```
; ****************************************************************************
;
; UNIX.ASM (RETRO UNIX 8086 Kernel - Only for 1.44 MB floppy disks)
; ----------------------------------------------------------------------------
;
; RETRO UNIX 8086 (Retro Unix == Turkish Rational Unix)
; Operating System Project (v0.1) by ERDOGAN TAN (Beginning: 11/07/2012)
; 1.44 MB Floppy Disk
; (11/03/2013)
;
; [ Last Modification: 15/04/2015 ]
;
; Derivation from UNIX Operating System (v1.0 for PDP-11)
; (Original) Source Code by Ken Thompson (1971-1972)
; <Bell Laboratories (17/3/1972)>
; <Preliminary Release of UNIX Implementation Document>
;
; ****************************************************************************

; 28/08/2014, 01/09/2014
; 20/07/2014, 21/07/2014, 23/07/2014, 24/07/2014, 27/07/2014, 28/07/2014
; 05/07/2014, 07/07/2014, 08/07/2014, 09/07/2014, 12/07/2014, 18/07/2014
; 26/06/2014, 27/06/2014, 30/06/2014, 01/07/2014, 03/07/2014, 04/07/2014
; 31/05/2014, 02/06/2014, 03/06/2014, 11/06/2014, 23/06/2014, 25/06/2014
; 05/05/2014, 19/05/2014, 20/05/2014, 22/05/2014, 26/05/2014, 30/05/2014
; 17/04/2014, 22/04/2014, 25/04/2014, 29/04/2014, 30/04/2014, 01/05/2014
; 24/03/2014, 04/04/2014, 10/04/2014, 11/04/2014, 14/04/2014, 15/04/2014
; 04/03/2014, 07/03/2014, 08/03/2014, 12/03/2014, 18/03/2014, 20/03/2014
; 14/02/2014, 17/02/2014, 23/02/2014, 25/02/2014, 28/02/2014, 03/03/2014
; 18/01/2014, 20/01/2014, 21/01/2014, 26/01/2014, 01/02/2014, 05/02/2014
; 10/01/2014, 12/01/2014, 13/01/2014, 14/01/2014, 16/01/2014, 17/01/2014
; 03/12/2013, 04/12/2013, 06/12/2013, 07/12/2013, 10/12/2013, 12/12/2013
; 24/10/2013, 30/10/2013, 04/11/2013, 18/11/2013, 19/11/2013, 30/11/2013
; 22/09/2013, 24/09/2013, 05/10/2013, 10/10/2013, 20/10/2013, 23/10/2013
; 30/08/2013, 26/08/2013, 03/09/2013, 13/09/2013, 17/09/2013, 20/09/2013
; 18/08/2013, 16/08/2013, 14/08/2013, 13/08/2013, 12/08/2013, 11/08/2013
; 09/08/2013, 08/08/2013, 05/08/2013, 03/08/2013, 02/08/2013, 01/08/2013
; 31/07/2013 user/u structure (u.rw and u.namei_r has been removed)
; 30/07/2013, 29/07/2013
; 28/07/2013 u.rw, u.namei_r, u.ttyn, u.errn
; 26/07/2013, 25/07/2013, 24/07/2013, 17/07/2013, 16/07/2013, 14/07/2013
; 13/07/2013 kernel initialization additions & modifications
; 09/07/2013
; 20/06/2013 set date & time (for 'sysstime' system call)
; 04/06/2013 ecore (sysexec)
; 03/06/2013 p_time (systime, sysmdate)
; 26/05/2013
; 24/05/2013 (end of core)
; 21/05/2013 com_stat: owner and status of COM/serial port (1&2)
; 10/05/2013 tty modifications (keyboard functions)
; 26/04/2013 device numbers, structure modifications
; 11/03/2013
nproc   equ     16   ; number of processes
nfiles  equ     50
ntty    equ      8   ; 8+1 -> 8 (10/05/2013)
nbuf    equ     6

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)

user    struc
        ; 10/10/2013
        ; 11/03/2013.
        ;Derived from UNIX v1 source code 'user' structure (ux).
        ;u.

         sp_     dw ? ; sp
        usp     dw ?
        r0      dw ?
        cdir    dw ?
        fp      db 10 dup(?)
        fofp    dw ?
        dirp    dw ?
        namep   dw ?
        off     dw ?
        base    dw ?
        count   dw ?
        nread   dw ?
        break_  dw ? ; break
        ttyp    dw ?
        dirbuf  db 10 dup(?)
        ;pri    dw ? ; 14/02/2014
        quant   db ? ; Retro UNIX 8086 v1 Feature only ! (uquant)
        pri     db ? ;
        intr    dw ?
        quit    dw ?
        ; emt   dw ? ; 10/10/2013
        ilgins  dw ?
        cdrv    dw ? ; cdev
        uid_    db ? ; uid
        ruid    db ?
        bsys    db ?
        uno     db ?
        ; user/program segment (12/03/2013)
         segmnt  dw ? ; 12/03/2013 - Retro Unix 8086 v1 feature only !
        ; tty number (rtty, rcvt, wtty)
        ttyn    db ? ; 28/07/2013 - Retro Unix 8086 v1 feature only !
        ; last error number (reserved)
        errn    db ? ; 28/07/2013 - Retro Unix 8086 v1 feature only !

user    ends

process  struc
        ; 05/02/2014 ttys -> waitc  (waiting channel, tty number)
        ; 17/09/2013 ttys (10 byte structure)
        ; 03/09/2013 ttyc (word -> byte) [ 10 bytes -> 9 bytes ]
        ; 14/08/2013 dska -> ttyc
        ; 11/03/2013.
        ;Derived from UNIX v1 source code 'proc' structure (ux).
        ;p.

         pid     dw nproc dup(?)
         ppid    dw nproc dup(?)
         break   dw nproc dup(?)
         ttyc    db nproc dup(?) ; console tty in Retro UNIX 8086 v1.
        waitc   db nproc dup(?) ; waiting channel in Retro UNIX 8086 v1.
        link    db nproc dup(?)
        stat    db nproc dup(?)

process ends

inode   struc ; 11/03/2013.
        ;Derived from UNIX v1 source code 'inode' structure (ux).
        ;i.

        flgs    dw ?
        nlks    db ?
        uid     db ?
         size_   dw ? ; size
        dskp    dw 8 dup(?) ; 16 bytes
        ctim    dd ?
        mtim    dd ?
        rsvd    dw ? ; Reserved (ZERO/Undefined word for UNIX v1.)

inode   ends

systm   struc ; 11/03/2013.
        ;Derived from UNIX v1 source code 'systm' structure (ux).
        ;s.

        dw      ?
        db      128 dup(?)
        dw      ?
        db      64 dup (?)
        time    dd ?
        syst    dd ?
         wait_   dd ? ; wait
        idlet   dd ?
        chrgt   dd ?
        drerr   dw ?

systm   ends

; fsp table entry (8 bytes) ;; 19/04/2013
;        inum           dw 0    ; inode number
;       devnum  dw 0    ; device number
;       ofsp    dw 0    ; offset pointer
;       oc      db 0    ; open count
;       df      db 0    ; deleted flag
;

phydrv  struc ; 26/04/2013  (09/07/2013)
        ; Physical drv parameters of Retro UNIX 8086 v1 devices
        ; Retro UNIX 8086 v1 feature only !
        err     db 6 dup(?) ; error status (>0 means error)
        pdn     db 6 dup(?) ; physical drive number
        spt     dw 6 dup(?) ; sectors per track
        hds     dw 6 dup(?) ; heads
phydrv  ends


; 14/07/2013
; 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 !

sys macro syscallnumber
    ; 14/07/2013
    ; Retro UNIX 8086 v1 system call.
    mov ax, syscallnumber
    int 20h
    endm


.8086

UNIX    SEGMENT PUBLIC PARA 'CODE'
                assume cs:UNIX,ds:UNIX,es:UNIX,ss:UNIX
START:

; 11/03/2013
; include files according to original UNIX v1 (except ux.s)
; (u0.s, u1.s, u2.s, u3.s, u34.s, u5.s, u6.s, u7.s, u8.s, u9.s)
;
include u0.asm ; u0.s (with major modifications for 8086 PC)
include u1.asm ; u1.s
include u2.asm ; u2.s
include u3.asm ; u3.s
include u4.asm ; u4.s
include u5.asm ; u5.s
include u6.asm ; u6.s
include u7.asm ; u7.s
include u8.asm ; u8.s
include u9.asm ; u9.s


; RETRO UNIX 8086 v1 special/private procedures
;
;

epoch:
        ; 09/04/2013
        ; Retro UNIX 8086 v1 feature/procedure only!
        ; '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: AX, DX, CX, BX))
        ;

        ; 21/7/2012
        ;push bx
        ;push cx

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

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

        mov cx, 3030h


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

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

        mov byte ptr [minute], al

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

        mov byte ptr [second], al


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

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

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

        add word ptr [year], ax


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

        mov byte ptr [month], al


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

        mov byte ptr [Day], al

convert_to_epoch:
        ; Derived from DALLAS Semiconductor
        ; Application Note 31 (DS1602/DS1603)
        ; 6 May 1998

        mov dx, word ptr [year]
        sub dx, 1970
        mov ax, 365
        mul dx
        xor bh, bh
        mov bl, byte ptr [month]
        dec bl
        shl bl, 1
        mov cx, word ptr DMonth[BX]
        mov bl, byte ptr [Day]
        dec bl
        add ax, cx
        adc dx, 0
        add ax, bx
        adc dx, 0
                              ; DX:AX = days since 1/1/1970
        mov cx, word ptr [year]
        sub cx, 1969
        shr cx, 1
        shr cx, 1
               ; (year-1969)/4
        add ax, cx
        adc dx, 0
                              ; + leap days since 1/1/1970

        cmp byte ptr [month], 2  ; if past february
        jna short @f
        mov cx, word ptr [year]
        and cx, 3 ; year mod 4
        jnz short @f
                              ; and if leap year
        add ax, 1 ; add this year's leap day (february 29)
        adc dx, 0
@@:                    ; compute seconds since 1/1/1970
        mov bx, 24
        call mul32

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

        mov bx, 60
        call mul32

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

        mov bx, 60
        call mul32

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

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

        ; 21/7/2012
        ;pop cx
        ;pop bx

        retn


mul32:
        ; push cx

        mov cx, bx
        mov bx, dx

        mul cx

        xchg ax, bx

        push dx

        mul cx

        pop cx

        add ax, cx
        adc dx, 0

        xchg bx, ax
        xchg dx, bx

        ; pop cx

        retn
```

```
set_date_time: ; 20/06/2013
convert_from_epoch:
        ; 20/06/2013
        ; Retro UNIX 8086 v1 feature/procedure only!
        ; 'convert_from_epoch' procedure prototype:
        ;                 UNIXCOPY.ASM, 10/03/2013
        ; 30/11/2012
        ; Derived from DALLAS Semiconductor
        ; Application Note 31 (DS1602/DS1603)
        ; 6 May 1998
        ;
        ; INPUT:
        ; DX:AX = Unix (Epoch) Time
        ;
        ; ((Modified registers: AX, DX, CX, BX))
        ;
        mov cx, 60
        call div32
        ;mov word ptr [imin], ax   ; whole minutes
        ;mov word ptr [imin]+2, dx ; since 1/1/1970
        mov word ptr [second], bx  ; leftover seconds
        ; mov cx, 60
        call div32
        ;mov word ptr [ihrs], ax   ; whole hours
        ;mov word ptr [ihrs]+2, dx ; since 1/1/1970
        mov word ptr [minute], bx  ; leftover minutes
        ; mov cx, 24
        mov cl, 24
        call div32
        ;mov word ptr [iday], ax  ; whole days
                                  ; since 1/1/1970
        ; mov word ptr [iday]+2, dx ; DX = 0
        mov word ptr [hour], bx   ; leftover hours
        add ax, 365+366                  ; whole day since
                                  ; 1/1/1968
        ; adc dx, 0                ;  DX = 0
        ; mov word ptr [iday], ax
        push ax
        mov cx, (4*365)+1         ; 4 years = 1461 days
        call div32
        pop cx
        ;mov word ptr [lday], ax  ; count of quadyrs (4 years)
        push bx
        ;mov word ptr [qday], bx  ;  days since quadyr began
        cmp bx, 31 + 29           ; if past feb 29 then
        cmc                       ; add this quadyr's leap day
        adc ax, 0                 ; to # of qadyrs (leap days)
        ;mov word ptr [lday], ax  ; since 1968
        ;mov cx, word ptr [iday]
        xchg cx, ax               ; CX = lday, AX = iday
        sub ax, cx                ; iday - lday
        mov cx, 365
        ;xor dx, dx               ; DX  = 0
        ; AX = iday-lday, DX = 0
        call div32
        ;mov word ptr [iyrs], ax   ; whole years since 1968
        ; jday = iday - (iyrs*365) - lday
        ;mov word ptr [jday], bx  ; days since 1/1 of current year
        add ax, 1968              ; compute year
        mov word ptr [year], ax
        mov dx, ax
        ;mov ax, word ptr [qday]
        pop ax
        cmp ax, 365               ; if qday <= 365 and qday >= 60
        ja short @f               ; jday = jday +1
        cmp ax, 60                 ; if past 2/29 and leap year then
         cmc                      ; add a leap day to the # of whole
        adc bx, 0                 ; days since 1/1 of current year
@@:
        ;mov word ptr [jday], bx
        mov cx, 12                ; estimate month
        xchg cx, bx               ; CX = jday, BX = month
        mov ax, 366               ; mday, max. days since 1/1 is 365
        and dx, 11b               ; year mod 4 (and dx, 3)
@@:     ; Month calculation       ; 0 to 11  (11 to 0)
        cmp cx, ax                ; mday = # of days passed from 1/1
        jnb short @f
        dec bx                    ; month = month - 1
        shl bx, 1
        mov ax, word ptr DMonth[BX] ; # 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 @b              ; then mday = mday + 1
        or dl, dl                 ; if past 2/29 and leap year then
        jnz short @b              ; add leap day (to mday)
        inc ax                    ; mday = mday + 1
        jmp short @b
@@:
        inc bx                    ; -> bx = month, 1 to 12
        mov word ptr [month], bx
        sub cx, ax                ; day = jday - mday + 1
        inc cx
        mov word ptr [day], cx

        ; ax, bx, cx, dx is changed at return
        ; output ->
        ; [year], [month], [day], [hour], [minute], [second]

        ; 20/06/2013
set_date:
        mov al, byte ptr [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, byte ptr [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, byte ptr [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, byte ptr [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
         mov ah, 05h
         int 1Ah
        ; retn
set_time:
         ; Read real-time clock time
         mov ah, 02h
         int 1Ah
        ; DL = 1 or 0 (day light saving time)
        mov al, byte ptr [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, byte ptr [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, byte ptr [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
         mov ah, 03h
         int 1Ah
        retn

div32:
        ; Input -> DX:AX = 32 bit dividend
        ;           CX = 16 bit divisor
        ; output -> DX:AX = 32 bit quotient
        ;           BX = 16 bit remainder
        mov  bx, dx
        xchg ax, bx
        xor  dx, dx
        div  cx          ; at first, divide DX
        xchg ax, bx     ; remainder is in DX
                      ; now, BX has quotient
                      ; save remainder
        div  cx         ; so, DX_AX divided and
                      ; AX has quotient
                      ; DX has remainder
        xchg dx, bx     ; finally, BX has remainder

        retn

;; 13/07/2013
unixbootdrive: db 0
;;
; Following (data) section is derived from UNIX v1 'ux.s' file
; 11/03/2013
;
align 2
; 13/07/2013
sb0:    db 4 dup(0) ; Retro UNIX 8086 v1 modification !
;systm:
;s:     db 218 dup(?)
s:      db 512 dup(0) ; Retro UNIX 8086 v1 modification !
;;inode:
;i:     db 32 dup(0)
sb1:    db 4 dup(0)    ; Retro UNIX 8086 v1 modification !
mount:  db 512 dup(0)  ; Retro UNIX 8086 v1 modification !
;mount: db 1024 dup(0)
;inode:
i:      db 32 dup(0)
;
;proc:
;p:     db 9*nproc dup(0) ; 03/09/2013
p:      db 10*nproc dup(0)
;tty:   db ntty*8 dup(0)
fsp:    db nfiles*8 dup(0)
bufp:   db ((nbuf*2)+4) dup(0) ; will be initialized (09/07/2013)
;;bufp: db ((nbuf*2)+6) dup(0)
;;sb0:  db 8 dup(0)
;sb0:   db 4 dup(0) ; Retro UNIX 8086 v1 modification !
;;sb1:  db 8 dup(0)
;sb1:   db 4 dup(0) ; Retro UNIX 8086 v1 modification !
;swp:   db 8 dup(0)
;;swp:  db 4 dup(0) ; Retro UNIX 8086 v1 modification !
ii:     dw 0
idev:   dw 0 ; device number is 1 byte in Retro UNIX 8086 v1 !
cdev:   dw 0 ; device number is 1 byte in Retro UNIX 8086 v1 !
;;deverr: db 12 dup(0)
;
; 26/04/2013 device/drive parameters
; Retro UNIX 8086 v1 feature only!
; there are 8 available Retro UNIX devices
;
; '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:   db 0 ; root device number ; Retro UNIX 8086 v1 feature only!
             ; as above, for physical drives numbers in following table
mdev:   db 0 ; mounted device number ; Retro UNIX 8086 v1 feature only!
             ; as above, for physical drives numbers in following table
; NOTE: the value of 'cdev' and 'u.drv' and 'idev' will be 0 or 1.
;       0 is for rdev, 1 is for mdev

drv: ; Retro UNIX 8086 v1 feature only!
drverr:
        db 6 dup(0FFh) ; error status (>0 means error)
drvpdn:
        db 6 dup(0FFh) ; physical drive number (FFh = invalid drive)
drvspt:
        dw 6 dup(0)    ; sectors per track
drvhds:
        dw 6 dup(0)    ; number of heads
;active: dw 0
```

```
active: db 0 ; 15/03/2013
brwdev: db 0 ; 26/04/2013 Retro UNIX 8086 v1 feature only !
;rfap:  dw 0
;rkap:  dw 0
;tcap:  dw 0
;tcstate:dw 0
;tcerrc:dw 0
mnti:   dw 0
;mntd:  dw 0 ; device number is 1 byte in Retro UNIX 8086 v1 !
mpid:   dw 0
;clockp: dw 0
rootdir:dw 0
;toutt: db 16 dup(0)
;touts:  db 32 dup(0)
;runq:  db 6 dup (0)
; 14/02/2014
; Major Modification: Retro UNIX 8086 v1 feature only!
;                     Single level run queue
;                     (in order to solve sleep/wakeup lock)
runq:   dw 0


;wlist: db 40 dup(0)
;cc:    db 30 dup(0)
;cf:    db 31 dup(0)
;cl_:   db 31 dup(0) ; cl
;clist: db 510 dup(0)

imod:   db 0
smod:   db 0
mmod:   db 0
;uquant: db 0  ; 14/02/2014 --> u.quant
sysflg: db 0
;pptiflg:db 0
;ttyoch: db 0

align 2

; Retro Unix 8086 v1 features only !
; 31/07/2013
; 07/04/2013
rw: db 0 ;; Read/Write sign
;; 07/08/2013 (reset in error routine)
;; mov word ptr [namei_r], 0 -> namei_r = 0,  mkdir_w = 0
; 26/07/2013
namei_r: db 0 ; the caller is 'namei' sign for 'dskr' (ES=CS)
; 01/08/2013
mkdir_w: db 0 ; the caller is 'mkdir' sign for 'dskw' (ES=CS)
;

align 2

; 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

; 10/05/2013
; Retro UNIX 8086 v1 feature only !
```

```
int09h: ; BIOS INT 09h handler (original)
        dw 0 ; offset
        dw 0 ; segment

; 03/06/2013
p_time: dd 0 ; present time (for systime & sysmdate)

; 04/12/2013  ('putc', 'write_tty' in U9.ASM)
crt_start: dw 0       ; starting address in regen buffer
                ; NOTE: active page only
cursor_posn: dw 8 dup(0) ; cursor positions for video pages

; 04/12/2013
active_page: ; = ptty  ('putc', 'write_tty' in U9.ASM)
; 10/05/2013
; Retro UNIX 8086 v1 feature only !
ptty: db 0 ; current tty
;nxtty: db 0 ; next tty (will be switched to)
; 16/07/2013
;getctty: db 0 ; for using in 'getc' routine
; 12/08/2013
;AltKeyDown: db 0 ; INT 09h


align 2

; 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    dW      offset KB_BUFFER ; OFFSET OF KEYBOARD BUFFER START
BUFFER_END      dW      offset KB_BUFFER + 32  ; OFFSET OF END OF BUFFER
BUFFER_HEAD     dw      offset KB_BUFFER ; POINTER TO HEAD OF KEYBOARD BUFFER
BUFFER_TAIL     dw      offset KB_BUFFER ; POINTER TO TAIL OF KEYBOARD BUFFER
; ------        HEAD = TAIL    INDICATES THAT THE BUFFER IS EMPTY
KB_BUFFER       dw      16 DUP (0)      ; ROOM FOR 15 SCAN CODE ENTRIES
;

;align 2

; 26/01/2014 'ttyl' lock table instead of 'ttyr' and 'ttyw'
;
; 16/08/2013 'ttypt' owner table -> 'ttyr', 'ttyw' lock table
; byte ptr [BX]+ttyl = owner/lock for read/write
;               (process number = locked, 0 = unlocked/free)
; byte ptr [BX]+ttyr+1 = count of open for read&write
;               (0 = free, >0 = in use)
;
;; Retro UNIX 8086 v1 feature only!
;;
;; (26/01/2014)
;; (13/01/2014)
;; 06/12/2013
;; <<<Major modification on TTY procedures>>>
;;
; Console TTY for process :
;   'sys fork' system call sets/copies parent process's
;   console TTY number as child process's console TTY number.
;   It is a zero based number (0 to 9) which is hold in 'p.ttyc'.
;   Console TTY setting can be changed by 'sys stty' system call.
; Recent TTY for process:
;   Recent TTY number during the last TTY read/write routine
;   by process. 'u.ttyp' (word pointer) is used for that purpose.
;   TTY num. of the last TTY Read is stored in low byte of 'u.ttyp'.
;   TTY num. of the last TTY write is stored in high byte of 'u.ttyp.
;
; TTY 'Open' conditions: (06/12/2013 <--- 16/08/2013)
;     1) A process can open a free/unlocked tty or a tty
;        which is locked by it or it's parent process. (13/01/2014)
;        (Open count is increased by 1 while a new instance of
;        tty is being open.)
;     2) The caller/process locks a tty if it is unlocked/free.
;     3) TTY open procedure sets 'u.ttyp' to related tty number + 1.
;        Open for read procedure sets the low byte and open for
;        write procedure sets the high byte.
;        NOTE: TTY read and write procedures change these recent tty
;        (u.ttyp) values. (06/12/2013)
;
; TTY 'close' conditions: (16/08/2013)
;     1) A tty is unlocked if it's open count becomes zero while
;        closing it. (26/01/2014)
;        (Open count is decreased by 1 when the instance of
;        tty is closed.)
;     2) TTY close procedure resets low byte or high byte of
;        'u.ttyp' if it was set to related tty number + 1.
;        Open for read procedure resets the low byte and open
;        for write procedure resets the high byte. (06/12/2013)
;
; NOTE: 'tty' functionality of 'Retro UNIX 8086 v1' is almost
;       different than original UNIX v1 (also v1 to recent
;       unix sys v versions). Above logic/methods is/are
;       developed by Erdogan Tan, for keeping 'multi screen',
;       'multi tasking' ability of 'Retro UNIX 8086 v1' (tty and
;       process switching by 'ALT + Function keys' and
;       for ensuring proper/stable process separation between
;       pseudo TTYs and serial ports).
;

; 09/07/2014 (tty8, tty9)
; 24/09/2013 (tty0 to tty7)
ttychr: ; (0 to 9)
    dw ntty+2 dup(0) ; ascii (lb) & scan code (hb) of keys
                  ; per every pseudo tty (video page)
; 26/01/2014 'ttyl' lock table instead of 'ttyr' and 'ttyw'
; 13/01/2014 (COM1 & COM2 have been added to pseudo TTYs)
; (ntty -> ntty + 2)
; 16/08/2013 (open mode locks for pseudo TTYs)
; [ major tty locks (return error in any conflicts) ]
ttyl: ; Retro UNIX 8086 v1 feature only !
    dw ntty+2 dup(0) ; opening locks for TTYs.
; 22/09/2013
wlist: db ntty+2 dup(0) ; wait channel list (0 to 9 for TTYs)
; 27/07/2014
tsleep: dw 0 ; Transmit sleep sign for port processes
             ; which use serial ports (COM1, COM2) as tty.

;; 16/07/2013
;; tty (keyboard) process/owner table (ttypt)
;ttypt: db ntty*2 dup(0)

;; 12/07/2014 -> communication status data is not needed here
;               <cancel>
; 16/07/2013
; 21/05/2013
;;com_stat:
; 13/01/2014
;;com1_stat:
;;       db 0 ; COM1 line status
;;       db 0 ; COM1 modem status
;;com2_stat:
;;       db 0 ; COM2 line status
;;       db 0 ; COM2 modem status

; 16/08/2013
; Communication parameters for serial ports
; Retro UNIX 8086 v1 default:
;;    11100011b ; E3h
   ;; (111) Baud rate: 9600, (00) parity: none,
   ;; (0) stop bits: 1, (11) word length: 8 bits
;
; NOTE: Default value (E3h) will be set again
; after an initalization error, even if 'sys stty'
; system call changes the value before
; an initialization error in tty 'open' routine.
; (Serial port initialization is performed
; when a tty 'open' routine runs for
; COM1 or COM2 while the tty is free/closed.)

;; 12/07/2014 -> sp_init set comm. parameters as 0E3h
;; 0 means serial port is not available
;;comprm: ; 25/06/2014
com1p: db 0  ;;0E3h
com2p: db 0  ;;0E3h

;Buffer:
        ;db ntty*140 dup(0)
        ;db nbuf*520 dup(0)

align 8
dd 0
Buffer:  ; Retro UNIX 8086 v1 modification !
        db nbuf*516 dup(0)
;user:
u: db 64 dup (0)   ; (Original Unix v1 'user' structure has 62 bytes)

; 14/07/2013
kernel_init_err_msg:
        db 0Dh, 0Ah
        db 07h
        db 'Kernel initialization ERROR !'
        db 0Dh, 0Ah, 0
kernel_init_ok_msg:
        db 07h
        db 'Welcome to Retro UNIX 8086 v1 Operating System !'
        db 0Dh, 0Ah
         db 'by Erdogan Tan - 15/04/2015'
        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

align 2

; sstack:
;       db 256 dup(0)

; 10/12/2013
; 'Enable Multi Tasking' system call (sys emt)
; (time-out enabling/disabling functionality)
; has been added to Retro UNIX 8086 v1 Kernel (in U1.ASM)

SizeOfFile equ $
; 08/03/2014 (system systack size = 256 - 64)
sstack  equ SizeOfFile + 256 - 64
;sstack equ SizeOfFile + 256 ; 24/07/2013

UNIX    ends

         end  START
```

```
; ****************************************************************************
;
; UNIX.ASM (RETRO UNIX 8086 Kernel - Only for 1.44 MB floppy disks)
; ----------------------------------------------------------------------------
; U0.ASM (include u0.asm) //// UNIX v1 -> u0.s

; RETRO UNIX 8086 (Retro Unix == Turkish Rational Unix)
; Operating System Project (v0.1) by ERDOGAN TAN (Beginning: 11/07/2012)
; 1.44 MB Floppy Disk
; (11/03/2013)
;
; [ Last Modification: 15/04/2015 ] ;;; completed ;;;
;
; Derivation from UNIX Operating System (v1.0 for PDP-11)
; (Original) Source Code by Ken Thompson (1971-1972)
; <Bell Laboratories (17/3/1972)>
; <Preliminary Release of UNIX Implementation Document>
;
; ****************************************************************************

; 23/07/2014, 27/07/2014, 28/07/2014
; 07/07/2014, 08/07/2014, 12/07/2014, 20/07/2014
; 30/06/2014, 03/07/2014, 04/07/2014, 05/07/2014
; 23/06/2014, 25/06/2014, 26/06/2014, 27/06/2014
; 22/05/2014, 26/05/2014, 02/06/2014, 03/06/2014
; 01/05/2014, 05/05/2014, 19/05/2014, 20/05/2014
; 14/04/2014, 25/04/2014, 29/04/2014, 30/04/2014
; 03/03/2014, 04/03/2014, 07/03/2014, 12/03/2014
; 05/02/2014, 14/02/2014, 23/02/2014, 28/02/2014
; 17/01/2014, 18/01/2014, 20/01/2014, 01/02/2014
; 30/10/2013, 04/12/2013, 06/12/2013, 10/12/2013
; 24/09/2013, 29/09/2013, 05/10/2013, 10/10/2013
; 30/08/2013, 03/09/2013, 17/09/2013, 20/09/2013
; 23/07/2013, 29/07/2013, 11/08/2013, 12/08/2013
; 16/07/2013, 17/07/2013, 18/07/2013, 22/07/2013
; 15/07/2013, 20/05/2013, 21/05/2013, 27/05/2013
; 15/05/2013, 17/05/2013, 13/07/2013, 14/07/2013
; 11/03/2013, 11/04/2013, 09/05/2013, 10/05/2013

; 29/04/2014 --> serial port (terminal) login functionality test
;                by using fake INT 14h, tty6, tty7
;                etc/init has been modified for leaving tty6 and tty7 free

kernel_init:
        ; 15/04/2015
        ; 07/03/2014
        ; 04/03/2013
        ; 28/02/2014
        ; 14/02/2014
        ; 05/02/2014
        ; 04/12/2013
        ; 05/10/2013
        ; 29/07/2013
        ; 18/07/2013
        ; 17/07/2013
        ; 14/07/2013
        ; 13/07/2013
        ; Retro UNIX 8086 v1 feature only !
        ;
        ; Retro UNIX 8086 v1
        ; kernel relies on data from its 'boot' program ...
        ;
        ;;mov   ax, cs
        ;mov    ds, ax
        ;mov    es, ax
        ;cli
        ;mov    ss, ax
        ;mov    sp, 32766
        ;sti
        ; mov   bp, sp
        mov      byte ptr [unixbootdrive], dl
        mov     ds, cx ; boot sector segment
        ; bx = boot sector buffer
        mov     ax, word ptr [BX]+2 ; 14/07/2013
        mov     dx, word ptr [BX]+4 ; 14/07/2013
        push    cs
        pop     ds
        cmp     ax, 'UR'
        jne     kernel_init_err ; jne short kernel_init_err
        cmp     dx, 'SF'
        jne     kernel_init_err ; jne short kernel_init_err
        ;
        call    drv_init
        jc       kernel_init_err ; jne short kernel_init_err
        ;
        ; 14/02/2014
        ; 14/07/2013
        mov     ax, 41
        mov     word ptr [rootdir], ax
        mov     word ptr [u.cdir], ax
        mov     ax, 1 ; 15/04/2015 (mov al, 1)
        mov     byte ptr [u.uno], al
        mov     word ptr [mpid], ax
        mov     word ptr [p.pid], ax
        mov     byte ptr [p.stat], al ; SRUN, 05/02/2014
        ;
        mov     al, time_count ; 30/08/2013
        ;; 29/07/2013
        ;;mov   byte ptr [s.wait_]+2, al
         ;;mov      byte ptr [s.idlet]+2, al
        ; 14/02/2014 uquant -> u.quant
        mov     byte ptr [u.quant], al ; 14/07/2013
        ; 22/07/2013
        mov     ax, cs
        mov     word ptr [u.segmnt], ax ; reset to CS
         ;
        call    epoch
        mov     word ptr [s.time], ax
        mov     word ptr [s.time]+2, dx
        ;
        call    kb_init
        ; ES = 0 (30/06/2014)
        ;
        ; 28/02/2014 INT 16h handler
        mov     ax, offset int_16h
        mov     di, 22*4 ; INT 16h vector - offset
        stosw
        mov     ax, cs
        stosw
        ;mov    es, ax ; 30/06/2014)
        ;
        ;; 10/12/2013
        ;; INT 1Ch handling disabled here,
        ;;      it will be enabled by 'sys emt'
        ;;      system call (in 'etc/init')
; INT 1Ch (clock/timer) transfer to unix kernel
        ;; 30/06/2014
        ;;xor   ax, ax
        ;;mov   es, ax ; 0
        ;; ES = 0
        ;mov    di, 28*4 ; INT 1Ch vector - offset
        ;cli
        ;mov    ax, offset clock
        ;stosw   ; offset
        ;mov    ax, cs
        ;stosw  ; segment
        ;sti
        ;
; setting up syscall vector (int 20h)
        mov     ax, offset sysent
        mov     di, 32*4 ; INT 20h for system calls
        stosw
        mov     ax, cs
        stosw
        ;mov    es, ax ; 14/04/2014
        ;
        ;
;;      13/07/2013
;;      Kernel is running message ... (temporary)
;
        mov     si, offset kernel_init_ok_msg
        ; 07/03/2014
        ;call   print_msg
        lodsb
        mov     ah, 0Eh
        mov     bx, 07h
@@:
        int     10h
        lodsb
        and     al, al
        jnz     short @b
        ;
        ; 17/01/2014
        ; ES = 0
        call    sp_init ; serial port interrupts
        ; 14/04/2014
        mov     ax, cs
        mov     es, ax
        ;
        ; 05/10/2013 Temporary
        xor     al, al ; mov al, 0
        ; mov byte ptr [u.ttyn], 0
        call    getc
        ; 16/07/2013
        ;xor    al, al
        ; 04/12/2013
        xor     bl, bl ; video page 0
@@:     ; clear video pages (reset cursor positions)
        call    vp_clr  ; 17/07/2013
        inc     bl
        cmp     bl, 8
        jb      short @b
        ;
        ; 17/07/2013
        ;mov     al, byte ptr [unixbootdrive]
        ;cmp    al, 80h ; 128 (80h->hd0)
        ;jna    short @f
        ;sub    al, 7Eh ; 126 (2->hd0)
;@@:
        ;mov    byte ptr [rdev], al
        ;
        call    bf_init ; buffer initialization ; 17/07/2013

;; original UNIX v1 (PDP-11) code here:
        ; / make current program a user
        ;
        ; mov    $41.,r0 / rootdir set to 41 and never changed
        ; mov    r0,rootdir / rootdir is i-number of root directory
        ; mov    r0,u.cdir / u.cdir is i-number of process current directory
        ; mov    $1,r0
        ; movb   r0,u.uno / set process table index for this process to 1
        ; mov    r0,mpid / initialize mpid to 1
        ; mov    r0,p.pid / p.pid identifies process
        ; movb   r0,p.stat / process status = 1 i.e., active
         ;                 /                 = 0 free
        ;                 /                 = 2 waiting for a child to die
         ;                 /                 = 3 terminated but not yet waited
         ;                 /                   for
        ; 18/01/2014
        ;sti
        ; 24/07/2013
        mov     bx, offset init_file
        mov     cx, offset init_argp
        ; (([u.segmnt] = CS))
        ; BX contains 'etc/init' asciiz file name address
        ; CX contains address of argument list pointer
        ;
        dec     byte ptr [sysflg] ; FFh = ready for system call
                                  ; 0 = executing a system call
        ;mov    ax, _exec
        ;int    20h
        sys     _exec  ; execute file
        ;
        jnc     short panic
        ;
        mov     si, offset etc_init_err_msg
        jmp     short @f

;; original UNIX v1 (PDP-11) code here:
 ; 1:
        ; decb sysflg / normally sysflag=0, indicates executing in system
        ; sys exec; 2f; 1f / generates trap interrupt; trap vector =
         ;                   / sysent; 0
        ; br  panic / execute file/etc/init

 ; 1:
        ; 2f;0
 ; 2:
        ; </etc/init\0> / UNIX looks for strings term, noted by nul\0

kernel_init_err:
        ;; NOTE: UNix kernel will load boot sector
        ;;
        mov     si, offset kernel_init_err_msg
@@:
        call    print_msg
        jmp     short key_to_reboot

align 2
init_argp:
        dw      offset init_file, 0
init_file:
        db      '/etc/init', 0

panic:
        ; 07/03/2014
        ; 05/10/2013 ('call getc' instead of 'int 16h')
        ; 14/07/2013 (panic_msg/print_msg)
        ; 10/04/2013
        ;
        ; Retro Unix 8086 v1 modification on original Unix v1 panic procedure!
        ;

        mov     si, offset panic_msg
        call    print_msg
key_to_reboot:
        ;hlt
        ; 05/10/2013
        xor     al, al
        call    getc
        ;
        mov     al, 0Ah
        mov     ah, byte ptr [ptty] ; [active_page]
        call    write_tty

        ;
        ; 15/07/2013
        ;mov    ah, 0Eh
        ;;mov   bx, 07h
        ;;mov   al, 0Dh
        ;;int   10h
        ;mov    al, 0Ah
        ;int 10h

cpu_reset:
        ; 07/03/2014
        ; CPU reset (power on) address
        db      0EAh  ; far jump  (jmp 0FFFFh:0000h)
        dw      0
        dw      0FFFFh  ; F000:0FFF0h

;khere: hlt
;       jmp     short khere

;@@:
        ; 24/09/2013
        ; Reset INT 09h vector for next start-up
        ;xor di, di
        ;mov es, di
        ;mov di, 4*9
        ;mov si, offset int09h
        ;movsw
        ;movsw
        ;
        ;int 19h

;       hlt
;       jmp short @b

        ; clr ps
;1:
        ; dec $0
        ; bne 1b
        ; dec $5
        ; bne 1b
        ; jmp *$173700 / rom loader address
```

```
print_msg:
        ; 07/03/2014
        ; (Modified registers: AX, BX, CX, DX, SI, DI)
        ;
        lodsb
@@:
        push    si
        mov     ah, byte ptr [ptty]
        call    write_tty
        pop     si
        lodsb
        and     al, al
        jnz     short @b
        retn

        ; 14/07/2013
        ; 13/07/2013
        ;lodsb
        ;mov    bx, 07h
        ;mov    ah, 0Eh
;@@:
        ;int    10h
        ;lodsb
        ;and    al, al
        ;jnz    short @b
        ;retn

kb_init:
        ; 30/06/2014
        ; 03/03/2014
        ; 11/08/2013
        ; 16/07/2013
        ; 15/07/2013
        ; 13/07/2013
        ; 21/05/2013
        ; 17/05/2013
        ; 10/05/2013
        ;
        ; Initialization of keyboard handlers
        ;
        ; Retro Unix 8086 v1 feature only!
        ;
        ; ((Modified registers: AX, CX, SI, DI, ES))
        ;
        xor     ax, ax ; 11/08/2013
        mov     di, offset int09h
        mov     ds, ax ; 0
        mov     ax, 9*4  ; INT 09h vector - offset
        mov     si, ax
        movsw           ; offset
        movsw           ; segment
        mov     di, ax
        mov     ax, ds
        mov     es, ax
        mov     ax, cs
        mov     ds, ax
        cli
        mov     ax, offset kb_int
        stosw
        mov     ax, cs
        stosw
        mov     ax, offset ctrlbrk
        mov     di, 27*4 ; INT 1Bh vector - offset
        stosw           ; offset
        mov     ax, cs
        stosw           ; segment
        sti
        ;mov    es, ax  ; 30/06/2014 (ES = 0)
        ;
        ; 03/03/2014
                        ; SETUP KEYBOARD PARAMETERS
        ;mov    si, offset KB_BUFFER
        ;mov    word ptr [BUFFER_HEAD], si
        ;mov    word ptr [BUFFER_TAIL], si
        ;mov    word ptr [BUFFER_START], si
        ;add    si, 32  ; DEFAULT BUFFER OF 32 BYTES
        ;mov    word ptr [BUFFER_END], si
        ;
        retn

ctrlbrk:
        ; 06/12/2013
        ; 20/09/2013
        ; 03/09/2013
        ; 09/05/2013
        ;
        ; INT 1Bh (control+break) handler
        ;
        ; Retro Unix 8086 v1 feature only!
        ;
        cmp     word ptr CS:[u.intr], 0
        ja      short cbrk1
        iret
cbrk1:
        ; 20/09/2013
        push    ax
        mov     al, byte ptr CS:[ptty]
        inc     al
        ; 06/12/2013
        cmp     al, byte ptr CS:[u.ttyp]
        je      short cbrk2
        cmp     al, byte ptr CS:[u.ttyp]+1
        jne     short cbrk3
cbrk2:
        ; 06/12/2013
        mov     ax, word ptr CS:[u.quit]
        and     ax, ax
        jz      short cbrk3
        xor     ax, ax ; 0
        dec     ax
        ; 0FFFFh = 'ctrl+brk' keystroke
        mov     word ptr CS:[u.quit], ax
cbrk3:
        pop     ax
        iret

;tty_sw: ; < tty switch >
        ; 23/02/2014
        ; 04/12/2013 'act_disp_page' (U9.ASM)
        ; 29/09/2013 (simplified)
        ; 29/09/2013 u1.asm -> u0.asm
        ; 22/09/2013
        ; 17/09/2013
        ; 03/09/2013
        ; 21/08/2013
        ; 18/08/2013
        ; 16/07/2013
        ; 15/07/2013
        ; 20/05/2013
        ;
        ; Retro UNIX 8086 v1 feature only !
        ;
        ; INPUTS:
        ;   AL = tty number to be switched on
        ; OUTPUTS:
        ;   Keyboard buffer will be reset and
        ;   active video page will be changed
        ;   according to the requested tty number.
