTES/SECTOR
        DB      09              ; EOT (LAST SECTOR ON TRACK)
        DB      02AH            ; GAP LENGTH
        DB      0FFH            ; DTL
        DB      050H            ; GAP LENGTH FOR FORMAT
        DB      0F6H            ; FILL BYTE FOR FORMAT
        DB      15              ; HEAD SETTLE TIME (MILLISECONDS)
        DB      8               ; MOTOR START TIME (1/8 SECONDS)
        DB      39              ; MAX. TRACK NUMBER
        DB      RATE_250        ; DATA TRANSFER RATE
;--------------------------------------------------------
;       360 KB MEDIA IN 1.2 MB DRIVE                   :
;--------------------------------------------------------
MD_TBL2:
        DB      11011111B       ; SRT=D, HD UNLOAD=0F - 1ST SPECIFY BYTE
        DB      2               ; HD LOAD=1, MODE=DMA - 2ND SPECIFY BYTE
        DB      MOTOR_WAIT      ; WAIT TIME AFTER OPERATION TILL MOTOR OFF
        DB      2               ; 512 BYTES/SECTOR
        DB      09              ; EOT (LAST SECTOR ON TRACK)
        DB      02AH            ; GAP LENGTH
        DB      0FFH            ; DTL
        DB      050H            ; GAP LENGTH FOR FORMAT
        DB      0F6H            ; FILL BYTE FOR FORMAT
        DB      15              ; HEAD SETTLE TIME (MILLISECONDS)
        DB      8               ; MOTOR START TIME (1/8 SECONDS)
        DB      39              ; MAX. TRACK NUMBER
        DB      RATE_300        ; DATA TRANSFER RATE
```

```
;-------------------------------------------------------
;       1.2 MB MEDIA IN 1.2 MB DRIVE                :
;-------------------------------------------------------
MD_TBL3:
        DB      11011111B       ; SRT=D, HD UNLOAD=0F - 1ST SPECIFY BYTE
        DB      2               ; HD LOAD=1, MODE=DMA - 2ND SPECIFY BYTE
        DB      MOTOR_WAIT      ; WAIT TIME AFTER OPERATION TILL MOTOR OFF
        DB      2               ; 512 BYTES/SECTOR
        DB      15              ; EOT (LAST SECTOR ON TRACK)
        DB      01BH            ; GAP LENGTH
        DB      0FFH            ; DTL
        DB      054H            ; GAP LENGTH FOR FORMAT
        DB      0F6H            ; FILL BYTE FOR FORMAT
        DB      15              ; HEAD SETTLE TIME (MILLISECONDS)
        DB      8               ; MOTOR START TIME (1/8 SECONDS)
        DB      79              ; MAX. TRACK NUMBER
        DB      RATE_500        ; DATA TRANSFER RATE
;-------------------------------------------------------
;       720 KB MEDIA IN 720 KB DRIVE                :
;-------------------------------------------------------
MD_TBL4:
        DB      11011111B       ; SRT=D, HD UNLOAD=0F - 1ST SPECIFY BYTE
        DB      2               ; HD LOAD=1, MODE=DMA - 2ND SPECIFY BYTE
        DB      MOTOR_WAIT      ; WAIT TIME AFTER OPERATION TILL MOTOR OFF
        DB      2               ; 512 BYTES/SECTOR
        DB      09              ; EOT (LAST SECTOR ON TRACK)
        DB      02AH            ; GAP LENGTH
        DB      0FFH            ; DTL
        DB      050H            ; GAP LENGTH FOR FORMAT
        DB      0F6H            ; FILL BYTE FOR FORMAT
        DB      15              ; HEAD SETTLE TIME (MILLISECONDS)
        DB      8               ; MOTOR START TIME (1/8 SECONDS)
        DB      79              ; MAX. TRACK NUMBER
        DB      RATE_250        ; DATA TRANSFER RATE
;-------------------------------------------------------
;       720 KB MEDIA IN 1.44 MB DRIVE               :
;-------------------------------------------------------
MD_TBL5:
        DB      11011111B       ; SRT=D, HD UNLOAD=0F - 1ST SPECIFY BYTE
        DB      2               ; HD LOAD=1, MODE=DMA - 2ND SPECIFY BYTE
        DB      MOTOR_WAIT      ; WAIT TIME AFTER OPERATION TILL MOTOR OFF
        DB      2               ; 512 BYTES/SECTOR
        DB      09              ; EOT (LAST SECTOR ON TRACK)
        DB      02AH            ; GAP LENGTH
        DB      0FFH            ; DTL
        DB      050H            ; GAP LENGTH FOR FORMAT
        DB      0F6H            ; FILL BYTE FOR FORMAT
        DB      15              ; HEAD SETTLE TIME (MILLISECONDS)
        DB      8               ; MOTOR START TIME (1/8 SECONDS)
        DB      79              ; MAX. TRACK NUMBER
        DB      RATE_250        ; DATA TRANSFER RATE
;-------------------------------------------------------
;       1.44 MB MEDIA IN 1.44 MB DRIVE                      :
;-------------------------------------------------------
MD_TBL6:
        DB      10101111B       ; SRT=A, HD UNLOAD=0F - 1ST SPECIFY BYTE
        DB      2               ; HD LOAD=1, MODE=DMA - 2ND SPECIFY BYTE
        DB      MOTOR_WAIT      ; WAIT TIME AFTER OPERATION TILL MOTOR OFF
        DB      2               ; 512 BYTES/SECTOR
        DB      18              ; EOT (LAST SECTOR ON TRACK)
        DB      01BH            ; GAP LENGTH
        DB      0FFH            ; DTL
        DB      06CH            ; GAP LENGTH FOR FORMAT
        DB      0F6H            ; FILL BYTE FOR FORMAT
        DB      15              ; HEAD SETTLE TIME (MILLISECONDS)
        DB      8               ; MOTOR START TIME (1/8 SECONDS)
        DB      79              ; MAX. TRACK NUMBER
        DB      RATE_500        ; DATA TRANSFER RATE
```

