```
; ============================================================================
;                  Litos - Keyboard driver
;; ============================================================================
%ifdef DEBUG
;%define DEBUG_SCAN               ; monitor keyboard scan code
%endif
                CODE_SECTION    32

; ------------- Character mapping code
; B5..B7: key category
;       0: control keys with ASCII control code: Esc, Tab
;       1: single characters: SpaceBar /*-+
;       2: alphabetic characters (with CapsLock): A..Z
;       3: numeric keys 0..9 (with NumLock): 0..9
;       4: double characters (with Shift): 0..9, `-=[]\;',./
;       5: control keys without ASCII code: Home, End...
;       6: modifiers and special keys: Shift, CapsLock, NumLock,...

KC_CTRL         EQU     0<<5    ; control keys with ASCII code (hardcoded)
KC_SING         EQU     1<<5    ; single characters
KC_ALPH         EQU     2<<5    ; alphabetic characters (+CapsLock) (hardcoded)
KC_NUM          EQU     3<<5    ; numeric keys (+NumLock)
KC_DBL          EQU     4<<5    ; double characters (+Shift)
KC_VIRT         EQU     5<<5    ; control keys without ASCII code
KC_MOD          EQU     6<<5    ; modifiers and special keys

KCM_NO          EQU     KC_VIRT ; unknown scan code

; ------------- Modifiers and special keys

KCM_LSHI EQU    0       ; Left Shift
KCM_RSHI EQU    1       ; Right Shift
KCM_LCTR EQU    2       ; Left Ctrl
KCM_RCTR EQU    3       ; Right Ctrl
KCM_LALT EQU    4       ; Left Alt
KCM_RALT EQU    5       ; Right Alt
KCM_NUML EQU    6       ; NumLock
KCM_CAPS EQU    7       ; CapsLock
KCM_SCRL EQU    8       ; ScrollLock
KCM_BACK EQU    9       ; \ | extended
KCM_RWIN EQU    10      ; Right Win
KCM_INSE EQU    11      ; Insert
KCM_PAUS EQU    12      ; Pause
KCM_0INS EQU    13      ; [0 Ins]

; ----------------------------------------------------------------------------
;           Lock/unlock 8042 keyboard controller
; ----------------------------------------------------------------------------
; ------------- Keyboard lock function

%ifdef  SMP
                LOCK_LockFnc KeybLock,KeyboardLock ; keyboard lock function
%endif

; ------------- Macro - call keyboard lock function
%macro          KEYBLOCK 0
%ifdef  SMP
                call    KeybLock ; call keyboard lock function
%endif
%endmacro

; ------------- Macro - keyboard unlock (it saves flags)
%macro          KEYBUNLOCK 0
%ifdef  SMP
                LOCK_Unlock KeyboardLock ; unlock keyboard
%endif
%endmacro
```

```
; ---------------------------------------------------------------------------
;                          Send byte to keyboard
; ---------------------------------------------------------------------------
; INPUT: AL = byte to send
; ---------------------------------------------------------------------------

KeybSendByte:    push    eax             ; push EAX
                 push    ecx             ; push ECX

; ------------- Wait for byte to accept (1.5 us one loop, aprox. 500 us total)

                 mov     ecx,350         ; ECX <- timeout counter (aprox.500 us)
KeybSendByte2:   in      al,64h          ; read keyboard controller status
                 test    al,B1           ; is input buffer full?
                 loopnz  KeybSendByte2   ; wait for freeing input buffer

; ------------- Pop registers

                 pop     ecx             ; pop ECX
                 pop     eax             ; pop EAX

; ------------- Send byte

                 out     60h,al          ; send byte to keyboard
                 ret

; ---------------------------------------------------------------------------
;                          Send data to keyboard
; ---------------------------------------------------------------------------
; INPUT: AL = byte to send
;                EBX = return address if response OK (KeybSendOK = completed)
; NOTES: It does not lock keyboard. Interupts must be disabled.
; ---------------------------------------------------------------------------

; ------------- Send data with completion address

KeybSendDataEnd:push     dword KeybSendOK ; push completion address into stack

; ------------- Send data with return address in [ESP]

KeybSendDataRet:pop      ebx             ; EBX <- return address

; ------------- Acknowledge routine

KeybSendData:    mov     [KeybSendACK],ebx ; acknowledge routine

; ------------- Initialize error service

                 mov     byte [KeybSendRepeat],5 ; error counter
                 mov     dword [KeybSendNACK],KeybSendDataN ; non acknowledge
                 call    KeybSendStartTO  ; start time-out timer

; ------------- Store last byte sent

                 mov     [KeybSendLast],al ; store last byte sent

; ------------- Send data to keyboard

                 jmp     short KeybSendByte ; send byte to keyboard

; ============= Stop sending data - error

KeybSendErr:     and     byte [KeybFlags+1],~(KEYB_PRESENT>>8) ; not present

; ============= Stop sending data - completed

KeybSendOK:      mov     dword [KeybSendNACK],KeybIntNoSend ; non-acknowledge
                 mov     dword [KeybSendACK],KeybIntNoSend ; acknowledge
```

```
                push    ebx             ; push EBX
                mov     ebx,KeybSendAlarm ; EBX <- alarm structure
                mov     dword [ebx+ALARM_Func],KeybSendNone ; callback
                call    AlarmStop       ; stop alarm timer
                pop     ebx             ; pop EBX
                and     byte [KeybFlags+1],~(KEYB_SENDING>>8) ; clear send flag
                ret

; ============= Non-acknowledge callback routine

KeybSendDataN:  dec     byte [KeybSendRepeat] ; error counter
                jz      KeybSendErr     ; it was last attempt
                push    ebx             ; push EBX
                mov     ebx,KeybSendAlarm ; EBX <- alarm structure
                call    AlarmStop       ; stop alarm timer
                pop     ebx             ; pop EBX

; ------------- Repeat last command

KeybSendDataRep:call    KeybSendStartTO ; start time-out timer
                push    eax             ; push EAX
                mov     al,[KeybSendLast] ; AL <- last byte sent
                call    KeybSendByte    ; re-send last byte
                pop     eax             ; pop EAX
                ret

; ============= Start time-out timer

KeybSendStartTO:push    eax             ; push EAX
                push    ebx             ; push EBX
                push    ecx             ; push ECX

                xor     eax,eax         ; EAX <- 0
                mov     al,200          ; EAX <- 200, time-out in [ms]
                mov     ebx,KeybSendAlarm ; EBX <- alarm structure
                xor     ecx,ecx         ; ECX <- 0, no repeat
                mov     dword [ebx+ALARM_Func],KeybSendDataTO ; callback
                call    AlarmSetInt     ; set alarm interval
                call    AlarmStart      ; start alarm timer

                pop     ecx             ; pop ECX
                pop     ebx             ; pop EBX
                pop     eax             ; pop EAX
KeybSendNone:   ret                     ; alarm callback - no function

; ============= Alarm callback - timeout

KeybSendDataTO: KEYBLOCK        ; lock keyboard
                dec     byte [KeybSendRepeat] ; error counter
                jz      KeybSendDataTO2 ; it was last attempt
                call    KeybSendDataRep ; repeat last command
                KEYBUNLOCK              ; unlock keyboard
                ret

KeybSendDataTO2:call    KeybSendErr     ; error
                KEYBUNLOCK              ; unlock keyboard
                ret

; ---------------------------------------------------------------------------
;                           Setup keyboard
; ---------------------------------------------------------------------------
; NOTES: This function only starts setup sequence or sets request.
;                It locks keyboard controller.
