20.4 The Keyboard BIOS Interface

Although MS-DOS provides a reasonable set of routines to read ASCII and extended character codes from the keyboard, the PC's BIOS provides much better keyboard input facilities. Furthermore, there are lots of interesting keyboard related variables in the BIOS data area you can poke around at. In general, if you do not need the I/O redirection facilities provided by MS-DOS, reading your keyboard input using BIOS functions provides much more flexibility.

To call the MS-DOS BIOS keyboard services you use the int 16h instruction. The BIOS provides the following keyboard functions:

BIOS Keyboard Support Functions

Function # (AH)	**Input Parameters**	**Output Parameters**	**Description**
0	-	`al`- ASCII character `ah`- scan code	Read character. Reads next available character from the system's type ahead buffer. Wait for a keystroke if the buffer is empty.
1	-	ZF- Set if no key. ZF- Clear if key available. `al`- ASCII code `ah`- scan code	Checks to see if a character is available in the type ahead buffer. Sets the zero flag if not key is available, clears the zero flag if a key is available. If there is an available key, this function returns the ASCII and scan code value in `ax`. The value in `ax` is undefined if no key is available.
2	-	al- shift flags	Returns the current status of the shift flags in al. The shift flags are defined as follows: � bit 7: Insert toggle bit 6: Capslock toggle bit 5: Numlock toggle bit 4: Scroll lock toggle bit 3: Alt key is down bit 2: Ctrl key is down bit 1: Left shift key is down bit 0: Right shift key is down
3	`al` = 5 `bh` = 0, 1, 2, 3 for 1/4, 1/2, 3/4, or 1 second delay `bl`= 0..1Fh for 30/sec to 2/sec.	-	Set auto repeat rate. The `bh` register contains the amount of time to wait before starting the autorepeat operation, the `bl` register contains the autorepeat rate.
5	`ch` = scan code `cl` = ASCII code	-	Store keycode in buffer. This function stores the value in the `cx` register at the end of the type ahead buffer. Note that the scan code in `ch` doesn't have to correspond to the ASCII code appearing in `cl`. This routine will simply insert the data you provide into the system type ahead buffer.
10h	-	`al`- ASCII character`ah`- scan code	Read extended character. Like `ah`=0 call, except this one passes all key codes, the `ah`=0 call throws away codes that are not PC/XT compatible.
11h	-	ZF- Set if no key. ZF- Clear if key available.`al`- ASCII code `ah`- scan code	Like the ah=01h call except this one does not throw away keycodes that are not PC/XT compatible (i.e., the extra keys found on the 101 key keyboard).
12h	-	al- shift flags ah- extended shift flags	Returns the current status of the shift flags in ax. The shift flags are defined as follows: � bit 15: SysReq key pressed bit 14: Capslock key currently down bit 13: Numlock key currently down bit 12: Scroll lock key currently down bit 11: Right alt key is down bit 10:Right ctrl key is down bit 9: Left alt key is down bit 8: Left ctrl key is down bit 7: Insert toggle bit 6: Capslock toggle bit 5: Numlock toggle bit 4: Scroll lock toggle bit 3: Either alt key is down (some machines, left only) bit 2: Either ctrl key is down bit 1: Left shift key is down bit 0: Right shift key is down