```
; << diskette.inc >>
; ++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++
;
;----------------------------------------
;       ROM BIOS DATA AREAS            :
;----------------------------------------

;DATA           SEGMENT AT 40H          ; ADDRESS= 0040:0000

;----------------------------------------
;       FIXED DISK DATA AREAS          :
;----------------------------------------

;DISK_STATUS1: DB       0               ; FIXED DISK STATUS
;HF_NUM:                DB      0               ; COUNT OF FIXED DISK DRIVES
;CONTROL_BYTE: DB       0               ; HEAD CONTROL BYTE
;@PORT_OFF      DB      ?               ;  RESERVED (PORT OFFSET)

;----------------------------------------
;       ADDITIONAL MEDIA DATA          :
;----------------------------------------

;@LASTRATE      DB      ?               ; LAST DISKETTE DATA RATE SELECTED
;HF_STATUS      DB      0               ; STATUS REGISTER
;HF_ERROR       DB      0               ; ERROR REGISTER
;HF_INT_FLAG    DB      0               ; FIXED DISK INTERRUPT FLAG
;HF_CNTRL       DB      0               ; COMBO FIXED DISK/DISKETTE CARD BIT 0=1
;@DSK_STATE     DB      ?               ; DRIVE 0 MEDIA STATE
;               DB      ?               ; DRIVE 1 MEDIA STATE
;               DB      ?               ; DRIVE 0 OPERATION START STATE
;               DB      ?               ; DRIVE 1 OPERATION START STATE
;@DSK_TRK       DB      ?               ; DRIVE 0 PRESENT CYLINDER
;               DB      ?               ; DRIVE 1 PRESENT CYLINDER

;DATA           ENDS                    ; END OF BIOS DATA SEGMENT
;
; ++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++

ERR_TBL:
        db      NO_ERR
        db      BAD_ADDR_MARK,BAD_SEEK,BAD_CMD,UNDEF_ERR
        db      RECORD_NOT_FND,UNDEF_ERR,BAD_ECC,BAD_SECTOR