; ---------------------------------------------------------------------------

; ------------- Disable interrupts

KeybSetSetup:   pushf                   ; push flags
```

```
                cli                     ; disable interrupts

; ------------- Lock keyboard

                KEYBLOCK        ; lock keyboard

; ------------- Set setup request

                or      byte [KeybFlags+1],KEYB_SENDSET>>8 ; set setup request

; ------------- Test and set sending flag

                bts     dword [KeybFlags],KEYB_SEND_BIT ; test and set sending
                jc      KeybSetSetup2   ; data is sending

; ------------- Clear setup request flag

                and     byte [KeybFlags+1],~(KEYB_SENDSET>>8); clear setup flag

; ------------- Push registers

                push    eax             ; push EAX
                push    ebx             ; push EBX

; ------------- Send command

                mov     al,KEYCMD_ENABLE ; command
                mov     ebx,KeybSendOK  ; continue address
                call    KeybSendData    ; send data to keyboard

; ------------- Pop registers

                pop     ebx             ; pop EBX
                pop     eax             ; pop EAX

; ------------- Unlock keyboard

KeybSetSetup2:  KEYBUNLOCK              ; unlock keyboard

; ------------- Enable interrupts

                popf                    ; pop flags
                ret

; ---------------------------------------------------------------------------
;                           Set LEDs
; ---------------------------------------------------------------------------
; NOTES: This function only starts LEDs sequence or sets LEDs request.
;                It locks keyboard controller.
; ---------------------------------------------------------------------------

; ------------- Disable interrupts

KeybSetLED:     pushf                   ; push flags
                cli                     ; disable interrupts

; ------------- Lock keyboard

                KEYBLOCK        ; lock keyboard

; ------------- Set LED request

                or      byte [KeybFlags+1],KEYB_SENDLED>>8 ; set LED request

; ------------- Test and set sending flag

                bts     dword [KeybFlags],KEYB_SEND_BIT ; test and set sending
                jc      KeybSetLED2     ; data is sending
```

```
; ------------- Clear LEDs request flag

                and     byte [KeybFlags+1],~(KEYB_SENDLED>>8) ; clear LED flag

; ------------- Push registers

                push    eax             ; push EAX
                push    ebx             ; push EBX

; ------------- Send command

                mov     al,KEYCMD_SETLED ; command
                mov     ebx,KeybSetLED4  ; continue address
                call    KeybSendData     ; send data to keyboard

; ------------- Pop registers

                pop     ebx             ; pop EBX
                pop     eax             ; pop EAX

; ------------- Unlock keyboard

KeybSetLED2:    KEYBUNLOCK              ; unlock keyboard

; ------------- Enable interrupts

                popf                    ; pop flags
                ret

; ------------- Send LEDs

KeybSetLED4:    mov     al,[KeybFlags]  ; AL <- keyboard flags
                and     al,B0+B1+B2     ; mask LED bits
KeybSetLED6:    call    KeybSendDataRet ; send data to keyboard

; Here it jumps after receiving ACK.

; ------------- Re-enable keyboard

                mov     al,KEYCMD_ENABLE ; command
                jmp     KeybSendDataEnd  ; re-enable keyboard

; ---------------------------------------------------------------------------
;            Set typematic rate
; ---------------------------------------------------------------------------
; NOTES: This function starts typematic rate sequence or sets request.
;                It locks keyboard controller.
; ---------------------------------------------------------------------------

; ------------- Disable interrupts

KeybSetRate:    pushf                   ; push flags
                cli                     ; disable interrupts

; ------------- Lock keyboard

                KEYBLOCK         ; lock keyboard

; ------------- Set rate request

                or      byte [KeybFlags+1],KEYB_SENDRATE>>8 ; set rate request

; ------------- Test and set sending flag

                bts     dword [KeybFlags],KEYB_SEND_BIT ; test and set sending
                jc      KeybSetRate2    ; data is sending
```

```
; ------------- Clear rate request flag

                and     byte [KeybFlags+1],~(KEYB_SENDRATE>>8); clear rate flag

; ------------- Push registers

                push    eax             ; push EAX
                push    ebx             ; push EBX

; ------------- Send command

                mov     al,KEYCMD_SETRATE ; command
                mov     ebx,KeybSetRate4 ; continue address
                call    KeybSendData    ; send data to keyboard

; ------------- Pop registers

                pop     ebx             ; pop EBX
                pop     eax             ; pop EAX

; ------------- Unlock keyboard

KeybSetRate2:   KEYBUNLOCK              ; unlock keyboard

; ------------- Enable interrupts

                popf                    ; pop flags
                ret

; ------------- Send typematic rate

KeybSetRate4:   mov     al,[KeybRate]   ; AL <- typematic rate
                jmp     short KeybSetLED6 ; send data to keyboard

; --------------------------------------------------------------------------
;               Debug: monitor scan code
; --------------------------------------------------------------------------
; INPUT: AL = scan code
; --------------------------------------------------------------------------

%ifdef DEBUG_SCAN

; ------------- Push registers

KeyIntDeb:      push    eax             ; push EAX
                push    ebx             ; push EBX
                push    ecx             ; push ECX
                push    edx             ; push EDX

; ------------- Prepare display descriptor

                mov     ebx,EGADrvDDPB ;CurDDPB   ; EBX <- display descriptor
;               or      byte [ebx+DDPB_Flags],DDPB_SINGLELF

; ------------- Change output position

                call    TXTGetPos       ; get output position
                push    ecx             ; push ECX (position)
                push    edx             ; push EDX (row)
                mov     ecx,[KeybDebOutPos] ; output position
                mov     edx,[KeybDebOutPos+4] ; output row
                call    TXTSetPos       ; set output position

; ------------- Display first HEX character

                push    eax             ; push EAX
                shr     al,4            ; AL <- high nibble
                add     al,"0"          ; AL <- convert high nibble to ASCII
```

```
                cmp     al,"9"          ; is it HEX character?
                jbe     KeybIntDeb2     ; it is not HEX character
                add     al,7            ; convert to HEX character 'A'..'F'
KeybIntDeb2:    call    TXTDispChar     ; display character
                pop     eax             ; pop EAX

; ------------- Display second HEX character

                push    eax             ; push EAX
                and     al,0fh          ; AL <- low nibble
                add     al,"0"          ; AL <- convert low nibble to ASCII
                cmp     al,"9"          ; is it HEX character?
                jbe     KeybIntDeb4     ; it is not HEX character
                add     al,7            ; convert to HEX character 'A'..'F'
KeybIntDeb4:    call    TXTDispChar     ; display character
                pop     eax             ; pop EAX

; ------------- Display space

                mov     ah,al           ; AH <- scan code
                mov     al," "          ; AL <- external separator
                cmp     ah,0e1h         ; prefix?
                je      KeybIntDel6     ; prefix
                cmp     ah,0e0h         ; prefix?
                je      KeybIntDel6     ; prefix
                or      ah,ah           ; key is up?
                js      KeybIntDel8     ; key is up
KeybIntDel6:    mov     al,"-"          ; AL <- internal separator
KeybIntDel8:    call    TXTDispCtrl     ; display separator

; ------------- Return output position

                call    TXTGetPos       ; get output position
                mov     [KeybDebOutPos],ecx ; output position
                mov     [KeybDebOutPos+4],edx ; output row
                pop     edx             ; pop EDX (row)
                pop     ecx             ; pop ECX (position)
                call    TXTSetPos       ; set output position

; ------------- Pop registers

                pop     edx             ; pop EDX
                pop     ecx             ; pop ECX
                pop     ebx             ; pop EBX
                pop     eax             ; pop EAX
                ret
%endif
; ---------------------------------------------------------------------------
;                Keyboard interrupt
; ---------------------------------------------------------------------------
; NOTES: Only CPU0 can accept keyboard interrupt to ensure right order of data.