        ;
        ; ((Modified registers: AX))
        ;
        ; 29/09/2013
        ; 03/09/2013
        ;
        ;mov    al, byte ptr [nxtty] ; tty number
        ;                            ; video page
        ;;;
        ; 04/12/2013
        ;;mov   ah, 5 ; Set video page
        ;;int   10h
        ;;mov   byte ptr [ptty], al ; byte ptr [active_page], al
        ;call   act_disp_page
        ; 23/02/2014
        ;mov    byte ptr [u.quant], 0
        ;retn

kb_int:
        ; INT 09h Keyboard Handler
        ;
        ; 30/06/2014
        ; 12/03/2014
        ; 07/03/2014
        ; 04/03/2014
        ; 03/03/2014 major modification
        ; 25/02/2013 ;;
        ; 23/02/2014
        ; 14/02/2014
        ; 01/02/2014
        ; 20/01/2014
        ; 18/01/2014
        ; 17/01/2014
         ; 10/10/2013
        ; 05/10/2013
        ; 29/09/2013
        ; 24/09/2013
        ; 03/09/2013
        ; 12/08/2013
        ; 11/08/2013
        ; 20/05/2013
        ; 15/05/2013
        ; 10/05/2013
        ;
        ; Retro Unix 8086 v1 feature only!

        ; 03/03/2014

        push    ds
        push    ax
        push    bx
        ;
        mov     ax, cs
        mov     ds, ax
        ;
        pushf
        ; 04/03/2014
        ;call   dword ptr [int09h]
        ; 07/03/2014
        push    cs
        call    int_09h
        ;
        ; 24/09/2013
        mov     ah, 1
        int     16h
        jz      short kb_int_4
        ;
        ; 04/03/2014
        mov     bl, byte ptr [ptty]
        xor     ah, ah
        int     16h
        ;
        and     al, al
        jnz     short kb_int_1
        ;
        cmp     ah, 68h ; ALT + F1 key
        jb      short kb_int_1
        cmp     ah, 6Fh  ; ALT + F8 key
        ja      short kb_int_1
        ;
        mov     bh, bl
        add     bh, 68h
        cmp     bh, ah
        je      short kb_int_1
        mov     al, ah
        sub     al, 68h
        ;
        ;mov    byte ptr [ptty], al  ; [active_page]
        ;
        call    tty_sw
        xor     ax, ax ; 0      ; 07/03/2014
        ; 12/03/2014
        mov     bl, byte ptr [ptty]
kb_int_1:
        xor     bh, bh
        shl     bl, 1
        add     bx, offset ttychr
        ; 12/03/2014
        or      ax, ax
        jz      short kb_int_2
        ; 29/09/2013
        cmp     word ptr [BX], 0
        ja      short kb_int_3
kb_int_2:
         ;
         ; 24/09/2013
        mov     word ptr [BX], ax  ; Save ascii code
                                  ; and scan code of the character
                                  ; for current tty (or last tty
                                  ; just before tty switch).
kb_int_3:
         ; 10/10/2013
         mov      al, byte ptr [ptty]
        ; 14/02/2014
        ;mov    bx, offset runq
        call    wakeup
        ;
kb_int_4:
        pop     bx ; 24/09/2013
        pop     ax
        pop     ds
        ;
        iret

vp_clr:
        ; Reset/Clear Video Page
        ;
        ; 04/12/2013 scroll_up (U9.ASM)
        ;
        ; 30/10/2013
        ; 17/09/2013
        ; 17/07/2013
        ; 21/05/2013
        ;
        ; Retro UNIX 8086 v1 feature only !
        ;
        ; INPUTS ->
        ;   AL = video page number
        ;
        ; OUTPUT ->
        ;   none
        ; ((Modified registers: AX, BH, CX, DX, SI, DI))
        ;
        ; 04/12/2013
        sub     al, al
        ; al = 0 (clear video page)
        ; bl = video page
        mov     bh, 07h
        ; bh = 7 (attribute/color)
        call    scroll_up
        ; bh = 7
        ; bl = video page
        xor     dx, dx ; 0
        ;call   set_cpos
        ;retn

        jmp     set_cpos

        ; 30/10/2013
        ;push   es
        ;xor    ah, ah
        ;;push  ax
        ;mov    di, 0B800h
        ;mov    es, di
        ;mov    cx, 2000
        ;sub    dx, dx ; 30/10/2013
        ;or     al, al
        ;jz     short @f
        ;; 30/10/2013
        ;shl    al, 1
        ;; 17/09/2013
        ;push   ax
        ;mul    cx
        ;pop    dx
  ;@@:
        ;mov    di, ax ; 17/09/2013
        ;mov    ah, 07h ; color
        ;rep    stosw
        ;;pop   ax
        ;;mov   bh, al ; video page
        ;;mov   ah, 2 ; set cursor position
        ;;xor   dx, dx
        ;;int   10h
        ;;xor   ax, ax
        ;xor    ah, ah
        ;;pop   di     ; Video page number
        ;;shl   di, 1
        ;mov    di, dx
        ;mov    es, ax ; 0
        ;add    di, 450h ; 40h:50h or 0h:450h
        ;; di = cursor position of the video page.
        ;stosw  ; reset cursor position
        ;pop    es
        ;retn

com2_int:
        ; 28/07/2014
        ; 27/07/2014
        ; 23/07/2014
        ; 20/07/2014 (null chr)
        ; 07/07/2014
        ; 05/07/2014
        ; 04/07/2014
        ; < serial port 2 interrupt handler >
        ;
        ; Retro UNIX 8086 v1 feature only !
        ;
        push    dx
        push    ax
        mov     dx, 2FAh    ; interrupt identification register
        mov     ax, 9      ; tty number of com2
        jmp     short @f

com1_int:
        ; 28/07/2014
        ; 27/07/2014
        ; 23/07/2014
        ; 20/07/2014 (null chr)
        ; 07/07/2014
        ; 05/07/2014
        ; 04/07/2014
        ; < serial port 1 interrupt handler >
        ;
        ; Retro UNIX 8086 v1 feature only !
        ;
        push    dx
        push    ax
        mov     dx, 3FAh    ; interrupt identification register
        mov     ax, 8      ; tty number of com1
@@:
        push    ds
        push    bx
        push    cs
        pop     ds
        push    ax
        ;
        mov     bx, ax
        in      al, dx     ; read register
        and     al, 0Fh     ; leave lowernibble only
        ; 28/07/2014
        cmp     al, 2
        jne     short com_rdei
        ;
        add     bx, offset tsleep - 8
        cmp     byte ptr [BX], ah ; 0
        jna     short @f
        mov     byte ptr [BX], ah ; 0
        jmp     short com_eoi
@@:
        mov     al, 20h
        out     20h, al    ; end of interrupt
        pop     ax
        jmp     short com_iret

com_rdei:
        cmp     al, 4      ; is it receiver data available interrupt?
        jne     short com_eoi ; no, leave interrupt handler
        ;
        sub     dx, 3FAh-3F8h ; data register (3F8h, 2F8h)
        in      al, dx      ; read character
        ; 27/07/2014
        and     al, al
        jnz     short @f
        ; null chr (al=0, ah=0)
        dec     ah ; 0FFh
@@:     ; 27/07/2014
        ; 09/07/2014
        shl     bl, 1
        add     bx, offset ttychr
        ; 23/07/2014 (always overwrite)
        ;;cmp   word ptr [BX], 0
        ;;ja    short com_eoi
        ;
        mov     word ptr [BX], ax   ; Save ascii code
                                   ; scan code = 0
com_eoi:
        mov     al, 20h
        out     20h, al    ; end of interrupt
        ;
        pop     ax ; al = tty number (8 or 9)
         call   wakeup
com_iret:
        pop     bx
        pop     ds
        pop     ax
        pop     dx
        iret

sp_init:
        ; 28/07/2014
        ; 27/07/2014
        ; 12/07/2014
        ; 08/07/2014
        ; 05/07/2014
        ; 03/07/2014
        ; 17/01/2014
        ;
        ; Initialization of serial port interrupt handlers
        ;
        ; Retro Unix 8086 v1 feature only!
        ;
        ; ((Modified registers: AX, CX, DX, DI))
        ;
        ; ES = 0
        ;
        ; Set communication parameters for COM1
        ;
        mov     cl, 0E3h
        xor     ah, ah
        mov     al, cl     ; Communication parameters (E3h)
                          ; 9600 baud, parity none, one stop bit
        xor     dx, dx    ; COM1 (DX=0)
        int     14h
        ; 12/07/2014
        test    ah, 80h
        jnz     short @f
                ; (Note: Serial port interrupts will be disabled here...)
                ; (INT 14h initialization code disables interrupts.)
        mov     byte ptr [com1p], cl ; 0E3h
        ;
        ;; Hook serial port (COM1) interrupt
        ;
        mov     di, 12 * 4   ; 0Ch, COM1 (IRQ 4) interrupt vector
        ;cli
        mov     ax, offset com1_int
        stosw
        mov     ax, cs
        stosw
        ;sti
        ;
        ;; COM1 - enabling IRQ 4
        mov     dx, 3FCh    ;modem control register
        in      al, dx     ;read register
        or      al, 8       ;enable bit 3 (OUT2)
        out     dx, al      ;write back to register
        mov     dx, 3F9h   ;interrupt enable register
        in      al, dx     ;read register
        ;or     al, 1      ;receiver data interrupt enable
        ; 27/7/2014         ; and
        or      al, 3     ;Transmitter empty interrupt enable
        ;
        out     dx, al    ;write back to register
        in      al, 21h    ;read interrupt mask register
        and     al, 0EFh   ;enable IRQ 4 (COM1)
        out     21h, al    ;write back to register

        ;
        ; Set communication parameters for COM2
        ;
        mov     dx, 1     ; COM2
        sub     ah, ah
         mov      al, cl    ; Communication parameters (E3h)
                         ; 9600 baud, parity none, one stop bit
        int     14h
        ; 12/07/2014
        test    ah, 80h
        jnz     short @f
                ; (Note: Serial port interrupts will be disabled here...)
                ; (INT 14h initialization code disables interrupts.)
        mov     byte ptr [com2p], cl ; 0E3h
        ;
        ;; Hook serial port (COM2) interrupt
        ;
        mov     di, 11 * 4  ; 0Bh, COM2 (IRQ 3) interrupt vector
        ;cli
        mov     ax, offset com2_int
        stosw
        mov     ax, cs
        stosw
        ;sti
        ;
        ;; COM2 - enabling IRQ 3
        mov     dx, 2FCh   ;modem control register
        in      al, dx    ;read register
        or      al, 8      ;enable bit 3 (OUT2)
        out     dx, al     ;write back to register
        mov     dx, 2F9h   ;interrupt enable register
        in      al, dx     ;read register
        ;or     al, 1      ;receiver data interrupt enable
        ; 27/7/2014       ; and
        or      al, 3     ;Transmitter empty interrupt enable
        ;
        out     dx, al    ;write back to register
        in      al, 21h    ;read interrupt mask register
        and     al, 0F7h   ;enable IRQ 3 (COM2)
        out     21h, al    ;write back to register
@@:
        retn
```

```
; ****************************************************************************
;
; UNIX.ASM (RETRO UNIX 8086 Kernel - Only for 1.44 MB floppy disks)
; ----------------------------------------------------------------------------
; U1.ASM (include u1.asm) //// UNIX v1 -> u1.s

; RETRO UNIX 8086 (Retro Unix == Turkish Rational Unix)
; Operating System Project (v0.1) by ERDOGAN TAN (Beginning: 11/07/2012)
; 1.44 MB Floppy Disk
; (11/03/2013)
;
; [ Last Modification: 12/07/2014 ] ;;; completed ;;;
;
; Derivation from UNIX Operating System (v1.0 for PDP-11)
; (Original) Source Code by Ken Thompson (1971-1972)
; <Bell Laboratories (17/3/1972)>
; <Preliminary Release of UNIX Implementation Document>
;
; ****************************************************************************

; 11/06/2014, 26/06/2014, 04/07/2014
; 07/03/2014, 10/04/2014, 15/04/2014, 22/04/2014, 30/04/2014
; 18/01/2014, 26/01/2014, 05/02/2014, 14/02/2014, 23/02/2014
; 12/01/2014, 13/01/2014, 14/01/2014, 16/01/2014, 17/01/2014
; 18/11/2013, 04/12/2013, 06/12/2013, 07/12/2013, 10/12/2013
; 20/10/2013, 23/10/2013, 24/10/2013, 30/10/2013, 04/11/2013
; 03/09/2013, 16/09/2013, 17/09/2013, 22/09/2013, 29/09/2013
; 14/08/2013, 18/08/2013, 19/08/2013, 21/08/2013, 30/08/2013
; 26/07/2013, 02/08/2013, 07/08/2013, 08/08/2013, 11/08/2013
; 15/07/2013, 16/07/2013, 22/07/2013, 23/07/2013, 24/07/2013
; 27/05/2013, 30/05/2013, 02/06/2013, 03/06/2013, 14/07/2013
; 20/05/2013, 22/05/2013, 23/05/2013, 24/05/2013, 26/05/2013
; 26/04/2013, 04/05/2013, 09/05/2013, 15/05/2013, 16/05/2013
; 11/03/2013, 10/04/2013, 16/04/2013, 17/04/2013, 19/04/2013
;
unkni: ; / used for all system calls
sysent: ; < enter to system call >
        ; 18/01/2014
        ; 26/07/2013
        ; 24/07/2013
        ; 14/07/2013
        ; 24/05/2013
        ; 16/04/2013
        ; 10/04/2013
        ;
        ; '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 AX register.
        ;
        ;        Other parameters are in DX, BX, CX, SI, DI, BP registers
        ;       depending of function details.
        ;
        ; 16/04/2013 segment changing
        push    cs
        pop     ds
        ;
        inc     byte ptr [sysflg]
                ; incb sysflg / indicate a system routine is in progress
         sti ; 18/01/2014
        jnz      panic ; 24/05/2013
        ;jz     short @f
                ; beq 1f
        ;jmp    short panic
                ; jmp panic ; / called if trap inside system
@@: ;1:
        ; 24/05/2013
        mov     word ptr [u.r0], ax
        mov     word ptr [u.usp], sp

; 16/04/2013 stack segment changing
        ;mov    ax, ss
        ;mov    word ptr [u.segmnt], ax
        mov     ax, cs
        ; 24/05/2013
        ;;;mov es, ax ; 14/07/2013
        cli
        ; 24/07/2013
        mov     sp, sstack ; offset sstack ; swap stack
                    ; (System/Kernel stack in Retro UNIX 8086 v1 !)
        mov     ss, ax
        sti
        ; 24/05/2013
        push    word ptr [u.usp]  ; user's stack pointer (old sp)
                                 ; which points to top of user's stack
                                 ; (Retro UNIX 8086 v1 modification!)
        ;
        push    dx
        push    cx
        push    bx
        push    si
        push    di
        push    bp
        ;
        mov     word ptr [u.sp_], sp
        ;;mov   ax, word ptr [s.syst+2]
        ;;mov   word ptr [clockp], ax
                ; 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
        mov     ax, word ptr [u.r0]
        shl     ax, 1
                ; asl r0 / multiply by 2 to jump indirect in bytes
        cmp     ax, offset @f - offset syscalls
                ; cmp r0,$2f-1f / limit of table (35) exceeded
        ;jnb    short badsys
                ; bhis badsys / yes, bad system call
        ; 16/04/2013
        cmc
        pushf
        push    ax
        ; 24/05/2013
        mov     bp, word ptr [u.usp]
        ; 26/07/2013
        ;mov    ax, 0FFFEh
        mov     al, 0FEh ; 11111110b
        adc     al, 0 ; al = al + cf
        ;and    word ptr ES:[BP]+4, ax ; flags
        ;;mov   ax, word ptr [u.segmnt]
        ;;mov   es, ax
        and     byte ptr ES:[BP]+4, al ; flags (reset carry flag)
                ; bic $341,20.(sp) / set users processor priority to 0
                                   ; / and clear carry bit
        mov     ax, ds ; 14/07/2013
        mov     es, ax ; 17/07/2013
        ;pop    ax
        ;mov    bp, ax
        ;shr    ax, 1
        pop     bp ; ax
        ;mov    ax, word ptr [u.r0]
        popf
         jc      badsys
        mov     ax, word ptr [u.r0]
        ; system call registers: AX, DX, CX, BX, SI, DI
        jmp     word ptr [BP]+syscalls
                ; jmp *1f(r0) / jump indirect thru table of addresses
                            ; / to proper system routine.
syscalls: ; 1:
        dw offset sysrele ; / 0
        dw offset sysexit ; / 1
        dw offset sysfork ; / 2
        dw offset sysread ; / 3
        dw offset syswrite ; / 4
        dw offset sysopen ; / 5
        dw offset sysclose ; / 6
        dw offset syswait ; / 7
        dw offset syscreat ; / 8
        dw offset syslink ; / 9
        dw offset sysunlink ; / 10
        dw offset sysexec ; / 11
        dw offset syschdir ; / 12
        dw offset systime ; / 13
        dw offset sysmkdir ; / 14
        dw offset syschmod ; / 15
        dw offset syschown ; / 16
        dw offset sysbreak ; / 17
        dw offset sysstat ; / 18
        dw offset sysseek ; / 19
        dw offset systell ; / 20
        dw offset sysmount ; / 21
        dw offset sysumount ; / 22
        dw offset syssetuid ; / 23
        dw offset sysgetuid ; / 24
        dw offset sysstime ; / 25
        dw offset sysquit ; / 26
        dw offset sysintr ; / 27
        dw offset sysfstat ; / 28
        dw offset sysemt ; / 29
        dw offset sysmdate ; / 30
        dw offset sysstty ; / 31
        dw offset sysgtty ; / 32
        dw offset sysilgins ; / 33
        dw offset syssleep ; 34 ; Retro UNIX 8086 v1 feature only !
                              ; 11/06/2014
@@: ;2:

error:
        ; 07/08/2013
        ; 26/05/2013
        ; 24/05/2013
        ; 22/05/2013
        ; 04/05/2013
        ; 18/04/2013
        ; 16/04/2013
        ; 10/04/2013
        ; '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     sp, word ptr [u.sp_]
         mov     bp, sp
                ; mov u.sp,r1
         mov     bx, word ptr [BP]+12 ; user's stack pointer
        ;
        mov     ax, word ptr [u.segmnt]
        mov     es, ax
        ;;push  ds
        ;;mov   ds, ax
        ;
        ;;; word ptr ES:[BX]  -> IP
        ;;; word ptr ES:[BX]+2 -> CS
        ;;; word ptr ES:[BX]+4 -> FLAGS

        ;;or    byte ptr [BX]+4, 1
        or      byte ptr ES:[BX]+4, 1  ; set carry bit of flags register
                                      ; in user's stack
                ; bis $1,20.(r1) / set c bit in processor status word below
                               ; / users stack
        ;;pop   ds
        mov     ax, cs
        mov     es, ax
        ; 07/08/2013
        mov word ptr [namei_r], 0 ; namei_r, mkdir_w reset

sysret: ; < return from system call>
        ; 23/02/2014
        ; 07/08/2013
        ; 24/05/2013
        ; 04/05/2013
        ; 26/04/2013
        ; 10/04/2013
        ;
        ; '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
        inc     al ; 04/05/2013
        cmp     byte ptr [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    sp, word ptr [u.sp_] ; 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     byte ptr [smod], al ; 0
                ; tstb  smod / has the super block been modified
        jna     short @f
                ; beq   1f / no, 1f
        mov     byte ptr [smod], al ; 0
                ; clrb smod / yes, clear smod
        mov     bx, offset sb0 ;; 07/08//2013
        or      word ptr [BX], 200h ;;
        ;or     word ptr [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
@@: ;1:
        cmp     byte ptr [mmod], al ; 0
                ; tstb  mmod / has the super block for the dismountable file
                           ; / system
        jna     short @f ; 23/02/2014 (@f location has been changed to u.quant check)
                ; beq 1f / been modified?  no, 1f
        mov     byte ptr [mmod], al ; 0
                ; clrb  mmod / yes, clear mmod
         ;mov    ax, word ptr [mntd]
         ;;mov   al, byte ptr [mdev]  ; 26/04/2013
        mov     bx, offset sb1 ;; 07/08//2013
         ;;mov  byte ptr [BX], al
        ;mov     byte ptr [sb1], al
                ; movb  mntd,sb1 / set the I/O queue
        or      word ptr [BX], 200h
        ;or     word ptr [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:
;       cmp     byte ptr [uquant], al ; 0
;               ; tstb uquant / is the time quantum 0?
;         ja      short @f
;         ;ja      short swapret
                ; bne 1f / no, don't swap it out

sysrele: ; < release >
        ; 07/03/2014
        ; 23/02/2014
        ; 14/02/2014 uquant -> u.quant
        ; 18/01/2014
        ; 07/12/2013
        ; 20/10/2013
        ; 22/09/2013
        ; 16/05/2013
        ; 08/05/2013
        ; 16/04/2013
        ; 11/04/2013
        ; 10/04/2013
        ;
        ; '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 (@@)
        ; 22/09/2013
@@: ;1:
        cmp     byte ptr [u.quant], 0 ; 16/05/2013
                ; tstb uquant / is the time quantum 0?
        ja       short @f
         ;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:
        ; 26/05/2013
        ; 'sp' must be already equal to 'word ptr [u.sp_]' here !
        ;mov    sp, word ptr [u.sp_] ; 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.)
        pop     bp
        pop     di
        pop     si
        pop     bx
        pop     cx
        pop     dx
                ; mov (sp)+,sc / restore user registers
                ; mov (sp)+,mq
                ; mov (sp)+,ac
                ; mov (sp)+,r5
                ; mov (sp)+,r4
                ; mov (sp)+,r3
                ; mov (sp)+,r2
        ; 22/09/2013
        call    isintr
        ; 20/10/2013
        jz      short @f
        call    intract
                ; jsr r0,isintr / is there an interrupt from the user
                ;      br intract / yes, output gets flushed, take interrupt
                                ; / action
@@:
                ; mov (sp)+,r1
        pop     ax ; user's stack pointer
                   ; (was pushed on system stack by 'sysenter'.)
                ; mov (sp)+,r0
        ; 24/05/2013
        ; 18/01/2014
        ;cli    ; disable (hardware) interrupts
        mov     sp, ax ; user's stack pointer
        mov     ax, word ptr [u.segmnt]
        mov     ss, ax ; user's stack segment
        ; 18/01/2014
        ;;sti   ; enable interrupts   ;; 07/03/2014
                        ; 'sti' is not needed here
                        ; (because 'iret' will restore interrupt flag)
        mov     es, ax
        ;;;mov  ax, word ptr [s.chrgt]+2
        ;;;mov  word ptr [clockp], ax
        ; 20/10/2013
        mov     ax, word ptr [u.r0] ; ((return value in AX))
        dec     byte ptr [sysflg]
                ; decb sysflg / turn system flag off
        push    es
        pop     ds
        iret
                ; rti / no, return from interrupt

badsys:
        ; 27/05/2013
        ; 11/04/2013
        inc     byte ptr [u.bsys]
                ; incb u.bsys / turn on the user's bad-system flag
        mov     word ptr [u.namep], offset badsys_3 ;   3f
                ; mov $3f,u.namep / point u.namep to "core\0\0"
        call    namei
                ; jsr r0,namei / get the i-number for the core image file
        ;or     ax, ax ; Retro UNIX 8086 v1 modification !
                       ; ax = 0 -> file not found
        ;jz     short badsys_1
        jc      short badsys_1 ; 27/05/2013
                ; br 1f / error
        neg     ax ; AX = r1
                ; neg r1 / negate the i-number to open the core image file
                       ; / for writing
        call    iopen
                ; jsr r0,iopen / open the core image file
        call    itrunc
                ; jsr r0,itrunc / free all associated blocks
        jmp     short badsys_2
                ; br 2f
badsys_1: ;1:
        mov     ax, 15 ; mode 17
                ; mov $17,r1 / put i-node mode (17) in r1
        call    maknod
                ; jsr r0,maknod / make an i-node
        mov     ax, word ptr [u.dirbuf] ; i-number
                ; mov u.dirbuf,r1 / put i-node number in r1
badsys_2: ;2:
        ; 19/04/2013
        mov     si, offset user
        mov     di, ecore
        mov     cx, word ptr [u.segmnt]
        mov     es, cx
        mov     cx, 32
        rep     movsw
        mov     dx, ds
        mov     es, dx

         mov     word ptr [u.base], core
                ; mov $core,u.base / move address core to u.base
        mov     word ptr [u.count], ecore - core + 64
                ; mov $ecore-core,u.count / put the byte count in u.count
        mov     word ptr [u.fofp], offset u.off
                ; mov $u.off,u.fofp / more user offset to u.fofp
        mov     word ptr [u.off], cx ; 0
                ; clr u.off / clear user offset
        call    writei
                ; jsr r0,writei / write out the core image to the user
        ;mov    word ptr [u.base], offset user
                ; mov $user,u.base / pt. u.base to user
        ;mov    word ptr [u.count], 64
                ; mov $64.,u.count / u.count = 64
        ;call   writei
                ; jsr r0,writei / write out all the user parameters
        neg     ax ; r1
                ; neg r1 / make i-number positive
        call    iclose
                ; jsr r0,iclose / close the core image file
        jmp     short sysexit
                ; br sysexit /
badsys_3: ;3:
        db      'core',0,0
                ; <core\0\0>

@@:     ; 22/09/2013
        retn

intract: ; / interrupt action
        ; 07/12/2013
        ; 06/12/2013
        ; 20/10/2013
        ; 22/09/2013
        ; 03/09/2013
        ; 16/05/2013 task/process/tty switch
        ; 15/05/2013 (ptty, set video page)
        ; 09/05/2013
        ; 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.
        ; 07/12/2013
        inc     word ptr [u.quit]
        jz      short @f ; FFFFh -> 0
        dec     word ptr [u.quit]
        jmp     short @b
@@:
        ; 20/10/2013
        pop     ax ; call intract -> retn
        pop     ax ; user's stack pointer ('sysrele')
        ;
        xor     ax, ax
        inc     al      ; mov ax, 1
        ; 06/12/2013
        ;mov    word ptr [u.quit], ax ; reset to
                                ; 'ctrl+brk' enabled
        ;jmp    sysexit
;;;
        ; 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>
        ; 14/02/2014
        ; 05/02/2014
        ; 17/09/2013
        ; 30/08/2013
        ; 19/04/2013
        ;
        ; '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 AX register.
        ;
        ;        Other parameters are in DX, BX, CX, SI, DI, BP registers
        ;       depending of function details.
        ;
        ; ('swap' procedure is mostly different than original UNIX v1.)
        ;
; / terminate process
        ; AX = 1
        dec     ax ; 0
        mov     word ptr [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
        xor     bh, bh ; 0
        mov     bl, byte ptr [u.uno]
                ; movb u.uno,r1 / yes, move dying process's number to r1
        mov     byte ptr [BX]+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, word ptr [BX]+p.pid-2
                ; mov p.pid-2(r1),r3 / move dying process's name to r3
        mov     dx, word ptr [BX]+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     si, si ; 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     word ptr [BX]+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 ptr [BX]+p.stat-1, 3 ; SZOMB, 05/02/2014
                ; cmpb p.stat-1(r2),$3 / is the child of this
                                    ; / dying process a zombie
        jne     short sysexit_3
                ; bne 2f / no
        mov     byte ptr [BX]+p.stat-1, ah ; 0, SFREE, 05/02/2014
                ; clrb p.stat-1(r2) / yes, free the child process
sysexit_3: ; 2:
        ;shr    bx, 1
        shl     bl, 1
                ; asl r2
sysexit_4: ; 3:
                ; / search the process name table
                ; / for the dying process's parent
        cmp     word ptr [BX]+p.pid-2, dx ; 17/09/2013
                ; cmp p.pid-2(r2),r4 / found it?
        jne     short sysexit_5
                ; bne 3f / no
        mov     si, bx
                ; mov r2,r5 / yes, put index to p.pid table (parents
                          ; / process # x2) in r5
sysexit_5: ; 3:
        ;cmp    bx, nproc + nproc
        cmp     bl, nproc + nproc
                ; cmp r2,$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     si, si ; 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, byte ptr [SI]+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, byte ptr [u.uno]
                ; movb u.uno,r3 / move dying process's number to r3
        mov     byte ptr [BX]+p.stat-1, 3
                ; 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 ptr [SI]+p.stat-1, 1 ; SRUN ; 05/02/2014
        ;dec    byte ptr [SI]+p.stat-1
                ; decb p.stat-1(r1) / awaken it by putting it (parent)
        mov     ax, si ; r1  (process number in AL)
        ; 14/02/2014
        ;mov    bx, offset runq + 4
                ; mov $runq+4,r2 / on the runq
        call    putlu
                ; jsr r0, putlu
sysexit_6: ; 2:
                ; / the process dies
        mov     byte ptr [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
        ; 30/08/2013
        ;mov    sp, word ptr [u.sp_] ; Retro Unix 8086 v1 modification!
        ;jmp     @b
         ;;jmp    swapret ; Retro UNIX 8086 v1 modification !
hlt_sys:
        ;sti ; 18/01/2014
@@:
        hlt
        ;jmp    short hlt_sys
        jmp     short @b
                ; 0 / and thereby kill it; halt?
```

```
syswait: ; < wait for a processs to die >
        ; 05/02/2014
        ; 10/12/2013
        ; 04/11/2013
        ; 30/10/2013
        ; 23/10/2013
        ; 24/05/2013
        ; '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, nothinh 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:
        xor     bh, bh
        mov     bl, byte ptr [u.uno]
                ; 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, word ptr [BX]+p.pid-2
                ; mov p.pid-2(r1),r1 / get the name of this process
        xor     si, si
                ; clr r2
        xor     cx, cx ; 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, word ptr [SI]+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 ptr [SI]+p.stat-1, 3 ; SZOMB, 05/02/2014
                ; cmpb p.stat-1(r2),$3 / is the child process a zombie?
        jne     short syswait_2
                ; bne 2f / no, skip it
        mov     byte ptr [SI]+p.stat-1, bh ; 0
                ; clrb p.stat-1(r2) / yes, free it
        shl     si, 1
                ; asl r2 / r2x2 to get index into p.pid table
        mov     ax, word ptr [SI]+p.pid-2
        mov     word ptr [u.r0], ax
                ; mov p.pid-2(r2),*u.r0
                              ; / put childs process name in (u.r0)
        jmp     sysret
                ; br sysret1 / return cause child is dead
syswait_2: ; 2:
        shl     si, 1
                ; asl r2 / r2x2 to get index into p.ppid table
syswait_3: ; 3:
        cmp     si, nproc+nproc
                ; cmp r2,$nproc+nproc / have all processes been checked?
        jb      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 @f
        ;jz     error
                ; beq error1 / there are no children, error
        mov     word ptr [u.r0], cx ; 0
        jmp     error
@@:
        mov     bl, byte ptr [u.uno]
                ; movb u.uno,r1 / there are children so put
                              ; / parent process number in r1
        inc     byte ptr [BX]+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 >
        ; 14/02/2014
        ; 05/02/2014
        ; 07/12/2013
        ; 06/12/2013
        ; 18/11/2013
        ; 17/09/2013
        ; 16/09/2013
        ; 30/08/2013
        ; 08/08/2013
        ; 22/07/2013
        ; 26/05/2013
        ; 24/05/2013
        ; '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
        ; BX = return address for child process
             ; (Retro UNIX 8086 v1 modification !)
        xor     si, si
                ; clr r1
sysfork_1: ; 1: / search p.stat table for unused process number
        inc     si
                ; inc r1
        cmp     byte ptr [SI]+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       ; 08/08/2013
                ; 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'
        ;       system call.
        ;       so, it is not needed to increment return address
        ;       of system call on the top of the user's stack.
        ;       If the routine would be same with original UNIX v1
        ;       'sysfork' routine, 'add word ptr [SP]+12, 2'
        ;       instruction would be put here.
        ;;      add word ptr [SP]+12, 2
        ;;      jmp error
                ;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 ; 08/08/2013
sysfork_2: ; 1:
        ; Retro UNIX 8086 v1. modification !
        ; 08/08/2013
        mov     ax, offset sysret
        push    ax ; *
        mov     word ptr [u.usp], sp
        ;;push es
         ; 08/08/2013
        ; Return address for the parent process is already set
        ; by sysenter routine.
        ;mov    ax, word ptr [u.segmnt]
        ;mov    es, ax
        ;mov    bp, sp
        ;mov    di, word ptr [BP]+12 ; user's stack pointer
        ;;pop   es
        ;push   word ptr ES:[DI]
        ;;;mov  ax, word ptr ES:[DI] ; return address (IP)
        ;;;push ax  ; **** return address for the parent process
        ;;mov   ax, cs
        ;;mov   es, ax
        ;;
        push    word ptr [u.segmnt] ; **
                              ; Retro UNIX 8086 v1 feature only !
        ;
        ; 06/12/2013
        ;push   word ptr [u.uno] ; ***
                ; movb u.uno,-(sp) / save parent process number
        xor     ah, ah
        mov     al, byte ptr [u.uno] ; parent process number
        push    ax ; ***
        mov     di, ax
        ; 07/12/2013
         mov     al, byte ptr [DI]+p.ttyc-1 ; console tty (parent)
         mov     byte ptr [SI]+p.ttyc-1, al ; set child's console tty
        ; 05/02/2014 (p.ttys has been removed)
         ;mov     byte ptr [SI]+p.ttys-1, al ; set parent's console tty
        mov      byte ptr [SI]+p.waitc-1, al ; set parent's console tty
        ; 22/07/2013
        mov     ax, si
        mov     byte ptr [u.uno], al
        ;
        ;mov    word ptr [u.uno], si
                ;movb r1,u.uno / set child process number to r1
        inc     byte ptr [SI]+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
        ;;and   di, di
        ;;jz    short sysfork_3
                 ; beq 2f / branch, if no such tty assigned
        ;; ????
                ; clrb 6(r2) / clear interrupt character in tty buffer
sysfork_3: ; 2:
        push    bx   ; * return address for the child process
                     ; * Retro UNIX 8086 v1 feature only !
        ;;mov   ax, si ;; 22/07/2013
        ; 14/02/2014
        ;mov    bx, offset runq + 2 ; middle priority !
                ; (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 ptr [mpid]
                ; inc mpid / increment m.pid; get a new process name
        mov     ax, word ptr [mpid]
        mov     word ptr [SI]+p.pid-2, ax
                ;mov mpid,p.pid-2(r1) / put new process name
                                    ; / in child process' name slot
        pop     dx   ; * return address for the child process
                     ; * Retro UNIX 8086 v1 feature only !

        ; 08/08//2013
        pop     bx ; ***
        push    bx ; ***
        ;mov     bp, sp
         ;mov     bx, word ptr [BP]  ; ***
                ; movb (sp),r2 / put parent process number in r2
        xor     bh, bh ; 08/08/2013
        shl     bx, 1
                ;asl r2 / multiply by 2 to get index into below tables
        mov     ax, word ptr [BX]+p.pid-2
                ; mov p.pid-2(r2),r2 / get process name of parent
                                   ; / process
        mov     word ptr [SI]+p.ppid-2, ax
                ; mov r2,p.ppid-2(r1) / put parent process name
                           ; / in parent process slot for child
        mov     word ptr [u.r0], ax
                ; mov r2,*u.r0 / put parent process name on stack
                             ; / at location where r0 was saved
        ; 22/07/2013
        call    segm_sw  ; User segment switch
        ; BX = New user segment ; 24/07/2013
        ;
        mov     ax, word ptr [u.segmnt] ; 08/08/2013
        mov     word ptr [u.segmnt], bx ; 24/07/2013
        mov     es, bx
        xor     si, si
        xor     di, di
        mov     cx, 16384
        mov     ds, ax ; 08/08/2013
        rep     movsw ; copy process (in current segment) to
                      ;new process segment
        ; 08/08/2013
        mov     ax, cs
        mov     ds, ax
        mov     ax, bx ; new user segment
        mov     bp, word ptr [u.sp_]
        mov     bx, word ptr [BP]+12 ; user's stack pointer
        mov     word ptr ES:[BX], dx ; *, CS:IP -> IP
                           ; * return address for the child process
        mov     word ptr ES:[BX]+2, ax ; CS:IP -> CS
                           ; * return address for the child process
        ;mov    ax, cs
        ;mov    es, ax
        ;*
        ;;mov   ax, offset sysret
        ;;push  ax ; *
                ; mov $sysret1,-(sp) /
        ;mov    word ptr [u.usp], 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
        ; ES = u.segmnt
        ; 06/12/2013
        ;push   word ptr [u.intr] ; ****
        ; 30/08/2013
        push    word ptr [u.ttyp] ; *****
        xor     ax, ax
        mov     word ptr [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
        ; ES = DS
        ;;mov   sp, word ptr [u.usp]
        ; 30/08/2013
        pop     word ptr [u.ttyp] ; *****
        ; 06/12/2013
        ;pop    word ptr [u.intr] ; ****
        ;;pop   ax ; *
                ; tst (sp)+ / bump stack pointer
        ;pop    word ptr [u.uno] ; ***
        pop     ax ; *** 22/07/2013
        mov     byte ptr [u.uno], al
                ;movb (sp)+,u.uno / put parent process number in u.uno
        ;
        pop     word ptr [u.segmnt] ; **
                               ; Retro UNIX 8086 v1 feature only !
        ;
        mov     ax, word ptr [mpid]
        mov     word ptr [u.r0], ax
                ; mov mpid,*u.r0 / put child process name on stack
                               ; / where r0 was saved
        ; 08/08/2013
        ; Return address for the parent process is already set
        ; by sysenter routine.
        ;pop    dx ; **** return address for the parent process
        ;mov    ax, word ptr [u.segmnt]
        ;mov    es, ax
        ;mov    word ptr ES:[BX]+2, ax ; user's CS for iret <- ax
        ;mov    word ptr ES:[BX], dx ; user's IP for iret <- dx
                ; add $2,18.(sp) / add 2 to pc on stack; gives parent
                                ; / process return
        ;pop    ax ; * 08/08/2013
         ;
         xor     si, si
                ;clr r1
sysfork_4: ; 1: / search u.fp list to find the files
              ; / opened by the parent process
         mov     bl, byte ptr [SI]+u.fp
                ; movb u.fp(r1),r2 / get an open file for this process
         or      bl, bl
        jz      short sysfork_5
                ; beq 2f / file has not been opened by parent,
                       ; / so branch
        xor      bh, bh ; 18/11/2013
         shl     bx, 1
                ; asl r2 / multiply by 8
         shl     bx, 1
                ; asl r2 / to get index into fsp table
         shl     bx, 1
                ; asl r2
         inc     byte ptr [BX]+fsp-2
                ; incb fsp-2(r2) / increment number of processes
                             ; / using file, because child will now be
                             ; / using this file
sysfork_5: ; 2:
         inc     si
                ; 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
        ; 08/08/2013
        retn    ; * -> sysret
        ;jmp    sysret
                ; br sysret1

segm_sw:
        ; 24/07/2013
        ; 23/07/2013
        ; 22/07/2013
        ; Retro UNIX 8086 v1 feature only !
        ; (User segment switch)
        ; INPUT -> none
        ; OUTPUT -> bx = new user segment
        ;           (word ptr [u.segmnt] = ax)
        ; ((Modified registers: cx))
        ;
        mov     cl, byte ptr [u.uno] ; 23/07/2013
        mov     bx, csgmnt ; segment of process 1
@@:
        dec     cl
        jz      short @f
        add     bx, 2048   ; (32768/16)
        jmp     short @b
@@:
        ;;mov   word ptr [u.segmnt], bx ;; 24/07/2013
        retn


sysread: ; < read from file >
        ; 23/05/2013
        ; '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,
        ;       Retro UNIX 8086 v1 argument transfer method 3 is used
        ;       to get sysread system call arguments from the user;
        ;       * 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.
        ;
        ;       NOTE: Retro UNIX 8086 v1 'arg' routine gets these
        ;             arguments in these registers;
        ;             (BX= file descriptor)
        ;             (CX= buffer address in user's program segment)
        ;             (DX= number of bytes)
        ;             then
        ;             * file descriptor (in BX) is moved into AX
        ;             * buffer address (in CX) is moved into 'u.base'.
        ;             * byte count (in DX) is moved into 'u.count'.
        ;
        call    rw1
                ; 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 @f
                ; br 1f
```

```
syswrite: ; < write to file >
        ; 23/05/2013
        ; '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,
        ;       Retro UNIX 8086 v1 argument transfer method 3 is used
        ;       to get syswrite system call arguments from the user;
        ;       * 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.
        ;
        ;       NOTE: Retro UNIX 8086 v1 'arg' routine gets these
        ;             arguments in these registers;
        ;             (BX= file descriptor)
        ;             (CX= buffer address in user's program segment)
        ;             (DX= number of bytes)
        ;             then
        ;             * file descriptor (in BX) is moved into AX
        ;             * buffer address (in CX) is moved into 'u.base'.
        ;             * byte count (in DX) is moved into 'u.count'.
        call    rw1
                ; jsr r0,rw1 / get i-number in r1 of file to write
        test    ah, 80h
                ; tst r1 / positive i-number ?
        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
@@: ; 1:
        mov     ax, word ptr [u.nread]
        mov     word ptr [u.r0], ax
                ; mov u.nread,*u.r0 / put no. of bytes transferred
                                ; / into (u.r0)
        jmp     sysret
                ; br sysret1
rw1:    ; 23/05/2013
        ; 'rw1' returns i-number of the file for 'sysread' & 'syswrite'.
        ; Retro UNIX 8086 v1 modification:
        ;       'arg' routine is different than 'arg' in original Unix v1.
        ;mov    ax, 3 ; number of arguments
        ;call   arg
        ; 24/05/2013
        ; System call registers: bx, cx, dx (through 'sysenter')
        mov     word ptr [u.base], cx ; buffer address/offset
                                   ; (in the user's program segment)
        mov     word ptr [u.count], dx
                ;
                ; jsr r0,arg; u.base / get buffer pointer
                ; jsr r0,arg; u.count / get no. of characters
        ;;mov   ax, bx ; file descriptor
                ; mov *u.r0,r1 / put file descriptor
                             ; / (index to u.fp table) in r1
        ;; call getf
         ; BX = 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
        retn
                ; rts r0
```

```
sysopen: ;<open file>
        ; 27/05/2013
        ; 24/05/2013
        ; 22/05/2013
        ; '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,
        ;       Retro UNIX 8086 v1 argument transfer method 2 is used
        ;       to get sysopen system call arguments from the user;
        ;       * 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).
        ;
        ;       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).