; 17/12/2014 (mov ax, [cfd])
; 11/12/2014
cfd:            db 0                    ; current floppy drive (for GET_PARM)
; 17/12/2014                            ; instead of 'DISK_POINTER'
pfd:            db 1                    ; previous floppy drive (for GET_PARM)
                                        ; (initial value of 'pfd
                                        ; must be different then 'cfd' value
                                        ; to force updating/initializing
                                        ; current drive parameters)
align 2

HF_PORT:        dw      1F0h  ; Default = 1F0h
                              ; (170h)
HF_REG_PORT:    dw      3F6h  ; HF_PORT + 206h

; 05/01/2015
hf_m_s:         db      0     ; (0 = Master, 1 = Slave)

; ****************************************************************************
```

```
; Retro UNIX 386 v1 Kernel - DISKBSS.INC
; Last Modification: 10/07/2015
;       (Unnitialized Disk Parameters Data section for 'DISKIO.INC')
;
; ****************************************************************************

alignb 2

;----------------------------------------
;       TIMER DATA AREA              :
;----------------------------------------

TIMER_LH:      ; 16/02/205
TIMER_LOW:      resw  1              ; LOW WORD OF TIMER COUNT
TIMER_HIGH:     resw  1              ; HIGH WORD OF TIMER COUNT
TIMER_OFL:      resb  1              ; TIMER HAS ROLLED OVER SINCE LAST READ

;----------------------------------------
;       DISKETTE DATA AREAS          :
;----------------------------------------

SEEK_STATUS:   resb    1
MOTOR_STATUS:  resb    1
MOTOR_COUNT:   resb    1
DSKETTE_STATUS: resb   1
NEC_STATUS:    resb    7

;----------------------------------------
;       ADDITIONAL MEDIA DATA        :
;----------------------------------------

LASTRATE:      resb    1
HF_STATUS:     resb    1
HF_ERROR:      resb    1
HF_INT_FLAG:   resb    1
HF_CNTRL:      resb    1
DSK_STATE:     resb    4
DSK_TRK:       resb    2

;----------------------------------------
;       FIXED DISK DATA AREAS        :
;----------------------------------------

DISK_STATUS1:  resb    1            ; FIXED DISK STATUS
HF_NUM:        resb    1            ; COUNT OF FIXED DISK DRIVES
CONTROL_BYTE:  resb    1            ; HEAD CONTROL BYTE
;@PORT_OFF     resb    1            ; RESERVED (PORT OFFSET)
;port1_off     resb    1            ; Hard disk controller 1 - port offset
;port2_off     resb    1            ; Hard idsk controller 2 - port offset

alignb 4

;HF_TBL_VEC:   resd    1            ; Primary master disk param. tbl. pointer
;HF1_TBL_VEC:  resd    1            ; Primary slave disk param. tbl. pointer
HF_TBL_VEC: ; 22/12/2014
HDPM_TBL_VEC:  resd    1            ; Primary master disk param. tbl. pointer
HDPS_TBL_VEC:  resd    1            ; Primary slave disk param. tbl. pointer
HDSM_TBL_VEC:  resd    1            ; Secondary master disk param. tbl. pointer
HDSS_TBL_VEC:  resd    1            ; Secondary slave disk param. tbl. pointer

; 03/01/2015
LBAMode:       resb    1

; ****************************************************************************
```

```
; Retro UNIX 386 v1 Kernel - ux.s
; Last Modification: 13/11/2015
;
; ///////// RETRO UNIX 386 V1 SYSTEM DEFINITIONS //////////////
; (Modified from
;       Retro UNIX 8086 v1 system definitions in 'UNIX.ASM', 01/09/2014)
; ((UNIX.ASM (RETRO UNIX 8086 V1 Kernel), 11/03/2013 - 01/09/2014))
; ----------------------------------------------------------------------------
; Derived 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>
; (Section E10 (17/3/1972) - ux.s)
; ****************************************************************************

alignb 2

inode:
        ; 11/03/2013.
        ;Derived from UNIX v1 source code 'inode' structure (ux).
        ;i.

        i.flgs:  resw 1
        i.nlks:  resb 1
        i.uid:   resb 1
         i.size:  resw 1 ; size
        i.dskp:  resw 8 ; 16 bytes
        i.ctim:  resd 1
        i.mtim:  resd 1
        i.rsvd:   resw 1 ; Reserved (ZERO/Undefined word for UNIX v1.)