; ---------------------------------------------------------------------------
; Registers:    AL =  scan code
;               AH =  controller status
;               EBX = key code
;               ECX = counter of inputs
;               EDX = console descriptor
;               EDI = keyboard flags

; ------------- Push registers
KeybInt: pusha                  ; push all registers
                push    ds              ; push DS
                push    es              ; push ES
; ------------- Initialize registers
                mov     eax,SYSTEM_DS   ; EAX <- system data segment
                mov     ds,eax          ; DS <- system data segment
                mov     es,eax          ; ES <- system data segment
                cld                     ; direction up
```

```
; ------------- Lock keyboard controller

                KEYBLOCK         ; lock keyboard controller

; ------------- Prepare pointer to keyboard flags (-> EDI)

                mov     edi,KeybFlags    ; EDI <- keyboard flags

; ------------- Check if controller has any data

                mov     ecx,100          ; ECX <- maximal number of loops
KeybInt2:       in      al,64h           ; read keyboard controller status
                test    al,B0            ; any data byte pending?
                jz      near KeybInt8    ; no data byte ready
                mov     ah,al            ; AH <- save controller status

; ------------- Read keyboard data (-> AL)

                in      al,60h           ; AL <- read data byte from keyboard

; ------------- DEBUG: Monitor scan code
%ifdef DEBUG_SCAN
                call    KeyIntDeb        ; monitor scan code
%endif
; ------------- Check data error

                test    ah,B6+B7 ; data error?
                jnz     short KeybInt33  ; data error, get next data byte

; ------------- Set keyboard present flag

                or      byte [edi+1],KEYB_PRESENT>>8 ; keyboard present

; ------------- Check if it is mouse data

                test    ah,B5            ; is it mouse data?
                jnz     near KeybInt5    ; it is mouse data

; ------------- Acknowledge (service can destroy EAX and EBX)

                cmp     al,KEYSTATE_ACK  ; command acknowledge?
                jne     short KeybInt22  ; not command acknowledge
                call    dword [KeybSendACK] ; ACK response
                jmp     short KeybInt33  ; next data byte

; ------------- Not acknowledge (service can destroy EAX and EBX)

KeybInt22:      cmp     al,KEYSTATE_NACK ; command non-acknowledge?
                jne     short KeybInt3   ; not command non-acknowledge
                call    dword [KeybSendNACK] ; NACK response
                jmp     short KeybInt33  ; next data byte

; ------------- Callback - not waiting for reply

KeybIntNoSend:  pop     ebx              ; destroy CALL return address

; ------------- Shift global random generator with an event

KeybInt3:       RNDSHIFTADD eax          ; shift global random generator

; ------------- Prefix 1

                cmp     al,KEYSTATE_PREF ; extended prefix?
                jne     short KeybInt32  ; not extended prefix
KeybInt31:      or      byte [edi+1],KEYB_EXT>>8 ; set extended prefix
                jmp     short KeybInt33  ; next data byte
```

```
; ------------- Prefix 2

KeybInt32:      cmp     al,KEYSTATE_PREF2 ; extended prefix2?
                jne     short KeybInt34  ; not extended prefix 2
                or      byte [edi+1],KEYB_EXT2>>8 ; set extended prefix 2
KeybInt33:      jmp     short KeybInt7   ; next data byte

; ------------- Prefix 2 service (ignore next scan code and set prefix 1)

KeybInt34:      btr     dword [edi],KEYB_EXT2_BIT ; extended prefix 2?
                jc      short KeybInt31  ; prefix 2, change to prefix 1

; ------------- Prepare key code (-> EBX)

                movzx   ebx,al           ; EBX <- scan code
                shl     bl,1             ; clear release key bit
                btr     dword [edi],KEYB_EXT_BIT ; test extended prefix
                rcr     bl,1             ; BL <- bit 7, extended prefix

; ------------- Get mapping code (-> ESI)

                movzx   esi,byte [KeybMapTab+ebx] ; ESI <- mapping code

; ------------- Set release flag and change pressed flag

                or      al,al            ; is key released?
                jns     short KeybInt36  ; key is not released
                btr     dword [KeybMap],ebx ; reset press flag
                or      bh,KEYCODE_UP    ; set released flag
                jmp     short KeybInt38

KeybInt36:      bts     dword [KeybMap],ebx ; set press flag

; ------------- Add modifiers

KeybInt38:      mov     al,[KeybFlags]   ; AL <- keyboard flags
                shr     al,4             ; AL <- shift modifiers
                and     al,B0+B1+B2      ; mask modifiers
                or      bh,al            ; add modifiers to the flags

; ------------- Shift key code to finish position (-> EBX)

                shl     ebx,16           ; shift scan code

; ------------- Jump to key service

                mov     eax,esi          ; EAX <- key mapping code
                shr     esi,5            ; ESI <- key category
                jmp     dword [KeybIntTab+esi*4] ; jump to service

; ------------- Mouse service

KeybInt5:       ; TODO

                jmp     short KeybInt7

; ------------- DEBUG: Monitor scan code

KeybIntSet:
%ifdef DEBUG_SCAN
                or      ebx,ebx          ; is it pressed key?
                js      KeybIntSet24     ; it is not press
                bt      ebx,VIRTKEY_CHAR_BIT ; is it valid ASCII?
                jnc     KeybIntSet24     ; it is not ASCII
                mov     al,bl            ; AL <- ASCII character
                cmp     al,CR            ; is it CR?
                jne     KeybIntSet22     ; it is not CR
                mov     al,LF            ; substitute CR with LF
```

```
KeybIntSet22:    push    ebx              ; push EBX
                 mov     ebx,EGADrvDDPB ;CurDDPB   ; EBX <- display descriptor
                 call    TXTDispCtrl      ; display character
                 pop     ebx              ; pop EBX
KeybIntSet24:
%endif

; ------------- Store key code
; TODO: national remap key code

                 xchg    eax,ebx          ; EAX <- key code
                 call    KeybWrite        ; write key code into buffer

; ------------- Read next data byte

KeybInt7:        dec     ecx              ; loop counter
                 jnz     near KeybInt2    ; read next data byte

; ------------- Unlock keyboard controller

KeybInt8:        KEYBUNLOCK               ; unlock keyboard controller

; ------------- Enable keyboard interrupt

                 IRQLOCK                  ; lock 8259A interrupt controller
                 in      al,21h           ; release interrupt controller 1
                 mov     al,60h+KEYB_IRQ ; AL <- keyboard IRQ
                 out     20h,al           ; acknowledge interrupt
                 IRQUNLOCK                ; unlock 8259A interrupt controller

; ------------- Send data to keyboard

                 test    byte [KeybFlags+1],KEYB_REQMASK>>8 ; any request?
                 jz      KeybInt9 ; no request
                 test    byte [KeybFlags+1],KEYB_SENDING>>8 ; sending data?
                 jnz     KeybInt9 ; sending data

; ------------- Send setup

                 test    byte [KeybFlags+1],KEYB_SENDSET>>8 ; send setup?
                 jz      KeybInt82        ; no setup
                 call    KeybSetSetup     ; setup keyboard
                 jmp     short KeybInt9

; ------------- Send LED state

KeybInt82:       test    byte [KeybFlags+1],KEYB_SENDLED>>8 ; send LED?
                 jz      KeybInt84        ; no LED
                 call    KeybSetLED       ; setup LED
                 jmp     short KeybInt9

; ------------- Send rate

KeybInt84:       test    byte [KeybFlags+1],KEYB_SENDRATE>>8 ; send rate?