        ; * 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
        ; 24/05/2013
        ; system call registers: bx, cx (through 'sysenter')

        mov     word ptr [u.namep], bx
        push    cx
        call    namei
                ; jsr r0,namei / i-number of file in r1
        ;and    ax, ax
        ;jz     error ; File not found
        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 @f
                ; beq 1f / yes, leave i-number positive
        neg     ax
                ; neg r1 / open for writing so make i-number negative
@@: ;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 @f
                ; beq op1 / is open for read op1
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
        ;; perhaps iclose then makes it positive ??? E. Tan [22/05/2013]
@@: ;op1:
        xor      si, si
                ; clr r2 / clear registers
        xor      bx, bx
                ; clr r3
@@: ;1: / scan the list of entries in fsp table
        cmp      byte ptr [SI]+u.fp, bl ; 0
                ; tstb u.fp(r2) / test the entry in the u.fp list
        jna       short @f
                ; beq 1f / if byte in list is 0 branch
        inc      si
                ; inc r2 / bump r2 so next byte can be checked
        cmp      si, 10
                ; cmp r2,$10. / reached end of list?
        jb      short @b
                ; blt 1b / no, go back
        jmp     error
                ; br error2 / yes, error (no files open)
@@: ; 1:
        cmp      word ptr [BX]+fsp, 0
                ; tst fsp(r3) / scan fsp entries
        jna      short @f
                ; beq 1f / if 0 branch
        add      bx, 8
                ; add $8.,r3 / add 8 to r3
                        ; / to bump it to next entry mfsp table
        cmp      bx, nfiles*8
                ; cmp r3,$[nfiles*8.] / done scanning
        jb      short @b
                ; blt 1b / no, back
        jmp     error
                ; br error2 / yes, error
@@: ; 1: / r2 has index to u.fp list; r3, has index to fsp table
        mov      word ptr [BX]+fsp, ax
                ; mov r1,fsp(r3) / put i-number of open file
                        ; / into next available entry in fsp table,
        mov     di, word ptr [cdev] ; word ? byte ?
        mov      word ptr [BX]+fsp+2, di
                ; mov cdev,fsp+2(r3) / put # of device in next word
        xor     di, di
        mov      word ptr [BX]+fsp+4, di
                ; clr fsp+4(r3)
        mov      word ptr [BX]+fsp+6, di
                ; clr fsp+6(r3) / clear the next two words
        shr      bx, 1
                ; asr r3
        shr      bx, 1
                ; asr r3 / divide by 8
        shr      bx, 1
                ; asr r3 ; / to get number of the fsp entry-1
        ;inc     bx
        inc      bl
                ; inc r3 / add 1 to get fsp entry number
        mov      byte ptr [SI]+u.fp, bl
                ; movb r3,u.fp(r2) / move entry number into
                        ; / next available slot in u.fp list
        mov      word ptr [u.r0], si
                ; mov r2,*u.r0 / move index to u.fp list
                             ; / into r0 loc on stack
        jmp     sysret
                ; br sysret2
```

```
syscreat: ; < create file >
        ; 27/05/2013
        ; '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,
        ;       Retro UNIX 8086 v1 argument transfer method 2 is used
        ;       to get syscreate system call arguments from the user;
        ;       * 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).
        ;
        ;       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).
        ; * 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     word ptr [u.namep], bx ; file name address
        push    cx ; mode
        call    namei
                ; jsr r0,namei / get the i-number
         ;and   ax, ax
        ;jz     short @f
        jc      short @f
                ; 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     short op0
                ; br op0
@@: ; 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, word ptr [u.dirbuf]
                ; mov u.dirbuf,r1 / put i-number
                                ; / for this new file in r1
         jmp     short op0
                ; br op0 / open the file
```

```
sysmkdir: ; < make directory >
        ; 02/08/2013
        ; 27/05/2013
        ; '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,
        ;      Retro UNIX 8086 v1 argument transfer method 2 is used
        ;      to get sysmkdir system call arguments from the user;
        ;      * 1st argument, name is pointed to by BX register
        ;      * 2nd argument, mode is in 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.

; / 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    word ptr [u.namep], bx
        push   cx
        call   namei
               ; jsr r0,namei / get the i-number
               ;     br .+4 / if file not found branch around error
         ;xor  ax, ax
        ;jnz   error
        jnc    error
               ; br  error2 / directory already exists (error)
        cmp    byte ptr [u.uid_], 0 ; 02/08/2013
               ;tstb u.uid / is user the super user
        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 /
```

```
sysclose: ;<close file>
        ; 26/05/2013
        ; 22/05/2013
        ; '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   ax, 1 ; one/single argument, put argument in BX
        ;;call  arg
        ;mov    bx, word ptr [u.sp_] ; points to user's BP register
        ;add    bx, 6 ; bx now points to BX on stack
        ;mov    ax, word ptr [BX]
                ; mov *u.r0,r1 / move index to u.fp list into r1
        mov     ax, bx ; 26/05/2013
        call    fclose
                ; jsr r0,fclose / close the file
        jc      error
                        ; br error2 / unknown file descriptor
        jmp     sysret
                ; br sysret2

sysemt:
        ; 10/04/2014 Bugfix [u.uid --> u.uid_]
        ; 18/01/2014
        ; 10/12/2013
        ; 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 ptr [u.uid_], 0   ; BugFix u.uid --> u.uid_
        ja      error
        push    es
        xor     ax, ax
        mov     es, ax ; 0
        mov     di, 28*4 ; INT 1Ch vector - offset
        ; 18/01/2014
        cli
        and     bx, bx
        jz      short emt_2
        ; Enable INT 1Ch time-out functionality.
        mov     ax, offset clock
emt_1:
        stosw    ; offset
        mov     ax, cs
        stosw   ; segment
        ; 18/01/2014
        sti
        pop     es
        jmp     sysret
emt_2:
        ; Disable INT 1Ch time-out functionality.
        mov     ax, offset emt_iret
        jmp     short emt_1
emt_iret:
        iret
        ; 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:
        ; 03/06/2013,
        ; Retro UNIX 8086 v1 modification:
        ;       not a valid system call ! (not in use)
        ;
        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 >
        ; 02/08/2013
        ; 03/06/2013
        ; '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    word ptr [u.namep], bx
        call    namei
               ; jsr r0,namei / get its i-number
         jc     error
               ; br error2 / no, such file
        call    iget
               ; jsr r0,iget / get i-node into core
        mov     al, byte ptr [u.uid_] ; 02/08/2013
        cmp     al, byte ptr [i.uid]
               ; cmpb u.uid,i.uid / is user same as owner
        je      short @f
               ; beq 1f / yes
        and     al, al
                ; tstb u.uid / no, is user the super user
        jnz     error
                ; bne error2 / no, error
@@: ;1:
        call    setimod
                ; jsr r0,setimod / fill in modification data,
                               ; / time etc.
        ; Retro UNIX 8086 v1 modification !
        mov     si, offset p_time
        mov     di, offset i.mtim
        movsw
        movsw
                ; mov 4(sp),i.mtim / move present time to
                ; mov 2(sp),i.mtim+2 / modification time
         jmp    sysret
                ; br sysret2

@@:
        retn

sysstty: ; < set tty status and mode >
        ; 12/07/2014
        ; 04/07/2014
        ; 26/06/2014
        ; 15/04/2014
        ; 18/01/2014
        ; 17/01/2014
        ; 16/01/2014
        ; 14/01/2014
        ; 13/01/2014
        ; 12/01/2014
        ; 07/12/2013
        ; 04/12/2013
        ; 30/10/2013
        ; 24/10/2013
        ; 03/09/2013
        ; 19/08/2013
        ; 15/08/2013 (set console tty)
        ; 11/08/2013
        ; 16/07/2013
        ; 15/07/2013
        ; 02/06/2013
        ;
        ; '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 CH = 0
        ;             set console tty for (current) process
        ;             CL = tty number (0 to 9)
        ;             (If ch = 0, character will not be written)
        ;           If CH > 0
        ;             set cursor position or comm. parameters only
        ;              If CL = FFh
        ;               set cursor position for console tty
        ;              or CL = tty number (0 to 9)
        ;             CH = character will be written
        ;               at requested cursor position (in DX)
        ;              (For tty numbers 0 to 7, if CH = FFh, character
        ;             will not be written)
        ;          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 -->
        ;              CL = 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)
        ;              CH = 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.
        ;
        ; 14/01/2014
        mov     word ptr [u.r0], 0FFFFh
        and     bx, bx
         jnz      sysstty_6
; set console tty
        ; 17/01/2014
        cmp     cl, 9
        jna     short sysstty_0
        or      ch, ch
        jz      error
        cmp     cl, 0FFh
        jb      error
        mov     bl, byte ptr [u.uno] ; process number
        mov     cl, byte ptr [BX]+p.ttyc-1 ; current/console tty
sysstty_0:
        cmp     cl, 8
        jb      short sysstty_2
        ;
        cmp     dh, 0FFh
        je      short sysstty_2
                ; set communication parameters for serial ports
        mov     si, offset com1p
        ; 12/07/2014
        cmp     cl, 9
        jb      short sysstty_1
        inc     si
sysstty_1:
        mov     byte ptr [SI], dl ; comm. parameters
sysstty_2:
        push    dx
        push    cx
        xor     dl, dl  ; sysstty call sign
        mov     al, cl
        mov     byte ptr [u.r0], al
        ; AH = 0
        ;cbw
        ; ah = 0
        call    ottyp
        pop     cx
        pop     dx
        ;
        jc      error
        xor     bh, bh
        ; 17/01/2014
        and     ch, ch  ; set cursor position
                        ; or comm. parameters ONLY
        jnz     short sysstty_3
        mov     bl, byte ptr [u.uno] ; process number
        mov     byte ptr [BX]+p.ttyc-1, cl ; current/console tty
sysstty_3:
        ; 16/01/2014
        mov     al, ch ; character  ; 0 to FFh
        cmp     cl, 7
         jna     short 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     byte ptr [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_4:
        ; 12/07/2014
        xchg    ah, al ; al = 0 -> al = ah, ah = 0
        sub     al, 8
        mov     dx, ax ; 0 or 1
        mov     ah, 3  ; Get serial port status
        int     14h
sysstty_5:
        mov     byte ptr [u.r0]+1, ah ; line status
        pushf
        xor     dl, dl ; sysstty call sign
        mov     al, byte ptr [u.ttyn] ; 26/06/2014
        cbw     ; ax = tty number (ah=0)
        call    cttyp
        popf
        jc      error
        jmp     sysret

sysstty_6:
        push    dx
        push    cx
        mov     word ptr [u.namep], bx
        call    namei
        pop     cx
        pop     dx
        jc      error
        cmp     ax, 19  ; inode number of /dev/COM2
        ja      error
        cmp     al, 10 ; /dev/tty0 .. /dev/tty7
                       ; /dev/COM1, /dev/COM2
        jb      short sysstty_7
        sub     al, 10
        jmp     short sysstty_8

sysstty_7:
        cmp     al, 1 ; /dev/tty
        jne     error
        xor     bh, bh
        mov     bl, byte ptr [u.uno] ; process number
        mov     al, byte ptr [BX]+p.ttyc-1 ; current/console tty
sysstty_8:
        mov     byte ptr [u.r0], al
        push    dx
        push    ax
        push    cx
        call    ottyp
        pop     cx
        pop     ax
        pop     dx
        jc      error
        ; 12/07/2014
        xchg    al, cl
        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
        mov     bl, cl ; (tty number = video page number)
        ;xor    bh, bh
        call    set_cpos
        pop     ax
        pop     cx
sysstty_10:
        ; 17/01/2014
        inc     ch
        jz       short sysstty_11 ; ch = FFh
        dec     ch
        jz       short sysstty_11 ; ch = 0
                ; ch > 0 and ch < FFh
        ; write a character at current cursor position
        mov     ah, 07h ; ah = 7 (color/attribute), al = char
        ; 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
        mov     al, cl
        cbw
        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 >
        ; 12/07/2014
        ; 22/04/2014
         ; 26/01/2014
        ; 17/01/2014
        ; 16/01/2014
        ; 07/12/2013
        ; 04/12/2013
        ; 03/09/2013
        ; 15/08/2013
        ; 16/07/2013
        ; 02/06/2013
        ; 30/05/2013
        ; '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 7)
        ;            CL = 1 --> return video page status (tty 0 to 7)
        ;            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
        ;

sysgtty_0:
gtty:    ; get (requested) tty number
        ; 12/07/2014
        ; 22/04/2014
        ; 15/04/2014
         ; 26/01/2014
        ; 17/01/2014
        ; 16/01/2014
        ; 07/12/2013
        ; 04/12/2013
        ; 03/09/2013
        ; 19/08/2013
        ; 16/07/2013
        ; 02/06/2013
        ; 30/05/2013
        ; Retro UNIX 8086 v1 modification !
        ;
        ; ((Modified registers: AX, BX, CX, DX, SI, DI, BP))
        ;
        ; 16/01/2014
        mov     word ptr [u.r0], 0FFFFh
        cmp     cl, 1
        ja      error
        ;
        and     bx, bx
        jz      short sysgtty_1
        ;
        mov     word ptr [u.namep], bx
        call    namei
        jc      error
        ;
        xor     bh, bh
        cmp     ax, 1
        jna     short sysgtty_2
        sub     ax, 10
        cmp     ax, 9
        ja      error
        mov     ch, al
        jmp     short sysgtty_4
sysgtty_1:
        ; 16/01/2014
        cmp     ch, 10
        ja      error
        dec     ch ; 0 -> FFh (negative)
        jns     short sysgtty_3 ; not negative
        ;
sysgtty_2:
        ; get tty number of console tty
        mov     ah, byte ptr [u.uno]
        mov     bl, ah
        ;xor    bh, bh
        mov     ch, byte ptr [BX]+p.ttyc-1
sysgtty_3:
        mov     al, ch
sysgtty_4:
        mov     byte ptr [u.r0], al
        ;cmp    ch, 9
        ;ja     error
        cmp     ch, 8 ; cmp al, 8
        jb      short sysgtty_6
        ;
        ; 12/07/2014
        mov     dx, 0
        je      short sysgtty_5
        inc     dl
sysgtty_5:
        ; 12/07/2014
        mov     ah, 3 ; get serial port status
        int     14h
        xchg    ah, al
        mov     word ptr [BP]+6, ax ; serial port status
        mov     ah, byte ptr [u.uno]
        mov     byte ptr [u.r0]+1, ah
        mov     word ptr [BP]+8, 0 ; data status (0 = not ready)
        test    al, 80h
        jnz     error
        test    al, 1
        jz      sysret
        dec     word ptr [BP]+8 ; data status (FFFFh = ready)
        jmp     sysret
sysgtty_6:
        mov     bp, word ptr [u.sp_]
        mov     byte ptr [u.ttyn], al ; tty number
        ;xor    bh, bh
        mov     bl, al ; tty number (0 to 7)
        shl     bl, 1  ; aligned to word
        ; 22/04/2014
         add     bx, offset ttyl
        mov     ah, byte ptr [BX]
        cmp     ah, byte ptr [u.uno]
        je      short sysgtty_7
        and     ah, ah
        ;jz     short sysgtty_7
        jnz     short sysgtty_8
        ;mov    ah, 0FFh
sysgtty_7:
        mov     byte ptr [u.r0]+1, ah
sysgtty_8:
        or      cl, cl
        jnz     short sysgtty_9
        mov     al, 1  ; test a key is available
        call    getc
        mov     word ptr [BP]+6, ax ; bx, character
        jmp     sysret
sysgtty_9:
        mov     bl, byte ptr [u.ttyn]
        ; bl = video page number
        call    get_cpos
        ; dx = cursor position
        mov     word ptr [BP]+6, dx ; bx
        ;mov    bl, byte ptr [u.ttyn]
        ; bl = video page number
        call    read_ac_current
        ; ax = character and attribute/color
        mov     word ptr [BP]+8, ax ; cx
        jmp     sysret

; 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
```

```
; ****************************************************************************
;
; UNIX.ASM (RETRO UNIX 8086 Kernel - Only for 1.44 MB floppy disks)
; ----------------------------------------------------------------------------
; U2.ASM (include u2.asm) //// UNIX v1 -> u2.s

; RETRO UNIX 8086 (Retro Unix == Turkish Rational Unix)
; Operating System Project (v0.1) by ERDOGAN TAN (Beginning: 11/07/2012)
; 1.44 MB Floppy Disk
; (11/03/2013)
;
; [ Last Modification: 24/03/2014 ] ;;; completed ;;;
;
; Derivation from UNIX Operating System (v1.0 for PDP-11)
; (Original) Source Code by Ken Thompson (1971-1972)
; <Bell Laboratories (17/3/1972)>
; <Preliminary Release of UNIX Implementation Document>
;
; ****************************************************************************
; 24/03/2014 sysbreak
; 12/01/2014 fclose
; 06/12/2013 sysexec
; 19/11/2013 sysbreak
; 18/11/2013 getf (getf1)
; 24/10/2013 sysexec
; 03/09/2013 sysexec (u.intr, u.quit reset -> enabled)
; 05/08/2013 fclose, seektell
; 02/08/2013 maknod, (u.uid -> u.uid_)
; 01/08/2013 mkdir
; 31/07/2013 u.namei_r -> namei_r, maknod
; 30/07/2013 fclose
; 28/07/2013 namei (u.namei_r)
; 26/07/2013 namei (namei_r)
; 25/07/2013 sysexec (arguments)
; 24/07/2013 sysexec
; 22/07/2013 sysexec, namei
; 18/07/2013 sysexec, namei
; 17/07/2013 maknod (inode->i)
; 09/07/2013 namei (rootdir)
; 07/07/2013 sysseek, systell, sysintr, sysquit, syssetuid, sysgetuid
; 07/07/2013 syschmod, syschown
; 20/06/2013 syschmod, syschown, systime, sysstime, sysbreak
; 19/06/2013 syslink, sysunlink, sysstat, sysfstat, syschdir
; 04/06/2013 sysexec
; 03/06/2013 sysexec
; 27/05/2013 namei (stc)
; 23/05/2013 getf1
; 02/05/2013 maknod
; 29/04/2013 mkdir
; 25/04/2013 anyi
; 24/04/2013 namei
; 19/04/2013 fclose
; 11/03/2013

syslink:
        ; 19/06/2013
        ; '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,
        ;       Retro UNIX 8086 v1 argument transfer method 2 is used
        ;       to get syslink system call arguments from the user;
        ;       * 1st argument, name 1 is pointed to by BX register
        ;       * 2nd argument, name 2 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.
        ;

; / name1, name2
        ;call   arg2
        ;       jsr r0,arg2 / u.namep has 1st arg u.off has 2nd
        mov     word ptr [u.namep], bx
        push    cx
        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
        call    iget
                ; jsr r0,iget / get the i-node into core
        pop     word ptr [u.namep] ; cx
                ; 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 ptr [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
        pop     cx
        cmp     cx, word ptr [cdev]
                ; cmp (sp)+,cdev / u.dirp now points to
                              ; / end of current directory
        jne     error
                 ; bne error9
        pop     ax
        push    ax
        mov     word ptr [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 ptr [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:
        ; 02/08/2013
        ; 04/05/2013
        ; '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: AX, DX, BX, CX, SI, DI, BP))
        ;
        ; / 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 ptr [u.uid_], 0
                ; tstb u.uid / super user
        jna     short @f
                ; beq 1f / yes, don't care
        push    word ptr [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 ptr [i.flgs], 4000h ; Bit 14 : Directory flag
                ; bit $40000,i.flgs / is it a directory
        jnz     error
                ; bne error9 / yes, error
        pop     ax
                ; mov (sp)+,r1 / no, put current i-number in r1 (ii)
        call    iget
                ; jsr r0,iget / get it back in
@@: ; 1:
        retn
                ; rts r0

sysunlink:
        ; 19/06/2013
        ; '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   ax, 1 ; one/single argument, put argument in BX
        ;;call  arg
        ;mov    bp, word ptr [u.sp_] ; points to user's BP register
        ;add    bp, 6 ; bx now points to BX on stack
        ;mov    bx, word ptr [BP]
        mov     word ptr [u.namep], bx
                ;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
        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     word ptr [u.dirbuf], ax ; 0
                ; clr u.dirbuf / no, clear the location that will
                           ; / get written into the i-number portion
                         ; / of the entry
        sub     word ptr [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 ptr [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:
        ; 01/08/2013
        ; 29/04/2013
        ; '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: AX, DX, BX, CX, SI, DI, BP))
        ;

        mov     cx, 4
        xor     ax, ax
         mov     di, offset u.dirbuf+2
        mov     si, di
        rep     stosw
                ; jsr r0,copyz; u.dirbuf+2; u.dirbuf+10. / clear this
        mov     di, si
        mov     si, word ptr [u.namep]
                ; mov u.namep,r2 / r2 points to name of directory entry
                ; mov $u.dirbuf+2,r3 / r3 points to u.dirbuf+2
mkdir_1: ; 1:
        ; / put characters in the directory name in u.dirbuf+2 - u.dirbuf+10
         ; 01/08/2013
        push    cs ; push ds
        mov     ax, word ptr [u.segmnt]
        mov     ds, ax
@@:
        lodsb
                ; movb (r2)+,r1 / move character in name to r1
        and     al, al
        jz      short mkdir_2
                ; beq 1f / if null, done
        cmp     al, '/'
                ; cmp r1,$'/ / is it a "/"?
        je      short @f
        ;je     error
                ; beq error9 / yes, error
         cmp     di, offset u.dirbuf+10
                ; cmp r3,$u.dirbuf+10. / have we reached the last slot for
                                    ; / a char?
        je      short @b
        ;je     short mkdir_1
                ; beq 1b / yes, go back
        stosb
                ; movb r1,(r3)+ / no, put the char in the u.dirbuf

        ; 01/08/2013
        jmp     short @b
        ; jmp   short mkdir_1
                ; br 1b / get next char
```

```
@@:
        ; 01/08/2013
        pop     ds
        jmp     error

mkdir_2: ; 1:
        ; 01/08/2013
        pop     ds
        ;
        mov     ax, word ptr [u.dirp]
        mov     word ptr [u.off], ax
                ; mov u.dirp,u.off / pointer to empty current directory
                                ; / slot to u.off
wdir: ; 29/04/2013
         mov     word ptr [u.base], offset u.dirbuf
                ; mov $u.dirbuf,u.base / u.base points to created file name
         mov     word ptr [u.count], 10
                ; mov $10.,u.count / u.count = 10
        mov     ax, word ptr [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 ptr [mkdir_w] ; the caller is 'mkdir' sign
        call    writei
                ; jsr r0,writei / write into directory
        retn
                ; rts r0

sysexec:
        ; 06/12/2013
        ; 24/10/2013, 22/09/2013, 03/09/2013
        ; 02/08/2013, 25/07/2013, 24/07/2013
        ; 22/07/2013, 18/07/2013, 03/06/2013
        ; 'sysexec' initiates execution of a file whose path name if
        ; pointed to by 'name' in the sysexec call.
        ; 'ssysexec' 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 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
        mov     word ptr [u.namep], bx ; argument 1
        push    cx      ; argument 2
        call    namei
                ; jsr r0,namei / namei returns i-number of file
                            ; / named in sysexec call in r1
        jc      error
                ; br error9
        call    iget
                ; jsr r0,iget / get i-node for file to be executed
        test    word ptr [i.flgs], 10h
                ; bit $20,i.flgs / is file executable
        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 ptr [i.flgs], 20h
                ; bit $40,i.flgs / test user id on execution bit
        jz      short sysexec_1
                ; beq 1f
        cmp     byte ptr [u.uid_], 0 ; 02/08/2013
                ; tstb u.uid / test user id
        jna     short sysexec_1
                ; beq 1f / super user
        mov     cl, byte ptr [i.uid]
        mov     byte ptr [u.uid_], cl ; 02/08/2013
                ; movb i.uid,u.uid / put user id of owner of file
                                ; / as process user id
sysexec_1: ; 1:
        ; 22/07/2013
        call    segm_sw   ; User segment switch
        ; BX = New user segment ; 24/07/2013
        ;
        pop     cx
                ; 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
        ; 24/07/2013
        mov     sp, sstack ; offset sstack
                ; mov $sstack,sp / stack space used during swapping
        ;push   cx
                ; mov r5,-(sp) / save arguments pointer on stack
        mov     di, ecore
                ; mov $ecore,r5 / r5 has end of core
        ;mov    bp, core
        xor     bp, bp   ; core = 0
                ; mov $core,r4 / r4 has start of users core
        mov     word ptr [u.base], bp
                ; mov r4,u.base / u.base has start of users core
        ; 24/07/2013
        mov     es, bx ; new user segment
                ; If the caller is a user, es = word ptr [u.segmnt]
                ; If the caller is system (sysexec for '/etc/init')
                ;     es = csgmnt and word ptr [u.segmnt] = cs
        mov     dx, word ptr [u.segmnt]
        mov     ds, dx
        mov     bx, cx
                ; mov (sp),r2 / move arguments list pointer into r2
sysexec_2: ; 1:
        ; AX = i-number of the file (at return of 'iopen' call)
        mov     dx, word ptr [BX]
        and     dx, dx
        jz      short @f
                ; tst (r2)+ / argument char = "nul"
                ; bne 1b
        inc     bx
        inc     bx
        jmp     short sysexec_2
@@:
        ; tst -(r2) / decrement r2 by 2; r2 has addr of end of
                  ; / argument pointer list
sysexec_3: ; 1:
             ; / move arguments to bottom of users core
        dec     bx
        dec     bx
        ;mov    si, word ptr [BX]
                ;; mov -(r2),r3 / (r3) last non zero argument ptr
        cmp     bx, cx
                ; cmp r2,(sp) / is r2 = beginning of argument
                          ; / ptr list
        jb      short sysexec_6
                ; blo 1f / branch to 1f when all arguments
                       ; / are moved
        mov     si, word ptr [BX]
                ; mov -(r2),r3 / (r3) last non zero argument ptr
sysexec_4: ; 2:
        mov     dl, byte ptr [SI]
        and     dl, dl
                ; tstb (r3)+
        jz      short sysexec_5
        inc     si
        jmp     short sysexec_4
                ; bne 2b / scan argument for \0 (nul)
sysexec_5: ; 2:
        dec     di
        mov     byte ptr ES:[DI], dl ; 24/07/2013
                ; movb -(r3),-(r5) / move argument char
                                ; / by char starting at "ecore"
        cmp     si, word ptr [BX]
                ; cmp r3,(r2) / moved all characters in
                          ; / this argument
                ; bhi 2b / branch 2b if not
        jna     short @f
        dec     si
        mov     dl, byte ptr [SI]
        jmp     short sysexec_5
@@:
        mov     word ptr ES:[BP], di ; 24/07/2013
        inc     bp
        inc     bp
                ; mov r5,(r4)+ / move r5 into top of users core;
                           ; / r5 has pointer to nth arg
        jmp     sysexec_3
                ; br 1b / string
sysexec_6: ; 1:
        dec     di
        dec     di ; 24/10/2013
        ;mov    byte ptr ES:[DI], 0 ; 24/07/2013
                ; clrb -(r5)
        shr     di, 1
        shl     di, 1
                ; bic $1,r5 / make r5 even, r5 points to
                        ; / last word of argument strings
        ;mov    si, core
        xor     si, si ; core = 0
                ; mov $core,r2
        mov     word ptr ES:[DI], si ; 24/07/2013
sysexec_7: ; 1: / move argument pointers into core following
             ; / argument strings
        cmp     si, bp
                ; cmp r2,r4
        jnb     short sysexec_8
                ; bhis 1f / branch to 1f when all pointers
                        ; / are moved
        mov     dx, word ptr ES:[SI] ; 25/07/2013
        inc     si
        dec     di
        inc     si
        dec     di
        mov     word ptr ES:[DI], dx ; 24/07/2013
                ; mov  (r2)+,-(r5)
        jmp     short sysexec_7
                ; br 1b
sysexec_8: ; 1:
        ;sub    bp, core ; core  = 0
                ; sub $core,r4 / gives number of arguments *2
        shr     bp, 1
                ; asr r4 / divide r4 by 2 to calculate
                       ; / the number of args stored
        dec     di
        dec     di
        mov     word ptr ES:[DI], bp ; 24/07/2013
                ; mov r4,-(r5) / save number of arguments ahead
                            ; / of the argument pointers
        xor     cx, cx
        pushf
        pop     dx
        dec     di
        dec     di
        ; 24/07/2013 (ES:[DI])
        mov     word ptr ES:[DI], dx ; FLAGS (for 'IRET')
                ; clr -(r5) / popped into ps when rti in
                         ; / sysrele is executed
        mov     bx, es  ; 24/07/2013
        dec     di
        dec     di
        mov     word ptr ES:[DI], bx ; CS (for 'IRET')
        ;mov    cx, core ; core = 0
        dec     di
        dec     di
        mov     word ptr ES:[DI], cx ; IP (for 'IRET')
                ; mov $core,-(r5) / popped into pc when rti
                                ; / in sysrele is executed
                ;mov r5,0f / load second copyz argument
                ;tst -(r5) / decrement r5
        mov     bx, cs
        mov     ds, bx
        mov     word ptr [u.r0], cx ; ax = 0
        mov     word ptr [u.usp], di
        push    di ; user's stack pointer
        push    cx ; dx = 0
        push    cx ; cx = 0
        push    cx ; bx = 0
        push    cx ; si = 0
        push    cx ; di = 0
        push    cx ; bp = 0
        mov     word ptr [u.sp_], sp
        mov     cx, di
        ; 24/07/2013
        xor     di, di ; 0
        push    ax ; i-number
        xor     ax, ax ; 0
        shr     cx, 1  ; cx/2 -> word count
        ; ES = word ptr [u.segmnt] or csgmnt
        rep     stosw  ; clear user's core/memory segment
        mov     ax, es ; 24/07/2013
        mov     word ptr [u.segmnt], ax ; 24/07/2013
        mov     es, bx ;  es = ds = cs
        pop     ax ; i-number
                ; 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
        mov     word ptr [u.break_], cx ; 0
                ; clr u.break
                ; mov r5,sp / point sp to user's stack
        mov     word ptr [u.count], 12
                ; mov $14,u.count
        mov     word ptr [u.fofp], offset u.off
                ; mov $u.off,u.fofp
        mov     word ptr [u.off], cx ; 0
                ; clr u.off / set offset in file to be read to zero
        ; 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     cx, word ptr [u.usp]
                ; mov sp,r5 / put users stack address in r5
        sub     cx, core+40 ; 40 bytes will be reserved
                        ;           for user stack
                ; sub $core+40.,r5 / subtract $core +40,
                               ; / from r5 (leaves number of words
                               ; / less 26 available for
                               ; / program in user core
        mov     word ptr [u.count], cx
                ; mov r5,u.count /
        mov     bx, word ptr [u.segmnt]
        mov     es, bx
        ;mov    bx, core ; 0
        xor     bx, bx ; 0
        cmp     word ptr ES:[BX], 0AEBh ; EBh, 0Ah -> jump to +12
                ; cmp core,$405 / br .+14 is first instruction
                              ; / if file is standard a.out format
        jne     short   sysexec_9
                ; bne 1f / branch, if not standard format
        add     bl, 2
        ;add    cx, word ptr ES:[BX]+2
        add     cx, word ptr ES:[BX]
                ; mov core+2,r5 / put 2nd word of users program in r5;
                              ; / number of bytes in program text
        mov     dx, ds
        mov     es, dx
        sub     cx, 12
                ; sub $14,r5 / subtract 12
        cmp     cx, word ptr [u.count]
                ; cmp r5,u.count /
        jg      short sysexec_9
                ; bgt 1f / branch if r5 greater than u.count
        mov     word ptr [u.count], cx
                ; mov r5,u.count
        push    bx
        call    readi
                ; jsr r0,readi / read in rest of user's program text
        mov     bx, word ptr [u.segmnt]
        mov     es, bx
        pop     bx
        ;mov    cx,  word ptr ES:[BX]+8
        add     bl, 6 ; 2+6 = 8
        mov     cx, word ptr ES:[BX]
        ;
        mov     bx, ds
        mov     es, bx
        ;
        mov     word ptr [u.nread], cx
                ; add core+10,u.nread / add size of user data area
                                   ; / to u.nread
        jmp     short sysexec_10
                ; br 2f
sysexec_9: ; 1:
        call    readi
                ; jsr r0,readi / read in rest of file
sysexec_10: ; 2:
        mov     cx, word ptr [u.nread]
        add     cx, core+12 ; 18/07/2013
        ;mov    word ptr [u.break_], cx
                ; mov u.nread,u.break / set users program break to end of
                                   ; / user code
        ;add    word ptr [u.break_], core+12 ; 12
                ; add $core+14,u.break / plus data area
        mov     word ptr [u.break_], cx ; 18/07/2013
        call    iclose
                ; jsr r0,iclose / does nothing
        ;; mov  sp , word ptr [u.sp_]
        ; 06/12/2013
        xor     ax, ax
        inc     al
        mov     word ptr [u.intr], ax ; 1 (interrupt/time-out is enabled)
        mov     word ptr [u.quit], ax ; 1 ('crtl+brk' signal is enabled)
        ;
        jmp     sysret
                ; br sysret3 / return to core image at $core
```

```
sysfstat:
        ; 19/06/2013
        ; '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,
        ;       Retro UNIX 8086 v1 argument transfer method 2 is used
        ;       to get sysfstat system call arguments from the user;
        ;       * 1st argument, file descriptor is in BX register
        ;       * 2nd argument, buf is pointed to by CX register

; / set status of open file
        ;call   arg2
                ; jsr r0,arg; u.off / put buffer address in u.off
        push    cx
                ; mov u.off,-(sp) / put buffer address on the stack
                ; mov ax, word ptr [u.r0]
                ; 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
        cmp     ah, 80h
        jb      short @f
                ; bgt 1f / if i-number is negative (open for writing)
        neg     ax
                ; neg r1 / make it positive, then branch
        jmp     short @f
                ; br 1f / to 1f
sysstat:
        ; 19/06/2013
        ; '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
        ;call   arg2
                ; jsr r0,arg2 / get the 2 arguments
        mov     word ptr [u.namep], bx
        push    cx
        call    namei
                ; jsr r0,namei / get the i-number for the file
        jc      error
                ; br error3 / no such file, error
@@: ; 1:
        call    iget
                ; jsr r0,iget / get the i-node into core
        mov     si, word ptr [u.segmnt]
        pop     di
                ; mov (sp)+,r3 / move u.off to r3 (points to buffer)
        mov     es, si
        stosw
                ; mov r1,(r3)+ / put i-number in 1st word of buffer
        ;mov    si, offset inode
        mov     si, offset i
                ; mov $inode,r2 / r2 points to i-node
@@: ; 1:
        movsw
                ; mov (r2)+,(r3)+ / move rest of i-node to buffer
        cmp     si, offset i + 32
                ; cmp r2,$inode+32 / done?
        jne     short @b
                ; bne 1b / no, go back
        mov     ax, ds
        mov     es, ax
        jmp     sysret
                ; br sysret3 / return through sysret

fclose:
        ; 12/01/2014
        ; 05/08/2013, 30/07/2013, 19/04/2013
        ; 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: DX, BX, CX, SI, DI, BP))
        ;
        ; Retro UNIX 8086 v1 modification : CF = 1
        ;              if i-number of the file is 0. (error)
        ;
        mov     dx, 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 inumber 0?
        jb      short fclose_2
                ; beq 1f / yes, i-node not active so return
                ; tst (r0)+ / no, jump over error return
        mov     bx, dx ; **
        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 ptr [BX]+u.fp, 0 ; 30/07/2013
                ; clrb u.fp(r1) / clear that entry in the u.fp list
        mov     bx, word ptr [u.fofp]
                ; mov u.fofp,r1 / r1 points to 3rd word in fsp entry
@@:
        dec     byte ptr [BX]+2
                ; 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     word ptr [BX]-4, ax ; 0
                ; clr -4(r1) / clear 1st word of fsp entry
        ; 12/1/2014 (removing Retro UNIX 8086 v1 modification, 30/7/2013)
        ;           (returning to original unix v1 code)
        mov     al, byte ptr [BX]+3
                ; 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:   ; 18/11/2013 (mov ax, bx)
        ; 19/04/2013
        ; / get the device number and the i-number of an open file
        mov     bx, ax
getf1: ;; Calling point from 'rw1' (23/05/2013)
        cmp     bx, 10
                ; cmp r1,$10. / user limited to 10 open files
         jnb     error
                ; bhis error3 / u.fp is table of users open files,
                           ; / index in fsp table
        mov     bl, byte ptr [BX]+u.fp
                ; movb u.fp(r1),r1 / r1 contains number of entry
                                  ; / in fsp table
        or      bl, bl
        jnz     short @f ; 18/11/2013
        ;jz     short @f
                ; beq 1f / if its zero return
        ; 18/11/2013
        mov     ax, bx ; 0
        retn
@@:
        shl     bx, 1
                ; asl r1
        shl     bx, 1
                ; asl r1 / multiply by 8 to get index into
                       ; / fsp table entry
        shl     bx, 1
                ; asl r1
        add     bx, offset fsp - 4
                ; add $fsp-4,r1 / r1 is pointing at the 3rd word
                             ; / in the fsp entry
        mov     word ptr [u.fofp], bx
                ; mov r1,u.fofp / save address of 3rd word
                             ; / in fsp entry in u.fofp
        dec     bx
        dec     bx
        mov     ax, word ptr [BX]
        ;mov    byte ptr [cdev], al ; ;;Retro UNIX 8086 v1 !
        mov     word ptr [cdev], ax ; ;;in fact (!)
                                   ; ;;dev number is in 1 byte
                ; mov -(r1),cdev / remove the device number  cdev
        dec     bx
        dec     bx
        mov     ax, word ptr [BX]
                ; mov -(r1),r1 / and the i-number  r1
;@@:    ; 1:
        retn
                ; rts r0

namei:
        ; 31/07/2013
        ; 28/07/2013
        ; 26/07/2013 (namei_r)
        ; 22/07/2013
        ; 18/07/2013
        ; 09/07/2013 mov ax, word ptr [rootdir]
        ; 27/05/2013 (cf=1 return for indicating 'file not found')
        ; 24/04/2013
        ; '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: DX, BX, CX, SI, DI, BP))
        ;

        ;;push  es ; Retro UNIX 8086 v1 Feature only !
        mov     ax, word ptr [u.segmnt] ; Retro UNIX 8086 v1 Feature only !
        mov     es, ax ; Retro UNIX 8086 v1 Feature only !

        mov     ax, word ptr [u.cdir]
                ; mov u.cdir,r1 / put the i-number of current directory
                              ; / in r1
        mov     dx, word ptr [u.cdrv]
        mov     word ptr [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
        xor     dx, dx ; 18/07/2013
        mov     si, word ptr [u.namep]
        cmp     byte ptr ES:[SI], '/'
                ; cmpb *u.namep,$'/ / is first char in file name a /
        jne     short namei_1
                ; bne 1f
        inc     si  ; go to next char
        mov     word ptr [u.namep], si
                ; inc u.namep / go to next char
        mov     ax, word ptr [rootdir] ; 09/07/2013 (mov ax, rootdir)
                ; mov rootdir,r1 / put i-number of rootdirectory in r1
        ;xor    dx, dx
        mov     word ptr [cdev], dx
                ; clr cdev / clear device number
namei_1: ; 1:
        ;; 18/07/2013
        mov     dl, byte ptr ES:[SI]
        mov     cx, cs
         mov    es, cx
        and     dl, dl
         jz       short nig
        ;;
        ;cmp    byte ptr ES:[SI], dl ; 0
                ; tstb *u.namep / is the character in file name a nul
        ;;jna   nig
                ; beq nig / yes, end of file name reached;
                        ; / branch to "nig"
```

```
namei_2: ; 1:
        ;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 ptr [i.flgs], 4000h
                ; bit $40000,i.flgs / directory i-node?
        jz      error
                ; beq error3 / no, got an error
        mov     ax, word ptr [i.size_]
        mov     word ptr [u.dirp], ax
                ; mov i.size,u.dirp / put size of directory in u.dirp
        xor     ax, ax
        mov     word ptr [u.off], ax ; 0
                ; clr u.off / u.off is file offset used by user
        mov     word ptr [u.fofp], offset u.off
                ; mov $u.off,u.fofp / u.fofp is a pointer to
                                 ; / the offset portion of fsp entry
namei_3: ; 2:
        mov     word ptr [u.base], offset u.dirbuf
                ; mov $u.dirbuf,u.base / u.dirbuf holds a file name
                                   ; / copied from a directory
        mov     word ptr [u.count], 10
                ; mov $10.,u.count / u.count is byte count
                                ; / for reads and writes
        mov     ax, word ptr [ii]
        ; 31/07/2013
        inc      byte ptr [namei_r] ; the caller is 'namei' sign
         ; 28/07/2013 nameir -> u.nameir
         ; 26/07/2013
         ;;inc      byte ptr [u.namei_r] ; the caller is 'namei' sign
        call    readi
        ; ES = DS after 'readi' !
                ; jsr r0,readi / read 10. bytes of file
                      ; with i-number (r1); i.e. read a directory entry
        mov     cx, word ptr [u.nread]
        or      cx, cx
                ; tst u.nread
        jz      short nib
                ; ble nib / gives error return
        ;
        mov     bx, word ptr [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     ax, word ptr [u.off]
        sub     ax, 10
        mov     word ptr [u.dirp], ax
                ; mov u.off,u.dirp
                ; sub $10.,u.dirp
        jmp     short namei_3
                ; br 2b
        ; 18/07/2013
nib:
        xor     ax, ax
        stc
nig:
        retn

namei_4: ; 3:
        mov     ax, word ptr [u.segmnt] ; Retro UNIX 8086 v1 Feature only !