I_SIZE  equ $ - inode

process:
        ; 06/05/2015
        ; 11/03/2013 - 05/02/2014
        ;Derived from UNIX v1 source code 'proc' structure (ux).
        ;p.

         p.pid:   resw nproc
         p.ppid:  resw nproc
         p.break: resw nproc
         p.ttyc:  resb nproc ; console tty in Retro UNIX 8086 v1.
        p.waitc: resb nproc ; waiting channel in Retro UNIX 8086 v1.
        p.link:  resb nproc
        p.stat:  resb nproc

        ; 06/05/2015 (Retro UNIX 386 v1 fetaure only !)
        p.upage: resd nproc ; Physical address of the process's
                            ; 'user' structure


P_SIZE  equ $ - process
```

```
; fsp table (original UNIX v1)
;
;Entry
;          15                                      0
;  1     |---|---------------------------------------|
;        |r/w|       i-number of open file           |
;        |---|---------------------------------------|
;        |                device number              |
;        |-------------------------------------------|
;    (*) | offset pointer, i.e., r/w pointer to file |
;        |-------------------------------------------|
;        |  flag that says    | number of processes  |
;        |   file deleted     | that have file open  |
;        |-------------------------------------------|
;  2     |                                           |
;        |-------------------------------------------|
;        |                                           |
;        |-------------------------------------------|
;        |                                           |
;        |-------------------------------------------|
;        |                                           |
;        |-------------------------------------------|
;  3     |                                           |
;        |                                           |
;
; (*) Retro UNIX 386 v1 modification: 32 bit offset pointer


; 15/04/2015
fsp:     resb nfiles * 10 ; 11/05/2015 (8 -> 10)
bufp:    resd (nbuf+2) ; will be initialized
ii:      resw 1
idev:    resw 1 ; device number is 1 byte in Retro UNIX 8086 v1 !
cdev:     resw 1 ; device number is 1 byte in Retro UNIX 8086 v1 !
; 18/05/2015
; 26/04/2013 device/drive parameters (Retro UNIX 8086 v1 feature only!)
; 'UNIX' device numbers (as in 'cdev' and 'u.cdrv')
;       0 -> root device (which has Retro UNIX 8086 v1 file system)
;       1 -> mounted device (which has Retro UNIX 8086 v1 file system)
; 'Retro UNIX 8086 v1' device numbers: (for disk I/O procedures)
;       0 -> fd0 (physical drive, floppy disk 1), physical drive number = 0
;       1 -> fd1 (physical drive, floppy disk 2), physical drive number = 1
;       2 -> hd0 (physical drive, hard disk 1), physical drive number = 80h
;       3 -> hd1 (physical drive, hard disk 2), physical drive number = 81h
;       4 -> hd2 (physical drive, hard disk 3), physical drive number = 82h
;       5 -> hd3 (physical drive, hard disk 4), physical drive number = 83h
rdev:    resb 1 ; root device number ; Retro UNIX 8086 v1 feature only!
                ; as above, for physical drives numbers in following table
mdev:    resb 1 ; mounted device number ; Retro UNIX 8086 v1 feature only!
; 15/04/2015
active:  resb 1
         resb 1 ; 09/06/2015
mnti:    resw 1
mpid:    resw 1
rootdir: resw 1
; 14/02/2014
; Major Modification: Retro UNIX 8086 v1 feature only!
;                     Single level run queue
;                     (in order to solve sleep/wakeup lock)
runq:    resw 1
imod:    resb 1
smod:    resb 1
mmod:    resb 1
sysflg:  resb 1
```

```
alignb 4

user:
        ; 18/10/2015
        ; 12/10/2015
        ; 21/09/2015
        ; 24/07/2015
        ; 16/06/2015
        ; 09/06/2015
        ; 11/05/2015
        ; 16/04/2015 (Retro UNIX 386 v1 - 32 bit modifications)
        ; 10/10/2013
        ; 11/03/2013.
        ;Derived from UNIX v1 source code 'user' structure (ux).
        ;u.