                 jz      KeybInt9 ; no rate
                 call    KeybSetRate      ; setup rate

; ------------- Pop registers

KeybInt9:        pop     es               ; pop ES
                 pop     ds               ; pop DS
                 popa                     ; pop all registers
                 iret

; ============= Keyboard service - control keys with ASCII code

KeybIntCTRL:     mov     bl,al            ; BL <- virtual code
KeybIntSetChar:  bts     ebx,KEYCODE_CHAR_BIT+24 ; set character flag
```

```
                jmp     KeybIntSet ; store key code

; ============= Keyboard service - single characters

KeybIntSING:    and     al,1fh           ; EAX <- key
                mov     bl,[KeyMapTabSing+eax] ; BL <- character code
                jmp     short KeybIntSetChar ; store key code

; ============= Keyboard service - alphabetic characters

KeybIntALPH:    test    byte [edi],KEYB_CAPS ; is Caps Lock on?
                jnz     KeybIntALPH2     ; Caps Lock is on
                xor     al,20h           ; shift to lower letter
KeybIntALPH2:   test    byte [edi],KEYB_SHIFT ; is Left or Right Shift on?
                jz      KeybIntALPH4     ; Shift is not on
                xor     al,20h           ; shift letter
KeybIntALPH4:   test    byte [edi],KEYB_CTRL ; is Left or Right Ctrl on?
                jz      KeybIntALPH6     ; Ctrl is not on
                and     al,1fh           ; change to control character
KeybIntALPH6:   mov     bl,al            ; BL <- character code
                jmp     short KeybIntSetChar ; store key code

; ============= Keyboard service - numeric keys

KeybIntNUM:     and     al,1fh           ; EAX <- key subcode
                test    byte [edi],KEYB_NUM ; is NumLock on?
                jz      KeybIntNUM4      ; NumLock is not on
                test    byte [edi],KEYB_SHIFT|KEYB_CTRL ; modifier?
                jnz     KeybIntNUM6      ; change to control code
KeybIntNUM2:    mov     bl,[KeyMapTabNum1+eax] ; BL <- character code
                jmp     short KeybIntSetChar ; store key code

KeybIntNUM4:    test    byte [edi],KEYB_SHIFT|KEYB_CTRL ; modifier?
                jnz     KeybIntNUM2      ; change to character code
KeybIntNUM6:    mov     bl,[KeyMapTabNum2+eax] ; BL <- virtual code
                jmp     short KeybIntSetVirt ; store key code

; ============= Keyboard service - double characters

KeybIntDBL:     and     al,1fh           ; EAX <- key subcode
                mov     esi,KeyMapTabDbl2 ; table with Shift
                test    byte [edi],KEYB_SHIFT ; is Shift on?
                jnz     KeybIntDBL2      ; Shift is on
                mov     esi,KeyMapTabDbl1 ; table without Shift
KeybIntDBL2:    mov     bl,[esi+eax]     ; BL <- character code
                jmp     short KeybIntSetChar ; store key code

; ============= Keyboard service - key without ASCII code

KeybIntVIRT:    mov     eax,ebx          ; EAX <- key code
                shr     eax,16           ; AL <- scan code
                mov     bl,al            ; BL <- virtual key code
                jmp     short KeybIntSetVirt ; store key code

; ============= Keyboard service - modifiers

KeybIntMOD:     and     al,1fh           ; EAX <- key subcode
                jmp     dword [KeybIntModTab+eax*4] ; jump to service

; ============= Modifiers service - Left and Right Shift

KeybModRSHI:
KeybModLSHI:    mov     bl,KEY_SHIFT     ; BL <- virtual key code
                or      ebx,ebx          ; key is pressed?
                js      KeybModSHI2      ; key si not pressed
                bts     ebx,VIRTKEY_SHIFT_BIT ; set Shift flag
                or      byte [edi],KEYB_SHIFT ; set Shift flag
                jmp     short KeybIntSetVirt ; store key code
```

```
KeybModSHI2:     btr      ebx,VIRTKEY_SHIFT_BIT ; reset Shift flag
                 and      byte [edi],~KEYB_SHIFT ; clear Shift flag
                 jmp      short KeybIntSetVirt ; store key code

; ============= Modifiers service - Left and Right Ctrl

KeybModRCTR:
KeybModLCTR:     mov      bl,KEY_CTRL      ; BL <- virtual key code
                 or       ebx,ebx          ; key is pressed?
                 js       KeybModCTR2      ; key si not pressed
                 bts      ebx,VIRTKEY_CTRL_BIT ; set Ctrl flag
                 or       byte [edi],KEYB_CTRL ; set Ctrl flag
                 jmp      short KeybIntSetVirt ; store key code

KeybModCTR2:     btr      ebx,VIRTKEY_CTRL_BIT ; reset Ctrl flag
                 and      byte [edi],~KEYB_CTRL ; clear Ctrl flag
                 jmp      short KeybIntSetVirt ; store key code

; ============= Modifiers service - Left and Right Alt

KeybModRALT:
KeybModLALT:     mov      bl,KEY_ALT       ; BL <- virtual key code
                 or       ebx,ebx          ; key is pressed?
                 js       KeybModALT2      ; key si not pressed
                 bts      ebx,VIRTKEY_ALT_BIT ; set Alt flag
                 or       byte [edi],KEYB_ALT ; set Alt flag
                 jmp      short KeybIntSetVirt ; store key code

KeybModALT2:     btr      ebx,VIRTKEY_ALT_BIT ; reset Alt flag
                 and      byte [edi],~KEYB_ALT ; clear Alt flag
                 jmp      short KeybIntSetVirt ; store key code

; ============= Modifiers service - NumLock

KeybModNUML:     mov      bl,KEY_NUMLOCK   ; BL <- virtual key code
                 or       ebx,ebx          ; is key pressed?
                 js       KeybIntSetVirt   ; key is not pressed
                 xor      byte [edi],KEYB_NUM ; flip NumLock flag
KeybModNUML2:    or       byte [edi+1],KEYB_SENDLED>>8 ; set LEDs request
KeybIntSetVirt:  jmp      KeybIntSet       ; store key code

; ============= Modifiers service - CapsLock

KeybModCAPS:     mov      bl,KEY_CAPSLOCK  ; BL <- virtual key code
                 or       ebx,ebx          ; is key pressed?
                 js       KeybIntSetVirt   ; key is not pressed
                 xor      byte [edi],KEYB_CAPS ; flip CapsLock flag
                 jmp      short KeybModNUML2 ; store key code

; ============= Modifiers service - ScrollLock

KeybModSCRL:     bts      ebx,KEYCODE_CHAR_BIT+24 ; set character flag
                 mov      bl,KEY_SCROLL    ; BL <- virtual key code
                 or       ebx,ebx          ; is key pressed?
                 js       KeybIntSetVirt   ; key is not pressed
                 xor      byte [edi],KEYB_SCROLL ; flip ScrollLock flag
                 jmp      short KeybModNUML2 ; store key code

; ============= Modifiers service - \ | extended

KeybModBACK:     mov      bl,KEY_BACKSLASH ; BL <- virtual key code
                 jmp      KeybIntSetChar   ; store key code

; ============= Modifiers service - Right Win

KeybModRWIN:     bts      ebx,KEYCODE_CHAR_BIT+24 ; set character flag
                 mov      bl,KEY_WIN       ; BL <- virtual key code
```

```
                jmp     short KeybIntSetVirt ; store key code

; ============= Modifiers service - Pause

KeybModPAUS:    mov     bl,KEY_PAUSE     ; BL <- virtual key code
                or      ebx,ebx          ; is key pressed?