        ;
        mov     si, word ptr [u.namep]
                ; mov u.namep,r2 / u.namep points into a file name string
        mov     di, offset u.dirbuf + 2
                ; mov $u.dirbuf+2,r3 / points to file name of directory entry
        mov     dx, offset u.dirbuf + 10
        ; AX = user segment
        mov     ds, ax ; Retro UNIX 8086 v1 Feature only !
namei_5: ; 3:
        lodsb    ; mov al, byte ptr [SI] ; inc si    (al = r4)
                ; movb (r2)+,r4 / move a character from u.namep string into r4
        or      al, al
        jz      short namei_6
                ; beq 3f / if char is nul, then the last char in string
                       ; / has been moved
        cmp     al, '/'
                ; cmp r4,$'/ / is char a </>
        je      short namei_6
                ; beq 3f
        cmp     di, dx ; offset u_dirbuf + 10
                ; cmp r3,$u.dirbuf+10. / have I checked
                                    ; / all 8 bytes of file name
        je      short namei_5
                ; beq 3b
        scasb
                ; cmpb (r3)+,r4 / compare char in u.namep string to file name
                              ; / char read from directory
        je      short namei_5
                ; beq 3b / branch if chars match
        mov     ax, cs ; Retro UNIX 8086 v1 Feature only !
        mov     ds, ax ; Retro UNIX 8086 v1 Feature only !
         jmp     short namei_3 ; 2b
                ; br 2b / file names do not match go to next directory entry
namei_6: ; 3:
        ; 22/07/2013
        mov     cx, cs ; Retro UNIX 8086 v1 Feature only !
        mov     ds, cx ; Retro UNIX 8086 v1 Feature only !
        ;
        cmp     di, dx
                ; cmp r3,$u.dirbuf+10. / if equal all 8 bytes were matched
        je      short namei_7
                ; beq 3f
        mov     ah, byte ptr [DI]
        ;inc    di
        and     ah, ah
                ; tstb (r3)+ /
         jnz     short namei_3
                ; bne 2b
namei_7: ; 3
        mov     word ptr [u.namep], si
                ; mov r2,u.namep / u.namep points to char
                               ; / following a / or nul
        ;mov    bx, word ptr [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
         jnz    namei_2
                ; bne 1b
        ; AX = i-number of the file
;;nig:
        ;;pop   es ; Retro UNIX 8086 v1 Feature only !
        retn
                ; tst (r0)+ / gives non-error return
;;nib:
;;      xor     ax, ax ; Retro UNIX 8086 v1 modification !
                       ; ax = 0 -> file not found
        ;;pop   es ; Retro UNIX 8086 v1 Feature only !
;;      stc     ; 27/05/2013
;;      retn
                ; rts r0

syschdir:
        ; 19/06/2013
        ; '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.
        ;        (argument transfer method 1)
; / makes the directory specified in the argument
; / the current directory
        ;;mov   ax, 1 ; one/single argument, put argument in BX
        ;;call  arg
        ;mov    bp, word ptr [u.sp_] ; points to user's BP register
        ;add    bp, 6 ; bx now points to BX on stack
        ;mov    bx, word ptr [BP]
        mov     word ptr [u.namep], bx
                ;jsr r0,arg; u.namep / u.namep points to path name
        call    namei
                ; jsr r0,namei / find its i-number
        jc      error
                ; br error3
        call    access
                ; jsr r0,access; 2 / get i-node into core
        test    word ptr [i.flgs], 4000h
                ; bit $40000,i.flgs / is it a directory?
        jz      error
                ; beq error3 / no error
        mov     word ptr [u.cdir], ax
                ; mov r1,u.cdir / move i-number to users
                              ; / current directory
        mov     ax, word ptr [cdev]
        mov     word ptr [u.cdrv], ax
                ; mov cdev,u.cdev / move its device to users
                                ; / current device
        jmp     sysret
                ; br sysret3

syschmod: ; < change mode of file >
        ; 07/07/2013
        ; 20/06/2013
        ; '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,
        ;       Retro UNIX 8086 v1 argument transfer method 2 is used
        ;       to get syschmod system call arguments from the user;
        ;       * 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 ptr [i.flgs], 4000h
                ; bit   $40000,i.flgs / directory?
        jz      short @f
                ; 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
```

```
@@: ; 2:
        mov     byte ptr [i.flgs], al
                ; movb r2,i.flgs / move remaining mode to i.flgs
        jmp     short @f
                ; br 1f

isown:
        ; 07/07/2013
        ; 27/05/2013, 04/05/2013
        ; '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: AX, DX, BX, CX, SI, DI, BP))
        ;
        ;;call  arg2
        ;;      ; jsr r0,arg2 / u.namep points to file name
        ;; ! 2nd argument on top of stack !
        ;; 07/07/2013
        mov     word ptr [u.namep], bx ;; 1st argument
        push    cx ;; 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
        call    iget
                ; jsr r0,iget / get i-node into core
        mov     al, byte ptr [u.uid_] ; 02/08/2013
        or      al, al
                ; tstb u.uid / super user?
        jz      short @f
                ; beq 1f / yes, branch
        cmp     al, byte ptr [i.uid]
                ; cmpb i.uid,u.uid / no, is this the owner of
                                ; / the file
        jne     error
                ; beq 1f / yes
                ; jmp error3 / no, error
@@: ; 1:
        call    setimod
                ; jsr r0,setimod / indicates
                ;               ; / i-node has been modified
        pop     ax ; 2nd argument
                ; mov (sp)+,r2 / mode is put in r2
                        ; / (u.off put on stack with 2nd arg)
        retn
                ; rts r0
```

```
syschown: ; < change owner of file >
        ; 02/08/2013
        ; 07/07/2013, 20/06/2013
        ; '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,
        ;       Retro UNIX 8086 v1 argument transfer method 2 is used
        ;       to get syschown system call arguments from the user;
        ;       * 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 ptr [u.uid_], 0 ; 02/08/2013
                ; tstb u.uid / super user
        jz      short @f
                ; beq 2f / yes, 2f
        test    byte ptr [i.flgs], 20h ; 32
                ; bit $40,i.flgs / no, set userid on execution?
        jnz     error
                ; bne 3f / yes error, could create Trojan Horses
@@: ; 2:
        ; AL = owner (number/ID)
        mov     byte ptr [u.uid_], al ; 02/08/2013
                ;  movb r2,i.uid / no, put the new owners id
                              ; / in the i-node
        jmp     sysret
        ; 1:
                ; jmp sysret4
        ; 3:
                ; jmp   error

;;arg:    ; < get system call arguments >
        ; 22/05/2013 'method 4' has been modified (corrected)
        ; 04/05/2013
        ; '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)
        ;
        ; RETRO UNIX 8086 v1 Modification !
        ;       Retro Unix 8086 v1 system call argument
        ;       transfer methods:
        ;       1) Single argument in BX register
        ;          ('arg' routine is called with AX=1)
        ;       2) Two arguments,
        ;               1st argument in BX register
        ;               2nd argument in CX register
        ;          ('arg' routine is called with AX=2)
        ;       3) Three arguments
        ;               3rd argument in DX register
        ;          ('arg' routine is called with AX=3)
        ;       4) Argument list address in BP register
        ;          ('arg' routine is called with AX=0)
        ; 'arg' routine will return arguments in same registers
        ;  except method 4 will return current argument
        ;  which is pointed by BP register and 'arg' will
        ;  increase value of (user's) BP register (on stack)
        ;  in order to point next argument. AX register will
        ;  return address of current argument.
        ; 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.
        ;
        ;    ((Modified registers: AX, DX, CX, BX))

; Retro UNIX 8086 v1 modification !
; [ sysunlink, sysfstat, syschdir, sysbreak, sysseek (seektell),
;  sysintr, sysquit, rw1 (sysread, syswrite), sysemt, sysilgins
; sysmdate, gtty (sysgtty) etc. call arg.]
;
; Note: If all of system calls which call 'arg' routine will have
; only 1 argument, this 'arg' routine may be simplified
; and system calls with 2 arguments may be changed to use 'arg1'
; instead of 'arg' (04/05/2013).

;;      mov     bx, word ptr [u.sp_] ; points to user's BP register
;;      mov     cx, ax
;;      or      cx, cx
;;      jnz     short @f
;arg_bp: ; method 4
;;      mov     ax, word ptr [BX] ; value of BP register on stack
        ; (sAX = uBP)
;;      mov     dx, ax
        ; AX = 1st argument or current argument (method 4)
;;      inc     dx
;;      inc     dx
;;      mov     word ptr [BX], dx ; BP will point to next argument
        ; (uBP = uBP+2)
;;      retn
; method 1, 2, 3
;;@@:
;;      add     bx, 6 ; bx now points to BX on stack
;;@@:
;;      mov     dx, word ptr [BX]
;;      push    dx ; 1st or 2nd or 3rd argument (depends on CX)
;;      dec     cx
;;      jz      short @f
;;      inc     bx
;;      inc     bx
;;      jmp     short @b
;;@@:
;;      dec     ax
;;      jz      short @f
;;      pop     cx ; 2nd or 3rd argument (depends on value in AX)
;;      dec     ax
;;      jz      short @f
;;      mov     dx, cx ; 3rd argument
;;      pop     cx ; 2nd argument
;;@@:
;;      pop     bx ; 1st argument
;;      retn

; UNIX v1 original 'arg' routine here:
               ; 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>
        ; 22/05/2013 arg1 modified (corrected)
        ; 04/05/2013
        ; '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
        ;
        ; RETRO UNIX 8086 v1 Modification !
        ;       Retro Unix 8086 v1 system call argument
        ;       transfer methods:
        ;       1) Single argument in BX register
        ;           ('arg' routine is called with AX=1)
        ;       2) Two arguments,
        ;               1st argument in BX register
        ;               2nd argument in CX register
        ;           ('arg' routine is called with AX=2)
        ;       3) Three arguments
        ;               3rd argument in DX register
        ;           ('arg' routine is called with AX=3)
        ;       4) Argument list address in BP register
        ;           ('arg' routine is called with AX=0)
        ; 'arg2' routine uses method 2 when calling 'arg' routine
        ;  then puts 1st argument (BX) in u.namep and pushes
        ;  2nd argument (CX) on stack.
        ;  (Retro UNIX 8086 v1 does not put 2nd argument in u.off)
        ;
        ; INPUTS ->
        ;    u.sp, r0
        ;
        ; OUTPUTS ->
        ;    u.namep
        ;    u.off
        ;    u.off pushed on stack
        ;    r1
        ;
        ;    ((Modified registers: AX, DX, CX, BX))
        ;
; arg2 (1) -- 04/05/2013 (1)
;       mov     ax, 2 ; two arguments, method 2
;       call    arg
;       ; BX = 1st argument
;       ; CX = 2nd argument

; arg2 (modified for arg1 call) -- 04/05/2013 (2)

; Retro UNIX 8086 v1 modification !
; Direct argument handling instead of using 'arg' call.
; [ sysexec, sysmount, sysopen, syslink, sysstat,
;  isown (syschmod, syschown),sysopen, syscreat, sysmkdir, sysmount
; call arg2 ]

;;      call    arg1 ; 04/05/2013
;;      mov     word ptr [u.namep], ax ; 1st argument
;;      pop     dx ; return address
;;      push    cx ; 2nd argument
;;      push    dx
        ; warning !
        ; ! Caller must pop 2nd argument on stack !
;;      retn

;;arg1: ; Retro UNIX 8086 v1 feature only !
        ; 22/05/2013 modified (corrected)
;;      mov     bx, word ptr [u.sp_] ; points to user's BP register
;;      add     bx, 6
;,      mov     ax, [BX] ; points to user's BX register
        ;(sAX = uBX)
;;      inc     bx
;;      inc     bx
;,      mov     cx, [BX] ; points to user's CX register
        ;(sCX = uCX)
;       retn

;; arg2 (2) -- 04/05/2013 (1)
;       mov     word ptr [u.namep], bx ; file name pointer
;       ;mov    word ptr [u.off], cx ; 2nd argument
;       pop     dx ; return address
;       push    cx
;       push    dx
;       ; warning !
;       ; ! Caller must pop 2nd argument on stack !
;       retn
; UNIX v1 original 'arg2' routine here:
               ; 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

systime:
        ; 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 !!
; / get time of year
        call    epoch
        mov     word ptr [u.r0], ax
        mov     bp, word ptr [u.sp_]
        add     bp, 10 ; points to the user's DX register
        mov     word ptr [BP], dx
                ; mov s.time,4(sp)
                ; mov s.time+2,2(sp) / put the present time
                                ; / on the stack
                ; br sysret4
        jmp     sysret

sysstime:
        ; 02/08/2013
        ; 20/06/2013
        ; '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 !!
; / set time
        cmp     byte ptr [u.uid_], 0 ; 02/08/2013
                ; tstb u.uid / is user the super user
        ja      error
                ; bne error4 / no, error
        ; CX:BX = unix (epoch) time (from user)
        mov     dx, cx
        mov     ax, bx
        ; DX:AX = unix (epoch) time (to subroutine)
        ;call convert_from_epoch
        call    set_date_time
                ; mov 4(sp),s.time
                ; mov 2(sp),s.time+2 / set the system time
        jmp     sysret
                ; br sysret4
```

```
sysbreak:
        ; 24/03/2014
        ; 19/11/2013
        ; 20/06/2013
        ; '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)

        ;cmp    word ptr [u.break], core
                ; mov u.break,r1 / move users break point to r1
                ; cmp r1,$core / is it the same or lower than core?
        ;ja     short sysbreak_3
                ; blos 1f / yes, 1f
        mov     di, word ptr [u.break]
        cmp     di, word ptr [u.usp]
                ; cmp r1,sp / is it the same or higher
                          ; / than the stack?
         jnb     short sysbreak_3
                ; bhis 1f / yes, 1f
        mov     ax, word ptr [u.segmnt]
        mov     es, ax
        xor     ax, ax
        test    di, 1
                ; bit $1,r1 / is it an odd address
        jz      short sysbreak_1
                ; beq 2f / no, its even
        stosb
                ; clrb (r1)+ / yes, make it even
sysbreak_0:  ; 2: / clear area between the break point and the stack
        cmp     di, word ptr [u.usp] ; 24/03/2014
                ; cmp r1,sp / is it higher or same than the stack
        jnb     short sysbreak_2
                ; bhis 1f / yes, quit
sysbreak_1:
        stosw
                ; clr (r1)+ / clear word
        jmp     short sysbreak_0
                ; br 2b / go back
sysbreak_2: ; 1:
        mov     ax, ds
        mov     es, ax
sysbreak_3:
        mov     word ptr [u.break], bx
                ; jsr r0,arg; u.break / put the "address"
                        ; / in u.break (set new break point)
        jmp     sysret
                ; br sysret4 / br sysret
```

```
maknod:
        ; 02/08/2013
        ; 31/07/2013
        ; 17/07/2013
        ; 02/05/2013
        ; '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: AX, DX, BX, CX, SI, DI, BP))

        ; / 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, word ptr [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
@@: ; 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
           ; BX (R2) has byte address of the byte with allocation bit
        test    byte ptr [BX], dl
                ; bitb mq,(r2) / is the i-node active
        jnz     short @b
                ; bne 1b / yes, try the next one
        or      byte ptr [BX], 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 ptr [i.flgs], 8000h
                ; tst i.flgs / is i-node already allocated
        jnz     short @b
                ; blt 1b / yes, look for another one
        mov     word ptr [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, word ptr [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     cx, 16
        xor     ax, ax ; 0
        ;mov    di, offset inode
        mov     di, offset i ; 17/07/2013
        rep     stosw
        ;
        pop     word ptr [i.flgs]
                ; mov (sp)+,i.flgs / fill flags
        mov     cl, byte ptr [u.uid_] ; 02/08/2013
        mov     byte ptr [i.uid], cl
                ; movb u.uid,i.uid / user id
        mov      byte ptr [i.nlks], 1
                ; movb $1,i.nlks / 1 link
        ;call   epoch   ; Retro UNIX 8086 v1 modification !
         ;mov   ax, word ptr [s.time]
         ;mov   dx, word ptr [s.time]+2
         ;mov   word ptr [i.ctim], ax
         ;mov   word ptr [i.ctim]+2, dx
                ; 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
        ; 05/08/2013
        ; 07/07/2013
        ; '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,
        ;       Retro UNIX 8086 v1 argument transfer method 3 is used
        ;       to get sysseek system call arguments from the user;
        ;       * 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
                ; jsr r0,seektell / get proper value in u.count
        ; AX = u.count
        ; BX = *u.fofp
                ; add u.base,u.count / add u.base to it
        add     ax, word ptr [u.base] ; add offset (u.base) to base
        mov     word ptr [BX], ax
                ; mov u.count,*u.fofp / put result into r/w pointer
        jmp     sysret
                ; br sysret4

systell: ; / get the r/w pointer
        ; 05/08/2013
        ; 07/07/2013
        ; 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    cx, cx ; 0
        mov     dx, 1 ; 05/08/2013
        ;call   seektell
        call    seektell0 ; 05/08/2013
        ;mov    bx, word ptr [u.fofp]
        mov     ax, word ptr [BX]
        mov     word ptr [u.r0], ax
        jmp     sysret

; Original unix v1 'systell' system call:
                ; jsr r0,seektell
                ; br error4

seektell:
        ; 05/08/2013 (return AX as base for offset)
        ; 07/07/2013
        ; '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     word ptr [u.base], cx ; offset
                ; jsr r0,arg; u.base / puts offset in u.base
seektell0:
        mov     word ptr [u.count], dx
                ; 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
        ;push   ax
        cmp     ah, 80h
        jb      short @f
                ; bgt .+4 / if its positive jump
        neg     ax
                ; neg r1 / if not make it positive
@@:
        call    iget
                ; jsr r0,iget / get its i-node into core
        mov     bx, word ptr [u.fofp] ; 05/08/2013
        cmp     byte ptr [u.count], 1
                ; cmp u.count,$1 / is ptr name =1
        ja      short @f
                ; blt 2f / no its zero
        je      short seektell_1
                ; beq 1f / yes its 1
        xor     ax, ax
        ;jmp    short seektell_2
        retn
@@:
         mov     ax, word ptr [i.size_]
                 ; mov i.size,u.count /  put number of bytes
                                    ; / in file in u.count
        ;jmp    short seektell_2
                ; br 2f
        retn
```

```
seektell_1: ; 1: / ptrname =1
        ;mov    bx, word ptr [u.fofp]
        mov     ax, word ptr [BX]
                ; mov *u.fofp,u.count / put offset in u.count
;seektell_2: ; 2: / ptrname =0
        ;mov    word ptr [u.count], ax
        ;pop    ax
                ; mov (sp)+,r1 / i-number on stack  r1
        retn
                ; rts r0

sysintr: ; / set interrupt handling
        ; 07/07/2013
        ; '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     word ptr [u.intr], bx
        ;jmp    short @f
                ;jsr r0,arg; u.intr / put the argument in u.intr
                ; br 1f / go into quit routine
        jmp     sysret

sysquit:
        ; 07/07/2013
        ; '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     word ptr [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
        ; 02/08/2013
        ; 07/07/2013
        ; '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, byte ptr [u.ruid]
                ; cmpb r1,u.ruid / is it equal to the real user
                               ; / id number
        je      short @f
                ; beq 1f / yes
        cmp     byte ptr [u.uid_], 0 ; 02/08/2013
                ; tstb u.uid / no, is current user the super user?
        ja      error
                ; bne error4 / no, error
        mov     byte ptr [u.ruid], bl
@@: ; 1:
        mov     byte ptr [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 >
        ; 07/07/2013
        ; '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.
        ;
        ;xor    ah, ah
        mov     al, byte ptr [u.ruid]
        mov     word ptr [u.r0], ax
                ; movb  u.ruid,*u.r0 / move the real user id to (u.r0)
        jmp     sysret
                ; br sysret4 / systerm return, sysret
```

```
anyi:
        ; 25/04/2013
        ; '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: DX, CX, BX, SI, DI, BP))
        ;
                ; / r1 contains an i-number
        mov     bx, offset fsp
                ; mov $fsp,r2 / move start of fsp table to r2
anyi_1: ; 1:
        cmp     ax, word ptr [BX]
                ; cmp r1,(r2) / do i-numbers match?
        je      short anyi_2
                ; beq 1f / yes, 1f
        neg     ax
                ; neg r1 / no complement r1
        cmp     ax, word ptr [BX]
                ; cmp r1,(r2) / do they match now?
        je      short anyi_2
                ; beq 1f / yes, transfer
                ; / i-numbers do not match
        add     bx, 8
                ; add $8,r2 / no, bump to next entry in fsp table
        cmp     bx, offset fsp + (nfiles*8)
                ; cmp r2,$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 @f
        neg     ax
                ; neg r1 / make i-number positive
@@:
        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
         ;; BX (R2) has address of the byte with allocation bit
        ; not   dx
        not     dl ;; 0 at calculated bit position, other bits are 1
         ;and   word ptr [BX], dx
        and     byte ptr [BX], 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 ptr [i.flgs], 0
                ; clr i.flgs / clear all flags in the i-node
        retn
                ;rts    r0 / return
anyi_2: ; 1: / i-numbers match
        inc     byte ptr [BX]+7
                ;incb 7(r2) / increment upper byte of the 4th word
                   ; / in that fsp entry (deleted flag of fsp entry)
        retn
                ; rts r0
; ****************************************************************************
;
; UNIX.ASM (RETRO UNIX 8086 Kernel - Only for 1.44 MB floppy disks)
; ----------------------------------------------------------------------------
; U3.ASM (include u0.asm) //// UNIX v1 -> u3.s

; RETRO UNIX 8086 (Retro Unix == Turkish Rational Unix)
; Operating System Project (v0.1) by ERDOGAN TAN (Beginning: 11/07/2012)
; 1.44 MB Floppy Disk
; (11/03/2013)
;
; [ Last Modification: 08/03/2014 ]  ;;; completed ;;;
;
; Derivation from UNIX Operating System (v1.0 for PDP-11)
; (Original) Source Code by Ken Thompson (1971-1972)
; <Bell Laboratories (17/3/1972)>
; <Preliminary Release of UNIX Implementation Document>
;
; ****************************************************************************

; 08/03/2014 wswap, rswap, swap
; 25/02/2014 swap
; 23/02/2014 putlu, swap
; 14/02/2014 swap ('SRUN' check), putlu (single level runq)
; 05/02/2014 swap (SSLEEP/SWAIT/SRUN, p.waitc)
; 23/10/2013 swap (consistency check), idle
; 10/10/2013 idle
; 24/09/2013 swap, wswap, rswap, tswap (consistency check)
; 20/09/2013 swap
; 30/08/2013 swap
; 09/08/2013 swap
; 08/08/2013 putlu, wswap, rswap
; 03/08/2013
; 01/08/2013
; 29/07/2013
; 24/07/2013
; 23/07/2013
; 09/07/2013
; 26/05/2013
; 24/05/2013
; 21/05/2013
; 17/05/2013
; 16/05/2013 swap
; 19/04/2013 swap, wrswap
; 14/04/2013 tswap, swap
; 10/04/2013
; 11/03/2013

tswap:
        ; 14/02/2014 single level runq
        ; 24/09/2013 consistency check -> ok
        ; 26/05/2013 (swap, putlu modifications)
        ; 14/04/2013
        ; 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 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: DX, BX, CX, SI, DI))
        ;
        mov     al, byte ptr [u.uno]
                ; movb u.uno,r1 / move users process number to r1
        ;mov    bx, offset runq + 4
```

```
                ; 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
swap:
        ; 08/03/2014
        ; 25/02/2014
        ; 23/02/2014
        ; 14/02/2014 single level runq
        ; 05/02/2014 SSLEEP/SWAIT/SRUN, p.waitc
        ; 23/10/2013 consistency check -> ok
        ; 24/09/2013 consistency check -> ok
        ; 20/09/2013 ('call idle' enabled again)
        ; 30/08/2013
        ; 09/08/2013
        ; 29/07/2013
        ; 24/07/2013 sstack (= file size + 256)
        ; 26/05/2013 wswap and rswap (are come back!)
        ; 24/05/2013 (u.usp -> sp modification)
        ; 21/05/2013
        ; 16/05/2013
        ; 19/04/2013 wrswap (instead of wswap and rswap)
        ; 14/04/2013
        ; 'swap' is routine that controls the swapping of processes
        ; in and out of core.
        ;
        ; 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: AX, DX, BX, CX, SI, DI))
        ;
swap_0:
                ;mov $300,*$ps / processor priority = 6
        ; 14/02/2014
        mov     si, offset runq  ; 23/02/2014 BX -> DI -> SI
                ; mov $runq,r2 / r2 points to runq table
swap_1: ; 1: / search runq table for highest priority process
        mov     ax, word ptr [SI]
        and     ax, ax
                ; tst (r2)+ / are there any processes to run
                         ; / in this Q entry
        jnz     short swap_2
                ; bne 1f / yes, process 1f
                ; cmp r2,$runq+6 / if zero compare address
                              ; / to end of table
                ; bne 1b / if not at end, go back
;mov cl, byte ptr [u.uno]
;mov al, 'X'
;mov ah, 04Fh
```

```
;add cl, '0'
;mov ch, ah
;call write_sign
        ;; 25/02/2014
        ;;mov   al, byte ptr [ptty]
        ;;call  wakeup
        ;;or    al, al
        ;;jnz   short swap_1
        ;
        ;;mov   cx, word ptr [s.idlet]+2 ;; 29/07/2013
        ;;; 30/08/2013
        ; 20/09/2013
        call    idle ; 23/10/2013 (consistency check !)
                ; jsr r0,idle; s.idlet+2 / wait for interrupt;
                                      ; / all queues are empty
        ; 14/02/2014
        jmp     short swap_1
                ; br swap
swap_2: ; 1:
                ; tst -(r2) / restore pointer to right Q entry
                ; mov r2,u.pri / set present user to this run queue
        ;mov    ax, word ptr [SI]
                 ; movb (r2)+,r1 / move 1st process in queue to r1
        ;
        cmp     al, ah ; 16/05/2013
                ; 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
        ;
        mov     bl, al
        xor     bh, bh
        mov     ah, byte ptr [BX]+p.link-1
        mov     byte ptr [SI], 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
        ; 23/02/2014 BX -> SI
        mov     word ptr [SI], dx  ;16/05/2013
                ; clr -(r2) / zero the entry; no processes on the Q
        ;
        ; 26/05/2013 (swap_4 and swap_5)
swap_4: ; / write out core to appropriate disk area and read
       ; / in new process if required
                ; clr *$ps / clear processor status
        ; 09/08/2013
        mov     ah, byte ptr [u.uno]
        cmp     ah, al
        ;cmp    byte ptr [u.uno], al
                ; cmpb r1,u.uno / is this process the same as
                              ; / the process in core?
        je      short swap_6
                ; beq 2f / yes, don't have to swap
                ; mov r0,-(sp) / no, write out core; save r0
                           ; / (address in routine that called swap)
        mov     word ptr [u.usp], sp
                ; mov sp,u.usp / save stack pointer
        ; 09/08/2013
        ; 24/07/2013
        ;mov    sp, sstack ; offset sstack
                ; mov $sstack,sp / move swap stack pointer
                              ; / to the stack pointer
        ;push   ax
                ; mov r1,-(sp) / put r1 (new process #) on the stack
        ; 09/08/2013
        or      ah, ah
        ;cmp    byte ptr [u.uno], dl ; 0
                ; tstb u.uno / is the process # = 0
        jz      short swap_5
        ;jna    short swap_5
                ; beq  1f / yes, kill process by overwriting
        call    wswap
                ;jsr r0,wswap / write out core to disk
```

```
swap_5: ;1:
        ; pop   ax
                ; mov (sp)+,r1 / restore r1 to new process number
        ; 08/03/2014
        ; (protect 'rswap' return address from stack overwriting)
        cli
        mov     sp, sstack - 190 ; (SizeOfFile + 2)
        ;
        call    rswap
                ; jsr r0,rswap / read new process into core
                ; jsr r0,unpack / unpack the users stack from next
                              ; / to his program to its normal
        mov     sp, word ptr [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_6: ;2:
        ; 14/02/2014 uquant -> u.quant
        ; 30/08/2013
        ; RETRO UNIX 8086 v1 modification !
        mov     byte ptr [u.quant], time_count
        ;mov    byte ptr [uquant], 3
                ; movb    $30.,uquant / initialize process time quantum
        retn
                ; rts r0 / return

wswap:  ; < swap out, swap to disk >
        ; 08/03/2014 major modification
        ; 24/09/2013 consistency check -> ok
        ; 08/08/2013
        ; 24/07/2013
        ; 26/05/2013
        ; 'wswap' writes out the process that is in core onto its
        ; appropriate disk area.
        ;
        ; 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: CX, SI, DI))

        mov     di, sdsegmnt
        mov     es, di
        xor     cl, cl
        mov     ch, byte ptr [u.uno]
        dec     ch ; 0 based process number
        ;; 08/03/2014 (swap data space is 256 bytes for every process)
        ;;shr   cx, 1 ; swap data space is 128 bytes for every process
        mov     di, cx
        mov     cx, 32
        mov     si, offset u ; user structure
        rep     movsw
        ;
        mov     si, word ptr [u.usp]   ; sp  (system stack pointer)
        mov     cx, sstack
        sub     cx, si  ; NOTE: system stack size = 256-64 = 192 bytes
        rep     movsb
        ;
        mov     cx, ds
        mov     es, cx
        retn
        ;
        ; 08/08/2013, 14 -> 16, 7 -> 8
        ;mov    si, sstack - 16 ; 24/07/2013
                       ; offset sstack - 16 ;; = word ptr [u.sp_] - 2
        ;mov    cx, 8
        ;rep    movsw
        ;mov    cl, 32
        ;mov    si, offset u ; user structure
        ;rep    movsw
        ;mov    cx, ds
        ;mov    es, cx
        ;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 >
        ; 08/03/2014 major modification
        ; 24/09/2013 consistency check -> ok
        ; 08/08/2013
        ; 24/07/2013
        ; 26/05/2013
        ; 'rswap' reads a process whose number is in r1,
        ; from disk into core.
        ;
        ; 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: AX, CX, SI, DI))

        mov     ah, al
        dec     ah
        xor     al, al
        ;;shr   ax, 1 ; 08/03/2014 (256 bytes per process)
        mov     si, ax ; SI points copy of sstack in sdsegment
                       ; u.sp_ points sstack-12 (for 6 registers)
        mov     ax, sdsegmnt ; 17/05/2013
        mov     ds, ax ; sdsegment
        ; 08/03/2014
        mov     di, offset u
        mov     cx, 32
        rep     movsw
        mov     di, word ptr ES:[u.usp] ; system stack pointer location
        mov     cx, sstack
        sub     cx, di          ; Max. 256-64 bytes stack space
        rep     movsb
        mov     ax, cs
        mov     ds, ax
        retn
        ;
        ; 08/08/2013 14 -> 16, 7 ->8
        ; 24/07/2013
        ;mov    di, sstack - 16 ; offset sstack-14
        ;mov    cx, 8
        ;rep    movsw
        ;mov    di, offset u
        ;mov    cl, 32
        ;rep    movsw
        ;mov    ax, cs
        ;mov    ds, ax
        ;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:
        ; 23/02/2014
        ; 14/02/2014 single level run queue
        ; 08/08/2013
        ; 26/05/2013 (si -> di)
        ; 15/04/2013
        ;
        ; '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: DX, BX, DI))
        ;

; / r1 = user process no.; r2 points to lowest priority queue

        ; BX = r2
        ; AX = r1 (AL=r1b)

        ; 14/02/2014
        mov     bx, offset runq
        ; 23/02/2014
         mov     dx, word ptr [BX]
        inc     bx
        and     dx, dx
                ; tstb (r2)+ / is queue empty?
        jz      short putlu_1
                ; beq 1f / yes, branch
        mov     dl, dh
        xor     dh, dh
        mov     di, dx
                ; movb (r2),r3 / no, save the "last user" process number
                             ; / in r3
        mov     byte ptr [DI]+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     byte ptr [BX]-1, al ; 08/08/2013
                ; movb r1,-1(r2) / user is only user;
                            ; / put process no. at beginning and at end
```

```
putlu_2: ; 2:
        mov     byte ptr [BX], al
                ; movb r1,(r2) / user process in r1 is now the last entry
                             ; / on the queue
        ; 23/02/2014
        mov     dl, al
         mov     di, dx
         mov     byte ptr [DI]+p.link-1, dh ; 0
        ;
        ;14/02/2014
        ;dec    bx
                ; 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:
      ; 23/10/2013
      ; 10/10/2013
      ; 29/07/2013
      ; 09/07/2013
      ; 10/04/2013
      ; (idle & wait loop)
      ; Retro Unix 8086 v1 modification on original Unixv1 idle procedure!
      ; input -> CX = wait count

      ;sti
      ; 29/07/2013
      hlt
      nop ; 10/10/2013
      nop
      nop
      ; 23/10/2013
      nop
      nop
      nop
      nop
      retn

      ;sti
      ;;;push word ptr [clockp]
      ;or cx, cx
      ;jnz short @f
      ;inc cx
;@@:
      ;;;mov word ptr [clockp], cx
@@:
      ;hlt ; wait for interrupt (timer interrupt or keyboard interrupt etc.)
      ;;;dec word ptr [clockp]
      ;dec cx ; 09/07/2013 ;;;
      ;jnz short @b
      ;;; pop word ptr [clockp]
      ;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:
        ; 03/08/2013
        ; 01/08/2013
        ; 23/07/2013
        ; 09/04/2013
        ;
        ; '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: DX, CX, BX, SI, DI, BP))

        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     di, bx ; r5
        mov     dx, ax ; 01/08/2013
        mov     cx, 256
                ; mov $256.,r3
        xor     ax, ax
        rep     stosw ; 03/08/2013
        mov     ax, dx ; 01/08/2013

; 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
        ; AX (r1) = block number
        retn
                ; rts r0
; ****************************************************************************
;
; UNIX.ASM (RETRO UNIX 8086 Kernel - Only for 1.44 MB floppy disks)
; ----------------------------------------------------------------------------
; U4.ASM (include u4.asm) //// UNIX v1 -> u4.s

; RETRO UNIX 8086 (Retro Unix == Turkish Rational Unix)
; Operating System Project (v0.1) by ERDOGAN TAN (Beginning: 11/07/2012)
; 1.44 MB Floppy Disk
; (11/03/2013)
;
; [ Last Modification: 04/07/2014 ] !!! completed !!!
;
; Derivation from UNIX Operating System (v1.0 for PDP-11)
; (Original) Source Code by Ken Thompson (1971-1972)
; <Bell Laboratories (17/3/1972)>
; <Preliminary Release of UNIX Implementation Document>
;
; ****************************************************************************

; 04/07/2014 (swakeup has been removed)
; 11/06/2014 swakeup
; 02/06/2014 swakeup
; 30/05/2014 isintr
; 20/03/2014 sleep
; 18/03/2014 clock
; 25/02/2014 sleep
; 23/02/2014 wakeup, sleep
; 17/02/2014 wakeup
; 14/02/2014 clock
; 14/02/2014 sleep, wakeup (sigle level runq) ((to prevent s/w locking))
; 05/02/2014 sleep, wakeup (SSLEEP/SRUN, p.waitc)
; 26/01/2014
; 10/12/2013
; 07/12/2013 clock
; 23/10/2013 wakeup, sleep
; 20/10/2013 isintr, clock, wakeup, sleep
; 05/10/2013 clock, wakeup, sleep
; 24/09/2013 sleep, wakeup (consistency check)
; 22/09/2013 sleep, wakeup (completed/modified)
; 20/09/2013 clock, sleep
;       NOTE: 'sleep' and 'wakeup' need to be modified according to
;             original Unix v1 waiting channel feature.
;             Currently 'wakeup' is disabled and 'sleep' is not written
;             properly and clock, sleep, wakeup are not similar
;             to original unix v1 (musti tasking, time sharing feature).
; 03/09/2013 clock, isintr
; 30/08/2013 clock
; 21/08/2013
; 29/07/2013 sleep
; 09/07/2013 clock (INT 1Ch handler)
; 16/05/2013 'isintr' modifications
; 15/05/2013
; 09/05/2013
; 11/03/2013
;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
        ; 10/04/2014
        ; 18/03/2014
        ; 14/02/2014 uquant --> u.quant
        ; 10/12/2013
        ; 07/12/2013
;; Retro Unix 8086 v1 Modification: INT 1Ch interrupt handler !
;; 30/08/2013
;; 09/07/2013
       ;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:
;; 30/08/2013
;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;; 09/07/2013

        ; 20/10/2013
        push    ds
        push    cs
        pop     ds
        ;
        ;; 10/04/2014
        ;pushf
        ;call   dword ptr [int1Ch]  ; Old INT 1Ch
        ;                           ; (Turn off floppy motor)

        cmp     byte ptr [u.quant], 0
        ja      short clk_1

        ; 03/09/2013
        cmp     byte ptr [sysflg], 0FFh ; user or system space ?
        jne     short clk_2 ; system space (sysflg <> 0FFh)
        ;; 06/12/2013
        cmp     byte ptr [u.uno], 1 ; /etc/init ?