Note that many of these functions are not supported in every BIOS that was ever written. In fact, only the first three functions were available in the original PC. However, since the AT came along, most BIOSes have supported at least the functions above. Many BIOS provide extra functions, and there are many TSR applications you can buy that extend this list even farther. The following assembly code demonstrates how to write an int 16h TSR that provides all the functions above. You can easily extend this if you desire.

```
; INT16.ASM
;
; A short passive TSR that replaces the BIOS' int 16h handler.
; This routine demonstrates the function of each of the int 16h
; functions that a standard BIOS would provide.
;
; Note that this code does not patch into int 2Fh (multiplex interrupt)
; nor can you remove this code from memory except by rebooting.
; If you want to be able to do these two things (as well as check for
; a previous installation), see the chapter on resident programs. Such
; code was omitted from this program because of length constraints.
;
;
; cseg and EndResident must occur before the standard library segments!

cseg            segment para public 'code'
cseg            ends

; Marker segment, to find the end of the resident section.

EndResident     segment para public 'Resident'
EndResident     ends

                .xlist
                include         stdlib.a
                includelib      stdlib.lib
                .list


byp             equ     <byte ptr>

cseg            segment para public 'code'
                assume  cs:cseg, ds:cseg

OldInt16        dword   ?


; BIOS variables:

KbdFlags1       equ     <ds:[17h]>
KbdFlags2       equ     <ds:[18h]>
AltKpd          equ     <ds:[19h]>
HeadPtr         equ     <ds:[1ah]>
TailPtr         equ     <ds:[1ch]>
Buffer          equ     1eh
EndBuf          equ     3eh

KbdFlags3       equ     <ds:[96h]>
KbdFlags4       equ     <ds:[97h]>

incptr          macro   which
                local   NoWrap
                add     bx, 2
                cmp     bx, EndBuf
                jb      NoWrap
                mov     bx, Buffer
NoWrap:         mov     which, bx
                endm


; MyInt16-              This routine processes the int 16h function requests.
```

```
;
;               AH      Description
;               --      ------------------------------------------------
;               00h     Get a key from the keyboard, return code in AX.
;               01h     Test for available key, ZF=1 if none, ZF=0 and
;                       AX contains next key code if key available.
;               02h     Get shift status. Returns shift key status in AL.
;               03h     Set Autorepeat rate. BH=0,1,2,3 (delay time in
;                       quarter seconds), BL=0..1Fh for 30 char/sec to
;                       2 char/sec repeat rate.
;               05h     Store scan code (in CX) in the type ahead buffer.
;               10h     Get a key (same as 00h in this implementation).
;               11h     Test for key (same as 01h).
;               12h     Get extended key status. Returns status in AX.


MyInt16         proc    far
                test    ah, 0EFh        ;Check for 0h and 10h
                je      GetKey
                cmp     ah, 2           ;Check for 01h and 02h
                jb      TestKey
                je      GetStatus
                cmp     ah, 3           ;Check for AutoRpt function.
                je      SetAutoRpt
                cmp     ah, 5           ;Check for StoreKey function.
                je      StoreKey
                cmp     ah, 11h         ;Extended test key opcode.
                je      TestKey
                cmp     ah, 12h         ;Extended status call
                je      ExtStatus

; Well, it's a function we don't know about, so just return to the caller.

                iret

; If the user specified ah=0 or ah=10h, come down here (we will not
; differentiate between extended and original PC getc calls).

GetKey:         mov     ah, 11h
                int     16h             ;See if key is available.
                je      GetKey          ;Wait for keystroke.

                push    ds
                push    bx
                mov     ax, 40h
                mov     ds, ax
                cli                     ;Critical region! Ints off.
                mov     bx, HeadPtr     ;Ptr to next character.
                mov     ax, [bx]        ;Get the character.
                incptr  HeadPtr         ;Bump up HeadPtr
                pop     bx
                pop     ds
                iret                    ;Restores interrupt flag.

; TestKey-       Checks to see if a key is available in the keyboard buffer.
;               We need to turn interrupts on here (so the kbd ISR can
;               place a character in the buffer if one is pending).
;               Generally, you would want to save the interrupt flag here.