        u.sp:    resd 1 ; esp (kernel stack at the beginning of 'sysent')
        u.usp:   resd 1 ; esp (kernel stack points to user's registers)
        u.r0:    resd 1 ; eax
        u.cdir:  resw 1
        u.fp:    resb 10
        u.fofp:  resd 1
        u.dirp:  resd 1
        u.namep:  resd 1
        u.off:   resd 1
        u.base:  resd 1
        u.count:  resd 1
        u.nread:  resd 1
        u.break:  resd 1 ; break
        u.ttyp:  resw 1
        u.dirbuf: resb 10
        ;u.pri:  resw 1 ; 14/02/2014
        u.quant:  resb 1 ; Retro UNIX 8086 v1 Feature only ! (uquant)
        u.pri:   resb 1 ;
        u.intr:  resw 1
        u.quit:  resw 1
        ;u.emt:  resw 1 ; 10/10/2013
        u.ilgins: resw 1
        u.cdrv:  resw 1 ; cdev
        u.uid:   resb 1 ; uid
        u.ruid:  resb 1
        u.bsys:  resb 1
        u.uno:   resb 1
         u.upage:  resd 1 ; 16/04/2015 - Retro Unix 386 v1 feature only !
        ; tty number (rtty, rcvt, wtty)
        u.ttyn:  resb 1 ; 28/07/2013 - Retro Unix 8086 v1 feature only !
        ; last error number
        u.error:  resd 1 ; 28/07/2013 - 09/03/2015
                        ; Retro UNIX 8086/386 v1 feature only!
        u.pgdir:  resd 1 ; 09/03/2015 (page dir addr of process)
        u.ppgdir: resd 1 ; 06/05/2015 (page dir addr of the parent process)
        u.pbase:  resd 1 ; 20/05/2015 (physical base/transfer address)
        u.pcount: resw 1 ; 20/05/2015 (byte -transfer- count for page)
        ;u.pncount: resw 1
                ; 16/06/2015 (byte -transfer- count for page, 'namei', 'mkdir')
        ;u.pnbase:  resd 1
                ; 16/06/2015 (physical base/transfer address, 'namei', 'mkdir')
                         ; 09/06/2015
        u.kcall:  resb 1 ; The caller is 'namei' (dskr) or 'mkdir' (dskw) sign
        u.brwdev: resb 1 ; Block device number for direct I/O (bread & bwrite)
                         ; 24/07/2015 - 24/06/2015
        ;u.args:  resd 1 ; arguments list (line) offset from start of [u.upage]
                         ; (arg list/line is from offset [u.args] to 4096 in [u.upage])
                         ; ([u.args] points to argument count -argc- address offset)
                         ; 24/06/2015
        ;u.core:  resd 1 ; physical start address of user's memory space (for sys exec)
        ;u.ecore: resd 1 ; physical end address of user's memory space (for sys exec)
                         ; 21/09/2015 (debugging - page fault analyze)
        u.pfcount: resd 1 ; page fault count for (this) process (for sys geterr)

alignb 4

U_SIZE  equ $ - user
```

```
; 18/10/2015 - Retro UNIX 386 v1 (local variables for 'namei' and 'sysexec')
pcore:  resd 1 ; physical start address of user's memory space (for sys exec)
ecore:  resd 1 ; physical start address of user's memory space (for sys exec)
nbase: resd 1  ; physical base address for 'namei' & 'sysexec'
ncount: resw 1 ; remain byte count in page for 'namei' & 'sysexec'
argc:   resw 1 ; argument count for 'sysexec'
argv:   resd 1 ; argument list (recent) address for 'sysexec'

; 03/06/2015 - Retro UNIX 386 v1 Beginning
; 07/04/2013 - 31/07/2013 - Retro UNIX 8086 v1
rw:      resb 1 ;; Read/Write sign (iget)
rwdsk:   resb 1 ;; Read/Write function number (diskio) - 16/06/2015
retry_count: resb 1 ; Disk I/O retry count - 11/06/2015
         resb 1 ;; Reserved (16/06/2015)