        ; 14/02/2014
        jna     short clk_1 ; yes, do not swap out
        cmp     word ptr [u.intr], 0
        ; 14/02/2014
        jna     short clk_2
clk_0:
        ; 30/08/2013
        ;cli
        ;;push  cs
        ;;pop   ds
        ; 18/03/2014
        inc     byte ptr [sysflg] ; Now, we are in system spacee
        ;
        mov     word ptr [u.r0], ax
        ; 07/12/2013
        pop     ax ; DS (user)
        ;
        mov     word ptr [u.usp], sp
        ;; 07/12/2013
        ;;mov   ax, ss ; mov ax, es
        ;;mov   word ptr [u.segmnt], ax
        mov     ax, cs
        ;mov    es, ax ; 18/03/2014
        mov     sp, sstack
        mov     ss, ax
        ;
        push    word ptr [u.usp]
        push    dx
        push    cx
        push    bx
        push    si
        push    di
        push    bp
        ;
        mov     word ptr [u.sp_], sp
        ;sti
        ; 07/12/2013
         jmp     sysrelease ; 'sys release' by clock/timer
clk_1:
        dec     byte ptr  [u.quant]
clk_2:
        ; 20/10/2013
        pop     ds
        iret
```

```
;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;

       ;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

@@:     ; 22/09/2013
        retn
```

```
wakeup: ; / wakeup processes waiting for an event
        ; / by linking them to the queue
        ;
        ; 02/06/2014
        ; 23/02/2014
        ; 17/02/2014
        ; 14/02/2014 single level runq (BX input is not needed)
        ; 05/02/2014 SSLEEP/SRUN, p.waitc
        ; 23/10/2013 (consistency check is OK)
        ; 20/10/2013
        ; 10/10/2013
        ; 05/10/2013
        ; 24/09/2013 (consistency check is OK)
        ; 22/09/2013
        ; 18/08/2013 -> tty lock and console tty setting (p.ttyc)
        ; 15/05/2013
        ; 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)
        ;         ;;BX = Run queue (r2) offset
        ;
        ; ((modified registers: AX, BX))
        ;
        ; 20/10/2013
        ; 10/10/2013
        ;;cmp   byte ptr [u.uno], 2
        ;;jb    short wakeup_4
        ; 14/02/2014
        xor     bh, bh
        mov     bl, al
        add     bx, offset wlist
        ; 23/02/2014
        mov     al, byte ptr [BX] ; waiting list (waiting process number)

        and     al, al
        jz      short @f ; nothing to wakeup
        ;cmp    al, 1
        ;jb     short @f ; nothing to wakeup

        ; 23/02/2014
        ;
        xor     ah, ah
        mov     byte ptr [u.quant], ah ; 0 ; time quantum = 0
        mov     byte ptr [BX], ah ; 0 ; zero wait channel entry
        push    di
        push    dx
        call    putlu
        pop     dx
        pop     di
@@:
        retn

                ;mov     r1,-(sp) / put char on stack
                ;mov     (r0)+,r2 / r2 points to a queue
                ;mov     (r0)+,r3 / r3 = wait channel number
                ;movb    wlist(r3),r1 / r1 contains process number
                ;         / in that wait channel that was sleeping
                ;beq     2f / if 0 return, nothing to wakeup
                ;cmp     r2,u.pri / is runq greater than or equal
                ;               / to users process priority
                ;bhis    1f / yes, don't set time quantum to zero
                ;clrb    uquant / time quantum = 0
        ;1:
                ;clrb    wlist(r3) / zero wait channel entry
                ;jsr     r0,putlu / create a link from the last user
                ;  / on the Q to this process number that got woken
        ;2:
                ;mov     (sp)+,r1 / restore r1
                ;rts     r0

sleep:
        ; 20/03/2014
        ; 25/02/2014
        ; 23/02/2014
        ; 14/02/2014 single level runq
        ; 05/02/2014 SSLEEP/SRUN, p.waitc
        ; 26/01/2014
        ; 10/12/2013
        ; 23/10/2013 (consistency check is OK)
        ; 20/10/2013
        ; 05/10/2013 (u.uno = 1 --> /etc/init ?) (r1 = ah)
        ; 24/09/2013 consistency check -> OK
        ; 22/09/2013
        ; 20/09/2013
        ; 29/07/2013 ;;;
        ; 09/05/2013
        ; 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 ptr [u.uno], 1
         ;ja    short @f
        ;retn

        ; 20/03/2014
        ;mov    bx, word ptr [runq]
        ;cmp    bl, bh
        ;jne    short @f
        ; 25/02/2014
        ;cmp word ptr [runq], 0
        ;ja short @f
        ;retn
@@:
        ;
        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
        ; 20/10/2013
         xor     bh, bh
         mov     bl, ah
         ; 22/09/2013
        add     bx, offset wlist
        ; 23/02/2014
        mov     al, byte ptr [BX]
        and     al, al
        jz      short @f
        push    bx
        call    putlu
        pop     bx
@@:
        mov     al, byte ptr [u.uno]
        mov     byte ptr [BX], 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
         push    word ptr [cdev]
                ; mov cdev,-(sp) / nothing happened in isintr so
        call    swap
                ; jsr r0,swap / swap out process that needs to sleep
         pop     word ptr [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:
        ; 30/05/2014
        ; 20/10/2013
        ; 22/09/2013
        ; 03/09/2013
        ; 16/05/2013 tty/video_page switching
        ; 09/05/2013
        ; 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 ptr [u.ttyp], 0 ; has process got a tty ?
        jna     short isintr2 ; retn
        ; 03/09/2013
        ; (nothing to do)
        ;retn
        ; 22/09/2013
        cmp     word ptr [u.intr], 0
        jna     short isintr2 ; retn
        ; 30/05/2014
        push    ax
        mov     ax, word ptr [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
```

```
; ****************************************************************************
;
; UNIX.ASM (RETRO UNIX 8086 Kernel - Only for 1.44 MB floppy disks)
; ----------------------------------------------------------------------------
; U5.ASM (include u5.asm) //// UNIX v1 -> u5.s

; RETRO UNIX 8086 (Retro Unix == Turkish Rational Unix)
; Operating System Project (v0.1) by ERDOGAN TAN (Beginning: 11/07/2012)
; 1.44 MB Floppy Disk
; (11/03/2013)
;
; [ Last Modification: 07/08/2013 ] ;;; completed ;;;
;
; Derivation from UNIX Operating System (v1.0 for PDP-11)
; (Original) Source Code by Ken Thompson (1971-1972)
; <Bell Laboratories (17/3/1972)>
; <Preliminary Release of UNIX Implementation Document>
;
; ****************************************************************************
; 07/08/2013 iget
; 01/08/2013 alloc, (free3, free), itrunc
; 31/07/2013 u.rw -> rw, setimod, mget
; 28/07/2013 iget, icalc (u.rw)
; 21/07/2013 alloc, free, imap
; 18/07/2013 iget
; 17/07/2013 icalc (inode->i), iget
; 09/07/2013 iget (cdev=1)
; 29/04/2013 access modification
; 26/04/2013 imap, iget (mntd->mdev)
; 24/04/2013 access
; 23/04/2013 itrunc
; 07/04/2013 alloc, free, iget, icalc
; 02/04/2013 alloc
; 01/04/2013 alloc
; 24/03/2013 mget
; 22/03/2013 mget
; 11/03/2013
mget:
        ; 31/07/2013
        ; 24/03/2013
         ; 22/03/2013
        ; 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: DX, BX, CX, SI, DI, BP))

                ; 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     si, word ptr [u.fofp] ; 24/03/2013
         mov     bl, byte ptr [SI]+1
         xor     bh, bh
         ; BX = r2
         test   word ptr [i.flgs], 4096 ; 1000h
                                    ; is this a large or small file
        jnz     short mget_5 ; 4f ; large file

         test    bl, 0F0h ; !0Fh
                ; 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
        mov     ax, word ptr i.dskp[BX] ; AX = R1, physical block number
                ; mov i.dskp(r2),r1 / r1 has physical block number
        or      ax, ax
        jnz     short mget_1 ; if physical block number is zero
                ; bne 2f / if physical block num is zero then need a new block
                       ; / for file
        call    alloc
                ; jsr r0,alloc / allocate a new block
          ; AX (r1) = Physical block number
        mov     word ptr i.dskp[BX], 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:
         ; AX (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
         ; AX (r1) = Physical block number
        call    wslot
                ; jsr r0,wslot / set up I/O buffer for write, r5 points to
                             ; / first data word in buffer
         ; AX (r1) = Physical block number
        mov     cx, 8  ; R3, transfer old physical block pointers
                   ; into new indirect block area for the new
                   ; large file
        mov     di, bx ; r5
        mov     si, offset 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     word ptr [SI]-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
         ; AX (r1) = Physical block number
        mov     word ptr [i.dskp], ax
                ; mov r1,i.dskp / put pointer to indirect block in i-node
        or      word ptr [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     short mget_0
                ; br mget
mget_5:  ; 4 ; large file
        ; 05/03/2013 (UNIXCOPY.ASM)
         ;mov    ax, bx  ; ax <= 255 for this file (UNIX v1, RUFS) system
         ;mov    cx, 256 ; 01/03/2013 no need a division here
        ;xor     dx, dx  ; 01/03/2013 no need a division here
        ;div     cx           ; 01/03/2013 no need a division here
        ;and     bx, 1FEh  ; zero all bit but 1,2,3,4,5,6,7,8
                      ;gives offset in indirect block
        ;push    bx                    ; R2
        ;mov     bx, ax  ; calculate offset in i-node for pointer
                    ; to proper indirect block
        ;and     bx, 0Eh
        ;mov     ax, word ptr i.dskp[BX]  ; R1
```

```
                ; 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    bx  ; 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
        mov     ax, word ptr [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    word ptr i.dskp[BX], ax   ; R1, block number
        mov     word ptr [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
        ; AX = new block number
        call    clear
                ; jsr r0,clear / clear new block
mget_6: ;2
        ; 05/03/2013
        ; AX = r1, physical block number (of indirect block)
        call    dskrd ; read indirect block
                ; jsr r0,dskrd / read in indirect block
        pop     dx  ; R2, get offset (*)
                ; mov (sp)+,r2 / get offset
        ; AX = r1, physical block number (of indirect block)
        push    ax ; ** ; 24/03/2013
                ; mov r1,-(sp) / save block number of indirect block on stack
        ; BX (r5) = pointer to buffer (indirect block)
        add     bx, dx ; / 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
        mov     ax, word ptr [BX] ; 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     word ptr [BX], ax ; R1
                ; mov r1,(r2) / put new block number into proper location in
                            ; / indirect block
        pop     dx ; ** ; 24/03/2013
                ; mov (sp)+,r1 / get block number of indirect block
        push    dx ; ** ; 31/07/2013
        push    ax ; * ; 24/03/2013, 31/07/2013 (new block number)
        mov     ax, dx ; 24/03/2013
                ; mov (r2),-(sp) / save block number of new block
        ; AX (r1) = physical block number (of indirect block)
        call    wslot
                ; jsr r0,wslot
         ; AX (r1) = physical block number
        ; BX (r5) = pointer to buffer (indirect block)
        call    dskwr
        ; AX = r1 = physical block number (of indirect block)
                ; jsr r0,dskwr / write newly modified indirect block
                            ; / back out on disk
        pop     ax ; *  ; 31/07/2013
                ; mov (sp),r1 / restore block number of new block
        ; AX (r1) = physical block number of new block
        call    clear
                ; jsr r0,clear / clear new block
mget_7: ; 2
        pop     dx ; **
                ; tst (sp)+ / bump stack pointer
        ; AX (r1) = Block number of new block
        retn
                ; rts r0
```

```
alloc:
        ; 01/08/2013
        ; 21/07/2013
        ; 02/04/2013
        ; 01/04/2013
        ;
        ; 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   cx
        push    bx ; R2
        ;push   dx ; R3
        ;mov    bx, offset systm ; SuperBlock
        mov     bx, offset s ; 21/07/2013
                ; mov $systm,r2 / start of inode and free storage map for drum
        cmp     byte ptr [cdev], 0
                ; tst cdev
        jna     short alloc_1
                ; beq 1f / drum is device
        mov     bx, offset mount
                ; mov $mount,r2 / disk or tape is device, start of inode and
                              ; / free storage map
alloc_1: ; 1
         mov     ax, word ptr [BX]
                ; mov (r2)+,r1 / first word contains number of bytes in free
                            ; / storage map
        shl     ax, 1
                ; asl r1 / multiply r1 by eight gives
                ; number of blocks in device
        shl     ax, 1
                ; asl r1
        shl     ax, 1
                ; asl r1
        mov     cx, ax
        ;; push cx ;; 01/08/2013
                ; mov r1,-(sp) / save # of blocks in device on stack
        xor     ax, ax ; 0
                ; clr r1 / r1 contains bit count of free storage map
alloc_2: ; 1
        inc     bx ; 18/8/2012
        inc     bx ;
        mov     dx, word ptr [BX]
                ; 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
         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     word ptr [BX], dx   ;; 0 -> allocated
                ; bic r3,(r2) / set bit for this block
                             ; / i.e. assign block
                ; br 2f
        jmp     short alloc_5

free:
        ; 01/08/2013
        ; 21/07/2013
        ; 07/04/2013
        ;
        ; 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   cx
        push    bx ; R2
        ;push   dx ; R3

         call    free3
                ; jsr r0,3f  / set up bit mask and word no.
                                 ; / in free storage map for block
        or      word ptr [BX], 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   dx
                ; mov (sp)+,r3 / restore r2, r3
        pop     bx
                ; mov (sp)+,r2
        ; pop   cx
        cmp     byte ptr [cdev], 0
                ; tst cdev / cdev = 0, block structured, drum;
                        ; / cdev = 1, mountable device
        ja      short alloc_6 ; 1f
                ; bne 1f
        ;mov    byte ptr [smod], 1
        inc     byte ptr [smod]
                ; incb smod / set super block modified for drum
        ; AX (r1) = block number
        retn
                ; rts r0
free_2:
alloc_6: ; 1:
        ;mov byte ptr [mmod], 1
        inc     byte ptr [mmod]
                ; incb  mmod
                  ; / set super block modified for mountable device
        ; AX (r1) = block number
        retn
                ; rts r0
```

```
free3:
        ; 01/08/2013
        ; 02/04/2013
        ;
        ; free3 is called from 'alloc' and 'free' procedures
        ;
alloc_free_3: ; 3
        mov     dx, 1
        mov     cx, ax
                ; mov r1,r2 / block number, k, = 1
        and     cx, 0Fh  ; 0Fh <-- (k) mod 16
                ; bic $!7,r2 / clear all bits but 0,1,2; r2 = (k) mod (8)
        jz      short @f
                ; bisb 2f(r2),r3 / use mask to set bit in r3 corresponding to
                              ; / (k) mod 8
        shl     dx, cl
@@:
        mov     bx, ax
                ; mov r1,r2 / divide block number by 16
        shr     bx, 1
                ; asr r2
        shr     bx, 1
                ; asr r2
        shr     bx, 1
                ; asr r2
        shr     bx, 1
                ; 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 ; 21/8/2012
                ; asl r2 / multiply block number by 2; r2 = k/8
        ;add    bx, offset systm+2 ; SuperBlock+2
        add     bx, offset s + 2 ; 21/07/2013
                ; add $systm+2,r2 / address of word of free storage map for drum
                               ; / with block bit in it
        cmp     byte ptr [cdev], 0
                ; tst cdev
        jna     short alloc_free_5
                ; beq 1f / cdev = 0 indicates device is drum
        ;add    bx, offset mount - offset systm
        add     bx, offset sb1 - offset sb0 ; 21/07/2013
                ; 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:
        ; 07/08/2013
        ; 31/07/2013
        ; 28/07/2013
        ; 18/07/2013
        ; 17/07/2013
        ; 09/07/2013 (cdev,mdev)
        ; 26/04/2013 (mdev)
        ; 07/04/2013
        ;
        ; 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: DX, CX, BX, SI, DI, BP))

        mov     dl, byte ptr [cdev] ; 18/07/2013
        mov     dh, byte ptr [idev] ; 07/08/2013
        ;
        cmp     ax, word ptr [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 @f
                ; beq 2f
iget_1: ; 1:
        xor     bl, bl
        cmp     byte ptr [imod], bl ; 0
                ; tstb imod / has i-node of current file
                          ; / been modified i.e., imod set
        jna     short iget_2
                ; beq 1f
        mov     byte ptr [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, byte ptr [cdev]
        push    dx
                ; mov cdev,-(sp)
        mov     ax, word ptr [ii]
                ; mov ii,r1
        ;mov    dh, byte ptr [idev]
        mov     byte ptr [cdev], dh
                ; mov idev,cdev
        inc     bl ; 1
        ; 31/07/2013
        mov      byte ptr [rw], bl ; 1 == write
        ;;28/07/2013 rw -> u.rw
         ;;mov      byte ptr [u.rw], bl ; 1 == write
        call    icalc
                ; jsr r0,icalc; 1
        pop     dx
        mov     byte ptr [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, byte ptr [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, word ptr [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, byte ptr [mntd]
        inc     dl ; move dl, 1 ; 17/07/2013
         mov    byte ptr [cdev], dl ; 17/07/2013 - 09/07/2013
                ; mov mntd,cdev / make mounted device the current device
        mov     ax, word ptr [rootdir]
                ; mov rootdir,r1
iget_3: ; 1:
        mov     word ptr [ii], ax
                ; mov r1,ii
        mov     byte ptr [idev], dl ; cdev
                ; mov cdev,idev
        xor     bl, bl
         ; 31/07/2013
        mov      byte ptr [rw], bl ; 0 == read
        ;;28/07/2013 rw -> u.rw
         ;;mov     byte ptr [u.rw], bl ; 0 = read
        call    icalc
                ; jsr r0,icalc; 0 / read in i-node ii
iget_4: ; 2:
        mov     ax, word ptr [ii]
                ; mov ii,r1
@@:
        retn
                ; rts r0

icalc:
        ; 31/07/2013
        ; 28/07/2013
        ; 17/07/2013
        ; 07/04/2013
        ;
        ; 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)mod16 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: AX, DX, CX, BX, SI, DI, BP))
        ;

        add     ax, 47  ; add 47 to inode number
                ;  add $31.,r1 / add 31. to i-number
        push    ax
                ; mov r1,-(sp) / save i+31. on stack
        shr     ax, 1
                ; asr r1 / divide by 16.
        shr     ax, 1
                ; asr r1
        shr     ax, 1
                ; asr r1
        shr     ax, 1
                ; 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 ptr [rw], 0 ; Retro Unix 8086 v1 feature !
        ;; 28/07/2013 rw -> u.rw
         ;;cmp     byte ptr [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
        ; AX = r1  = block number
        call    wslot
                ; jsr r0,wslot / set up data buffer for write
                             ; / (will be same buffer as dskrd got)
        ; BX = r5 points to first word in data area for this block
icalc_1: ; 1:
        pop     dx
        and     dx, 0Fh ; (i+47) mod 16
                ; bic $!17,(sp) / zero all but last 4 bits;
                              ; / gives (i+31.) mod 16
        shl     dx, 1
        shl     dx, 1
        shl     dx, 1
        shl     dx, 1
        shl     dx, 1
        ; DX = 32 * ((i+47) mod 16)
        mov     si, bx  ; bx points 1st word of the buffer
        add     si, dx  ; dx is inode offset in the buffer
                ; SI (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    di, offset inode
        mov     di, offset i ; 17/07/2013
                ; mov $inode,r1 / inode is address of first word
                              ; / of current i-node
        mov     cx, 16 ; CX = r3
                ; mov $16.,r3
        ; 31/07/2013
        cmp      byte ptr [rw], ch ; 0  ;; Retro Unix 8086 v1 feature !
        ;;28/07/2013 rw -> u.rw
        ;;cmp      byte ptr [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    si, di
        ; over write old i-node (in buffer to be written)
        rep     movsw
                ; 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     movsw
                ; mov (r5)+,(r1)+ / read new i-node into
                                ; / "inode" area of core
                ; dec r3
                ; bgt 2b
        retn
                ; rts r0
```

```
access:
        ; 29/04/2013 (AX register preserved)
        ; 24/04/2013
        ; 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: CX, BX, SI, DI, BP))
        ;
        push    dx  ; flags
        call    iget
                ; jsr r0,iget / read in i-node for current directory
                            ; / (i-number passed in r1)
        mov     cx, word ptr [i.flgs]
                ; mov i.flgs,r2
        pop     dx
        mov     dh, byte ptr [u.uid_] ; 29/04/2013 al -> dh
        cmp     dh, byte ptr [i.uid] ; 29/04/2013
                ; cmpb i.uid,u.uid / is user same as owner of file
        jne     short access_1
                ; bne 1f / no, then branch
        shr     cl, 1
                ; asrb r2 / shift owner read write bits into non owner
                        ; / read/write bits
        shr     cl, 1
                ; 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 ; 29/04/2013 al -> dh
                ; tstb u.uid
        jnz     error
                ; beq 1f
                ; jmp error
access_2: ; 1:
        retn
                ; rts r0
setimod:
        ; 31/07/2013
        ; 09/04/2013
        ; '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: DX, CX, BX))
        ; push  dx
        push    ax
        mov     byte ptr [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     word ptr [i.mtim], ax
        mov     word ptr [i.mtim]+2, dx
        ; Retro UNIX 8086 v1 modification !
        mov     cx, word ptr [i.ctim]
        mov     bx, word ptr [i.ctim]+2
        test    cx, bx
        jnz     short @f
        mov     word ptr [i.ctim], ax
        mov     word ptr [i.ctim]+2, dx
@@: ; 31/07/2013
        pop     ax
        ;pop    dx
        retn
                ; rts r0
```

```
itrunc:
        ; 01/08/2013
        ; 23/04/2013
        ; '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: DX, CX, BX, SI, DI, BP))

        call    iget
                ; jsr r0,iget
        mov     si, offset i.dskp
                ; mov $i.dskp,r2 / address of block pointers in r2
itrunc_1: ; 1:
        lodsw
                ; mov (r2)+,r1 / move physical block number into r1
        or      ax, ax
        jz      short itrunc_5
                ; beq 5f
        push    si
                ; mov r2,-(sp)
        test     word ptr [i.flgs], 1000h
                ; bit $10000,i.flgs / test large file bit?
        jz      short itrunc_4
                ; beq 4f / if clear, branch
        push    ax
                ; mov r1,-(sp) / save block number of indirect block
        call    dskrd
                ; jsr r0,dskrd / read in block, 1st data word
                             ; / pointed to by r5
        ; BX = r5 = Buffer data address (the 1st word)
        mov     cx, 256
                ; mov $256.,r3 / move word count into r3
        mov     si, bx
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    cx
                ; mov r3,-(sp) / save r3, r5 on stack
        ;push   si
                ; mov r5,-(sp)
        call    free
                ; jsr r0,free / free block in free storage map
        ;pop    si
                ; mov(sp)+,r5
        pop     cx
                ; mov (sp)+,r3
itrunc_3: ; 3:
        loop    itrunc_2
                ; dec r3 / decrement word count
                ; bgt 2b / branch if positive
        pop     ax
                ; mov (sp)+,r1 / put physical block number of
                             ; / indirect block
        ; 01/08/2013
        and      word ptr [i.flgs], 0EFFFh ; 1110111111111111b
itrunc_4: ; 4:
        call    free
                ; jsr r0,free / free indirect block
        pop     si
                ; mov (sp)+,r2
```

```
itrunc_5: ; 5:
        cmp     si, offset 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 ptr [i.flgs], 0EFFFh ; 1110111111111111b
                ; bic $10000,i.flgs / clear large file bit
        mov     di, offset i.dskp
        mov     cx, 8
        xor     ax, ax
        mov     word ptr [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, word ptr [ii]
                ; mov ii,r1
        retn
                ; rts r0

imap:
        ; 26/04/2013
        ; '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: DX, CX, BX, SI))
        ;
                ; / 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
        mov     bx, 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 @f
        ;shl    dx, cl
        shl     dl, cl
                ; mov r3,lsh / move the 1 over (i-41) mod 8 positions
@@:                        ; / to the left to mask the correct bit
        shr     bx, 1
                ; asr r2
        shr     bx, 1
                ; asr r2
        shr     bx, 1
                ; 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    si, offset systm
        mov     si, offset s ; 21/07/2013
                ; mov $systm,r2 / r2 points to the in-core image of
                               ; / the super block for drum
        ;cmp    word ptr [cdev], 0
        cmp     byte ptr [cdev], 0
                ; tst cdev / is the device the disk
        jna     short @f
                ; beq 1f / yes
        ;add    si, offset mount - offset systm
        add     si, offset mount - offset s ; 21/07/2013
                ; add $mount-systm,r2 / for mounted device,
                        ; / r2 points to 1st word of its super block
@@: ; 1:
        add     bx, word ptr [SI] ;; add free map size to si
                ; add (r2)+,(sp) / get byte address of allocation bit
        add     bx, si
                ; add (sp)+,r2 / ?
        add     bx, 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
; ****************************************************************************
;
; UNIX.ASM (RETRO UNIX 8086 Kernel - Only for 1.44 MB floppy disks)
; ----------------------------------------------------------------------------
; U6.ASM (include u6.asm) //// UNIX v1 -> u6.s

; RETRO UNIX 8086 (Retro Unix == Turkish Rational Unix)
; Operating System Project (v0.1) by ERDOGAN TAN (Beginning: 11/07/2012)
; 1.44 MB Floppy Disk
; (11/03/2013)
;
; [ Last Modification: 23/07/2014 ]  ;;; completed ;;;
;
; Derivation from UNIX Operating System (v1.0 for PDP-11)
; (Original) Source Code by Ken Thompson (1971-1972)
; <Bell Laboratories (17/3/1972)>
; <Preliminary Release of UNIX Implementation Document>
;
; ****************************************************************************

; 23/07/2014 rtty
; 07/07/2014 wtty
; 27/06/2014 wtty (putc)
; 19/06/2014 rtty, wtty
; 03/06/2014 (rtty/wtty check is ok)
; 02/06/2014 wtty
; 26/05/2014 wtty
; 15/04/2014 rtty, wtty ('getc' and 'putc' error return modifications)
; 14/04/2014 wtty
; 23/02/2014 rtty
; 01/02/2014 rtty
; 13/01/2014 rtty, wtty
; 06/12/2013 rtty, wtty (major modification: p.ttyc, u.ttyp)
; 10/10/2013 rtty, wtty (tty read lock & tty write lock are removed)
; 05/10/2013 rtty, wtty
; 29/09/2013 rtty
; 20/09/2013 rtty & passc (tty read lock)
;            wtty & cpass (tty write lock), dskw, rmem, wmem
; 13/09/2013 rtty
; 26/08/2013 wtty
; 14/08/2013 rtty, rcvt, wtty, xmtt, cpass
; 03/08/2013 dskr (namei_r), dskw (mkdir_w)
; 01/08/2013 dskw (mkdir_w)
; 31/07/2013 dskr (namei_r), writei
; 29/07/2013 rtty, idle
; 28/07/2013 rtty, rcvt, wtty, u.namei_r
; 26/07/2013 readi
; 16/07/2013 rtty, rcvt, chk_ttyp, rmem, wmem modifications
; 27/05/2013 chk_ttyp
; 21/05/2013 chk_ttyp, chk_com_o
; 20/05/2013 chk_ttyp
; 15/05/2013 rcvt, xmtt, COM1, COM2
; 26/04/2013 readi, writei modifications
; 14/03/2013 -> writei
; 12/03/2013 -> writei, u.segment

; 11/03/2013

readi:
        ; 31/07/2013
        ; 26/07/2013 (namei_r check in 'dskr')
        ; 15/05/2013 COM1, COM2 (serial ports) modification
        ; 26/04/2013 (modification depending on 'dsrkd' modification)
        ; 12/03/2013 -> u.segment
        ; 11/03/2013
        ; 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: DX, BX, CX, SI, DI, BP))
        xor     dx, dx ; 0
        mov     word ptr [u.nread], dx ; 0
                 ; clr u.nread / accumulates number of bytes transmitted
        cmp     word ptr [u.count], dx ; 0
                 ; tst u.count / is number of bytes to be read greater than 0
        ja      short @f ; 1f
                 ; bgt 1f / yes, branch
        retn
                 ; rts r0 / no, nothing to read; return to caller
@@: ; 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
        push    ax ; because subroutines will jump to 'ret_'
@@: ; 1:
        mov     bx, ax
        shl     bx, 1
                 ; asl r1 / multiply inode number by 2
        add     bx, offset @f - 2
        jmp     word ptr [BX]
                 ; jmp *1f-2(r1)
@@: ; 1:
        dw      offset rtty ; tty, AX = 1 (runix)
                 ;rtty / tty; r1=2
                 ;rppt / ppt; r1=4
        dw      offset 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
        dw      offset rfd ; fd0, AX = 3 (runix only)
        dw      offset rfd ; fd1, AX = 4 (runix only)
        dw      offset rhd ; hd0, AX = 5 (runix only)
        dw      offset rhd ; hd1, AX = 6 (runix only)
        dw      offset rhd ; hd2, AX = 7 (runix only)
        dw      offset rhd ; hd3, AX = 8 (runix only)
        dw      offset rlpr ; lpr, AX = 9 (invalid, write only device !?)
        dw      offset rcvt ; tty0, AX = 10 (runix)
                 ;rcvt / tty0
        dw      offset rcvt ; tty1, AX = 11 (runix)
                 ;rcvt / tty1
        dw      offset rcvt ; tty2, AX = 12 (runix)
                 ;rcvt / tty2
        dw      offset rcvt ; tty3, AX = 13 (runix)
                 ;rcvt / tty3
        dw      offset rcvt ; tty4, AX = 14 (runix)
                 ;rcvt / tty4
        dw      offset rcvt ; tty5, AX = 15 (runix)
                 ;rcvt / tty5
        dw      offset rcvt ; tty6, AX = 16 (runix)
                 ;rcvt / tty6
        dw      offset rcvt ; tty7, AX = 17 (runix)
                 ;rcvt / tty7
        dw      offset rcvt ; COM1, AX = 18 (runix only)
                 ;rcrd / crd
        dw      offset rcvt ; COM2, AX = 19 (runix only)
```

```
rtty: ; / read from console tty
        ; 19/06/2014
        ; 15/04/2014 ('getc' error return modifications)
        ; 23/02/2014
        ; 01/02/2014
        ; 13/01/2014
        ; 06/12/2013 (major modification: p.ttyc, u.ttyp)
        ; 10/10/2013
        ; 05/10/2013
        ; 29/09/2013
        ; 20/09/2013 (tty read lock)
        ; 13/09/2013
        ; 14/08/2013
        ; 28/07/2013 u.ttyn
        ; 16/07/2013
        ; 16/07/2013 'getc' modifications
        ; 20/05/2013
        ; 15/05/2013 'getc' error return for serial ports
        ; 14/05/2013 'getc' modifications instead of INT 16h
        ; 11/03/2013
        ; 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
        mov     bl, byte ptr [u.uno] ; process number
        xor     bh, bh
        mov     al, byte ptr [BX]+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     byte ptr [u.ttyn], al
        ; 06/12/2013
        ;; 13/01/2014
        ;;cmp   al, 7
        ;;ja    short rtty_nc
        inc     al
        mov     byte ptr [u.ttyp], al ; tty number + 1
rtty_nc: ; 01/02/2014
        ; 29/09/2013
        mov     cx, 10
@@:     ; 01/02/2014
        push    cx ; 29/09/2013
        ; byte ptr [u.ttyn] = tty number (0 to 9)
        mov     al, 1
        call    getc
        pop     cx ; 29/09/2013
        ; 28/07/2013
        ; byte ptr [u.ttyn] = tty number
        ;; 15/04/2014
        ;;jc    error  ; 15/05/2013 (COM1 or COM2 serial port error)
          ;mov  ah, 01h  ; Test for available key, ZF=1 if none, ZF=0 and
          ;int  16h      ; AX contains next key code if key available.
        jnz     short @f
                ; bne 1f / 2nd word of console tty buffer contains number
                        ; / of chars. Is this number non-zero?
        ;dec    cx
        ;jnz    short rtty_idle
        loop    rtty_idle ; 01/02/2014
        ; 05/10/2013
        mov     ah, byte ptr [u.ttyn]
        ; 29/09/2013
        call    sleep
                ; jsr r0,canon; ttych / if 0, call 'canon' to get a line
                 ;            / (120 chars.)
        ;byte ptr [u.ttyn] = tty number (0 to 9)
        jmp     short rtty_nc ; 01/02/2014

rtty_idle:
        ; 16/07/2013
        ;; mov  cx, word ptr [s.idlet]+2 ;; 29/07/2013
        call    idle
        ; 29/09/2013
        jmp     short @b ; 01/02/2014
```

```
;1:
;rtty_nc:
        ;mov    al, 1
        ;call   getc
          ;mov ah, 01h   ; Test for available key, ZF=1 if none, ZF=0 and
          ;int 16h      ; AX contains next key code if key available.
         ;jz     short ret_
                ; 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
@@:
        xor     al, al
        call    getc
        ;; 23/07/0014
        ;;jc    error ; 15/05/2013 (COM1 or COM2 serial port error)
        ; AL = ascii code of the character
          ;xor  ah, ah
          ;int  16h
        ;
        call    passc
                ; jsr r0,passc / move the character to core (user)
        ; 19/06/2014
        jnz     short rtty_nc
        ; 23/07/2014
        ;jmp    short ret_
        pop     ax
        retn
;ret1:
                ; jmp ret / return to caller via 'ret'

rcvt:    ; < receive/read character from tty >
        ; 06/12/2013 (major modification: p.ttyc, u.ttyp)
        ; 28/07/2013 al = tty number (ah -> al)
        ; 16/07/2013 rttys
        ; 21/05/2013 owner checking for COM/serial ports
        ; 15/05/2013
        ;
        ; 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
         mov     si, word ptr [u.fofp]
@@:
         mov     bx, word ptr [SI]
                ; mov *u.fofp,r1 / save file offset which points to the char
                              ; / to be transferred to user
         inc     word ptr [BX] ; 16/07/2013
                ; inc *u.fofp / increment file offset to point to 'next'
                          ; / char in memory file
        mov     al, byte ptr [BX]
                ; 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
        ; 20/09/2013
        ;jmp    short @b
                ; br rmem / continue
        jnz     short @b
        ;
```

```
ret_:
        pop     ax
        retn
rlpr:
;1:
;rcrd:
        jmp     error
                ;jmp    error / see 'error' routine

dskr:
        ; 03/08/2013
        ; 31/07/2013
        ; 26/07/2013 (namei_r check)
        push    ax ; 26/04/2013
                ; 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
         mov     dx, word ptr [i.size_]
                ; mov i.size,r2 / file size in bytes in r2
        mov     bx, word ptr [u.fofp]
        sub     dx, word ptr [BX]
                ; sub *u.fofp,r2 / subtract file offset
         jna     short ret_
                ; blos ret
         cmp     dx, word ptr [u.count]
                ; cmp r2,u.count / are enough bytes left in file
                               ; / to carry out read
        jnb     short dskr_1
                ; bhis 1f
        mov     word ptr [u.count], dx
                ; mov r2,u.count / no, just read to end of file
dskr_1: ; 1:
        ; AX = i-number
        call    mget
                ; jsr r0,mget / returns physical block number of block
                            ; / in file where offset points
        ; AX = physical block number
        call    dskrd
                ; jsr r0,dskrd / read in block, r5 points to
                             ; / 1st word of data in buffer
        ; BX (r5) = system (I/O) buffer address
        call    sioreg
                ; jsr r0,sioreg
        xchg    si, di
        ; DI = file (user data) offset
        ; SI = sector (I/O) buffer offset
        ; CX = byte count
        ; 03/08/2013
        cmp     byte ptr [namei_r], 0
        ;;28/07/2013 namei_r -> u.namei_r
        ; 26/07/2013
        ;;dec   byte ptr [u.namei_r] ; the caller is 'namei' sign (=1)
        jna     short dskr_2      ; zf=0 -> the caller is 'namei'
        rep     movsb
        jmp     short dskr_3
dskr_2:
        ;;28/07/2013
        ; 26/07/2013
        ;;inc   byte ptr [u.namei_r] ; (=0)
         mov     ax, word ptr [u.segmnt] ; Retro Unix 8086 v1 feature only !
        mov     es, ax ; Retro Unix 8086 v1 feature: ES = user segment !
; 2:
        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
        mov     ax, ds
        mov     es, ax
dskr_3:
        ; 03/08/2013
        pop     ax
        cmp     word ptr [u.count], cx ; 0
                ; tst u.count / all bytes read off disk
                ; bne dskr
        ja      short dskr
        mov     byte ptr [namei_r], cl ; 0
        retn
        ;jna     short ret_
                ; br ret
         ;pop   ax ; 26/04/2013 (i-node number)
        ;jmp    short dskr

passc:
         mov     bx, word ptr [u.segmnt] ; Retro Unix 8086 v1 feature only !
        mov     es, bx  ; Retro Unix 8086 v1 feature: ES = user segment !

        mov     bx, word ptr [u.base]
        mov     byte ptr ES:[BX], al
                ; movb r1,*u.base / move a character to the next byte of the
                               ; / users buffer

        mov     bx, ds ; Retro Unix 8086 v1 feature: DS = system segment !
        mov     es, bx ; Retro Unix 8086 v1 feature: ES = system segment !

        inc     word ptr [u.base]
                ; inc u.base / increment the pointer to point to
                          ; / the next byte in users buffer
        inc     word ptr [u.nread]
                ; inc u.nread / increment the number of bytes read
        dec     word ptr [u.count]
                ; dec u.count / decrement the number of bytes to be read
        ; 20/09/2013 (;;)
        retn
        ;;jnz   short @f
                ; bne 1f / any more bytes to read?; yes, branch
        ;;pop   ax
        ;;      ; mov (sp)+,r0 / no, do a non-local return to the caller of
                             ; / 'readi' by:
;;ret_: ;/ (1) pop the return address off the stack into r0
;;      pop     ax
                ; mov (sp)+,r1 / (2) pop the i-number off the stack into r1
;;@@: ;1:
                ; clr   *$ps / clear processor status
;;      retn
                ; rts r0 / return to address currently on top of stack

writei:
        ; 31/07/2013
        ; 15/05/2013 COM1, COM2 (serial ports) modification
        ; 26/04/2013
        ; 14/03/2013 wslot, sioreg
        ; 12/03/2013
        ; 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     cx, cx
        mov     word ptr [u.nread], cx  ; 0
                ; clr u.nread / clear the number of bytes transmitted during
                            ; / read or write calls
        cmp     word ptr [u.count], cx
        ;       ; tst u.count / test the byte count specified by the user
        ja      short @f ; 1f
                ; bgt 1f / any bytes to output; yes, branch
        retn
        ;       ; rts r0 / no, return - no writing to do
@@: ;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
        ;
        push    ax ; because subroutines will jump to 'ret_'
        mov     bx, ax
        shl     bx, 1
                ; asl r1 / yes, calculate the index into the special file
        add     bx, offset @f - 2
        jmp     word ptr [BX]
                ; jmp *1f-2(r1)
                ; / jump table and jump to the appropriate routine
@@: ;1:
        dw      offset wtty ; tty, AX = 1 (runix)
                 ;wtty / tty; r1=2
                 ;wppt / ppt; r1=4
        dw      offset 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
        dw      offset wfd ; fd0, AX = 3 (runix only)
        dw      offset wfd ; fd1, AX = 4 (runix only)
        dw      offset whd ; hd0, AX = 5 (runix only)
        dw      offset whd ; hd1, AX = 6 (runix only)
        dw      offset whd ; hd2, AX = 7 (runix only)
        dw      offset whd ; hd3, AX = 8 (runix only)
        dw      offset wlpr ; lpr, AX = 9   (runix)
        dw      offset xmtt ; tty0, AX = 10 (runix)
                 ;xmtt / tty0
        dw      offset xmtt ; tty1, AX = 11 (runix)
                 ;xmtt / tty1
        dw      offset xmtt ; tty2, AX = 12 (runix)
                 ;xmtt / tty2
        dw      offset xmtt ; tty3, AX = 13 (runix)
                 ;xmtt / tty3
        dw      offset xmtt ; tty4, AX = 14 (runix)
                 ;xmtt / tty4
        dw      offset xmtt ; tty5, AX = 15 (runix)
                 ;xmtt / tty5
        dw      offset xmtt ; tty6, AX = 16 (runix)
                 ;xmtt / tty6
        dw      offset xmtt ; tty7, AX = 17 (runix)
                 ;xmtt / tty7
        dw      offset xmtt ; COM1, AX = 18 (runix only)
                ; / wlpr / lpr
        dw      offset xmtt ; COM2, AX = 19 (runix only)

wtty: ; write to console tty (write to screen)
        ; 07/07/2014
        ; 27/06/2014
        ; 19/06/2014
        ; 02/06/2014
        ; 26/05/2014 (putc_eot, putc_n, sleep bugfix)
        ; 15/04/2014 ('putc' error return modification)
        ; 14/04/2014 (serial port modification)
        ; 13/01/2014
        ; 06/12/2013 (major modification: p.ttyc, u.ttyp)
        ; 10/10/2013
        ; 05/10/2013
        ; 20/09/2013 (tty write lock)
        ; 13/09/2013
        ; 26/08/2013
        ; 14/08/2013
        ; 28/07/2013 u.ttyn
        ; 21/05/2013 owner checking
        ; 15/05/2013 'mov ah, byte ptr [ptty]', wtty_nc
        ; 14/05/2013 'putc' modifications instead of INT 10h
        ; 12/03/2013
        ; Console tty output is on on current video page
        ; Console tty character output procedure is changed here
        ; according to IBM PC compatible ROM BIOS video (text mode) functions.
        ;
        ; 06/12/2013
        mov     bl, byte ptr [u.uno] ; process number
        xor     bh, bh
        mov     ah, byte ptr [BX]+p.ttyc-1 ; current/console tty
        mov     al, ah ; 07/07/2014
```

```
wttys:  ;
        ; 10/10/2013
        mov     byte ptr [u.ttyn], ah
        ; 06/12/2013
        ;; 13/01/2014
        ;;cmp   ah, 7
        ;;ja    short @f
        ;mov    al, ah
        inc     al
        mov     byte ptr [u.ttyp]+1, al ; tty number + 1
;;@@:   ; 26/08/2013
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
@@:
        ; AH = tty number
        ; AL = ASCII code of the character
        ; 15/04/2014
        push    ax
        call    putc ; 14/05/2013
        jnc     short @f
        ; 02/06/2014
        mov     ah, byte ptr [u.ttyn]
        call    sleep
        pop     ax
        jmp     short @b
                ; 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
@@:
        ; 15/04/2014
        pop     ax
        jmp     short wtty_nc
                ; br wtty
wret:   ; 10/10/2013
        pop     ax
        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 >
        ; 06/12/2013 (major modification: p.ttyc, u.ttyp)
        ; 10/10/2013
        ; 14/08/2013
        ; 28/07/2013
        ; 21/05/2013 owner checking for COM/serial ports
        ; 15/05/2013
        ;
        ; 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:
        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
         mov     si, word ptr [u.fofp]
@@:
        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 ; @f
         mov     bx, word ptr [SI]
                ; mov *u.fofp,r1 / save file offset in r1
         inc     word ptr [BX] ; 16/07/2013
                ; inc *u.fofp / increment file offset to point to next
                            ; / available location in file
        mov     byte ptr [BX], al
                ; movb (sp)+,(r1) / pop char off stack, put in memory loc
                                ; / assigned to it
        jmp     short @b
                ; br wmem / continue
;1:
        ;jmp    error / ?
;@@:
;       ; 20/09/2013
;       pop     ax
;       retn

dskw: ; / write routine for non-special files
        ; 20/09/2013
        ; 03/08/2013
        ; 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     bx, word ptr [u.fofp]
        mov     dx, word ptr [BX]
                ; mov *u.fofp,r2 / put the file offset [(u.off) or the offset
                               ; / in the fsp entry for this file] in r2
        add     dx, word ptr [u.count]
                ; add u.count,r2 / no. of bytes to be written
                               ; / + file offset is put in r2
         cmp     dx, word ptr [i.size_]
                ; cmp r2,i.size / is this greater than the present size of
                              ; / the file?
        jna     short dskw_1
                ; blos 1f / no, branch
         mov     word ptr [i.size_], dx
                ; mov r2,i.size / yes, increase the f11e 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
        ; AX = Block number
                ; jsr r0,mget / get the block no. in which to write
                            ; /the next data byte
        mov      bx, word ptr [u.fofp]
        mov     dx, word ptr [BX]
        and     dx, 1FFh
                ; bit *u.fofp,$777 / test the lower 9 bits of the file offset
        jnz     short dskw_2
                ; bne 2f / if its non-zero, branch; if zero, file offset = 0,
                       ; / 512, 1024,...(i.e., start of new block)
        cmp     word ptr [u.count], 512
                ; cmp u.count,$512. / if zero, is there enough data to fill
                                 ; / an entire block? (i.e., no. of
        jnb     short dskw_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:
        ; AX (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
        ; BX (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
        ; SI = file (user data) offset
        ; DI = sector (I/O) buffer offset
        ; CX = byte count
        ;
         ; 03/08/2013
        ; 01/08/2013
        cmp     byte ptr [mkdir_w], 0
        jna     short dskw_4      ; zf=0 -> the caller is 'mkdir'
        rep     movsb
        jmp     short dskw_5
dskw_4:
        mov      ax, word ptr [u.segmnt] ; Retro Unix 8086 v1 feature only !
        mov     ds, ax ; Retro Unix 8086 v1 feature: ES = user segment !
; 2:
        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

        mov     ax, cs ; Retro Unix 8086 v1 feature: CS = system segment !
        mov     ds, ax ; Retro Unix 8086 v1 feature: DS = system segment !
dskw_5:
        call    dskwr
                ; jsr r0,dskwr / yes, write the block and the i-node
         cmp      word ptr [u.count], 0
                ; tst u.count / any more data to write?
        ja      short dskw_1
                ; bne 1b / yes, branch
        ; 03/08/2013
        mov     byte ptr [mkdir_w], 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
        cmp     word ptr [u.count], 0  ; 14/08/2013
                ; tst u.count / have all the characters been transferred
                            ; / (i.e., u.count, # of chars. left
        jna     short @f
                ; beq 1f / to be transferred = 0?) yes, branch
        dec     word ptr [u.count]
                ; dec u.count / no, decrement u.count
         ;
        mov      bx, word ptr [u.segmnt] ; Retro Unix 8086 v1 feature only !
        mov     es, bx  ; Retro Unix 8086 v1 feature: ES = user segment !
        ;
        mov     bx, word ptr [u.base]
        mov     al, byte ptr ES:[BX] ; Runix v1: get data from user segment!
```

```
                ; movb *u.base,r1 / take the character pointed to
                               ; / by u.base and put it in r1
        mov     bx, ds ; Retro Unix 8086 v1 feature: DS = system segment !
        mov     es, bx ; Retro Unix 8086 v1 feature: ES = system segment !