;               But BIOS always forces interrupts on, so there may be some
;               programs out there that depend on this, so we won't "fix"
;               this problem.
```

```
;
;               Returns key status in ZF and AX. If ZF=1 then no key is
;               available and the value in AX is indeterminate. If ZF=0
;               then a key is available and AX contains the scan/ASCII
;               code of the next available key. This call does not remove
;               the next character from the input buffer.

TestKey:        sti                     ;Turn on the interrupts.
                push    ds
                push    bx
                mov     ax, 40h
                mov     ds, ax
                cli                     ;Critical region, ints off!
                mov     bx, HeadPtr
                mov     ax, [bx]        ;BIOS returns avail keycode.
                cmp     bx, TailPtr     ;ZF=1, if empty buffer
                pop     bx
                pop     ds
                sti                     ;Inst back on.
                retf    2               ;Pop flags (ZF is important!)


; The GetStatus call simply returns the KbdFlags1 variable in AL.

GetStatus:      push    ds
                mov     ax, 40h
                mov     ds, ax
                mov     al, KbdFlags1   ;Just return Std Status.
                pop     ds
                iret


; StoreKey-     Inserts the value in CX into the type ahead buffer.

StoreKey:       push    ds
                push    bx
                mov     ax, 40h
                mov     ds, ax
                cli                     ;Ints off, critical region.
                mov     bx, TailPtr     ;Address where we can put
                push    bx              ; next key code.
                mov     [bx], cx        ;Store the key code away.
                incptr  TailPtr         ;Move on to next entry in buf.
                cmp     bx, HeadPtr     ;Data overrun?
                jne     StoreOkay       ;If not, jump, if so
                pop     TailPtr         ; ignore key entry.
                sub     sp, 2           ;So stack matches alt path.
StoreOkay:      add     sp, 2           ;Remove junk data from stk.
                pop     bx
                pop     ds
                iret                    ;Restores interrupts.


; ExtStatus-    Retrieve the extended keyboard status and return it in
;               AH, also returns the standard keyboard status in AL.

ExtStatus:      push    ds
                mov     ax, 40h
                mov     ds, ax

                mov     ah, KbdFlags2
                and     ah, 7Fh         ;Clear final sysreq field.
                test    ah, 100b        ;Test cur sysreq bit.
                je      NoSysReq        ;Skip if it's zero.
                or      ah, 80h         ;Set final sysreq bit.
NoSysReq:
                and     ah, 0F0h        ;Clear alt/ctrl bits.
                mov     al, KbdFlags3
                and     al, 1100b       ;Grab rt alt/ctrl bits.
                or      ah, al          ;Merge into AH.
                mov     al, KbdFlags2
                and     al, 11b         ;Grab left alt/ctrl bits.
                or      ah, al          ;Merge into AH.

                mov     al, KbdFlags1   ;AL contains normal flags.
                pop     ds
                iret

; SetAutoRpt-   Sets the autorepeat rate. On entry, bh=0, 1, 2, or 3 (delay
;               in 1/4 sec before autorepeat starts) and bl=0..1Fh (repeat
;               rate, about 2:1 to 30:1 (chars:sec).

SetAutoRpt:     push    cx
                push    bx

                mov     al, 0ADh                ;Disable kbd for now.
                call    SetCmd

                and     bh, 11b                 ;Force into proper range.
                mov     cl, 5
                shl     bh, cl                  ;Move to final position.
                and     bl, 1Fh                 ;Force into proper range.
                or      bh, bl                  ;8042 command data byte.
                mov     al, 0F3h                ;8042 set repeat rate cmd.
                call    SendCmd                 ;Send the command to 8042.
                mov     al, bh                  ;Get parameter byte
                call    SendCmd                 ;Send parameter to the 8042.

                mov     al, 0AEh                ;Reenable keyboard.
                call    SetCmd
                mov     al, 0F4h                ;Restart kbd scanning.
                call    SendCmd

                pop     bx
                pop     cx
                iret

MyInt16         endp



; SetCmd-       Sends the command byte in the AL register to the 8042
;               keyboard microcontroller chip (command register at
;               port 64h).

SetCmd          proc    near
                push    cx
                push    ax                      ;Save command value.