                js      KeybModPAUS2     ; key is not pressed
                xor     byte [edi],KEYB_PAUSE ; flip Pause flag
KeybModPAUS2:   jmp     short KeybIntSetVirt ; store key code

; ============= Modifiers service - [0 Ins]

KeybMod0INS:    test    byte [edi],KEYB_NUM ; is NumLock on?
                jz      KeybMod0INS4     ; NumLock is not on
                test    byte [edi],KEYB_SHIFT|KEYB_CTRL ; modifier?
                jnz     short KeybModINSE ; change to control code
KeybMod0INS2:   mov     bl,"0"           ; BL <- character code
                bts     ebx,KEYCODE_CHAR_BIT+24 ; set character flag
                jmp     KeybIntSetChar   ; store key code

KeybMod0INS4:   test    byte [edi],KEYB_SHIFT|KEYB_CTRL ; modifier?
                jnz     KeybMod0INS2     ; change to character code

; KeybModINSE must follow

; ============= Modifiers service - Insert

KeybModINSE:    mov     bl,KEY_INSERT    ; BL <- virtual key code
                or      ebx,ebx          ; is key pressed?
                js      KeybModINSE2     ; key is not pressed
                xor     byte [edi],KEYB_INSERT ; flip Insert flag
KeybModINSE2:   jmp     short KeybIntSetVirt ; store key code

; ---------------------------------------------------------------------------
;          Write key into all focused buffers
; ---------------------------------------------------------------------------
; INPUT: EAX = key code
; ---------------------------------------------------------------------------

KeybWrite:      push    ebx              ; push EBX
                push    ecx              ; push ECX
                push    edx              ; push EDX
                push    esi              ; push ESI

; ------------- Prepare mask of consoles for active consoles (-> EDX)

                mov     edx,[KeybConMask] ; EDX <- mask of consoles
                mov     ebx,edx          ; EBX <- push mask of consoles
                and     edx,[ConActMask] ; EDX <- active consoles
                jnz     KeybWrite2       ; found any valid console

; ------------- Prepare mask of consoles for open consoles (-> EDX)

                mov     edx,ebx          ; EDX <- mask of consoles
                and     edx,[ConOpenMask] ; EDX <- open consoles
                jnz     KeybWrite2       ; found any valid console

; ------------- Prepare mask of consoles for any console (-> EDX)

                mov     edx,ebx          ; EDX <- mask of any console

; ------------- Index of first console (-> EBX, ECX)

KeybWrite2:     bsf     ebx,edx          ; EBX <- index of first console
                jz      KeybWrite8       ; no console
                mov     cl,bl            ; CL <- index of first console
                shr     edx,cl           ; destroy unneeded bits
                mov     esi,[KeybBuffAddr+ebx*4-4] ; ESI <- console buffer-1
```

```
; ------------- Check if use this console

KeybWrite4:     add     esi,KEYBBUF_size ; ESI <- next keyboard buffer
                shr     edx,1            ; shift mask of consoles
                jnc     KeybWrite6       ; don't use this console

; ------------- Write to one console

                mov     ebx,[esi+KBUF_WKeyData] ; EBX <- private data
                call    dword [esi+KBUF_WKeyFnc] ; write to one console

; ------------- Next console

                or      edx,edx          ; any other console?
KeybWrite6:     jnz     KeybWrite4       ; next console

; ------------- Pop registers

KeybWrite8:     pop     esi              ; pop ESI
                pop     edx              ; pop EDX
                pop     ecx              ; pop ECX
                pop     ebx              ; pop EBX
                ret

; ---------------------------------------------------------------------------
;         Write key into keyboard buffer
; ---------------------------------------------------------------------------
; INPUT: EAX = key code
;                EBX = keyboard buffer
; ---------------------------------------------------------------------------

KeybWriteKey:   push    ecx              ; push ECX

; ------------- Disable interrupts

                pushf                    ; push flags
                cli                      ; disable interrupts

; ------------- Lock keyboard buffer (in EBX)
%ifdef SMP
                call    SpinLock ; lock keyboard buffer
%endif
; ------------- Write key code into buffer and shift write offset

                movzx   ecx,byte [ebx+KBUF_Write] ; ECX <- write offset
                mov     [ebx+KBUF_Buff+ecx],eax ; store key code into buffer
                add     cl,4             ; ECX <- next write offset
                mov     [ebx+KBUF_Write],cl ; store new write offset

; ------------- Shift key code read offset if buffer is full

                cmp     cl,[ebx+KBUF_ReadKey] ; is key code buffer full?
                jne     KeybWriteKey2    ; buffer is not full
                add     byte [ebx+KBUF_ReadKey],4 ; shift key code read offset

; ------------- Shift character read offset if buffer is full

KeybWriteKey2:  cmp     cl,[ebx+KBUF_ReadChar] ; is character buffer full?
                jne     KeybWriteKey4    ; buffer is not full
                add     byte [ebx+KBUF_ReadChar],4; shift character read offset

; ------------- Unlock keyboard buffer and enable interrupts

KeybWriteKey4:
%ifdef SMP
                LOCK_Unlock ebx          ; unlock keyboard buffer
%endif
```

```
                popf                    ; pop flags

; ------------ Pop registers

                pop     ecx             ; pop ECX
                ret

; ---------------------------------------------------------------------------
;               Read key from keyboard buffer
; ---------------------------------------------------------------------------
; INPUT: EBX = keyboard buffer
; OUTPUT:       EAX = key code (if NC, EAX = 0 on CY)
;               CY = no character ready (EAX = 0 on CY)
; ---------------------------------------------------------------------------

; ------------ Disable interrupts

KeybReadKey:    pushf                   ; push flags
                cli                     ; disable interrupts

; ------------ Lock keyboard buffer (in EBX)
%ifdef SMP
                call    SpinLock ; lock keyboard buffer
%endif
; ------------ Check if there is next key code

                movzx   eax,byte [ebx+KBUF_ReadKey] ; EAX <- read offset
                cmp     al,[ebx+KBUF_Write] ; any key code?
                je      KeybReadKey6    ; buffer is empty

; ------------ Get next key code (-> EAX)

                mov     eax,[ebx+KBUF_Buff+eax] ; EAX <- next key code
                add     byte [ebx+KBUF_ReadKey],4 ; shift read offset

; ------------ OK: Unlock keyboard buffer and enable interrupts
%ifdef SMP
                LOCK_Unlock ebx         ; unlock keyboard buffer
%endif
                popf                    ; pop flags
                clc                     ; clear error flag
                ret

; ------------ ERROR: Unlock keyboard buffer and enable interrupts

KeybReadKey6:
%ifdef SMP
                LOCK_Unlock ebx         ; unlock keyboard buffer
%endif
                popf                    ; pop flags
                xor     eax,eax         ; EAX <- 0, invalid key code
                stc                     ; set error flag
                ret

; ---------------------------------------------------------------------------
;               Read character from keyboard buffer
; ---------------------------------------------------------------------------
; INPUT: EBX = keyboard buffer
; OUTPUT:       AL = character (if NC, AL = 0 on CY)
;               CY = no character ready (AL = 0 on CY)
; ---------------------------------------------------------------------------

; ------------ Push registers

KeybReadChar:   push    ecx             ; push ECX

; ------------ Disable interrupts
```

```
                pushf                   ; push flags
                cli                     ; disable interrupts

; ------------- Lock keyboard buffer (in EBX)
%ifdef SMP
                call    SpinLock ; lock keyboard buffer
%endif
; ------------- Read character from cache (-> AL)

                mov     al,[ebx+KBUF_Cache] ; AL <- character from cache
                btr     dword [ebx+KBUF_Flags],KBUF_CACHE_BIT ; cached?