        ;
        inc     word ptr [u.nread]
                ; inc u.nread / increment no. of bytes transferred
        inc     word ptr [u.base]
                ; inc u.base / increment the buffer address to point to the
@@:     ; 20/09/2013 (;;)
        retn
                ; rts   r0 / next byte
;;@@: ; 1:
;;      pop     ax
                ; mov (sp)+,r0
                          ; / put return address of calling routine into r0
;;dskw_ret:
;;      pop     ax
                ; mov (sp)+,r1 / i-number in r1
;;      retn
                ; rts r0 / non-local return

sioreg:
        ; 22/07/2013
        ; 14/03/2013 bx -> si, ax input -> bx input
        ; 12/03/2013
        ; INPUTS ->
        ;     BX = system buffer (data) address (r5)
        ; OUTPUTS ->
        ;     SI = user data offset (r1)
        ;     DI = system (I/O) buffer offset (r2)
        ;     CX = byte count (r3)
         ; ((Modified registers: AX)) ; 22/07/2013

         mov     si, word ptr [u.fofp]
         mov     di, word ptr [SI]
                ; mov *u.fofp,r2 / file offset (in bytes) is moved to r2
        mov     cx, di
                ; mov r2,r3 / and also to r3
        or      cx, 0FE00h
                ; bis $177000,r3 / set bits 9,...,15 of file offset in r3
        and     di, 1FFh
                ; bic $!777,r2 / calculate file offset mod 512.
        add     di, bx ; BX = system buffer (data) address
                ; add r5,r2 / r2 now points to 1st byte in system buffer
                          ; / where data is to be placed
        mov     ax, word ptr [u.base] ; 22/07/2013
                 ; mov u.base,r1 / address of data is in r1
        neg     cx
                ; neg r3 / 512 - file offset (mod512.) in r3
                      ; / (i.e., the no. of free bytes in the file block)
        cmp     cx, word ptr [u.count]
                ; cmp r3,u.count / compare this with the no. of data bytes
                              ; / to be written to the file
        jna     short @f
                ; blos  2f / if less than branch. Use the no. of free bytes
                         ; / in the file block as the number to be written
        mov     cx, word ptr [u.count]
                ; mov u.count,r3 / if greater than, use the no. of data
                              ; / bytes as the number to be written
@@: ; 2:
        add     word ptr [u.nread], cx
                ; add r3,u.nread / r3 + number of bytes xmitted
                                ; / during write is put into u.nread
        sub     word ptr [u.count], cx
                ; sub r3,u.count / u.count = no. of bytes that still
                              ; / must be written or read
        add     word ptr [u.base], cx
                ; add r3,u.base / u.base points to the 1st of the remaining
                             ; / data bytes
       add      word ptr [SI], cx
                ; add r3,*u.fofp / new file offset = number of bytes done
                              ; / + old file offset
        mov     si, ax ; 22/07/2013
       retn
                ; rts r0
; ****************************************************************************
;
; UNIX.ASM (RETRO UNIX 8086 Kernel - Only for 1.44 MB floppy disks)
; ----------------------------------------------------------------------------
; U7.ASM (include u7.asm) //// UNIX v1 -> u7.s

; RETRO UNIX 8086 (Retro Unix == Turkish Rational Unix)
; Operating System Project (v0.1) by ERDOGAN TAN (Beginning: 11/07/2012)
; 1.44 MB Floppy Disk
; (11/03/2013)
;
; [ Last Modification: 13/07/2014 ]  ;;; completed ;;;
;
; Derivation from UNIX Operating System (v1.0 for PDP-11)
; (Original) Source Code by Ken Thompson (1971-1972)
; <Bell Laboratories (17/3/1972)>
; <Preliminary Release of UNIX Implementation Document>
;
; ****************************************************************************

; 13/07/2014 ottyp
; 12/07/2014 ottyp
; 15/04/2014 ottyp
; 26/01/2014 otty, ottyp, ctty, cttyp
; 17/01/0214 otty, ottyp, ottys, ctty, cttyp
; 13/01/2014 otty, ocvt, ottys, ctty, ccvt, ottyp, cttyp
; 12/01/2014 iclose
; 06/12/2013 otty, ocvt, ctty, ccvt (major modification: p.ttyc, u.ttyp)
; 04/12/2013 (getc, putc procedures have been moved to U9.ASM)
; 03/12/2013 putc (write_tty, beep, waitf)
; 30/11/2013 putc
; 04/11/2013 putc, symount, sysumount
; 30/10/2013 putc
; 20/10/2013 getc
; 10/10/2013 getc
; 05/10/2013 getc
; 24/09/2013 getc, otty, ocvt, ctty, ccvt, putc (consistency check)
; 20/09/2013 putc, getc
; 17/09/2013 otty (ottys), ctty, ccvt
; 16/09/2013 ocvt, ctty
; 13/09/2013 otty
; 03/09/2013 otty, ocvt, ctty, ccvt
; 27/08/2013 iopen, iclose, ocvt, ccvt
; 26/08/2013 putc
; 16/08/2013 iopen, iclose, otty, ctty
; 13/08/2013 ctty (cttys)
; 05/08/2013 ctty
; 30/07/2013 iclose, ctty, ccvt
; 29/07/2013
; 28/07/2013
; 16/07/2013 iopen, otty, ocvt, ctty, ccvt, getc, iclose modifications
; 15/07/2013
; 09/07/2013 - sysmount, sysumount


sysmount: ; / mount file system; args special; name
        ; 04/11/2013
        ; 09/07/2013
        ; '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 f'le 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,
        ;       Retro UNIX 8086 v1 argument transfer method 2 is used
        ;       to get sysmount system call arguments from the user;
        ;       * 1st argument, special is pointed to by BX register
        ;       * 2nd argument, name is in CX register
        ;
        ;       NOTE1: 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.
        ;       NOTE2: 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     word ptr [u.namep], bx
        push    cx
        cmp     word ptr [mnti], 0
                ; tst mnti / is the i-number of the cross device file
                         ; / zero?
        ja      error
                ; bne errora / no, error
        call    getspl
                ; jsr r0,getspl / get special files device number in r1
        ; 04/11/2013
        ;pop    cx ; file name pointer
        mov     bx, ax ; ; Retro UNIX 8086 v1 device number (0 to 5)
        cmp     byte ptr [BX]+drv.err, 0
        ja      error
        ;mov    word ptr [u.namep], cx
        pop     word ptr [u.namep]
                ; mov (sp)+,u.namep / put the name of file to be placed
                                  ; / on the device
        push    ax ; push bx
                ; 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
        mov     word ptr [mnti], ax
                ; mov r1,mnti / put it in mnti
        ; 04/11/2013
        mov     bx, offset sb1 ; super block buffer (of mounted disk)
@@: ;1:
         cmp    byte ptr [BX]+1, 0
                ; tstb sb1+1 / is 15th bit of I/O queue entry for
                          ; / dismountable device set?
         jna    short @f
                ; bne 1b / (inhibit bit) yes, skip writing
        call    idle ; 04/11/2013 (wait for hardware interrupt)
        jmp     short @b
@@:
        pop     ax ; Retro UNIX 8086 v1 device number/ID (0 to 5)
        mov     byte ptr [mdev], al
                ; mov  (sp),mntd / no, put the device number in mntd
        ; 04/11/2013
        mov     byte ptr [BX], al
                ; movb (sp),sb1 / put the device number in the lower byte
                             ; / of the I/O queue entry
        ;mov    byte ptr [cdev], 1 ; mounted device/drive
                ; mov (sp)+,cdev / put device number in cdev
         or     word ptr [BX], 400h ; Bit 10, 'read' flag/bit
                ; bis $2000,sb1 / set the read bit
        mov     byte ptr [BX]+2, 1 ; physical block number = 1
        call    diskio
        jnc     short @f
        xor     ax, ax
        mov     word ptr [mnti], ax ; 0
        mov     byte ptr [mdev], al ; 0
        ;mov    byte ptr [cdev], al ; 0
        mov     word ptr [BX], ax ; 0
        jmp     error
```

```
@@:
        mov     byte ptr [BX]+1, 0 ; 18/07/2013
        ;;call ppoke
                ; jsr r0,ppoke / read in entire file system
;@@: ;1:
        ;;cmp   byte ptr [sb1]+1, 0
                ; tstb   sb1+1 / done reading?
        ;;jna   sysret
        ;,call idle ; 04/11/2013 (wait for hardware interrupt)
        ;;jmp   short @b
                ;bne 1b / no, wait
                ;br sysreta / yes
        jmp     sysret

sysumount: ; / special dismount file system
        ; 04/11/2013
        ; 09/07/2013
        ; 'sysmount' 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,
        ;       Retro UNIX 8086 v1 argument transfer method 1 is used
        ;       to get sysmount system call argument from the user;
        ;       * 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    word ptr [u.namep], bx
        call    getspl
                ; jsr r0,getspl / get the device number in r1
        cmp     al, byte ptr [mdev]
                ; cmp r1,mntd / is it equal to the last device mounted?
        jne     error
                ; bne errora / no error
        xor     al, al ; ah = 0
@@: ;1:
        cmp     byte ptr [sb1]+1, al ; 0
                ; tstb sb1+1 / yes, is the device still doing I/O
                          ; / (inhibit bit set)?
        jna     short @f
                ; bne 1b / yes, wait
        call    idle ; 04/11/2013 (wait for hardware interrupt)
        jmp     short @b
@@:
        mov     byte ptr [mdev], al
                ; clr mntd / no, clear these
        mov     word ptr [mnti], ax
                ; clr mnti
         jmp    sysret
                ; br sysreta / return

getspl: ; / get device number from a special file name
         ; 09/07/2013
        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 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     error
```

```
                ; ble errora / less than 0?  yes, error
         cmp    ax, 5 ;
                ; cmp  r1,$9. / greater than 9  tap 7
        ja      error
                ; bgt errora / yes, error
         ; AX = Retro UNIX 8086 v1 Device Number (0 to 5)
@@:
        retn
                ; rts    r0 / return with device number in r1

iopen:
        ;27/08/2013
        ;16/08/2013
        ;16/07/2013
        ;21/05/2013
        ;
        ; 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: DX, BX, CX, SI, DI, BP))
        ;
; / 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 @f
                ;bgt  3f / no. 3f
        ja      short @b ; 16/08/2013
        push    ax
                ; mov r1,-(sp) / yes, figure out
        mov     bx, ax
        shl     bx, 1
                ; asl r1
         add     bx, offset iopen_1 - 2
        jmp     word ptr [BX]
                ; jmp *1f-2(r1) / which one and transfer to it
iopen_1: ; 1:
        dw      offset otty ; tty, AX = 1 (runix)
                 ;otty / tty ; r1=2
                 ;oppt / ppt ; r1=4
        dw      offset 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
         dw      offset sret ; fd0, AX = 3 (runix only)
         dw      offset sret ; fd1, AX = 4 (runix only)
         dw      offset sret ; hd0, AX = 5 (runix only)
         dw      offset sret ; hd1, AX = 6 (runix only)
         dw      offset sret ; hd2, AX = 7 (runix only)
         dw      offset sret ; hd3, AX = 8 (runix only)
        ;dw     offset error ; lpr, AX = 9 (error !)
         dw      offset sret ; lpr, AX = 9 (runix)
        dw      offset ocvt ; tty0, AX = 10 (runix)
                 ;ocvt / tty0
        dw      offset ocvt ; tty1, AX = 11 (runix)
                 ;ocvt / tty1
        dw      offset ocvt ; tty2, AX = 12 (runix)
                 ;ocvt / tty2
        dw      offset ocvt ; tty3, AX = 13 (runix)
                 ;ocvt / tty3
        dw      offset ocvt ; tty4, AX = 14 (runix)
                 ;ocvt / tty4
        dw      offset ocvt ; tty5, AX = 15 (runix)
                 ;ocvt / tty5
        dw      offset ocvt ; tty6, AX = 16 (runix)
                 ;ocvt / tty6
        dw      offset ocvt ; tty7, AX = 17 (runix)
                 ;ocvt / tty7
        dw      offset ocvt ; COM1, AX = 18 (runix only)
                 ;error / crd
        dw      offset ocvt ; COM2, AX = 19 (runix only)
;@@:
        ;retn

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 ptr [i.flgs], 4000h ; Bit 14 : Directory flag
                ;bit $40000,i.flgs / is it a directory?
        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 @b
                 ;bgt 3f / no, return
        ;push   ax
                ;mov r1,-(sp) / yes
        ;mov    bx, ax
        ;shl    bx, 1
                ; asl r1
        ;add    bx, offset ipen_3 - 2
        ;jmp    word ptr [BX]
                ; jmp *1f-2(r1) / figure out
                        ; / which special file it is and transfer
;iopen_3: ; 1:
;       dw      offset otty ; tty, AX = 1 (runix)
                 ;otty / tty ; r1=2
                 ;leadr / ppt ; r1=4
;       dw      offset 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
;       dw      offset sret ; fd0, AX = 3 (runix only)
;       dw      offset sret ; fd1, AX = 4 (runix only)
;       dw      offset sret ; hd0, AX = 5 (runix only)
;       dw      offset sret ; hd1, AX = 6 (runix only)
;       dw      offset sret ; hd2, AX = 7 (runix only)
;       dw      offset sret ; hd3, AX = 8 (runix only)
;       dw      offset sret ; lpr, AX = 9  (runix)
        ;dw     offset ejec ; lpr, AX = 9  (runix)
;       dw      offset sret ; tty0, AX = 10 (runix)
                 ;ocvt / tty0
;       dw      offset sret ; tty1, AX = 11 (runix)
                 ;ocvt / tty1
;       dw      offset sret ; tty2, AX = 12 (runix)
                 ;ocvt / tty2
;       dw      offset sret ; tty3, AX = 13 (runix)
                 ;ocvt / tty3
;       dw      offset sret ; tty4, AX = 14 (runix)
                 ;ocvt / tty4
;       dw      offset sret ; tty5, AX = 15 (runix)
                 ;ocvt / tty5
```

```
;       dw      offset sret ; tty6, AX = 16 (runix)
                 ;ocvt / tty6
;       dw      offset sret ; tty7, AX = 17 (runix)
                 ;ocvt / tty7
;       dw      offset ocvt ; COM1, AX = 18 (runix only)
                 ;/ ejec / lpr
;       dw      offset ocvt ; COM2, AX = 19 (runix only)


otty: ;/ open console tty for reading or writing
        ; 13/07/2014
        ; 12/07/2014
        ; 15/04/2014 (modification for serial ports)
        ; 26/01/2014
        ; 17/01/2014
        ; 13/01/2014
        ; 06/12/2013 (major modification: p.ttyc, u.ttyp)
        ; 24/09/2013 consistency check -> ok
        ; 17/09/2013
        ; 16/09/2013
        ; 13/09/2013
        ; 03/09/2013
        ; 16/08/2013
        ; 16/07/2013
        ; 15/07/2013
        ; 27/05/2013
        ; 21/05/2013
        ; Retro UNIX 8086 v1 modification !
        ;
        ; 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)
        ; ah = 0
        ; 06/12/2013
        mov     bl, byte ptr [u.uno] ; process number
        xor     bh, bh
        mov     al, byte ptr [BX]+p.ttyc-1 ; current/console tty
        ; 13/01/2014
        jmp     short ottyp
ocvt:
        sub     al, 10
ottyp:
        ; 13/07/2014
        ; 12/07/2014
        ; 15/04/2014 (modification for serial ports)
        ; 26/01/2014
        ; 13/01/2014
        ; 06/12/2013
        mov     dh, al ; tty number
        ; 16/08/2013
        mov     bx, ax ; AL = tty number (0 to 9), AH = 0
        shl     bl, 1  ; aligned to word
        ;26/01/2014
        add     bx, offset ttyl
        mov     cx, word ptr [BX]
                   ; CL = lock value (0 or process number)
                   ; CH = open count
        and     cl, cl
        ; 13/01/2014
        jz      short otty_ret
        ;
        cmp     cl, byte ptr [u.uno]
        je      short otty_ret
        ;
        mov     bl, cl ; the process which has locked the tty
        shl     bl, 1
        xor     bh, bh
        mov     ax, word ptr [BX]+p.pid-2
        mov     bl, byte ptr [u.uno]
        shl     bl, 1
        cmp     ax, word ptr [BX]+p.ppid-2
        je      short otty_ret
        ;;jne   short otty_err
                  ; the tty is locked by another process
                  ; except the parent process (p.ppid)
```

```
;;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_ret
ottys:
        ; 17/01/2013
        push    dx ; *
        mov     ah, dl ; open mode
        mov     dl, dh
        xor     dh, dh
        sub     dl, 8
        ;
        and     ah, ah ; sysstty system call check
        jz      short com_port_init
        ;
        and     cx, cx
        jz      short @f ; unlocked/free tty (serial port)
        ;
        ; 13/01/2014
        ; DX = port number (COM1=0, COM2=1)
        mov     ah, 3
        int     14h   ; Get serial port status
        ; 13/07/2014
        pop     dx ; *
        test    ah, 80h
        jz      short ottys_rtn
;;otty_err: ; 13/01/2014
        or      dl, dl ; DL = 0 -> called by sysstty
        jnz     error
        stc
        retn
@@:
        xor     ah, ah ; 0
com_port_init:
        mov     si, offset com1p
        or      dl, dl ; COM1 ?
        jz      short @f ; yes, it is COM1
        inc     si       ; no, it is COM2
@@:
        mov     al, byte ptr [SI] ; comm. parameters
        ;
        ; Initializing serial port parameters
        ;xor    ah, ah ; 0
        ; AL = Communication parameters
        ; DX = Serial port number (COM1 = 0, COM2 = 1)
        int     14h  ; Initialize serial port parameters
                ;
                ; (Note: Serial port interrupts
                ;       will be disabled here...)
                ; (INT 14h initialization code
                ;       disables interrupts.)
        ; 13/07/2014
        and     dl, dl
        jz      short com1p_eirq
        ;
        ;; COM2 - enabling IRQ 3
        mov     dx, 2FCh   ;modem control register
        in      al, dx    ;read register
        or      al, 8      ;enable bit 3 (OUT2)
        out     dx, al     ;write back to register
        mov     dx, 2F9h   ;interrupt enable register
        in      al, dx     ;read register
        or      al, 1      ;receiver data interrupt enable
        out     dx, al    ;write back to register
        in      al, 21h    ;read interrupt mask register
        and     al, 0F7h   ;enable IRQ 3 (COM2)
        out     21h, al    ;write back to register
        mov     dx, 1
        jmp     short comp_get_stat
com1p_eirq:
        ;; COM1 - enabling IRQ 4
        mov     dx, 3FCh   ;modem control register
        in      al, dx    ;read register
        or      al, 8      ;enable bit 3 (OUT2)
        out     dx, al      ;write back to register
        mov     dx, 3F9h    ;interrupt enable register
        in      al, dx      ;read register
        or      al, 1       ;receiver data interrupt enable
        out     dx, al     ;write back to register
        in      al, 21h     ;read interrupt mask register
        and     al, 0EFh    ;enable IRQ 4 (COM1)
        out     21h, al     ;write back to register
        xor     dx, dx
comp_get_stat:
        mov     ah, 3
        int     14h    ; Get serial port status
        ;
        test    ah, 80h
        jz      short comp_init_ok ; successfully initialized
        ; Initialization ERROR !
                ; 11100011b ; E3h
                 ; (111) Baud rate: 9600, (00) parity: none,
                 ; (0) stop bits: 1, (11) word length: 8 bits
        ; 15/04/2014
        cmp     byte ptr [SI], 0E3h
        je      short @f
        ;
        mov     byte ptr [SI], 0E3h ; Reset comm. parameters
        xor     ah, ah
        jmp     short @b
@@:
        ; 12/07/2014
        pop     dx ; *
        stc
        retn
comp_init_ok:
        ; 12/07/2014
        pop     dx ; *
ottys_ret:
        or      cl, cl   ; cl = lock/owner, ch = open count
        jnz     short @f
        mov     cl, byte ptr [u.uno]
ottys_rtn:
@@:
        inc     ch
        mov     word ptr [BX], cx ; set tty lock again
        ; 06/12/2013
        inc     dh ; tty number + 1
        mov     bx, offset u.ttyp
        ; 13/01/2014
        test    dl, 2 ; open for read sign
        jnz     short @f
        inc     bx
@@:
        ; Set 'u.ttyp' ('the recent TTY') value
        mov     byte ptr [BX], dh ; tty number + 1
sret:
        or      dl, dl ; sysstty system call check (DL=0)
        jz      short @f
        pop     ax
@@:
        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:
        ;13/01/2014
        ;12/01/2014
        ;27/08/2013
        ;16/08/2013
        ;30/07/2013
        ;16/07/2013
        ;21/05/2013
        ;
        ; 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: -BX-, DX))
;/ 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 @b  ; 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
        mov     bx, ax
        shl     bx, 1
                ; asl r1
        add     bx, offset iclose_1 - 2
        jmp     word ptr [BX]
                ; jmp *1f-2(r1) / compute jump address and transfer
iclose_1 :
        dw      offset ctty ; tty, AX = 1 (runix)
        dw      offset cret ; mem, AX = 2 (runix)
        dw      offset cret ; fd0, AX = 3 (runix only)
        dw      offset cret ; fd1, AX = 4 (runix only)
        dw      offset cret ; hd0, AX = 5 (runix only)
        dw      offset cret ; hd1, AX = 6 (runix only)
        dw      offset cret ; hd2, AX = 7 (runix only)
        dw      offset cret ; hd3, AX = 8 (runix only)
        dw      offset cret ; lpr, AX = 9 (runix)
        ;dw     offset error; lpr, AX = 9 (error !)
        ;;dw    offset ejec ;;lpr, AX = 9
        dw      offset ccvt ; tty0, AX = 10 (runix)
        dw      offset ccvt ; tty1, AX = 11 (runix)
        dw      offset ccvt ; tty2, AX = 12 (runix)
        dw      offset ccvt ; tty3, AX = 13 (runix)
        dw      offset ccvt ; tty4, AX = 14 (runix)
        dw      offset ccvt ; tty5, AX = 15 (runix)
        dw      offset ccvt ; tty6, AX = 16 (runix)
        dw      offset ccvt ; tty7, AX = 17 (runix)
        dw      offset ccvt ; COM1, AX = 18 (runix only)
        dw      offset 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)
        ;mov    bx, ax
        ;shl    bx, 1
                ;asl r1 / yes. compute jump address and transfer
        ;add    bx, offset iclose_3 - 2
        ;jmp    word ptr [BX]
                ;jmp *1f-2(r1) / figure out
;iclose_3:
        ;dw     offset ctty ; tty, AX = 1 (runix)
        ;dw     offset sret ; mem, AX = 2 (runix)
        ;dw     offset sret ; fd0, AX = 3 (runix only)
        ;dw     offset sret ; fd1, AX = 4 (runix only)
        ;dw     offset sret ; hd0, AX = 5 (runix only)
        ;dw     offset sret ; hd1, AX = 6 (runix only)
        ;dw     offset sret ; hd2, AX = 7 (runix only)
        ;dw     offset sret ; hd3, AX = 8 (runix only)
         ;dw    offset sret ; lpr, AX = 9
        ;dw     offset ejec ; lpr, AX = 9  (runix)
        ;dw     offset ccvt ; tty0, AX = 10 (runix)
        ;dw     offset ccvt ; tty1, AX = 11 (runix)
        ;dw     offset ccvt ; tty2, AX = 12 (runix)
        ;dw     offset ccvt ; tty3, AX = 13 (runix)
        ;dw     offset ccvt ; tty4, AX = 14 (runix)
        ;dw     offset ccvt ; tty5, AX = 15 (runix)
        ;dw     offset ccvt ; tty6, AX = 16 (runix)
        ;dw     offset ccvt ; tty7, AX = 17 (runix)
        ;dw     offset ccvt ; COM1, AX = 18 (runix only)
        ;dw     offset 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
        ; 26/01/2014
        ; 17/01/2014
        ; 13/01/2014
        ; 06/12/2013 (major modification: p.ttyc, u.ttyp)
        ; 24/09/2013 consistency check -> OK
        ; 17/09/2013
        ; 16/09/2013
        ; 03/09/2013
        ; 16/08/2013
        ; 13/08/2013
        ; 05/08/2013
        ; 30/07/2013
        ; 16/07/2013
        ; 27/05/2013
        ; 21/05/2013
        ; 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
        mov     bl, byte ptr [u.uno] ; process number
        xor     bh, bh
        mov     al, byte ptr [BX]+p.ttyc-1
        ; 13/01/2014
        jmp     short cttyp
ccvt:
        sub al, 10
```

```
cttyp:
        ; 26/01/2014
        ; 13/01/2014
        ; 24/09/2013 consistency check -> ok
        ; 16/08/2013
        ; AH = 0
        mov     bx, ax ; tty number (0 to 9)
        shl     bl, 1  ; aligned to word
        ; 26/01/2014
        add     bx, offset ttyl
        mov     dh, al ; tty number
        mov     ax, word ptr [BX]
                   ; AL = lock value (0 or process number)
                   ; AH = open count
        and     ah, ah
        ;jz     short ctty_err ; open count = 0, it is not open !
        jz      error
        ; 26/01/2014
ctty_ret:
        dec     ah ; decrease open count
        jnz     short @f
        xor     al, al ; unlock/free tty
@@:
        mov     word ptr [BX], ax ; close tty instance
        ;
        mov     bx, offset u.ttyp
        test    dl, 1 ; open for write sign
        jz      short @f
        inc     bx
@@:
        inc     dh ; tty number + 1
        cmp     dh, byte ptr [BX]
        jne     short cret
        ; Reset/Clear 'u.ttyp' ('the recent TTY') value
        mov     byte ptr [BX], 0
cret:
        or      dl, dl ; sysstty system call check (DL=0)
        jz      short @f
        pop     ax
@@:
        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 >
        ; 13/01/2014
        ; 06/12/2013 (major modification: p.ttyc, u.ttyp)
        ; 24/09/2013 consistency check -> ok
        ; 17/09/2013
        ; 03/09/2013
        ; 27/08/2013
        ; 16/08/2013
        ; 30/07/2013
        ; 16/07/2013
        ; 27/05/2013
        ; 21/05/2013
        ;
        ; 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'
```

```
; ****************************************************************************
;
; UNIX.ASM (RETRO UNIX 8086 Kernel - Only for 1.44 MB floppy disks)
; ----------------------------------------------------------------------------
; U8.ASM (include u8.asm) //// UNIX v1 -> u8.s

; RETRO UNIX 8086 (Retro Unix == Turkish Rational Unix)
; Operating System Project (v0.1) by ERDOGAN TAN (Beginning: 11/07/2012)
; 1.44 MB Floppy Disk
; (13/03/2013)
;
; [ Last Modification: 18/01/2014 ] ;;; completed ;;;
;
; Derivation from UNIX Operating System (v1.0 for PDP-11)
; (Original) Source Code by Ken Thompson (1971-1972)
; <Bell Laboratories (17/3/1972)>
; <Preliminary Release of UNIX Implementation Document>
;
; ****************************************************************************

; 18/01/2014
; 03/08/2013 dskwr
; 31/07/2013
; 29/07/2013
; 26/07/2013 bread, bwrite (bug) note
; 23/07/2013 poke
; 20/07/2013 poke, bufaloc, bread, bwrite, dskrd, dskwr, wslot
; 17/07/2013 poke
; 09/07/2013 bufaloc, poke
; 26/04/2013 device number modifications (cdev/0/1 -> 0/rdev, 1/mdev -> drv)
; 18/04/2013
; 24/03/2013 poke
; 15/03/2013 poke, diskio (runix)
; 14/03/2013
; 13/03/2013

;; 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

rfd:    ; 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)
        mov     cx, 2880 ; size of floppy disks (1.44 MB)
        call    bread ; **** returns to routine that called readi ('jmp ret')
wfd:     ; 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)
        mov     cx, 2880 ; size of floppy disks (1.44 MB)
        call    bwrite ; **** returns to routine that called writei ('jmp ret')
rhd:    ; 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)
        mov     cx, 0FFFFh ; size of fixed disks (32 MB, first 65535 sectors)
        call    bread ; **** returns to routine that called readi ('jmp ret')
```

```
whd:
        ; 14/03/2013 Retro UNIX 8086 v1 device (not an original unix v1 device)
        ;sub    ax, 3 ; zero based device number (Hard disk)
        mov     cx, 0FFFFh ; size of fixed disks (32 MB, first 65535 sectors)
        call    bwrite ; **** returns to routine that called writei ('jmp ret')

bread:
        ; 29/07/2013
        ; 20/07/2013
        ; 26/04/2013 Retro Unix 8086 v1 feature (device number) modifications
        ; 14/03/2013
        ; 13/03/2013 Retro UNIX 8086 v1 modification on original unix code
        ;; / 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: DX, CX, BX, SI, DI, BP))
        ;
        ; NOTE: Original UNIX v1 has/had a defect/bug here, even if read
        ;        byte count is less than 512, block number in *u.fopf (u.off)
        ;       is increased by 1. For example: If user/program request
        ;        to read 16 bytes in current block, 'sys read' increaces
        ;       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    cx ; **
        ;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     di, offset brwdev ; block device number for direct I/O
        mov     byte ptr [DI], al
        ;; 20/07/2013
        ;;xor   dx, dx ; 0 is needed for bufaloc_0
        ;
        mov     bx, word ptr [u.fofp]
        mov     ax, word ptr [BX]
                ; mov *u.fofp,r1 / restore r1 to initial value of the
                               ; / block #
        cmp     ax, cx
                ; cmp r1,(r0)+ / block # greater than or equal to maximum
                              ; / block number allowed
        jnb     error        ; 18/04/2013
                ; bhis error10 / yes, error
        inc     word ptr [BX]
                ; inc *u.fofp / no, *u.fofp has next block number
        ; AX = Block number (zero based)
                ;;jsr r0,preread / read in the block whose number is in r1
```

```
preread: ;; call preread
        call    bufaloc_0 ; 26/04/2013
        ;; jc   error
        ; BX = Buffer (Header) Address (r5) (ES=CS=DS, system/kernel segment)
         ; AX = Block/Sector number (r1)
                ; jsr r0,bufaloc / get a free I/O buffer (r1 has block number)
        ; 14/03/2013
         jz     short @f ; Retro UNIX 8086 v1 modification
                ; br 1f / branch if block already in a I/O buffer
        or      word ptr [BX], 400h ; set read bit (10) in I/O Buffer
                ; bis $2000,(r5) / set read bit (bit 100 in I/O buffer)
        call    poke
                ; jsr r0,poke / perform the read
        ;;jc    error ;2 0/07/2013
; 1:
                ; clr *$ps / ps = 0
                ; rts r0
;; return from of preread
@@:
        or      word ptr [BX], 4000h
                ; bis $40000,(r5)
                        ; / set bit 14 of the 1st word of the I/O buffer
@@: ; 1:
        test    word ptr [BX], 2400h
                ; bit $22000,(r5) / are 10th and 13th bits set (read bits)
        jz      short @f
                ; 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, word ptr [s.wait_]+2 ;; 29/07/2013
        call    idle
                ; jsr r0,idle; s.wait+2 / wait
        jmp     short @b
                ; br 1b
@@: ; 1: / 10th and 13th bits not set
        and     word ptr [BX], 0BFFFh ; 1011111111111111b
                ; bic $40000,(r5) / clear bit 14
                ; jsr r0,tstdeve / test device for error (tape)
        ;add    bx, 8
        ; 26/04/2013
        add     bx, 4 ; Retro Unix 8086 v1 modification !
                ; add $8,r5 / r5 points to data in I/O buffer
        ; BX = system (I/O) buffer address
        call    dioreg
                ; jsr r0,dioreg / do bookkeeping on u.count etc.
        ; AX =  [u.base] value before it gets updated
        ; CX =  Byte count to transfer
        ; BX is not changed in dioreg
;1: / r5 points to beginning of data in I/O buffer, r2 points to beginning
;   / of users data
        mov     si, bx
        mov     di, ax
        mov      ax, word ptr [u.segmnt]
                ; Retro Unix 8086 v1 feature only
        mov     es, ax
        rep     movsb
        mov     ax, ds
        mov     es, ax
                ; 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     cx ; **
        cmp     word ptr [u.count], 0
                ; tst u.count / done
        jna     short @f
                ; beq 1f / yes, return
                ; tst -(r0) / no, point r0 to the argument again
        jmp     short bread
                ; br bread / read some more
@@: ; 1:
        pop     ax ; ****
```

```
                ; mov (sp)+,r0
         jmp     ret_
                ;jmp ret  / jump to routine that called readi

bwrite: ; 20/07/2013
        ; 26/04/2013 Retro Unix 8086 v1 feature (device number) modifications
        ; 14/03/2013
        ;; / 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: DX, CX, BX, SI, DI, BP))
        ;
        ; 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' increaces
        ;       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    cx ; **
        ;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     di, offset brwdev ; block device number for direct I/O
        mov     byte ptr [DI], al
        ;; 20/07/2013
        ;;xor   dx, dx ; 0 is needed for bufaloc_0
        ;
        mov     bx, word ptr [u.fofp]
        mov     ax, word ptr [BX]
                ; mov *u.fofp,r1 / put the block number in r1
        cmp     ax, cx
                ; cmp r1,(r0)+ / does block number exceed maximum allowable #
                              ; / block number allowed
        jnb     error        ; 18/04/2013
                ; bhis error10 / yes, error
        inc     word ptr [BX]
                ; inc *u.fofp / no, increment block number
        call    bwslot ; 26/04/2013 (wslot -> bwslot)
                ; jsr r0,wslot / get an I/O buffer to write into
         call   dioreg
                ; jsr r0,dioreg / do the necessary bookkeeping
        ; AX = [u.base] before it gets updated
        ; CX = byte count
        ; BX is not changed
; 1: / r2 points to the users data; r5 points to the I/O buffers data area
        mov     di, bx ; system (I/O) buffer (data) address
        mov     si, ax ; beginning of user data
         mov     ax, word ptr [u.segmnt]
                ; Retro Unix 8086 v1 feature only
        mov     ds, ax
        rep     movsb
        mov     ax, cs
        mov     ds, ax
                ; 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     cx ; **
         cmp     word ptr [u.count], 0
                ; tst u.count / done
        jna     short @f
```

```
                ; beq 1f / yes, 1f
                ; tst -(r0) / no, point r0 to the argument of the call
        jmp     short bwrite
                ; br bwrite / go back and write next block
@@: ; 1:
        pop     ax ; ****
                ; mov (sp)+,r0
         jmp     ret_
                ; jmp ret / return to routine that called writei
;error10:
;        jmp     error  ; / see 'error' routine

dioreg:
        ; 14/03/2013
        ; bookkeeping on block transfers of data
        ;
        ; returns value of u.base before it gets updated, in AX (r2)
        ; returns byte count (to transfer) in CX (<=512)

        mov     cx, word ptr [u.count]
                ; mov u.count,r3 / move char count to r3
        cmp     cx, 512
                ; cmp r3,$512. / more than 512. char?
        jna     short @f
                ; blos 1f / no, branch
        mov     cx, 512
                ; mov $512.,r3 / yes, just take 512.
@@: ; 1:
        mov     ax, word ptr [u.base]
                 ; mov u.base,r2 / put users base in r2
        add     word ptr [u.nread], cx
                ; add r3,u.nread / add the number to be read to u.nread
        sub     word ptr [u.count], cx
                ; sub r3,u.count / update count
        add     word ptr [u.base], cx
                ; add r3,u.base / update base
         retn
                ; rts r0 / return
dskrd:
        ; 29/07/2013
        ; 20/07/2013, 26/04/2013, 14/03/2013
        ;
        ; '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: DX, CX, BX, SI, DI, BP))
        ;
        call    bufaloc
                ; jsr r0,bufaloc / shuffle off to bufaloc;
                               ; / get a free I/O buffer
        ;;jc    error ; 20/07/2013
        jz      short @f ; Retro UNIX 8086 v1 modification
                ; br 1f / branch if block already in a I/O buffer
        or      word ptr [BX], 400h ; set read bit (10) in I/O Buffer
                ; bis $2000,(r5) / set bit 10 of word 1 of
                               ; / I/O queue entry for buffer
        call    poke
                ; jsr r0,poke / just assigned in bufaloc,
                            ; /bit 10=1 says read
        ;;jc    error ; 20/07/2013
@@: ; 1:
                ;clr *$ps
        test    word ptr [BX], 2400h
                ; bit $22000,(r5) / if either bits 10, or 13 are 1;
                ; jump to idle
        jz      short @f
                ; beq 1f
        ;; mov     cx, word ptr [s.wait_]+2 ;; 29/07/2013
        call    idle
                ; jsr r0,idle; s.wait+2
        jmp short @b
                ; br 1b
@@: ; 1:
        ;add   bx, 8
        ; 26/04/2013
        add     bx, 4 ; Retro Unix 8086 v1 modification !
                ; add $8,r5 / r5 points to first word of data in block
                          ; / just read in
        retn
                ; rts r0

bwslot:
        ; 26/04/2013
        ; Retro UNIX 8086 v1 modification !
        ; ('bwslot' will be called from 'bwrite' only!)
        ; INPUT -> DI - points to device id (in bwdev)
        ;       -> AX = block number
        ;
        call    bufaloc_0
        ;; jc   error
        jmp     short @f

wslot:
        ; 29/07/2013
        ; 20/07/2013
        ; 26/04/2013
        ; 14/03/2013
        ;
        ; '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: DX, CX, BX, SI, DI, BP))

        call    bufaloc
                ; jsr r0,bufaloc / get a free I/O buffer; pointer to first
        ;;jc    error ; 20/07/2013
        ; BX = Buffer (Header) Address (r5) (ES=CS=DS, system/kernel segment)
        ; AX = Block/Sector number (r1)
        ; jz short @f
                ; br 1f / word in buffer in r5
@@: ;1:
        test    word ptr [BX], 2400h
                ; bit $22000,(r5) / check bits 10, 13 (read, waiting to read)
                               ; / of I/O queue entry
        jz      short @f
                 ; beq 1f   / branch if 10, 13 zero (i.e., not reading,
                     ; / or not waiting to read)

        ;; mov     cx, word ptr [s.wait_]+2 ; 29/07/2013
        call    idle
                ; jsr r0,idle; / if buffer is reading or writing to read,
                              ; / idle
        jmp     short @b
                ; br 1b / till finished
@@: ;1:
         or      word ptr [BX], 8200h
                ; bis $101000,(r5) / set bits 9, 15 in 1st word of I/O queue
                                ; / (write, inhibit bits)
                ; clr     *$ps / clear processor status
         ;add   bx, 8
        ; 26/04/2013
        add     bx, 4 ; Retro Unix 8086 v1 modification !
                ; add $8,r5 / r5 points to first word in data area
                          ; / for this block
        retn
                ; rts r0
dskwr:
        ; 03/08/2013
        ; 31/07/2013
        ; 20/07/2013
        ; 26/04/2013
        ; 14/03/2013
        ;
        ; '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: CX, DX, BX, SI, DI)
        ;
        ;
        ; 03/08/2013 (si -> bx)
        mov     bx, word ptr [bufp]
        and     word ptr [bx], 7FFFh ; 0111111111111111b
                ; bic $100000,*bufp / clear bit 15 of I/O queue entry at
                                    ; / bottom of queue
ppoke:
                ; mov $340,*$ps
                ; jsr r0,poke
                ; clr *$ps
                ; rts r0
poke:
        ; 18/01/2014
        ; 31/07/2013
        ; 23/07/2013
        ; 20/07/2013
        ; 17/07/2013
        ; 09/07/2013
        ; 26/04/2013
        ; 24/03/2013 AX (r1) -> push/pop (to save physical block number)
        ; 15/03/2013
        ; (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: CX, DX, SI, DI)

        ; 20/07/2013 modifications
        ;              (Retro UNIX 8086 v1 features only !)
        ; INPUTS ->
        ;         (BX = buffer header address)
        ; OUTPUTS ->
        ;        cf=0 -> successed r/w (at least, for the caller's buffer)
        ;        cf=1 -> error, word ptr [BX] = 0FFFFh
        ;               (drive not readi or r/w error!)
        ;        (word ptr [BX]+2 <> 0FFFFh indicates r/w success)
        ;        (word ptr [BX]+2 = FFFFh mean RW/IO error)
        ;         (also it indicates invalid buffer data)

        ; 17/07/2013
        push    bx
        ; 24/03/2013
                ; mov r1,-(sp)
                ; mov r2,-(sp)
                ; mov r3,-(sp)
        push    ax ; Physical Block Number (r1) (mget)
        ;mov    si, offset bufp + nbuf + nbuf + 6
                ; mov $bufp+nbuf+nbuf+6,r2 / r2 points to highest priority
                                       ; / I/O queue pointer
         mov     si, offset bufp + (2*nbuf) + (2*2)  ; 09/07/2013
poke_1: ; 1:
         dec    si
        dec     si
        mov     bx, word ptr [SI]
                ; mov -(r2),r1 / r1 points to an I/O queue entry
        mov     ax, word ptr [BX] ; 17/07/2013
        test    ah, 06h
        ;test   word ptr [BX], 600h ; 0000011000000000b
                ; bit $3000,(r1) / test bits 9 and 10 of word 1 of I/O
                              ; / queue entry
         jz      short poke_2
                ; beq 2f / branch to 2f if both are clear
        ; 31/07/2013
        ;test   ah, 0B0h ; (*)
        ;;test  word ptr [BX], 0B000h ; 1011000000000000b
                ; bit $130000,(r1) / test bits 12, 13, and 15
         ;jnz     short poke_2 ; 31/07/2013 (*)
                ; bne 2f / branch if any are set
        mov     cl, byte ptr [BX] ; 26/04/2013 ; Device Id
                ; movb (r1),r3 / get device id
        xor     ch, ch ; mov ch, 0 ; 26/04/2013
        ;mov    di, cx ; 26/04/2013
        xor     ax, ax ; 0
        ;cmp    byte ptr [DI]+drv.err, al ; 0 ; 26/04/2013
                ; tstb deverr(r3) / test for errors on this device
        ;jna    short poke_3
                ; beq 3f / branch if no errors
        ; 20/07/2013
        ;dec    ax
        ;mov    word ptr [BX]+2, ax ; FFFFh ; -1
                ; mov $-1,2(r1) / destroy associativity
        ;inc    ah ; 0
        ;mov    word ptr [BX], ax ; 00FFh, reset
                ; clrb 1(r1) / do not do I/O
        ;jmp     short poke_2
         ;        ; br 2f
                 ; rts r0
poke_3: ; 3:
        ; 26/04/2013 Modification
        inc     al ; mov ax, 1
        or      cl, cl ; Retro UNIX 8086 v1 device id.
        jz      short @f ; cl = 0
        shl     al, cl ; shl ax, cl
@@::
        ;test   word ptr [active], ax
        test    byte ptr [active], al
                ; bit $2,active / test disk busy bit
         jnz     short poke_2
                ; bne 2f / branch if bit is set
        ;or     word ptr [active], ax
        or      byte ptr [active], al
                ; bis $2,active / set disk busy bit
        push    ax ; 17/07/2013
        call    diskio ; Retro UNIX 8086 v1 Only !
        mov      byte ptr [DI]+drv.err, ah
        pop     ax
        jnc     short @f ; 20/07/2013
                ; tstb deverr(r3) / test for errors on this device
                ; beq 3f / branch if no errors
        ; 20/07/2013
        mov     word ptr [BX]+2, 0FFFFh ; -1
                ; mov $-1,2(r1) / destroy associativity
        mov     byte ptr [BX]+1, 0
                ; clrb 1(r1) / do not do I/O
        jmp      short poke_2
@@:     ; 20/07/2013
        ; 17/07/2013
        not     al
        and     byte ptr [active], al ; reset, not busy
        ; BX = system I/O buffer header (queue entry) address
seta: ; / I/O queue bookkeeping; set read/write waiting bits.
        mov     ax, word ptr [BX]
                ; 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 ptr [BX], 0F9FFh
                ; bic $3000,(r1) / clear only bits 9 and 10
        ;shl    ax, 1
        ;shl    ax, 1
        ;shl    ax, 1
                ; rol r3
                  ; rol r3
                  ; rol r3
        ; 23/07/2013
        shl     ah, 1
        shl     ah, 1
        shl     ah, 1
        or      word ptr [BX], ax
                ; bis r3,(r1) / or old value of bits 9 and 10 with
                           ; bits 12 and 13
        call    idle ; 18/01/2014
        ;; sti
        ;hlt    ; wait for a hardware interrupt
        ;; cli
        ; NOTE: In fact, disk controller's 'disk I/O completed'
         ; interrupt would be used to reset busy bits, but INT 13h
        ; returns when disk I/O is completed. So, here, as temporary
        ; method, this procedure will wait for a time according to
        ; multi tasking and time sharing concept.
        not     ax
        and     word ptr [BX], ax ; clear bits 12 and 13
poke_2: ;2:
         cmp     si, offset bufp
                ; cmp r2,$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    ax  ; Physical Block Number (r1) (mget)
        ; 17/07/2013
        pop     bx
        ; 20/07/2013
        cmp     word ptr [BX]+2, 0FFFFh
        je      error
        ; '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
        ;
        ;cmc
        retn
                 ; rts r0

bufaloc:
        ; 29/07/2013
        ; 20/07/2013
        ; 09/07/2013
        ; 26/04/2013 (device number/id modifications)
        ; 13/03/2013
        ; 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  si ; ***
                ; mov r2,-(sp) / save r2 on stack
                ; mov $340,*$ps / set processor priority to 7
        ; 20/07/2013
        ; 26/04/2013
        xor     bh, bh
        mov     bl, byte ptr [cdev] ; 0 or 1
        mov     di, offset rdev  ; offset mdev = offset rdev + 1
        add     di, bx
```

```
bufaloc_0: ; 26/04/2013 !! here is called from bread or bwrite !!