;alignb 4

; 22/08/2015
buffer: resb nbuf * 520

sb0:    resd 2
;s:
; (root disk) super block buffer
systm:
        ; 13/11/2015 (Retro UNIX 386 v1)
        ; 11/03/2013.
        ;Derived from UNIX v1 source code 'systm' structure (ux).
        ;s.

        resw 1
        resb 360 ; 2880 sectors ; original UNIX v1 value: 128
        resw 1
        resb 32  ; 256+40 inodes ; original UNIX v1 value: 64
        s.time:  resd 1
        s.syst:  resd 1
         s.wait_: resd 1 ; wait
        s.idlet: resd 1
        s.chrgt: resd 1
        s.drerr: resw 1

S_SIZE  equ $ - systm

        resb 512-S_SIZE ; 03/06/2015

sb1:    resd 2
; (mounted disk) super block buffer
mount:
        resb 512  ; 03/06/2015

;/ ux -- unix
;
;systm:
;
;       .=.+2
;       .=.+128.
;       .=.+2
;       .=.+64.
;       s.time: .=.+4
;       s.syst: .=.+4
;       s.wait: .=.+4
;       s.idlet:.=.+4
;       s.chrgt:.=.+4
;       s.drerr:.=.+2
;inode:
;       i.flgs: .=.+2
;       i.nlks: .=.+1
;       i.uid:  .=.+1
;       i.size: .=.+2
;       i.dskp: .=.+16.
;       i.ctim: .=.+4
;       i.mtim: .=.+4
;       . = inode+32.
;mount: .=.+1024.
```

```
;proc:
;       p.pid:  .=.+[2*nproc]
;       p.dska: .=.+[2*nproc]
;       p.ppid: .=.+[2*nproc]
;       p.break:.=.+[2*nproc]
;       p.link: .=.+nproc
;       p.stat: .=.+nproc
;tty:
;       . = .+[ntty*8.]
;fsp:   .=.+[nfiles*8.]
;bufp:  .=.+[nbuf*2]+6
;sb0:   .=.+8
;sb1:   .=.+8
;swp:   .=.+8
;ii:    .=.+2
;idev:  .=.+2
;cdev:  .=.+2
;deverr: .=.+12.
;active: .=.+2
;rfap:  .=.+2
;rkap:  .=.+2
;tcap:  .=.+2
;tcstate:.=.+2
;tcerrc: .=.+2
;mnti:  .=.+2
;mntd:  .=.+2
;mpid:  .=.+2
;clockp: .=.+2
;rootdir:.=.+2
;toutt: .=.+16.
;touts:  .=.+32.
;runq:  .=.+6
;
;wlist: .=.+40.
;cc:    .=.+30.
;cf:    .=.+31.
;cl:    .=.+31.
;clist: .=.+510.
;imod:  .=.+1
;smod:  .=.+1
;mmod:  .=.+1
;uquant: .=.+1
;sysflg: .=.+1
;pptiflg:.=.+1
;ttyoch: .=.+1
; .even
; .=.+100.; sstack:
;buffer: .=.+[ntty*140.]
;       .=.+[nbuf*520.]
;
; . = core-64.
;user:
;       u.sp:    .=.+2
;       u.usp:   .=.+2
;       u.r0:    .=.+2
;       u.cdir:  .=.+2
;       u.fp:    .=.+10.
;       u.fofp:  .=.+2
;       u.dirp:  .=.+2
;       u.namep: .=.+2
;       u.off:   .=.+2
;       u.base:  .=.+2
;       u.count: .=.+2
;       u.nread: .=.+2
;       u.break: .=.+2
;       u.ttyp:  .=.+2
;       u.dirbuf:.=.+10.
;       u.pri:   .=.+2
;       u.intr:  .=.+2
;       u.quit:  .=.+2
;       u.emt:   .=.+2
;       u.ilgins:.=.+2
;       u.cdev:  .=.+2
;       u.uid:   .=.+1
;       u.ruid:  .=.+1
;       u.bsys:  .=.+1
;       u.uno:   .=.+1
;. = core
```