                jc      KeybReadChar6   ; character is cached

; ------------- Check if there is another key code

KeybReadChar2:  movzx   ecx,byte [ebx+KBUF_ReadChar] ; ECX <- read offset
                cmp     cl,[ebx+KBUF_Write] ; any key code?
                je      KeybReadChar8   ; buffer is empty

; ------------- Get next key code (-> ECX)

                mov     ecx,[ebx+KBUF_Buff+ecx] ; ECX <- next key code
                add     byte [ebx+KBUF_ReadChar],4 ; shift read offset

; ------------- Check if it is valid character

                or      ecx,ecx         ; is key pressed?
                js      KeybReadChar2   ; no, key is released
                bt      ecx,KEYCODE_CHAR_BIT+24 ; is it character key?
                jnc     KeybReadChar2   ; it is not character key

; ------------- Save second byte of character code

                bt      ecx,KEYCODE_CH16_BIT+24 ; is it 16-bit character?
                jnc     KeybReadChar4   ; it is not 16-bit character
                mov     [ebx+KBUF_Cache],ch ; save second byte of character
                or      byte [ebx+KBUF_Flags],KBUF_CACHE ; set cache bit

; ------------- Character is OK

KeybReadChar4:  mov     al,cl           ; AL <- read character

; ------------- OK: Unlock keyboard buffer and enable interrupts

KeybReadChar6:
%ifdef SMP
                LOCK_Unlock ebx         ; unlock keyboard buffer
%endif
                popf                    ; pop flags

; ------------- OK: pop registers

                clc                     ; clear error flag
                pop     ecx             ; pop ECX
                ret

; ------------- ERROR: Unlock keyboard buffer and enable interrupts

KeybReadChar8:
%ifdef SMP
                LOCK_Unlock ebx         ; unlock keyboard buffer
%endif
                popf                    ; pop flags

; ------------- ERROR: Pop registers

                mov     al,0            ; AL <- 0, invalid character
```

```
                stc                     ; set error flag
                pop     ecx             ; pop ECX
                ret

; ----------------------------------------------------------------------------
;         Multi-read characters from keyboard buffer
; ----------------------------------------------------------------------------
; INPUT: EAX = input buffer
;                EBX = keyboard buffer
;                ECX = buffer size
; OUTPUT:        ECX = number of read bytes (if NC, ECX = 0 on CY)
;                CY = no character ready (ECX = 0 on CY)
; ----------------------------------------------------------------------------

; ------------- Push registers

KeybMReadChar:  push    edx             ; push EDX
                push    esi             ; push ESI

; ------------- Disable interrupts

                pushf                   ; push flags
                cli                     ; disable interrupts

; ------------- Lock keyboard buffer (in EBX)
%ifdef SMP
                call    SpinLock ; lock keyboard buffer
%endif
; ------------- Read character from cache (-> DL)

                xor     esi,esi         ; ESI <- 0, character counter
KeybMReadChar2: mov     dl,[ebx+KBUF_Cache] ; AL <- character from cache
                btr     dword [ebx+KBUF_Flags],KBUF_CACHE_BIT ; cached?
                jc      KeybMReadChar6  ; character is cached

; ------------- Check if there is another key code

KeybMReadChar4: movzx   edx,byte [ebx+KBUF_ReadChar] ; EDX <- read offset
                cmp     dl,[ebx+KBUF_Write] ; any key code?
                je      KeybMReadChar8  ; buffer is empty

; ------------- Get next key code (-> EDX)

                mov     edx,[ebx+KBUF_Buff+edx] ; EDX <- next key code
                add     byte [ebx+KBUF_ReadChar],4 ; shift read offset

; ------------- Check if it is valid character

                or      edx,edx         ; is key pressed?
                js      KeybMReadChar4  ; no, key is released
                bt      edx,KEYCODE_CHAR_BIT+24 ; is it character key?
                jnc     KeybMReadChar4  ; it is not character key

; ------------- Save second byte of character code

                bt      edx,KEYCODE_CH16_BIT+24 ; is it 16-bit character?
                jnc     KeybMReadChar6  ; it is not 16-bit character
                mov     [ebx+KBUF_Cache],dh ; save second byte of character
                or      byte [ebx+KBUF_Flags],KBUF_CACHE ; set cache bit

; ------------- Store character into buffer

KeybMReadChar6: mov     [eax+esi],dl    ; store character into buffer
                inc     esi             ; increase counter of characters
                loop    KeybMReadChar2  ; next character

; ------------- Unlock keyboard buffer and enable interrupts
```

```
KeybMReadChar8:
%ifdef SMP
                LOCK_Unlock ebx          ; unlock keyboard buffer
%endif
                popf                     ; pop flags

; ------------- Check number of read characters

                mov     ecx,esi          ; ECX <- number of bytes
                cmp     ecx,byte 1       ; any data read?

; ------------- Pop registers

                pop     esi              ; pop ESI
                pop     edx              ; pop EDX
                ret

; ---------------------------------------------------------------------------
;               Initialise keyboard driver
; ---------------------------------------------------------------------------

; ------------- Install keyboard driver

KeybInit:       mov     ebx,KeybDPB      ; EBX <- driver parameter block
                call    DrvLstInsert     ; insert driver into list

; ------------- Prepare buffer for next console

                mov     ebx,KeybBuff     ; EBX <- first keyboard input buffer
                xor     ecx,ecx          ; ECX <- console index
                xor     edx,edx          ; EDX <- 0
                inc     edx              ; EDX <- 1, console mask
KeybInit2:
%ifdef SMP
                LOCK3_Init ebx           ; initialize buffer lock
%endif
                mov     [ebx+KBUF_WKeyData],ebx ; private data
                mov     dword [ebx+KBUF_WKeyFnc],KeybWriteKey ; write function
                mov     [ebx+KBUF_Index],cl ; store console index
                test    edx,[KeybConMask] ; install for this console?