                      ;; DI points to device id.
        ; 20/07/2013
        mov     bl, byte ptr [DI] ; DI -> rdev/mdev or brwdev
        xor     bh, bh
        cmp     byte ptr [BX]+drv.pdn, 0FFh ; Drive not ready !
        je      error ; 20/07/2013
@@:
        mov     dx, bx  ; dh = 0, dl = device number (0 to 5)
        xor     bp, bp ; 0
        push    bp ; 0
         mov      bp, sp
        ;
bufaloc_1: ;1:
                ; clr -(sp) / vacant buffer
         mov    si, offset bufp
                ; mov $bufp,r2 / bufp contains pointers to I/O queue
                            ; / entrys in buffer area
bufaloc_2: ;2:
        mov     bx, word ptr [SI]
        inc     si
        inc     si
                ; mov (r2)+,r5 / move pointer to word 1 of an I/O
                           ; queue entry into r5
        test    word ptr [BX], 0F600h
                ; bit $173000,(r5) / lock+keep+active+outstanding
         jnz    short bufaloc_3
                ; bne 3f / branch when
                       ; / any of bits 9,10,12,13,14,15 are set
                        ; / (i.e., buffer busy)
         mov     word ptr [BP], si ; pointer to word 2 of I/0 queue
                                  ; entry
                 ; mov  r2,(sp) ;/ save pointer to last non-busy buffer
                       ; / found points to word 2 of I/O queue entry)
bufaloc_3: ;3:
        ;mov    dl, byte ptr [DI] ; 26/04/2013
        ;
        cmp     byte ptr [BX], dl
                ; cmpb (r5),cdev / is device in I/O queue entry same
                                ; / as current device
        jne     short bufaloc_4
                ; bne 3f
        cmp     word ptr [BX]+2, ax
                ; cmp 2(r5),r1 / is block number in I/O queue entry,
                            ; / same as current block number
        jne     short bufaloc_4
                ; bne 3f
        ;add    sp, 2
        pop     cx
                ; tst (sp)+ / bump stack pointer
        dec     si ; 09/07/2013
        dec     si ; 09/07/2013
        jmp     short bufaloc_7 ; Retro Unix 8086 v1 modification
                                ; jump to bufaloc_6 in original Unix v1
                ; br 1f / use this buffer
bufaloc_4: ;3:
        cmp     si, offset bufp + nbuf + nbuf
                ; cmp r2,$bufp+nbuf+nbuf
        jb      short bufaloc_2
                ; blo 2b / go to 2b if r2 less than bufp+nbuf+nbuf (all
                        ; / buffers not checked)
         pop    si
                ; mov (sp)+,r2 / once all bufs are examined move pointer
                            ; / to last free block
        or      si, si
        jnz     short bufaloc_5
                ; bne 2f / if (sp) is non zero, i.e.,
                 ; / if a free buffer is found branch to 2f
         ;; mov  cx, word ptr [s.wait_]+2 ;; 29/07/2013
        call    idle
                ; jsr r0,idle; s.wait+2 / idle if no free buffers
        ; 26/04/2013
        ;xor    dx, dx
        xor     dl, dl
        push    dx ; 0
        ;
        jmp     short bufaloc_1
                ; br 1b
```

```
bufaloc_5: ;2:
                ; tst (r0)+ / skip if warmed over buffer
        inc     dh ; Retro UNIX 8086 v1 modification
bufaloc_6: ;1:
        dec     si
        dec     si
         mov            bx, word ptr [SI]
                ; mov -(r2),r5 / put pointer to word 1 of I/O queue
                             ; / entry in r5
        ;; 26/04/2013
         ;mov   dl, byte ptr [DI] ; byte ptr [rdev] or byte ptr [mdev]
        mov     byte ptr [BX], dl
                ; movb cdev,(r5) / put current device number
                                ; / in I/O queue entry
        mov     word ptr [BX]+2, ax
                ; mov r1,2(r5) / move block number into word 2
                             ; / of I/O queue entry
bufaloc_7: ;1:
         cmp    si, offset 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)
        dec     si
        dec     si
        mov     cx, word ptr [SI]
        mov     word ptr [SI]+2, cx
                ; mov -(r2),2(r2) / job for a particular device
        jmp     short bufaloc_7
                ; br 1b
bufaloc_8: ;1:
         mov    word ptr [SI], bx
                ; mov r5,(r2)
        ;;pop   si ; ***
                ; mov (sp)+,r2 / restore r2
        or      dh, dh ; 0 or 1 ?
                ; Retro UNIX 8086 v1 modification
                ; zf=1 --> block already in a I/O buffer
                ; zf=0 --> a new I/O buffer has been allocated
        retn
                ; rts r0

diskio:
        ; 26/04/2013 Device ID modifications
        ; 15/03/2013
        ; 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 ->
        ;           BX = System I/O Buffer header address
        ; OUTPUTS -> cf=0 --> done
        ;             cf=1 ---> error code in AH
        ;
        ; (Modified registers: CX,DX,AX)

        ;; 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 -> ; 26/04/2013
        ;;            Word 3 = number of words in buffer (=256)
        ;; Original UNIX v1 -> ; 26/04/2013
        ;;            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) ; 26/04/2013
        ;;         0 = fd0
        ;;         1 = fd1
        ;;         2 = hd0
        ;;         3 = hd1
        ;;         4 = hd2
        ;;         5 = hd3

        mov     dx, 0201h ; Read 1 sector/block
        mov     ax, word ptr [BX]
        ; 26/04/2013
        push    si ; ****
        mov     cl, al
        xor     ch, ch
        mov     si, cx
        ;
        test    ah, 2
        ;test   ax, 200h ; Bit 9 of word 0 (status word)
                          ; write bit
        jz      short @f
        ;test   ah, 4
        ;;test  ax, 400h ; Bit 10 of word 0 (status word)
        ;               ; read bit
        ;jz     short diskio_ret
        inc     dh ; 03h = write
@@:
        ;mov    cx, 4 ; Retry Count
        mov     cl, 4

        ; push  ds
        ; pop   es
@@:
        push    dx ; ***
        push    bx ; ***
        push    cx ; ***
        push    dx ; ** ; I/O type (Int 13h function, r/w)
        inc     bx ; +1
        inc     bx ; +2
        mov     ax, word ptr [BX] ; Block/Sector number
        xor     dx, dx
        shl     si, 1 ; 2 * device number ; 26/04/2013
        mov     cx, word ptr [SI]+drv.spt
                               ; Sectors per track
        div     cx
        mov     cx, dx ; remainder, sector (zero based)
        inc     cx      ; sector (1 based)
        push    cx ; *
        mov     cx, word ptr [SI]+drv.hds ; Heads
        xor     dx, dx
        ; ax = track number
        div     cx
        mov     dh, dl ; head number (<=255)
        shr     si, 1 ; device number ; 26/04/2013
        mov      dl, byte ptr [SI]+drv.pdn ; 26/04/2013
                        ; Physical device number
        pop     cx ; * ; cx = sector of track (1 to spt)
        inc     bx ; +2
        inc     bx ; +3  ; I/O Buffer (Data)
        mov     ch, al ; low 8 bytes of cylinder number
        ror     ah, 1
        ror     ah, 1
        or      cl, ah
        pop     ax ; ** ; AH=2-read, AH=3-write
        int     13h     ; AL-count CH-track CL-sect
                         ; DH-head DL-drive ES:BX-buffer
                         ; CF-flag AH-stat AL-sec read
        pop     cx ; ***
        pop     bx ; ***
        jnc     short @f
        cmp     cl, 1
        jb      short @f
        xor     ah, ah ; Disk Reset
        int     13h
        dec     cx
        pop     dx ; ***
        jmp     short @b
@@:
        pop     dx ; ***
        pop     si ; ****
        retn
```

```
; ****************************************************************************
;
; UNIX.ASM (RETRO UNIX 8086 Kernel - Only for 1.44 MB floppy disks)
; ----------------------------------------------------------------------------
; U9.ASM (include u9.asm) //// UNIX v1 -> u9.s

; RETRO UNIX 8086 (Retro Unix == Turkish Rational Unix)
; Operating System Project (v0.1) by ERDOGAN TAN (Beginning: 11/07/2012)
; 1.44 MB Floppy Disk
; (11/03/2013)
;
; [ Last Modification: 01/09/2014 ] ;;; completed ;;;
;
; Derivation from UNIX Operating System (v1.0 for PDP-11)
; (Original) Source Code by Ken Thompson (1971-1972)
; <Bell Laboratories (17/3/1972)>
; <Preliminary Release of UNIX Implementation Document>
;
; ****************************************************************************

; 28/08/2014
; 28/07/2014
; 27/07/2014
; 23/07/2014
; 20/07/2014
; 12/07/2014
; 04/07/2014
; 30/06/2014
; 27/06/2014
; 25/06/2014
; 11/06/2014
; 03/06/2014
; 02/06/2014
; 05/05/2014
; 30/04/2014
; 17/04/2014
; 15/04/2014
; 04/04/2014 scroll_up
; 07/03/2014
; 04/03/2014 act_disp_page --> tty_sw
; 03/03/2014 int_09h, int_16h
; 28/02/2014 int_16h
; 17/02/2014
; 14/02/2014
; 01/02/2014 write_tty
; 18/01/2014
; 17/01/2014
; 13/01/2014 getc, putc
; 12/12/2013
; 10/12/2013
; 07/12/2013
; 04/12/2013 getc, putc, write_tty
; 04/11/2013 drv_init
; 24/07/2013 bf_init
; 20/07/2013 bf_init
; 19/07/2013 drv_init
; 18/07/2013 drv_init
; 17/07/2013 bf_init
; 14/07/2013
; 13/07/2013 drv_init, dparam (Retro UNIX 8086 v1 features only!)
; 21/05/2013 'ocvt' & 'ccvt' routines (in U7.ASM)
; 15/05/2013 'rcvt' & 'xmtt' routines (in U6.ASM)
; 11/03/2013

;;rcvt:
;; 'rcvt' routine is in U6.ASM (Retro UNIX 8086 v1 modification!)

;;xmtt:
;; 'xmtt' routine is in U6.ASM (Retro UNIX 8086 v1 modification!)

;;ocvt:
;; 'ocvt' routine is in U7.ASM (Retro UNIX 8086 v1 modification!)

;;ccvt:
;; 'ccvt' routine is in U7.ASM (Retro UNIX 8086 v1 modification!)
```

```
drv_init:
        ; 04/11/2013
        ; 19/07/2013
        ; 18/07/2013
        ; 14/07/2013
        ; 13/07/2013
        ; Retro UNIX 8086 v1 feature only !
        ;
        ; Derived from DRVINIT.ASM (DRVINIT4) file of TR-DOS project
        ; by Erdogan Tan, (26/09/2009 --> 07/08/2011)
        ;
        ; Modified/Simplified for Retro UNIX 8086 v1
        ;
        ; (LBA disks excluded, hard disk file systems excluded)
        ;
        ; ((RUFS and/or TRFS/SINGLIX partitions will be validated
        ;  in future RUNIX/TR-UNIX versions if they will be available.)
        ;
        ; Input: none
        ; Output:
        ;       cf = 0 -> disk drive initialization is ok.
        ;       cf = 1 -> error (error code in ah)
        ; ((Modified registers: AX, BX, CX, DX, SI, DI))
fd_init:
         xor    dx, dx  ; fd0
         xor    si, si ; 0
        call    dparam
        inc     si ; 1
        cmp     al, 2 ; 04/11/2013
        jb      short hd_init
        inc     dl ; fd1
        call    dparam
hd_init:
        inc     si ; 2
        mov     dl, 80h ; hd0
        call    dparam
        jc      short drv_init_lbs
        ; al = number of hard disk drives
        cmp     al, 2 ; 04/11/2013
        jb      short drv_init_lbs
        mov     byte ptr [brwdev], al ; 19/07/2013
@@:
        dec     byte ptr [brwdev] ; 19/07/2013
        jz      short drv_init_lbs
        inc     si
        inc     dl
        call    dparam
        jmp     short @b

drv_init_lbs:
        push    cs ; 14/07/2013
        pop     es ; 14/07/2013
        xor     bx, bx
        mov     dl, byte ptr [unixbootdrive]
@@:
        cmp     dl, byte ptr [BX]+drv.pdn
        je      short @f
         cmp     bx, si ; 19/07/2013
        jnb     short drv_init_err
        inc     bl
        jmp     short @b
drv_init_err:
        mov     ah, byte ptr [BX]+drv.err
        stc
        retn
@@:
        cmp     byte ptr [BX]+drv.err, 0
        ja      short drv_init_err
        mov     si, offset sb0 ; super block buffer
        mov     byte ptr [SI], bl ; Device Id
        mov     byte ptr [SI]+1, 4 ; Bit 10,
                                ; read bit
        mov     byte ptr [rdev], bl ; 19/07/2013
        mov     bx, si
        inc     byte ptr [BX]+2 ; physical block number = 1
        call    diskio
        mov     byte ptr [BX]+1, 0 ; 18/07/2013
        retn
```

```
dparam:
        ; 13/07/2013
        ; Retro UNIX 8086 v1 feature only !
        ;
         push   dx
         mov    ah, 08h
         int    13h
         mov    byte ptr [SI]+drv.err, ah
         jnc    short @f
dparam_error:
        pop     dx
        retn
@@:
        mov     al, dl ; Number of disk drives
        ;cmp    al, 1
        ;jb     short dparam_err
        ; dh = last head number
        inc     dh
         mov      dl, dh
         xor      dh, dh
         shl    si, 1 ; align to word ptr drv.hds
         mov      word ptr [SI]+drv.hds, dx
                               ; number of heads
         and      cx, 3Fh
        ; SI is already aligned for word ptr drv.spt
         mov      word ptr [SI]+drv.spt, cx
        shr     si, 1 ; align to byte ptr drv.pdn
        pop     dx
        mov     byte ptr [SI]+drv.pdn, dl
                             ; Physical drive number
        retn

bf_init:
        ; 24/07/2013 (from last to first)
        ; 20/07/2013 Device id reset (0FFh)
        ; 17/07/2013
        ; Buffer (pointer) initialization !
        ;
        ;   Retro UNIX 8086 v1 feature only !
        ;
        mov     cl, nbuf
        mov     di, offset bufp
        ; 24/07/2013
        mov     ax, offset Buffer + (nbuf*516)
        mov     dx, 0FFFFh
@@:
        ; 24/07/2013
        sub     ax, 516 ; 4 header + 512 data
        stosw
        mov     si, ax ; 24/07/2013
        ; mov   word ptr [SI], dx ; 0FF00h
        mov     byte ptr [SI], dl ; 0FFh
                          ; Not a valid device sign
        ;mov    word ptr [SI]+2, dx ; 0FFFFh
                     ; Not a valid block number sign
        dec     cl
        jnz     short @b
        mov     ax, offset sb0
        stosw
        mov     ax, offset sb1
        stosw
        ; 20/07/2013
        mov     si, ax ; offset sb1
        mov     byte ptr [SI], dl ; 0FFh
        ;mov    word ptr [SI]+2, dx ; 0FFFFh
         ;
        retn
```

```
getc:
        ;04/07/2014 (rcvc has been removed)
        ;           (serial port interrupts)
        ;27/06/2014 (rcvc, EOT)
        ;03/06/2014 (rcvc)
        ;02/06/2014 (rcvc has been moved here again)
        ;05/05/2014 (rcvc has been moved from here)
        ;17/04/2014
        ;15/04/2014 (rcvc)
        ;17/02/2014
        ;14/02/2014
        ;17/01/2014
        ;13/01/2014
        ;10/12/2013
        ;20/10/2013
        ;10/10/2013
        ;05/10/2013
        ;24/09/2013
        ;20/09/2013
        ;29/07/2013 (getc_s, sleep -> idle)
        ;28/07/2013 (byte ptr [u.ttyn] = tty number)
        ;16/07/2013
        ;20/05/2013
        ;14/05/2013 (AH input instead of 'mov ax, byte ptr [ptty]')
        ;13/05/2013
        ; 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: AX, BX, -CX-, -DX-, -SI-, -DI-))
        ;

        ; 16/07/2013
        ; mov   byte ptr [getctty], ah
        ;

        mov     ah, byte ptr [u.ttyn] ; 28/07/2013
getc_n:
        ; 10/10/2013
        mov     bx, offset ttychr
        and     ah, ah
        jz      short @f
        shl     ah, 1
        ; 17/02/2014
        add     bl, ah
        adc     bh, 0
        ; 24/09/2013
        ;mov    bl, ah
        ;xor    bh, bh
        ;shl    bl, 1
        ;add    bx, offset ttychr
@@:
        mov     cx, word ptr [BX] ; ascii & scan code
                                  ; (by kb_int)
        or      cx, cx
        jnz     short @f
        and     al, al
        jz      short getc_s
        xor     ax, ax
        retn
```

```
@@:
        and     al, al
        mov     ax, cx
        mov     cx, 0
        jnz     short @f
getc_sn:
        mov     word ptr [BX], cx ; 0, reset
        cmp     ax, cx   ; zf = 0
@@:
        retn
getc_s:
        ; 14/02/2014 uquant -> u.quant
        ; 10/12/2013
        ; 20/10/2013
        ; 05/10/2013
        ; 24/09/2013
        ; 20/09/2013
        ; 29/07/2013
        ; 28/07/2013
        ; 16/07/2013
        ; 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.)
        ;; 29/07/2013
        ; 20/09/2013
        ;((Modified registers: AX, BX, CX, DX, SI, DI))
        ;
        ; 05/10/2013
        ; ah = byte ptr [u.ttyn] ; (tty number)
        ;
        ; 10/10/2013
gcw0:
        mov     cl, 10 ; ch = 0
gcw1:
        call    idle
        mov     ax, word ptr [BX] ; ascii & scan code
                                ; (by kb_int)
        or      ax, ax
        jnz     short gcw3
        loop    gcw1
        ;
        mov     ah, byte ptr [u.ttyn] ; 20/10/2013
        ; 10/12/2013
        cmp     ah, byte ptr [ptty]
        jne     short gcw2
        ; 14/02/2014
        cmp     byte ptr [u.uno], 1
        jna     short gcw0
gcw2:
        call    sleep

        ; 20/09/2013
        mov     ah, byte ptr [u.ttyn]
        xor     al, al
        jmp     short getc_n
gcw3:
        ; 10/10/2013
        xor     cl, cl
        jmp     short getc_sn
```

```
sndc:    ; <Send character>
        ;
        ; 28/07/2014
        ; 27/07/2014
        ; 23/07/2014
        ; 20/07/2014
        ; 12/07/2014
        ; 04/07/2014
        ; 27/06/2014
        ; 25/06/2014
        ; 15/04/2014
        ; 13/01/2014
        ; 16/07/2013 bx
        ; 14/05/2013
        ;
        ; Retro UNIX 8086 v1 feature only !
        ;
        ; 12/07/2014
        xor     dh, dh
        mov     dl, ah
        ; 27/07/2014
        sub     dl, 8
        ; 25/06/2014
        push    ax
sndcs:
        ; 28/07/2014
;       ; 27/07/2014
;       mov     cx, 10
;@@:
        mov     ah, 3   ; Get serial port status
        int     14h
        test    ah, 20h ; Transmitter holding register empty ?
        jnz     short @f
;       call    idle
;       loop    @b
        ;
        push    dx
        push    bx
        ; 27/07/2014
        mov     bx, dx
        add     bx, offset tsleep
        ;
        mov     ah, byte ptr [u.ttyn]
        ;
        mov     byte ptr [BX], ah ; 27/07/2014
        ;
        call    sleep
        pop     bx
        pop     dx
        jmp     short sndcs
@@:
        pop     ax
@@:
        ;mov    ah, 1   ; Send character
        ;int    14h
        ; 13/07/2014
        push    dx
        or      dl, dl
        mov     dx, 2F8h   ;data port (COM2)
        jnz     short @f
        add     dx, 100h   ;3F8h, data port (COM1)
@@:
        out     dx, al     ;send on serial port
        pop     dx
        ; 27/07/2014
        call    idle
        ;
        mov     ah, 3   ; Get serial port status
        int     14h
        cmp     ah, 80h ; time out error
        cmc     ; cf = 0 (OK), cf = 1 (error!)
@@:
        retn
```

```
putc:
        ;27/07/2014
        ;23/07/2014, 20/07/2014
        ;27/06/2014 (sndc, EOT)
        ;25/06/2014, 05/05/2014, 15/04/2014, 13/01/2014
        ;04/12/2013 write_tty
        ;03/12/2013 write_tty, beep, waitf
        ;          (for video page switch bug-fixing)
        ;30/11/2013, 04/11/2013, 30/10/2013
        ;24/09/2013 consistency check -> ok
        ;20/09/2013 (cx = repeat count)
        ;  (int 10h, function 0Eh -> function 09h)
        ;  (video page can be selected in function 09h only!)
        ;26/08/2013, 14/05/2013
        ; 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: AX, BX, CX, DX, SI, DI))
        ;
        cmp     ah, 7
        ja      short sndc ; send character

write_tty:
        ; 01/02/2014
        ; 18/01/2014, 12/12/2013, 04/12/2013
        ; 03/12/2013
        ; (Modified registers: AX, BX, CX, DX, SI, DI)

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

        ; mov   bl, 07h

; 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                                               :
;--------------------------------------------------------------------------

        ;;push  ax              ; save character and video page number
        ;;mov   bh, ah          ; get page setting
        ;;mov   ah, 03h         ; (read cursor position)
        ;;int   10h
        ;;pop   ax              ; recover character and video page

        cli

        ; READ CURSOR (04/12/2013)
        xor     bh, bh
        mov     bl, ah
        shl     bl, 1
        add     bx, offset cursor_posn
        mov     dx, word ptr [BX]
        ;mov    cx, word ptr [cursor_mode]
        ;

        ;mov    bl, 07h         ;
        ;mov    bh, ah          ;
        mov     bl, ah          ; video page number
        ;xor    bh, bh

        ; 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:
        ;mov    ah, 0Ah         ; write character only command
        ;mov    cx, 1           ; only one character
        ;int    10h             ; write the character

        mov     ah, 07h ; attribute/color
        ; al = character
        ; bl = video page number (0 to 7)
        ;
        call    write_c_current

        ; position the cursor for next char

        inc     dl
        cmp     dl, 80          ; test for column overflow
        ;jne    short u7
         jne     set_cpos
        mov     dl, 0
        cmp     dh, 25-1        ; check for last row
        jne     short u6

        ; scroll required
u1:
        ;;mov   ah, 02h
        ;;int   10h             ; set the cursor
        ; SET CURSOR POSITION (04/12/2013)
        call    set_cpos

        ; determine value to fill with during scroll
u2:
        ;;mov   ah, 08h         ; get read cursor command
        ;;int   10h             ; read char/attr at current cursor

        ; 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, byte ptr [crt_mode]        ; move current mode into ah
        ;
        ; bl = video page number
        ;
        call    find_position  ; get regen location and port address
        ; dx = status port
        ;mov    si, di          ; establish addressing in si
        ; si = cursor location/address
        ;push   es              ; get regen segment for quick access
        ;pop    ds
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:
        ;lodsw                  ; get the character and attribute
        ;
        push    ds
        mov     ax, 0B800h
        mov     ds, ax
        mov     ax, word ptr [SI]
        pop     ds
        ;
        ; al = character, ah = attribute
        ;
        sti
        mov     bh, ah          ; store in bh
        ; 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, 80          ; lower right column
        ;dec    dl
        ;;mov   dl, 79

        ;call   scroll_up       ; 04/12/2013
        mov     al, 1
        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:
        push    cx
        mov     al, 20h
        call    write_tty
        pop     cx
        dec     cl
        jnz     short ts
        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     short u1        ; yes, scroll the screen

beeper: ; 18/01/2014 (sti)
        ; 17/01/2014 (call from 'kb_int')
        ;sti

        ; bell found
u11:
        sti ; 01/02/2014
        ; 12/12/2013
        cmp     bl, byte ptr [active_page]
        jne     short @f        ; 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:
        ; 18/01/2014
        ; 10/12/2013
        ; 07/12/2013 (sti)
        ; 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     cx, 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
        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
        out     PORT_B, al      ; and stop speaker timer
        ;popf                   ; restore interrupt flag state
        sti
        mov     cx, 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
@@:
        retn

REFRESH_BIT equ        00010000b      ; REFRESH TEST BIT

waitf:
        ; 03/12/2013
        ;
        ; TEST4.ASM - 06/10/85  POST AND BIOS UTILITY ROUTINES
        ;
        ; 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

        ; delay for (cx)*15.085737 us
        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
```

```
set_cpos:
        ; 01/09/2014
        ; 12/12/2013
        ; 04/12/2013
        ;
        ; 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 ID SET AT 6845 IF DISPLAY PAGE IS CURRENT DISPLAY

        ;mov    al, bh  ; move page number to work register
        mov     al, bl  ; page number
        cbw             ; convert page to word value
        mov     si, ax  ; ah = 0, al = video page number
        shl     si, 1   ; word offset
        mov     word ptr [SI + offset cursor_posn], dx ; save the pointer
        ; 01/09/2014
        cmp     byte ptr [active_page], bl ; al
        jne     short m17
        mov     cx, word ptr [crt_start]
        ;
        mov     ax, dx  ; get row/column to ax
        ;call   m18     ; CURSOR SET
;m17:                   ; SET_CPOS_RETURN
        ; 01/09/2014
;       retn
m18:
        call    position ; determine location in regen buffer
        ; 01/09/2014
        add     cx, ax  ; add to the start address for this page
        ;sar    cx, 1
        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, word ptr [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
m17:
        ; 01/09/2014
        retn
```

```
position:
        ; 04/12/2013
        ;
        ; 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

        push    bx      ; save register
        mov     bl, al
        mov     al, ah  ; rows to al
        ;mul    byte ptr [crt_cols] ; determine bytes to row
        mov     bh, 80
        mul     bh
        xor     bh, bh
        add     ax, bx  ; add in column value
        ;sal    ax, 1
        shl     ax, 1   ; * 2 for attribute bytes
        pop     bx
        retn

find_position:
        ; VIDEO.ASM - 06/10/85  VIDEO DISPLAY BIOS
        mov     cl, bl ; video page number
        xor     ch, ch
        mov     si, cx  ; ch = 0, cl = video page number
        shl     si, 1
        mov     ax, word ptr [SI + Offset cursor_posn]
        jz      short p21
        ;
        xor     si, si  ; else set buffer address to zero
        ;
p20:
        ;add    si, word ptr [crt_len] ; add length of buffer for one page
        add     si, 80*25*2
        loop    p20
p21:
        and     ax, ax
        jz      short @f
        call    position ; determine location in regen in page
        add     si, ax   ; add location to start of regen page
@@:
        ;mov    dx, word ptr [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
        ; cx = 0
        retn
```

```
scroll_up:
        ; 04/04/2014 (BugFix)
        ; 12/12/2013
        ; 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
        ;
        ; ((ah = 3))
        ; dl = 79
        ; dh = 24
        ;
        ; al = line count (0 or 1) ((0 == clear video page))
        ;       ((al = 1 for write_tty (putc) procedure))
        ; bl = video page number (0 to 7)
        ; bh = attribute to be used on blanked line

        ;cli
        push    ax
        cmp     bl, byte ptr [active_page]
        je      short n0
        xor     si, si
        and     bl, bl
        jz      short n9
        mov     cl, bl
@@:
        add     si, 25*80*2 ; 04/04/2014
        dec     cl
        jnz     short @b
        jmp     short n9
n0:
        mov     si,  word ptr [crt_start]
n1:     ; 04/04/2014
        ;mov    di, si
        ;
        ;inc    dh
        ;inc    dl      ; increment for origin
        ; dl = 80
        ; dh = 25
        ;cmp    bl, byte ptr [active_page]
        ;jne    short n9
        ;
        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
n9:
        pop     cx      ; al = line count
        ;
        mov     di, si  ; 04/04/2014
        ;
        push    es
        push    ds
        mov     ax, 0B800h
        mov     es, ax
        mov     ds, ax
        ;
        and     cl, cl
        jnz     short @f
        ; clear video page
        mov     cx, 25 * 80
        jmp     short n3
```

```
@@:
        ;mov    ax, 160
;       mov     al, 160 ;   2 * (80 columns)
;       mul     cl
        ;add    si, ax
        add     si, 160
;       ;mov    cx, 24
;n2:                    ; row loop
;       ;call   n10     ; move one row
;       ;add    si, ax
;       ;add    di, ax
;       ;loop   n2
;       mov     al, cl
;       mov     cl, 25
;       sub     cl, al
;       xor     ch, ch
;       ; cx = line count to move
;@@:
;       push    cx
n10:
        ;mov    cx, 80
        mov     cx, 24*80 ; 24 rows/lines
        rep     movsw   ; move one line (up)
        ;loop   n2
;       pop     cx
;       loop    @b
;       mov     cl, al
        mov     cl, 80
n3:                     ; clear entry
        mov     ah, bh  ; attribute in ah
        mov     al, 20h ; fill with blanks
        ; cx = word count to clear (80 or 25*80)
;@@:
;       push    cx
n11:
;       mov     cl, 80  ; get # of columns to clear
        rep     stosw   ; store the fill character
;       pop     cx
;       loop    @b
n5:                     ; SCROLL_END
        pop     ds
        cmp     bl, byte ptr [active_page]
        jne     short @f
        ;mov    al, byte ptr [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
@@:
        pop     es
        ;sti
        retn
```

```
write_c_current:
        ; 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
        ; si = 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, byte ptr [active_page]
        jne     short p44 ; 18/01/2014
        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:    ; 18/01/2014
        sti
p44:
        pop     ax      ; restore the character (al) & attribute (ah)
        push    ds
        mov     cx, 0B800h
        mov     ds, cx
        mov     word ptr [SI], ax
        pop     ds
        pop     dx
        retn
```

```
tty_sw:
        mov     byte ptr [u.quant], 0   ; 04/03/2014
        ;
;act_disp_page:
        ; 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    si ; 10/12/2013
        ;push   bx
        push    cx
        push    dx
        ;
        mov     byte ptr [active_page], al ; save active page value ; [ptty]
        ;mov    cx, word ptr [crt_len] ; get saved length of regen buffer
        mov     cx, 25*80*2
        cbw             ; convert AL to word
        push    ax      ; save page value
        mul     cx      ; display page times regen length
        ; 10/12/2013
        mov     word ptr [crt_start], ax ; save start address for later
        mov     si, ax
        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
        pop     bx      ; recover page value
        ;sal    bx, 1
        shl     bx, 1   ; *2 for word offset
        mov     ax, word ptr [BX + offset cursor_posn] ; get cursor for this page
        call    m18
        ;
        pop     dx
        pop     cx
        ;pop    bx
        pop     si ; 10/12/2013
        ;
        ;sti
        ;
        retn

get_cpos:
        ; 04/12/2013 (sysgtty)
        ;
        ; INPUT -> bl = video page number
        ; RETURN -> dx = cursor position

        push    bx
        xor     bh, bh
        shl     bl, 1
        add     bx, offset cursor_posn
        mov     dx, word ptr [BX]
        pop     bx
        retn
```

```
read_ac_current:
        ; 04/12/2013 (sysgtty)
        ;
        ; INPUT -> bl = video page number
        ; RETURN -> ax = character (al) and attribute (ah)

        call    find_position
        push    ds
        mov     ax, 0B800h
        mov     ds, ax
        mov     ax, word ptr [SI]
        pop     ds
        retn


; 11/06/2014
; Retro UNIX 8086 v1 feature only
; (INPUT -> none)
syssleep:
        mov     bl, byte ptr [u.uno] ; process number
        xor     bh, bh
        mov     ah, byte ptr [BX]+p.ttyc-1 ; current/console tty
        call    sleep
        jmp     sysret


; COMMENT $

; 28/02/2014
; Keyboard function variables (for INT 16h)
; DS = 40h
;;DDSDATA         equ 40h
;
;;KB_FLAG    equ 17h ; byte
;;;KB_FLAGS  equ 17h ; word ; initial value = 0
;;BUFF_HEAD  equ 1Ah ; word ; initial value = offset KB_BUFF
;;BUFF_TAIL  equ 1Ch ; word ; initial value = offset KB_BUFF
;;BUFF_START equ 80h ; word ; initial value = offset KB_BUFF
;;BUFF_END   equ 82h ; word ; initial value = offset KB_BUFF + 32
;;;KB_BUFF   equ 1Eh ; 32 bytes ; Keyboard buffer (circular queue buffer)

; 03/03/2014
BIOS_DSEGM      equ     40h
RESET_FLAG      equ     72h     ; WORD=1234H IF KEYBOARD RESET UNDERWAY
                                ; (40h:72h)
;----------------------------------------
;       VIDEO DISPLAY DATA AREA            ;
;----------------------------------------
CRT_MODE        equ     49h     ; CURRENT DISPLAY MODE (TYPE)
CRT_MODE_SET    equ     65h     ; CURRENT SETTING OF THE 3X8 REGISTER

;--------- 8042 COMMANDS ----------------------------------------------------
ENA_KBD         equ     0AEh    ; ENABLE KEYBOARD COMMAND
DIS_KBD         equ     0ADh    ; DISABLE KEYBOARD 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

;---------- KEYBOARD SCAN CODES ------------------------------------------------
ID_1            equ     0ABh            ; 1ST ID CHARACTER FOR KBX
ID_2            equ     041h            ; 2ND ID CHARACTER FOR KBX
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
NUM_KEY         equ     69              ; SCAN CODE FOR         NUMBER LOCK KEY
RIGHT_KEY       equ     54              ; SCAN CODE FOR         RIGHT SHIFT
SCROLL_KEY      equ     70              ; SCAN CODE FOR         SCROLL LOCK KEY
SYS_KEY         equ     84              ; SCAN CODE FOR         SYSTEM KEY
```

```
;---------- 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 HAS BEEN TOGGLED
NUM_STATE      equ     00100000b      ; NUM LOCK STATE HAS BEEN TOGGLED
CAPS_STATE     equ     01000000b      ; CAPS LOCK STATE HAS BEEN TOGGLED
INS_STATE      equ     10000000b      ; INSERT STATE IS ACTIVE

;---------- FLAG EQUATES WITHIN      @KB_FLAG_1 -------------------------------------
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     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      ; REYBOARD TRANSMIT ERROR FLAG

;----------- FLAGS EQUATES WITHIN @KB_FLAG_3 -----------------------------------
KBX            equ     00000001b      ; KBX INSTALLED
LC_HC          equ     00000010b      ; LAST SCAN CODED WAS A HIDDEN CODE
GRAPH_ON       equ     00000100b      ; ALL GRAPHICS KEY DOWN (W.T. ONLY)
;              equ     00011000b      ; RESERVED (MUST BE ZERO)
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)
;
;----- THIS CODE CONTAINS THE KBX SUPPORT FOR INT 09H
;      EQUATES
F11_M          equ     217            ; FUNC 11 MAKE
F11_B          equ     215            ; FUNC 11 BREAK
F12_M          equ     218            ; FUNC 12 MAKE
F12_B          equ     216            ; FUNC 12 BREAK
K102_M         equ     86             ; KEY 102 MAKE
K102_B         equ     214            ; KEY 102 BREAK
;
INS_M          equ     82             ; INSERT KEY MAKE
DEL_M          equ     83             ; DELETE KEY MAKE
LEFT_M         equ     75             ; CURSOR LEFT MAKE
RIGHT_M        equ     77             ; CURSOR RIGHT MARE
UP_M           equ     72             ; CURSOR UP MAKE
DN_M           equ     80             ; CURSOR DOWN MAKE
PGUP_M         equ     73             ; PG UP MAKE
PGDN_M         equ     81             ; PG DN MAKE
HOME_M         equ     71             ; HOME MAKE
END_M          equ     79             ; END MAKE
;
FUNC11         equ     133            ; FUNCTION 11 KEY
HC             equ     224            ; HIDDEN CODE
;----------- INTERRUPT EQUATES -------------------------------------------------
EOI            equ     020h           ; END OF INTERRUPT COMMAND TO 8259
INTA00         equ     020h           ; 8259 PORT
```

```
int_16h:
        ; 30/06/2014
        ; 03/03/2014
        ; 28/02/2014
        ; Derived from "KEYBOARD_IO_1" procedure of IBM "pc-at"
        ; rombios source code (06/10/1985)
        ;       'keybd.asm', INT 16H, KEYBOARD_IO
        ;
        ; 06/10/85  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).      :
        ;                                    :
        ;       (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.       :
        ;       (AH)= 02H      RETURN THE CURRENT SHIFT STATUS IN (AL) REGISTER    :
        ;                       THE BIT SETTINGS FOR THIS CODE ARE INDICATED IN THE :
        ;                       EQUATES FOR @KB_FLAG          :
        ; OUTPUT                                        :
        ;       AS NOTED ABOVE, ONLY (AX) AND FLAGS CHANGED :
        ;       ALL REGISTERS RETAINED        :
        ;--------------------------------------------------------------------------------

        sti
        push    ds                      ; SAVE CURRENT DS
        push    bx                      ; SAVE BX TEMPORARILY
         mov     bx, cs
        mov     ds, bx                  ; PUT SEGMENT VALUE OF DATA AREA INTO DS
        or      ah, ah                  ; CHECK FOR (AH)= 00H
        jz      short k1b               ; ASCII_READ
        ;
        dec     ah
        jz      short k2                ; CHECK FOR (AH)= 01H
                                        ; ASCII_STATUS
        dec     ah                      ; CHECK FOR (AH)= 02H
        jz      short k3                ; SHIFT STATUS
        pop     bx                      ; RECOVER REGISTER
        pop     ds                      ; RECOVER SEGMENT
        iret                            ; INVALID COMMAND EXIT

        ;----- READ THE KEY TO FIGURE OUT WHAT TO DO
k1b:
         mov     bx, word ptr [BUFFER_HEAD] ; GET POINTER TO HEAD OF BUFFER
         cmp     bx, word ptr [BUFFER_TAIL] ; TEST END OF BUFFER
        ;; 28/08/2014
        ;;jne   short k1c               ; IF ANYTHING IN BUFFER SKIP INTERRUPT
        jne     short k1d
        ;;mov   ax, 09002h              ; MOVE IN WAIT CODE A TYPE
        ;;int   15h                     ; PERFORM OTHER FUNCTION
k1:                                     ; ASCII READ
        sti                             ; INTERRUPTS BACK ON DURING LOOP
        nop                             ; ALLOW AN INTERRUPT TO OCCUR
k1c:    cli                             ; INTERRUPTS BACK OFF
         mov     bx, word ptr [BUFFER_HEAD] ; GET POINTER TO HEAD OF BUFFER
         cmp     bx, word ptr [BUFFER_TAIL] ; TEST END OF BUFFER
k1d:            ; 30/06/2014 (original code again)
        push    bx                      ; SAVE ADDRESS
        pushf                           ; SAVE FLAGS
        call    make_led                ; GO GET MODE INDICATOR DATA BYTE
        mov     bl, byte ptr [KB_FLAG_2]  ; GET PREVIOUS BITS
        xor     bl, al                  ; SEE IF ANY DIFFERENT
        and     bl, KB_LEDS             ; ISOLATE INDICATOR BITS
        jz      short k1a               ; IF NO CHANGE BYPASS UPDATE
        call    snd_led1
        cli
k1a:
        popf                            ; RESTORE FLAGS
        pop     bx                      ; RESTORE ADDRESS
        jz      short k1                ; LOOP UNTIL SOMETHING IN BUFFER
        ;
        mov     ax, word ptr [BX]      ; GET SCAN CODE AND ASCII CODE
        call    k4                     ; MOVE POINTER TO NEXT POSITION
        ; 03/03/2014
         mov     word ptr [BUFFER_HEAD], bx ; STORE VALUE IN VARIABLE
        pop     bx                     ; RECOVER REGISTER
        pop     ds                     ; RECOVER SEGMENT
        iret                           ; RETURN TO CALLER

        ;----- ASCII STATUS
k2:
        cli                            ; INTERRUPTS OFF
         mov     bx, word ptr [BUFFER_HEAD] ; GET HEAD POINTER
         cmp     bx, word ptr [BUFFER_TAIL] ; IF EQUAL (Z=1) THEN NOTHING THERE
        mov     ax, word ptr [BX]
        ; 30/06/2014 (original code again)
        pushf                          ; SAVE FLAGS
        push    ax                     ; SAVE CODE
        call    make_led               ; GO GET MODE INDICATOR DATA BYTE
        mov     bl, byte ptr [KB_FLAG_2] ; GET PREVIOUS BITS
        xor     bl, al                 ; SEE IF ANY DIFFERENT
        and     bl, KB_LEDS            ; ISOLATE INDICATOR BITS
        jz      short sk2              ; IF NO CHANGE BYPASS UPDATE
        ;
        call    snd_led1
sk2:
        pop     ax                     ; RESTORE CODE
        popf                           ; RESTORE FLAGS
        sti                            ; INTERRUPTS BACK ON
        pop     bx                     ; RECOVER REGISTER
         pop     ds                       ; RECOVER SEGMENT
        retf    2                      ; THROW AWAY FLAGS

        ;----- SHIFT STATUS
k3:
        mov     al, byte ptr [KB_FLAG] ; GET THE SHIFT STATUS FLAGS
        pop     bx                     ; RECOVER REGISTERS
        pop     ds
        iret                   ; RETURN TO CALLER

        ; 03/03/2014
        ;----- INCREMENT A BUFFER POINTER
k4:     inc     bx
        inc     bx                     ; MOVE TO NEXT WORD IN LIST
         cmp     bx, word ptr [BUFFER_END] ; AT END OF BUFFER?
        ;jne    short k5               ; NO, CONTINUE
        jb      short k5
         mov     bx, word ptr [BUFFER_START] ; YES, RESET TO BUFFER BEGINNING
k5:
        retn
```

```
int_09h:
        ; 07/03/2014
        ; 03/03/2014
        ; Derived from "KEYBOARD_INT_1" procedure of IBM "pc-at"
        ; rombios source code (06/10/1985)
        ;       'keybd.asm', INT 16H, KEYBOARD_IO
        ;
        ; 06/10/85  KEYBOARD BIOS
        ;
        ;--- HARDWARE INT 09 H - ( IRQ LEVEL 1 )---------------------------------------
        ;
        ;       KEYBOARD INTERRUPT ROUTINE
        ;
        ;------------------------------------------------------------------------------

        sti                             ; ENABLE INTERRUPTS
        push    bp
        push    ax
        push    bx
        push    cx
        push    dx
        push    si
        push    di
        push    ds
        push    es
        cld                             ; FORWARD DIRECTION
        ;call   dds                     ; SET UP ADDRESSING
        ;mov    ax, offset DDSData      ;
        mov     ax, cs
        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
        ;sub    cx, cx                  ; SET MAXIMUM TIMEOUT
        xor     cx, cx
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 INTERRUPT LEVEL 9HI
        ;mov    ah, 04Fh                ; SYSTEM INTERCEPT - KEY CODE FUNCTION
        ;stc                            ; SET CY= 1 (IN CASE OF IRET)
        ;int    15h                     ; CASSETTE CALL   (AL)= KEY SCAN CODE
                                        ; RETURNS CY= 1 FOR INVALID FUNCTION
        ;jc     short kb_int_02                 ; CONTINUE IF CARRY FLAG SET ((AL)=CODE)
        ;
        ;jmp    short k26               ; EXIT IF SYSTEM HANDLED SCAN CODE
                                        ; EXIT HANDLES HARDWARE EOI AND ENABLE
        ;jnc     k26

        ;
        ;----- 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_03            ; GO IF RESEND
        ;
        ;----- CHECK FOR RESPONSE TO A COMMAND TO KEYBOARD
        cmp     al, KB_ACK              ; IS THE INPUT AN ACKNOWLEDGE
         jne     short kb_int_04           ; GO IF NOT
        ;
        ;----- A COMMAND TO THE KEYBOARD WAS ISSUED
        cli                             ; DISABLE INTERRUPTS
        or      byte ptr [KB_FLAG_2], KB_FA ; INDICATE ACK RECEIVED
         jmp     k26                        ; RETURN IF NOT (ACK RETURNED FOR DATA)
        ;
        ;----- RESEND THE LAST BYTE
kb_int_03:
        cli                             ; DISABLE INTERRUPTS
        or      byte ptr [KB_FLAG_2], KB_FE ; INDICATE RESEND RECEIVED
         jmp     k26                       ; RETURN IF NOT ACK RETURNED FOR DATA)
        ;
```

```
kb_int_04:
        ;----- UPDATE MODE INDICATORS IF CHANGE IN STATE
        push    ax                      ; SAVE DATA IN
        call    make_led                ; GO GET MODE INDICATOR DATA BYTE
        mov     bl, byte ptr [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
        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
        ;jne    short k16               ; NO, TEST FOR SHIFT KEY
        ;jmp    short k62               ; BUFFER_FULL_BEEP
         je      k62
        ;
k16:
        and     al, 07Fh                ; REMOVE BREAK BIT
        ;push   cs
        ;pop    es                      ; ESTABLISH ADDRESS OF TABLES
        ;
        test    byte ptr [KB_FLAG_3], 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     ah, ID_1                ; IS THIS THE 1ST ID CHARACTER?
        jne     short rst_rd_id
        or      byte ptr [KB_FLAG_3], LC_AB ; INDICATE 1ST ID WAS OK
rst_rd_id:
        and     byte ptr [KB_FLAG_3], NOT RD_ID      ; RESET THE READ ID FLAG
        ;jmp    short do_ext
         jmp     k26
        ;
tst_id_2:
        and     byte ptr [KB_FLAG_3], NOT LC_AB      ; RESET FLAG
        cmp     ah, ID_2                ; IS THIS THE 2ND ID CHARACTER?
        ;jne    short do_ext            ; LEAVE IF NOT
         jne     k26
        ;
        ;----- A READ ID SAID THAT IT WAS KBX
        or      byte ptr [KB_FLAG_3], KBX ; INDICATE KBX WAS FOUND
        test    byte ptr [KB_FLAG_3], SET_NUM_LK ; SHOULD WE SET NUM LOCK?
        ;jz     short do_ext            ; EXIT IF NOT
         jz      k26
        or      byte ptr [KB_FLAG], NUM_STATE ; FORCE NUM LOCK ON
        call    snd_led                 ; GO SET THE NUM LOCK INDICATOR
        ;jmp    short exit
         jmp     k26
        ;
not_id:
        test    byte ptr [KB_FLAG_3], LC_HC ; WAS THE LAST CHARACTER A HIDDEN CODE
        jz      short not_lc_hc                 ; JUMP IF NOT
        ;
        ;----- THE LAST CHARACTER WAS A HIDDEN CODE
        and     byte ptr [KB_FLAG_3], NOT LC_HC      ; RESET LAST CHAR HIDDEN CODE FLAG
        cmp     al, INS_M               ; WAS IT THE INSERT KEY?
        je      short not_i
        test    ah, 80h                 ; IS THIS A BREAK CODE
        ;jnz    short exit              ; IGNORE BREAK ON REST OF THESE KEYS
         jnz     k26
not_i:
        mov     di, offset K_TAB1       ; TEST FOR ONE OF THE KEYPAD CURSOR FUNC
        mov     cx, L_TAB1
        repne   scasb                   ; SCAN FOR THE KEY
        jne     short not_cur           ; GO ON IF NOT FOUND
        test    byte ptr [KB_FLAG_1], HOLD_STATE ; ARE WE IN HOLD STATE?
        jz      short n_hld
        and     byte ptr [KB_FLAG_1], NOT HOLD_STATE ; EXIT HOLD STATE
;do_ext:
;       jmp     short exit              ; IGNORE THIS KEY
         jmp     k26
n_hld:
        test    byte ptr [KB_FLAG], ALT_SHIFT ; IS ALT DOWN?
        jz      short not_alt
        test    byte ptr [KB_FLAG], CTL_SHIFT ; HOW ABOUT CTRL?
        ;jz     short exit              ; IGNORE ALL IF ONLY ALT DOWN
         jz      k26
        cmp     al, DEL_M               ; WAS IT THE DELETE KEY'
        ;jne    short exit              ; IGNORE IF NOT
         jne     k26
         jmp     k29                        ; GO DO THE CTL, ALT, DEL RESET
        ;
not_alt:
        test    byte ptr [KB_FLAG], CTL_SHIFT ; IS CTL DOWN?
        jnz     short ctl_on            ; SPECIAL CASE IF SO
        cmp     al, INS_M               ; IS THIS THE INSERT KEY?
        ;jne    short n_ins
         jne     k49
        ;
        ;----- SPECIAL HANDLING FOR INSERT KEY
        mov     al, ah                  ; RECOVER SCAN CODE
        mov     ah, INS_SHIFT           ; AH = MASK FOR INSERT
        test    al, 80h                 ; WAS THIS A BREAK CODE?
        ;jnz    short b_c
         jnz     k24
         jmp     k22                        ; GO HANDLE INSERT SHIFT
;b_c:
;       jmp     short k24               ; HANDLE BREAK
;n_ins:
;       jmp     short k49               ; HANDLE & IGNORE NUMLOCK
ctl_on:
         cmp     cl, 5                      ; WAS IT INS, DEL, UP OR DOWN?
        ;ja     short exit              ; IGNORE IF DO
         ja      k26
         jmp     k42                        ; GO HANDLE CTRL CASE
        ;
not_lc_hc:                              ; LAST CHARACTER WAS NOT A HIDDEN CODE
        cmp     ah, HC                  ; IS THIS CHARACTER A HIDDEN CODE?
        jne     short not_cur
        or      byte ptr [KB_FLAG_3], LC_HC+KBX ; SET LAST CHAR WAS A HIDDEN CODE & KOX
;exit:
         jmp     k26                        ; THROW AWAY THIS CODE
        ;
not_cur:
        cmp     ah, F11_M               ; WAS IT F11?
        jne     short t_f12             ; HANDLE IF SO
        mov     cl, FUNC11              ; SET BASE FUNCTION 11
        cmp     ah, F11_B               ; IS THIS A BREAK CODE
        ;je     short exit              ; IGNORE SPEAK CODES
         je      k26
        cmp     ah, F12_B               ; IS THIS A BREAK CODE
        ;je     short exit              ; IGNORE BREAK CODES
         je      k26
        jmp     short do_fn
t_f12:
        cmp     ah, F12_M               ; WAS IT F12?
        jne     short t_sys_key                 ; GO TEST FOR SYSTEM KEY
        mov     cl, FUNC11+1            ; SET BASE FUNCTION 12
do_fn:
        test    byte ptr [KB_FLAG_1], HOLD_STATE ; ARE WE IN HOLD STATE?
        jz      short n_hld1
        and     byte ptr [KB_FLAG_1], NOT HOLD_STATE ; EXIT HOLD STATE
        ;jmp    short exit              ; IGNORE THIS KEY
         je      k26
n_hld1:
        mov     ah, cl
        ;
        test    byte ptr [KB_FLAG], ALT_SHIFT ; ARE WE IN ALT
        jz      short t_ctl
         add     ah, 6                       ; CNVT TO ALT FN 11-12
        jmp     short set_fn
t_ctl:
        test    byte ptr [KB_FLAG], CTL_SHIFT ; ARE WE IN CTRL
        jz      short t_shf
        add     ah, 4                   ; CNVT TO CTRL FN 11-12
        jmp     short set_fn
t_shf:
        test    byte ptr [KB_FLAG], LEFT_SHIFT+RIGHT_SHIFT ; IS EITHER SHIFT ON?
        jz      short set_fn
        add     ah, 2                   ; CNVT TO SHIFT FN 11-12
set_fn:
        sub     al, al                  ; FORCE PSEUDO SCAN CODE
         jmp     k61                        ; PUT IT INTO BUFFER
        ;
```

```
;----- 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 ptr [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 ptr [KB_FLAG_1], SYS_SHIFT      ; INDICATE SYSTEM KEY DEPRESSED
        mov     al, EOI                 ; END OF INTERRUPT COMMAND
        out     INTA00, al              ; SEND COMMAND TO INTERRUPT CONTROL PORT
                                        ; INTERRUPT-RETURN-NO-EOI
        mov     al, ENA_KBD             ; INSURE KEYBOARD 15 ENABLED
        call    ship_it                 ; EXECUTE ENABLE
        ;mov    ax, 8500h               ; FUNCTION VALUE FOR MAKE OF SYSTEM KEY
        ;sti                            ; MAKE SURE INTERRUPTS ENABLED
        ;int    15h                     ; USER INTERRUPT
         jmp     k27a                      ; END PROCESSING
;k16b:
;       jmp     short k26                       ; IGNORE SYSTEM KEY

k16c:
        and     byte ptr [KB_FLAG_1], NOT SYS_SHIFT ; TURN OFF SHIFT KEY HELD DOWN
        mov     al, EOI                 ; END OF INTERRUPT COMMAND
        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, 08501h              ; FUNCTION VALUE FOR BREAK OF SYSTEM KEY
        ;sti                            ; MAKE SURE INTERRUPTS ENABLED
        ;int    15h                     ; USER INTERRUPT
         jmp     k27a                      ; IGNORE SYSTEM KEY
k16a:
        mov     di, offset K6           ; SHIFT KEY TABLE
        mov     cx, K6L                 ; LENGTH
        repne   scasb                   ; LOOK THROUGH THE TABLE FOR A MATCH
        mov     al, ah                  ; RECOVER SCAN CODE
        ;je     short k17               ; JUMP IF MATCH FOUND
        ;jmp    short k25               ; IF NO MATCH, THEN SHIFT NOT FOUND
         jne     k25
        ;
        ;------SHIFT KEY FOUND
k17:
        sub     di, offset K6+1                 ; ADJUST PTR TO SCAN CODE MATCH
         add    di, offset K7
        mov      ah, byte ptr [DI]         ; GET MASK INTO AH
        test    al, 80h                 ; TEST FOR BREAK KEY
        ;jz     short k17c              ; BREAK_SHIFT_FOUND
        ;jmp    short k23               ; CONTINUE
        jnz     short k23
        ;
        ;----- 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      byte ptr [KB_FLAG], ah; TURN ON SHIFT BIT
         jmp     k26                       ; INTERRUPT_RETURN
        ;
        ;----- TOGGLED SHIFT KEY, TEST FOR 1ST MAKE OR NOT
k18:                                    ; SHIFT-TOGGLE
        test    byte ptr [KB_FLAG], CTL_SHIFT ; CHECK CTL SHIFT STATE
        jnz     short k25               ; JUMP IF CTL STATE
        ;
        cmp     al, INS_KEY             ; CHECK FOR INSERT KEY
        jnz     short k22               ; JUMP IF NOT INSERT KEY
        test    byte ptr [KB_FLAG], ALT_SHIFT ; CHECK FOR ALTERNATE SHIFT
        jnz     short k25               ; JUMP IF ALTERNATE SHIFT
        ;
        test    byte ptr [KB_FLAG], NUM_STATE ; CHECK FOR BASE STATE
        jnz     short k21               ; JUMP IF NUM LOCK IS ON
        test    byte ptr [KB_FLAG], LEFT_SHIFT+RIGHT_SHIFT
        jz      short k22               ; JUMP IF BASE STATE
        ;
```

```
k20:                                    ; NUMERIC ZERO, NOT INSERT KEY
        mov     ax, 5230h               ; PUT OUT AN ASCII ZERO
         jmp     k57                       ; BUFFER FILL
k21:                                    ; MIGHT BE NUMERIC
        test    byte ptr [KB_FLAG], LEFT_SHIFT+RIGHT_SHIFT
        jz      short k20               ; JUMP NUMERIC, NOT INSERT
        ;
k22:                                    ; SHIFT TOGGLE KEY HIT; PROCESS IT
        test    ah, byte ptr [KB_FLAG_1] ; IS KEY ALREADY DEPRESSED
        jz      short k22a0             ; GO IF NOT
        jmp     short k26               ; JUMP IF KEY ALREADY DEPRESSED
k22a0:
         or      byte ptr [KB_FLAG_1], ah ; INDICATE THAT THE KEY IS DEPRESSED
        xor     byte ptr [KB_FLAG], ah; TOGGLE THE SHIFT STATE
        ;
        ;----- TOGGLE LED IF CAPS OR NUM 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     short k26               ; JUMP IF NOT INSERT KEY
        mov     ax, INS_KEY*100h        ; SET SCAN CODE INTO AH, 0 INTO AL
         jmp     k57                       ; PUT INTO OUTPUT BUFFER
        ;
        ;----- BREAK SHIFT FOUND
k23:                                    ; BREAK-SHIFT-FOUND
        cmp     ah, SCROLL_SHIFT        ; IS THIS A TOGGLE KEY
        jae     short k24               ; YES, HANDLE BREAK TOGGLE
        not     ah                      ; INVERT MASK
        and     byte ptr [KB_FLAG], ah; TURN OFF SHIFT BIT
        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, byte ptr [ALT_INPUT]
        mov     ah, 0                   ; SCAN CODE OF 0
        mov     byte ptr [ALT_INPUT], ah ; ZERO OUT THE FIELD
        cmp     al, 0                   ; WAS THE INPUT=0
        je      short k26               ; INTERRUPT_RETURN
         jmp     k58                       ; IT WASN'T, SO PUT IN BUFFER
        ;
k24:                                    ; BREAK-TOGGLE
        not     ah                      ; INVERT MASK
        and     byte ptr [KB_FLAG_1], ah ; INDICATE NO LONGER DEPRESSED
        jmp     short k26               ; INTERRUPT_RETURN
        ;
        ;----- TEST FOR HOLD STATE
k25:                                    ; NO-SHIFT-FOUND
        cmp     al, 80h                 ; TEST FOR BREAK KEY
        jae     short k26               ; NOTHING FOR BREAK CHARS FROM HERE ON
        test    byte ptr [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 ptr [KB_FLAG_1], NOT HOLD_STATE ; TURN OFF THE HOLD STATE BIT
        ;
k26:                                    ; INTERRUPT-RETURN
        cli                             ; TURN OFF INTERRUPTS
        mov     al, EOI                 ; END OF INTERRUPT COMMAND
        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     di
        pop     si
        pop     dx
        pop     cx
        pop     bx
        pop     ax
        pop     bp
        iret                            ; RETURN, INTERRUPTS ON WITH FLAG CHANGE
        ;-----  NOT IN HOLD STATE
k28:                                ; NO-HOLD-STATE
        test    byte ptr [KB_FLAG], ALT_SHIFT ; ARE WE IN ALTERNATE SHIFT
        ;jnz    short k29           ; JUMP IF ALTERNATE SHIFT
        ;jmp    short k38           ; JUMP IF NOT ALTERNATE
         jz      short k38
        ;
        ;-----  TEST FOR CONTROL KEY AND RESET KEY SEQUENCE (CTL ALT DEL)
k29:                                ; TEST-RESET
        test    byte ptr [KB_FLAG], CTL_SHIFT ; ARE WE IN CONTROL SHIFT ALSO
        jz      short k31           ; NO RESET
        cmp     al, NUM_KEY         ; CHECK FOR INVALID NUM LOCK KEY
        je      short k26           ; THROW AWAY IF (ALT-CTL)+NUM-LOCK
        cmp     al, SCROLL_KEY      ; CHECK FOR INVALID SCROLL-LOCK KEY
        je      short k26           ; THROW AWAY IF (ALT-CTL)+SCROLL_LOCK
        cmp     al, DEL_KEY         ; CTL-ALT STATE, TEST FOR DELETE KEY
        jne     short k31           ; NO-RESET
        ;
        ;-----  CTL-ALT-DEL HAS BEEN FOUND
        ;;mov   byte ptr [RESET_FLAG], 1234h ; SET FLAG FOR RESET FUNCTION
        ;;jmp   short START_1       ; JUMP TO POWER ON DIAGNOSTICS
         mov     bx, BIOS_DSEGM
         mov     ds, bx
         mov     bx, RESET_FLAG
         mov     word ptr [BX], 1234h ; warm reset
        ; 07/03/2014
        jmp     cpu_reset
;cpu_reset:
        ; 07/03/2014
        ; CPU reset (power on) address
        ;db     0EAh  ; far jump  (jmp 0FFFFh:0000h)
        ;dw     0
        ;dw     0FFFFh ; F000:0FFF0h

;khere: hlt
;       jmp     short khere

        ;
        ;-----  IN ALTERNATE SHIFT, RESET NOT FOUND
k31:                                ; NO-RESET
        cmp     al, 57              ; TEST FOR SPACE KEY
        jne     short k32           ; NOT THERE
        mov     al, ' '             ; SET SPACE CHAR
         jmp     k57                   ; BUFFER_FILL
        ;
        ;-----  LOOK FOR KEY PAD ENTRY
k32:                                ; ALT-KEY-PAD
        mov     di, offset K30      ; ALT-INPUT-TABLE
        mov     cx, 10              ; LOOK FOR ENTRY USING KEYPAD
        repne   scasb               ; LOOK FOR MATCH
        jne     short k33           ; NO_ALT_KEYPAD
        sub     di, offset K30+1    ; DI-NOW-HAS ENTRY VALUE
        mov     al, byte ptr [ALT_INPUT] ; GET THE CURRENT BYTE
        mov     ah, 10              ; MULTIPLY BY 10
        mul     ah
        add     ax, di              ; ADD IN THE LATEST ENTRY
        mov     byte ptr [ALT_INPUT], al ; STORE IT AWAY
        jmp     short k26           ; THROW AWAY THAT KEYSTROKE
        ;
        ;-----  LOOK FOR SUPERSHIFT ENTRY
k33:                                ; NO-ALT-KEYPAD
        mov     byte ptr [ALT_INPUT], 0       ; ZERO ANY PREVIOUS ENTRY INTO INPUT
        mov     cx, 26              ; (DI),(ES) ALREADY POINTING
        repne   scasb               ; LOOK FOR MATCH IN ALPHABET
        jne     short k34           ; NOT FOUND, FUNCTION KEY OR OTHER
        mov     al, 0               ; ASCII CODE OF ZERO
         jmp     k57                   ; PUT IT IN THE BUFFER
        ;
        ;-----  LOOK FOR TOP ROW OF ALTERNATE SHIFT
k34:                                ; ALT-TOP-ROW
        cmp     al, 2               ; KEY WITH '1' ON IT
        je      short k35           ;  NOT ONE OF INTERESTING KEYS
        cmp     al, 14              ;  IS IT IN THE REGION
        jae     short k35           ;  ALT-FUNCTION
        add     ah, 118             ;  CONVERT PSEUDO SCAN CODE TO RANGE
        mov     al, 0               ;  INDICATE AS SUCH
       jmp      k57                  ;  BUFFER_FILL
        ;
        ;----- TRANSLATE ALTERNATE SHIFT PSEUDO SCAN CODES
k35:                                ; ALT-FUNCTION
        ; 59 =  scan code of F1 key
        cmp     al, 59              ; TEST FOR IN TABLE
        ;jae    short k37           ; ALT-CONTINUE
         jb      k26
;k36:                               ; CLOSE-RETURN
;       jmp      short k26          ; IGNORE THE KEY
k37:                                ; ALT-CONTINUE
        cmp     al, 71              ; IN KEYPAD REGION
        ;jae    short k36           ; IF SO, IGNORE
         jae     k26

        mov     bx, offset K13      ; ALT SHIFT PSEUDO SCAN TABLE
         jmp     k63                   ; TRANSLATE THAT
        ;
        ;----- NOT IN ALTERNATE SHIFT
k38:                                ; NOT-ALT-SHIFT
        test    byte ptr [KB_FLAG], CTL_SHIFT ; ARE WE IN CONTROL SHIFT
        jz      short k44           ; NOT-CTL-SHIFT
        ;
        ;----- CONTROL SHIFT, TEST SPECIAL CHARACTERS
        ;----- TEST FOR BREAK AND PAUSE KEYS
        cmp     al, SCROLL_KEY      ; TEST FOR BREAK
        jne     short k39           ; NO-BREAK
        mov     bx , word ptr [BUFFER_START] ; RESET BUFFER TO EMPTY
        mov     word ptr [BUFFER_HEAD], bx
        mov     word ptr [BUFFER_TAIL], bx
        mov     byte ptr [BIOS_BREAK], 80h ; TURN ON @BIOS_BREAK BIT
        ;
        ;----- ENABLE KEYBOARD
        mov     al, ENA_KBD         ; ENABLE KEYBOARD
        call    ship_it             ; EXECUTE ENABLE
        int     1Bh                 ; BREAK INTERRUPT VECTOR
        sub     ax, ax              ; PUT OUT DUMMY CHARACTER
         jmp     k57                   ; BUFFER_FILL
k39:                                ; NO_BREAK
        cmp     al, NUM_KEY         ; LOOK FOR PAUSE KEY
        jne     short k41           ; NO-PAUSE
        or      byte ptr [KB_FLAG_1], HOLD_STATE ; TURN ON THE HOLD FLAG
        ;
        ;----- ENABLE KEYBOARD
        mov     al, ENA_KBD         ; ENABLE KEYBOARD
        call    ship_it             ; EXECUTE ENABLE
        mov     al, EOI             ; END OF INTERRUPT TO CONTROL PORT
        out     INTA00, al          ; ALLOW FURTHER KEYSTROKE INTERRUPTS
        ;
        ;----- DURING PAUSE INTERVAL, TURN COLOR CRT BACK ON
         push    ds
         mov     bx, BIOS_DSEGM
         mov     ds, bx
         mov     bx, offset CRT_MODE
         cmp     byte ptr [BX], 7      ; IS THIS THE MONOCHROME CARD
         je      short k40p            ; YES, NOTHING TO DO
        mov     dx, 03D8h           ; PORT FOR COLOR CARD
         mov     al, byte ptr [CRT_MODE_SET] ; GET THE VALUE OF THE CURRENT MODE
        out     dx, al              ; SET THE CRT MODE, SO THAT CRT 15 ON
        ;
        ;----- SUSPEND SYSTEM OPERATION (LOOP) TILL NEXT KEY CLEARS HOLD STATE FLAG
k40p:
         pop     ds
k40:                                ; PAUSE-LOOP
        test    byte ptr [KB_FLAG_1], HOLD_STATE ; CHECK HOLD STATE FLAG
        jnz     short k40           ; LOOP UNTIL FLAG TURNED OFF
        ;
         jmp     k27a                  ; INTERRUPT_RETURN_NO_EOI
         ;
        ;----- TEST SPECIAL CASE KEY 55
k41:                                ; NO-PAUSE
        cmp     al, 55
        jne     short k42           ; NOT-KEY-55
        mov     ax, 114*100h        ; START/STOP PRINTING SWITCH
         jmp     k57                   ; BUFFER_FILL
        ;
        ;----- SET UP TO TRANSLATE CONTROL SHIFT
k42:                                ; NOT-KEY-55
        mov     bx, offset K8       ; SET UP TO TRANSLATE C7L
        cmp     al, 59              ; IS IT IN TABLE
        js      short k56           ; YES, GO TRANSLATE CHAR
```

```
                                        ; CTL-TABLE-TRANSLATE
        mov     bx, offset K9           ; CTL TABLE SCAN
        jmp     k63                     ; TRANSLATE_SCAN
        ;
        ;----- NOT IN CONTROL SHIFT
k44:                                    ; NOT-CTL-SHIFT
        cmp     al, 71                  ; TEST FOR KEYPAD REGION
        jae     short k48               ; HANDLE KEYPAD REGION
        test    byte ptr [KB_FLAG], LEFT_SHIFT+RIGHT_SHIFT
        jz      short k54               ; TEST FOR SHIFT STATE
        ;
        ;----- UPPER CASE, HANDLE SPECIAL CASES
        cmp     al, 15                  ; BACK TAB KEY
        jne     short k45               ; NOT-BACK-TAB
        mov     ax, 15*100h             ; SET PSEUDO SCAN CODE
        jmp     short k57               ; BUFFER_FILL
        ;
k45:                                    ; NOT-BACK-TAB
        cmp     al, 55                  ; PRINT SCREEN KEY
        jne     short k46               ; NOT-PRINT-SCREEN
        ;
        ;----- ISSUE INTERRUPT TO INDICATE PRINT SCREEN FUNCTION
        mov     al, ENA_KBD             ; INSURE KEYBOARD IS ENABLED
        call    ship_it                 ; EXECUTE ENABLE
        mov     al, EOI                 ; END OF CURRENT INTERRUPT
        out     INTA00, al              ; SO FURTHER THINGS CAN HAPPEN
        ;push   bp                      ; SAVE POINTER
        ;int    05h                     ; ISSUE PRINT SCREEN INTERRUPT
        ;pop    bp                      ; RESTORE POINTER
         jmp     k27                       ; GO BACK WITHOUT EOI OCCURRING
        ;
k46:                                    ; NOT-PRINT-SCREEN
        cmp     al, 59                  ; FUNCTION KEYS
        js      short k47               ; NOT-UPPER-FUNCTION
        mov     bx, offset K12          ; UPPER CASE PSEUDO SCAN CODES
         jmp     k63                       ; TRANSLATE_SCAN
        ;
k47:                                    ; NOT-UPPER-FUNCTION
        mov     bx, offset K11          ; POINT TO UPPER CASE TABLE
        jmp     short k56               ; OK, TRANSLATE THE CHAR
        ;
        ;----- KEYPAD KEYS, MUST TEST NUM LOCK FOR DETERMINATION
k48:                                    ; KEYPAD-REGION
        test byte ptr [KB_FLAG], NUM_STATE ; ARE WE IN NUM LOCK
        jnz     short k52               ; TEST FOR SURE
        test    byte ptr [KB_FLAG], LEFT_SHIFT+RIGHT_SHIFT ; ARE WE IN SHIFT STATE
        jnz     short k53               ; IF SHIFTED, REALLY NUM STATE
        ;
        ;----- BASE CASE FOR KEYPAD
k49:                                    ; BASE-CASE
        cmp     al, 74                  ; SPECIAL CASE FOR A COUPLE OF KEYS
        je      short k50               ; MINUS
        cmp     al, 78
        je      short k51
        sub     al, 71                  ; CONVERT ORIGIN
        mov     bx, offset K15          ; BASE CASE TABLE
         jmp     k64                       ; CONVERT TO PSEUDO SCAN
k50:
        mov     ax, (74*100h)+'-'       ; MINUS
        jmp     short k57               ; BUFFER_FILL
k51:
        mov     ax, (78*100h)+'+'               ; PLUS
        jmp     short k57               ; BUFFER_FILL
        ;
        ;----- MIGHT BE NUM LOCK, TEST SHIFT STATUS
k52:                                    ; ALMOST-NUM-STATE
        test    byte ptr [KB_FLAG], LEFT_SHIFT+RIGHT_SHIFT
        jnz     short k49               ; SHIFTED TEMP OUT OF NUM STATE
k53:                                    ; REALLY NUM STATE
        sub     al, 70                  ; CONVERT ORIGIN
        mov     bx, offset K14          ; NUM STATE TABLE
        jmp     short k56               ; TRANSLATE_CHAR
        ;
        ;----- PLAIN OLD LOWER CASE
k54:                                    ; NOT-SHIFT
        cmp     al, 59                  ; TEST FOR FUNCTION KEYS
        jb      short k55               ; NOT-LOWER-FUNCTION
        mov     al, 0                   ; SCAN CODE IN AH ALREADY
        jmp     short k57               ; BUFFER_FILL
```

```
k55:                                    ; NOT-LOWER-FUNCTION
        mov     bx, offset K10          ; LC TABLE
        ;
        ;----- TRANSLATE THE CHARACTER
k56:                                    ; TRANSLATE-CHAR
        dec     al                      ; CONVERT ORIGIN
        xlat                            ; CONVERT THE SCAN CODE TO ASCII
        ;
        ;----- 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
        cmp     ah, -1                  ; LOOK FOR -1 PSEUDO SCAN
        ;je     short k59               ; NEAR_INTERRUPT_RETURN
         je      k26
        ;
;       ; 07/03/2014
;; DELETE key handling (ASCII = 127)
;; (This code part was not in original INT 09h handler)
;; AX =  53E0h => AX = 007Fh <= AX = 5300h
;       cmp     ah, DEL_KEY
;       jne     short k58
;       cmp     al, 0E0h
;       je      short @f
;       and     al, al
;       jnz     short k58
;@@:
;       mov     ax, 127
;       jmp     short k61
        ;
        ;
        ;----- HANDLE THE CAPS LOCK PROBLEM
k58:                                    ; BUFFER_FILL-NOTEST
        test    byte ptr [KB_FLAG], CAPS_STATE ; ARE WE IN CAPS LOCK STATE
        jz      short k61               ; SKIP IF NOT
        ;
        ;----- IN CAPS LOCK STATE
        test    byte ptr [KB_FLAG], LEFT_SHIFT+RIGHT_SHIFT ; TEST FOR SHIFT STATE
        jz      short k60               ; IF NOT SHIFT, CONVERT LOWER TO UPPER
        ;
        ;----- CONVERT ANY UPPER CASE TO LOWER CASE
        cmp     al, 'A'                 ; FIND OUT IF ALPHABETIC
        jb      short k61               ; NOT-CAPS-STATE
        cmp     al, 'Z'
        ja      short k61               ; NOT_CAPS STATE
        add     al, 'a'-'A'             ; CONVERT TO LOWER CASE
        jmp     short k61               ; NOT_CAPS_STATE
        ;
;k59:                                   ; NEAR-INTERRUPT-RETURN
;       jmp     short k26               ; INTERRUPT_RETURN
        ;
        ;----- CONVERT ANY LOWER CASE TO UPPER CASE
k60:                                    ; LOWER-TO-UPPER
        cmp     al, 'a'                 ; FIND OUT IF ALPHABETIC
        jb      short k61               ; NOT_CAPS_STATE
        cmp     al, 'z'
        ja      short k61               ; NOT CAPS STATE
        sub     al, 'a'-'A'             ; CONVERT TO UPPER CASE
        ;
k61:                                    ; NOT-CAPS-STATE
        mov     bx, word ptr [BUFFER_TAIL] ; GET THE END POINTER TO THE BUFFER
        mov     si, bx                  ; SAVE THE VALUE
        call    k4                      ; ADVANCE THE TAIL
        cmp     bx, word ptr [BUFFER_HEAD] ; HAS THE BUFFER WRAPPED AROUND
        je      short k62               ; BUFFER_FULL_BEEP
        mov     word ptr [SI], ax       ; STORE THE VALUE
        mov     word ptr [BUFFER_TAIL], bx ; MOVE THE POINTER UP
        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, 09102h              ; MOVE IN POST CODE & TYPE
        ;int    15h                     ; PERFORM OTHER FUNCTION
         jmp     k27a                      ; INTERRUPT_RETURN
        ;
        ;----- TRANSLATE SCAN FOR PSEUDO SCAN CODES
k63:                                ; TRANSLATE-SCAN
        sub     al, 59              ; CONVERT ORIGIN TO FUNCTION KEYS
k64:                                ; TRANSLATE-SCAN-ORGD
        xlat                         ; CTL TABLE SCAN
        mov     ah, al              ; PUT VALUE INTO AH
        mov     al, 0               ; ZERO ASCII CODE
        jmp     short k57           ; PUT IT INTO THE BUFFER
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

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    cx
        mov     bh, al              ; SAVE TRANSMITTED BYTE FOR RETRIES
        mov     bl, 3               ; LOAD RETRY COUNT SOOT
        cli                         ; DISABLE INTERRUPTS
        and     byte ptr [KB_FLAG_2], not (KB_FE+KB_FA) ; CLEAR ACK AND RESEND FLAGS
        ;
        ;----- WAIT FOR ANY PENDING COMMAND TO BE ACCEPTED
        sub     cx, cx              ; MAXIMUM WAIT COUNT
sd1:
        in      al, STATUS_PORT             ; READ KEYBOARD PROCESSOR STATUS PORT
        test    al, INPT_BUF_FULL   ; CHECK FOR ANY PENDING COMMAND
        loopnz  sd1                 ; 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+
        xor     cx, cx
sd3:
        test    byte ptr [KB_FLAG_2], KB_FE+KB_FA ; SEE IF EITHER BIT SET
        jnz     short sd7           ; IF SET, SOMETHING RECEIVED GO PROCESS
        ;
        loop    sd3                 ; OTHERWISE WAIT
sd5:
        dec     bl                  ; DECREMENT RETRY COUNT
        jnz     short sd1           ; RETRY TRANSMISSION
        ;
        or      byte ptr [KB_FLAG_2], KB_ERR ; TURN ON TRANSMIT ERROR FLAG
        jmp     short sd9           ; RETRIES EXHAUSTED FORGET TRANSMISSION
sd7:
        test    byte ptr [KB_FLAG_2], KB_FA  ; SEE IF THIS IS AN ACKNOWLEDGE
        jz      short sd5           ; IF NOT, GO RESEND
sd9:
        pop     cx                  ; RESTORE REGISTERS
        pop     bx
        pop     ax
        retn                        ; RETURN, GOOD TRANSMISSION

snd_led:
        ; ------------------------------------------------------------------------------
        ; SND_LED
        ; SND_LED1
        ;
        ;       THIS ROUTINES TURNS ON THE MODE INDICATORS.
        ;
        ;-------------------------------------------------------------------------------
        ;
        cli                         ; TURN OFF INTERRUPTS
        test    byte ptr [KB_FLAG_2], KB_PR_LED      ; CHECK FOR MODE INDICATOR UPDATE
        jnz short sl9               ; DON'T UPDATE AGAIN IF UPDATE UNDERWAY
        ;
        or      byte ptr [KB_FLAG_2], KB_PR_LED      ; TURN ON UPDATE IN PROCESS
        mov     al, EOI             ; END OF INTERRUPT COMMAND
        out     INTA00, al              ; SEND COMMAND TO INTERRUPT CONTROL PORT
        jmp     short sl3               ; GO SEND MODE INDICATOR COMMAND

snd_led1:
        cli                             ; TURN OFF INTERRUPTS
        test    byte ptr [KB_FLAG_2], KB_PR_LED ; CHECK FOR MODE INDICATOR UPDATE
        jnz     short sl9               ; DON'T UPDATE AGAIN IF UPDATE UNDERWAY
        ;
        or      byte ptr [KB_FLAG_2], KB_PR_LED      ; TURN ON UPDATE IN PROCESS
sl3:
        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 ptr [KB_FLAG_2], not KB_LEDS ; CLEAR MODE INDICATOR BITS
        or      byte ptr [KB_FLAG_2], al ; SAVE INDICATORS STATES FOR NEXT TIME
        test    byte ptr [KB_FLAG_2], KB_ERR ; TRANSMIT ERROR DETECTED
        jnz     short sl5               ; IF SO, BYPASS SECOND BYTE TRANSMISSION
        ;
        call    snd_data                ; SEND DATA TO KEYBOARD
        cli                             ; TURN OFF INTERRUPTS
        test    byte ptr [KB_FLAG_2], KB_ERR ; TRANSMIT ERROR DETECTED
        jz      short sl7               ; IF NOT, DON'T SEND AN ENABLE COMMAND
sl5:
        mov     al, KB_ENABLE           ; GET KEYBOARD CSA ENABLE COMMAND
        call    snd_data                ; SEND DATA TO KEYBOARD
        cli                             ; TURN OFF INTERRUPTS
sl7:
        and     byte ptr [KB_FLAG_2], not (KB_PR_LED+KB_ERR) ; TURN OFF MODE INDICATOR
sl9:                                    ; 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, byte ptr [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
        and     al, 07h                 ; MAKE SURE ONLY MODE BITS ON
        pop     cx
        retn                            ; RETURN TO CALLER

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
        sub     cx, cx                  ; CLEAR TIMEOUT COUNTER
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
```

```
;----- TABLE OF SHIFT KEYS AND MASK VALUES (EARLY PC)
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

;----- SHIFT_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

;----- SCAN CODE TABLES
K8:     db      27,-1,0,-1,-1,-1,30,-1,-1,-1,-1,31
        db      -1,127,-1,17,23,5,18,20,25,21,9,15
        db      16,27,29,10,-1,1,19,4,6,7,8,10
        db      11,12,-1,-1,-1,-1,28,26,24,3,22,2
        db      14,13,-1,-1,-1,-1,-1,-1,' ',-1

;----- CTL TABLE SCAN
K9:     db      94,95,96,97,98,99,100,101,102,103,-1,-1
        db      119,-1,132,-1,115,-1,116,-1,117,-1,118,-1
        db      -1

;----- LC TABLE
K10:    db      01Bh,'1234567890-=',08h,09h
        db      'qwertyuiop[]',0Dh,-1,'asdfghjkl;',027h
        db      60h,-1,5Ch,'zxcvbnm,./',-1,'*',-1,' '

;----- UC TABLE
K11:    db      27,'!@#$',37,05Eh,'&*()_+',08h,0
        db      'QWERTYUIOP{}',0Dh,-1,'ASDFGHJKL:"'
        db      07Eh,-1,'|ZXCVBNM<>?',-1,0,-1,' ',-1

;----- UC TABLE SCAN
K12:    db      84,85,86,87,88,89
        db      90,91,92,93

;----- ALT TABLE SCAN
K13:    db      104,105,106,107,108
        db      109,110,111,112,113

;----- NUM STATE TABLE
K14:    db      '789-456+1230.'

;----- BASE CASE TABLE
K15:    db      71,72,73,-1,75,-1
        db      77,-1,79,80,81,82,83

;----- TABLE OF KEYPAD CURSOR; CONTROL KEYS
K_TAB1:
        db      UP_M, DN_M, INS_M, DEL_M, LEFT_M, RIGHT_M
        db      PGUP_M, PGDN_M, HOME_M, END_M
L_TAB1  equ     $-K_TAB1

;----- ALT-INPUT-TABLE
K30:    db      82,79,80,81,75,76
        db      77,71,72,73             ; 10 NUMBERS 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

; $
```