                jz      KeybInit4        ; no

; ------------- Install keyboard console interface

                mov     eax,KeybReadChar ; EAX <- read function
                call    ConRegRead       ; install read
                mov     eax,KeybReadKey  ; EAX <- key-read function
                call    ConRegKRead      ; install key-read
                mov     eax,KeybMReadChar ; EAX <- multi-read function
                call    ConRegMRead      ; install multi-read

; ------------- Next keyboard buffer

KeybInit4:      add     ebx,KEYBBUF_size ; EBX <- next keyboard input buffer
                inc     ecx              ; ECX <- increase console index
                shl     edx,1            ; EDX <- next console mask
                jnz     KeybInit2        ; next console
                ret

; ---------------------------------------------------------------------------
;               Data
; ---------------------------------------------------------------------------

                DATA_SECTION

; ------------- Mask of consoles using this keyboard
```

```
                align   4, db 0
KeybConMask:    dd      -1              ; mask of consoles using this keyboard

; ------------- Keyboard driver

                align   8, db 0
KeybDPB: RBTREENODE             ; red-black tree node
                SPINLOCK        ; driver lock
                db      0, DRV_INP_KEYB  ; index, class and subclass
                db      DPB_STATIC,0     ; flags and class flags
                db      0,0,0,1          ; driver version
                dd      DrvVendorName    ; pointer to vendor name
                dd      KeybName ; pointer to driver name
                dd      KeybModel        ; pointer to model name
                dd      EmptySText       ; modul path
                dd      DrvStdFuncTab    ; pointer to function table
                LISTHEAD        ; resource list

KeybName:       STEXT   'Keyboard'
KeybModel:      STEXT   'Standard 101/102-key keyboard'

; ------------- IRQ handler for keyboard

                align   8, db 0
KeybIRQHandler: LISTHEAD        ; link to next IRQ handler
                dd      0               ; pointer to IRQ descriptor
                dw      IRQ_PRIVATE|IRQ_ACTIVE ; IRQ flags
                db      1               ; current IRQ number
                db      1               ; recomended best IRQ number
                dd      B1              ; mask of usable IRQs (1=enabled)
                dd      0               ; user data (NULL=disabled)
                dd      0               ; counter for slow interrupts
                dd      KeybInt         ; fast handler (NULL=none)
                dd      0               ; slow handler (NULL=none)
                dd      0               ; callback (NULL=none)

; ------------- 8042 keyboard controller lock
%ifdef SMP
                align   4, db 0
KeyboardLock:   SPINLOCK        ; 8042 keyboard controller lock
%endif

; ------------- DEBUG: Scan codes output position

%ifdef DEBUG_SCAN
                align   4, db 0
KeybDebOutPos:  dd      0,10
%endif

; ------------- Communication with keyboard

                align   8, db 0
KeybSendAlarm:  ALARMTIMER KeybSendNone,0 ; alarm structure for time-out
KeybSendACK:    dd      KeybIntNoSend   ; callback if ACK response
KeybSendNACK:   dd      KeybIntNoSend   ; callback if NACK response
KeybSendRepeat: db      1               ; repeat counter to send data
KeybSendLast:   db      0               ; last byte sent

; ------------- Keyboard driver flags

                align   4, db 0
KeybFlags:      dd      KEYB_DEFAULT    ; keyboard driver flags

; ------------- Keyboard typematic rate

KeybRate:       db      0 + (0<<5)      ; current keyboard typematic rate

; ------------- Keyboard character mapping table, no modifiers
```

```
KeybMapTab:     db      0       +KC_VIRT ; 00: no key
                db      ESC                      ; 01: Esc
                db      0       +KC_DBL          ; 02: 1 !
                db      1       +KC_DBL          ; 03: 2 @
                db      2       +KC_DBL          ; 04: 3 #
                db      3       +KC_DBL          ; 05: 4 $
                db      4       +KC_DBL          ; 06: 5 %
                db      5       +KC_DBL          ; 07: 6 ^
                db      6       +KC_DBL          ; 08: 7 &
                db      7       +KC_DBL          ; 09: 8 *
                db      8       +KC_DBL          ; 0A: 9 (
                db      9       +KC_DBL          ; 0B: 0 )
                db      10      +KC_DBL          ; 0C: - _
                db      11      +KC_DBL          ; 0D: = +
                db      BS                       ; 0E: Backspace
                db      TAB                      ; 0F: Tab
                db      'Q'                      ; 10: Q
                db      'W'                      ; 11: W
                db      'E'                      ; 12: E
                db      'R'                      ; 13: R
                db      'T'                      ; 14: T
                db      'Y'                      ; 15: Y
                db      'U'                      ; 16: U
                db      'I'                      ; 17: I
                db      'O'                      ; 18: O
                db      'P'                      ; 19: P
                db      12      +KC_DBL          ; 1A: [ {
                db      13      +KC_DBL          ; 1B: ] }
                db      CR                       ; 1C: Enter
                db      KCM_LCTR+KC_MOD          ; 1D: Left Ctrl
                db      'A'                      ; 1E: A
                db      'S'                      ; 1F: S
                db      'D'                      ; 20: D
                db      'F'                      ; 21: F
                db      'G'                      ; 22: G
                db      'H'                      ; 23: H
                db      'J'                      ; 24: J
                db      'K'                      ; 25: K
                db      'L'                      ; 26: L
                db      14      +KC_DBL          ; 27: ; :
                db      15      +KC_DBL          ; 28: ' "
                db      16      +KC_DBL          ; 29: ` ~
                db      KCM_LSHI+KC_MOD          ; 2A: Left Shift
                db      17      +KC_DBL          ; 2B: \ |
                db      'Z'                      ; 2C: Z
                db      'X'                      ; 2D: X
                db      'C'                      ; 2E: C
                db      'V'                      ; 2F: V
                db      'B'                      ; 30: B
                db      'N'                      ; 31: N
                db      'M'                      ; 32: M
                db      18      +KC_DBL          ; 33: , <
                db      19      +KC_DBL          ; 34: . >
                db      20      +KC_DBL          ; 35: / ?
                db      KCM_RSHI+KC_MOD          ; 36: Right Shift
                db      0       +KC_SING ; 37: Gray [*]
                db      KCM_LALT+KC_MOD          ; 38: Left Alt
                db      1       +KC_SING ; 39: SpaceBar
                db      KCM_CAPS+KC_MOD          ; 3A: CapsLock
                db      0       +KC_VIRT ; 3B: F1
                db      0       +KC_VIRT ; 3C: F2
                db      0       +KC_VIRT ; 3D: F3
                db      0       +KC_VIRT ; 3E: F4
                db      0       +KC_VIRT ; 3F: F5
                db      0       +KC_VIRT ; 40: F6
                db      0       +KC_VIRT ; 41: F7
                db      0       +KC_VIRT ; 42: F8
```

```
db      0       +KC_VIRT ; 43: F9
db      0       +KC_VIRT ; 44: F10
db      KCM_NUML+KC_MOD          ; 45: NumLock
db      KCM_SCRL+KC_MOD          ; 46: ScrollLock
db      0       +KC_NUM          ; 47: [7 Home]
db      1       +KC_NUM          ; 48: [8 Up]
db      2       +KC_NUM          ; 49: [9 PgUp]
db      2       +KC_SING ; 4A: Gray [-]
db      3       +KC_NUM          ; 4B: [4 Left]
db      4       +KC_NUM          ; 4C: [5]
db      5       +KC_NUM          ; 4D: [6 Right]
db      3       +KC_SING ; 4E: Gray [+]
db      6       +KC_NUM          ; 4F: [1 End]
db      7       +KC_NUM          ; 50: [2 Down]
db      8       +KC_NUM          ; 51: [3 PgDn]
db      KCM_0INS+KC_MOD          ; 52: [0 Ins]
db      9       +KC_NUM          ; 53: [. Del]
db      0       +KC_VIRT ; 54: SysRq
db      KCM_NO                   ; 55: ----
db      KCM_BACK+KC_MOD          ; 56: \ | alternative
db      0       +KC_VIRT ; 57: F11
db      0       +KC_VIRT ; 58: F12
times 5+B7-59h db  KCM_NO ; 59 to 04+B7: ----
db      0       +KC_VIRT ; 05+B7: Messenger
db      KCM_NO                   ; 06+B7: ----
db      0       +KC_VIRT ; 07+B7: Edit Redo
db      0       +KC_VIRT ; 08+B7: Edit Undo
db      0       +KC_VIRT ; 09+B7: Application Left
db      0       +KC_VIRT ; 0A+B7: Edit Paste
db      0       +KC_VIRT ; 0B+B7: Scroll Normal
times 10h-0Ch db  KCM_NO  ; 0C+B7 to 0F+B7: ----
db      0       +KC_VIRT ; 10+B7: Media Prev
db      0       +KC_VIRT ; 11+B7: Scroll Fast
db      0       +KC_VIRT ; 12+B7: Scroll Faster
db      0       +KC_VIRT ; 13+B7: Word
db      0       +KC_VIRT ; 14+B7: Excel
db      0       +KC_VIRT ; 15+B7: Calendar
db      0       +KC_VIRT ; 16+B7: Log Off
db      0       +KC_VIRT ; 17+B7: Edit Cut
db      0       +KC_VIRT ; 18+B7: Edit Copy
db      0       +KC_VIRT ; 19+B7: Media Next
times 2         db  KCM_NO       ; 1A+B7, 1B+B7: ----
db      CR                       ; 1C+B7: Gray [Enter]
db      KCM_RCTR+KC_MOD          ; 1D+B7: Right Ctrl
db      0       +KC_VIRT ; 1E+B7: Application Right
db      0       +KC_VIRT ; 1F+B7: Scroll Fastest
db      0       +KC_VIRT ; 20+B7: Volume Mute
db      0       +KC_VIRT ; 21+B7: Calculator
db      0       +KC_VIRT ; 22+B7: Media Play
db      0       +KC_VIRT ; 23+B7: Spell
db      0       +KC_VIRT ; 24+B7: Media Stop
times 2ah-25h    db  KCM_NO       ; 25+B7 to 29+B7: ----
db      KCM_LSHI+KC_MOD          ; 2A+B7: fake Left Shift
times 2eh-2bh    db  KCM_NO       ; 2B+B7 to 2D+B7: ----
db      0       +KC_VIRT ; 2E+B7: Volune-
db      KCM_NO                   ; 2F+B7: ----
db      0       +KC_VIRT ; 30+B7: Volune+
db      KCM_NO                   ; 31+B7: ----
db      0       +KC_VIRT ; 32+B7: WWW Home
times 2         db  KCM_NO       ; 33+B7, 34+B7: ----
db      4       +KC_SING ; 35+B7: Gray [/]
db      KCM_RSHI+KC_MOD          ; 36+B7: fake Right Shift
db      0       +KC_VIRT ; 37+B7: PrintScr
db      KCM_RALT+KC_MOD          ; 38+B7: Right Alt
times 2         db  KCM_NO       ; 39+B7, 3A+B7: ----
db      0       +KC_VIRT ; 3B+B7: Help
db      0       +KC_VIRT ; 3C+B7: My Music
db      0       +KC_VIRT ; 3D+B7: Task Pane
```

```
                db      0       +KC_VIRT ; 3E+B7: File New
                db      0       +KC_VIRT ; 3F+B7: File Open
                db      0       +KC_VIRT ; 40+B7: File Close
                db      0       +KC_VIRT ; 41+B7: Email Reply
                db      0       +KC_VIRT ; 42+B7: Email Forward
                db      0       +KC_VIRT ; 43+B7: Email Send
                db      KCM_NO                  ; 44+B7: ----
                db      KCM_PAUS+KC_MOD         ; 45+B7: Pause
                db      0       +KC_VIRT ; 46+B7: Break
                db      0       +KC_VIRT ; 47+B7: Home
                db      0       +KC_VIRT ; 48+B7: Up
                db      0       +KC_VIRT ; 49+B7: Page Up
                db      KCM_NO                  ; 4A+B7: ----
                db      0       +KC_VIRT ; 4B+B7: Left
                db      KCM_NO                  ; 4C+B7: ----
                db      0       +KC_VIRT ; 4D+B7: Right
                db      KCM_NO                  ; 4E+B7: ----
                db      0       +KC_VIRT ; 4F+B7: End
                db      0       +KC_VIRT ; 50+B7: Down
                db      0       +KC_VIRT ; 51+B7: Page Down
                db      KCM_INSE+KC_MOD         ; 52+B7: Insert
                db      0       +KC_VIRT ; 53+B7: Delete
                times 57h-54h   db  KCM_NO      ; 54+B7 to 56+B7: ----
                db      0       +KC_VIRT ; 57+B7: File Save
                db      0       +KC_VIRT ; 58+B7: File Print
                times 2         db  KCM_NO      ; 59+B7, 5A+B7: ----
                db      0       +KC_VIRT ; 5B+B7: Left Win
                db      KCM_RWIN+KC_MOD         ; 5C+B7: Right Win
                db      0       +KC_VIRT ; 5D+B7: Win Menu
                db      0       +KC_VIRT ; 5E+B7: Power
                db      0       +KC_VIRT ; 5F+B7: Sleep
                times 63h-60h   db  KCM_NO      ; 60+B7: to 62+B7: ----
                db      0       +KC_VIRT ; 63+B7: Wake Up
                db      0       +KC_VIRT ; 64+B7: My Picture
                db      0       +KC_VIRT ; 65+B7: File Search
                db      0       +KC_VIRT ; 66+B7: WWW Favorites
                db      0       +KC_VIRT ; 67+B7: WWW Refresh
                db      0       +KC_VIRT ; 68+B7: WWW Stop
                db      0       +KC_VIRT ; 69+B7: WWW Forward
                db      0       +KC_VIRT ; 6A+B7: WWW Back
                db      0       +KC_VIRT ; 6B+B7: My Computer
                db      0       +KC_VIRT ; 6C+B7: E-mail
                db      0       +KC_VIRT ; 6D+B7: Media Select
                times 255-6Eh   db  KCM_NO      ; 6E+B7: to FF+B7: ----

; ------------- Keyboard character map - single characters

KeyMapTabSing:  db      '* -+/'


; ------------- Keyboard character map - numeric keys

KeyMapTabNum1:  db      '789456123.'            ; with NumLock
KeyMapTabNum2:  db      KEY_HOME        ; without NumLock
                db      KEY_UP
                db      KEY_PAGEUP
                db      KEY_LEFT
                db      KEY_NONE
                db      KEY_RIGHT
                db      KEY_END
                db      KEY_DOWN
                db      KEY_PAGEDOWN
                db      KEY_DELETE

; ------------- Keyboard character map - double characters

KeyMapTabDbl1:  db      '1234567890-=[];',27h,'`\,./'     ; without Shift
KeyMapTabDbl2:  db      '!@#$%^&*()_+{}:',22h,'~|<>?'     ; with Shift
```

```
; ------------- Keyboard service jump table

                align   4, db 0
KeybIntTab:     dd      KeybIntCTRL             ; control keys with ASCII code
                dd      KeybIntSING             ; single characters
                dd      KeybIntALPH             ; alphabetic characters
                dd      KeybIntNUM              ; numeric keys
                dd      KeybIntDBL              ; double characters
                dd      KeybIntVIRT             ; key without ASCII code
                dd      KeybIntMOD              ; modifiers

; ------------- Modifiers jump table

                align   4, db 0
KeybIntModTab:  dd      KeybModLSHI             ; Left Shift
                dd      KeybModRSHI             ; Right Shift
                dd      KeybModLCTR             ; Left Ctrl
                dd      KeybModRCTR             ; Right Ctrl
                dd      KeybModLALT             ; Left Alt
                dd      KeybModRALT             ; Right Alt
                dd      KeybModNUML             ; NumLock
                dd      KeybModCAPS             ; CapsLock
                dd      KeybModSCRL             ; ScrollLock
                dd      KeybModBACK             ; \ | extended
                dd      KeybModRWIN             ; Right Win
                dd      KeybModINSE             ; Insert
                dd      KeybModPAUS             ; Pause
                dd      KeybMod0INS             ; [0 Ins]

; ------------- Pointers to keyboard input buffers (+ one invalid)

                align   4, db 0
                dd      KeybBuff - KEYBBUF_size   ; invalid address
KeybBuffAddr:
%assign  KBUFINX  0
%rep     CONSOLE_NUM
                dd      KeybBuff + KEYBBUF_size*KBUFINX
%assign  KBUFINX  KBUFINX+1
%endrep

; ---------------------------------------------------------------------------
;                                Uninitialized data
; ---------------------------------------------------------------------------

                BSS_SECTION

; ------------- Keyboard map of pressed key (1=key is pressed)

                align   4, resb 1
KeybMap: resb   256/8           ; keyboard map

; ------------- Keyboard input buffers

                align   4, resb 1


KeybBuff:       resb    KEYBBUF_size*CONSOLE_NUM ; keyboard input buffers
```