                cli                             ;Critical region, no ints now.

; Wait until the 8042 is done processing the current command.
```

```
                xor     cx, cx                  ;Allow 65,536 times thru loop.
Wait4Empty:     in      al, 64h                 ;Read keyboard status register.
                test    al, 10b                 ;Input buffer full?
                loopnz  Wait4Empty              ;If so, wait until empty.

; Okay, send the command to the 8042:

                pop     ax              ;Retrieve command.
                out     64h, al
                sti                     ;Okay, ints can happen again.
                pop     cx
                ret
SetCmd          endp




; SendCmd-      The following routine sends a command or data byte to the
;               keyboard data port (port 60h).

SendCmd         proc    near
                push    ds
                push    bx
                push    cx
                mov     cx, 40h
                mov     ds, cx
                mov     bx, ax          ;Save data byte

                mov     bh, 3           ;Retry cnt.
RetryLp:        cli                     ;Disable ints while accessing HW.

; Clear the Error, Acknowledge received, and resend received flags
; in KbdFlags4

                and     byte ptr KbdFlags4, 4fh

; Wait until the 8042 is done processing the current command.

                xor     cx, cx          ;Allow 65,536 times thru loop.
Wait4Empty:     in      al, 64h         ;Read keyboard status register.
                test    al, 10b         ;Input buffer full?
                loopnz  Wait4Empty      ;If so, wait until empty.

; Okay, send the data to port 60h

                mov     al, bl
                out     60h, al
                sti                     ;Allow interrupts now.

; Wait for the arrival of an acknowledgement from the keyboard ISR:

                xor     cx, cx          ;Wait a long time, if need be.
Wait4Ack:       test    byp KbdFlags4, 10 ;Acknowledge received bit.
                jnz     GotAck
                loop    Wait4Ack
                dec     bh              ;Do a retry on this guy.
                jne RetryLp

; If the operation failed after 3 retries, set the error bit and quit.

                or      byp KbdFlags4, 80h ;Set error bit.
```

```
GotAck:         pop     cx
                pop     bx
                pop     ds
                ret
SendCmd         endp



Main            proc

                mov     ax, cseg
                mov     ds, ax

                print
                byte    "INT 16h Replacement",cr,lf
                byte    "Installing....",cr,lf,0

; Patch into the INT 9 and INT 16 interrupt vectors. Note that the
; statements above have made cseg the current data segment,
; so we can store the old INT 9 and INT 16 values directly into
; the OldInt9 and OldInt16 variables.

                cli                             ;Turn off interrupts!
                mov     ax, 0
                mov     es, ax
                mov     ax, es:[16h*4]
                mov     word ptr OldInt16, ax
                mov     ax, es:[16h*4 + 2]
                mov     word ptr OldInt16+2, ax
                mov     es:[16h*4], offset MyInt16
                mov     es:[16h*4+2], cs
                sti                             ;Okay, ints back on.


; We're hooked up, the only thing that remains is to terminate and
; stay resident.

                print
                byte    "Installed.",cr,lf,0

                mov     ah, 62h                 ;Get this program's PSP
                int     21h                     ; value.

                mov     dx, EndResident         ;Compute size of program.
                sub     dx, bx
                mov     ax, 3100h               ;DOS TSR command.
                int     21h
Main            endp
cseg            ends

sseg            segment para stack 'stack'
stk             db      1024 dup ("stack ")
sseg            ends

zzzzzzseg       segment para public 'zzzzzz'
LastBytes       db      16 dup (?)
zzzzzzseg       ends
                end     Main
```

20.3 The Keyboard DOS Interface

MS-DOS provides several calls to read characters from the keyboard. The primary thing to note about the DOS calls is that they only return a single byte. This means that you lose the scan code information the keyboard interrupt service routine saves in the type ahead buffer.

If you press a key that has an extended code rather than an ASCII code, MS-DOS returns two keycodes. On the first call MS-DOS returns a zero value. This tells you that you must call the get character routine again. The code MS-DOS returns on the second call is the extended key code.

Note that the Standard Library routines call MS-DOS to read characters from the keyboard. Therefore, the Standard Library `getc` routine also returns extended keycodes in this manner. The `gets` and `getsm` routines throw away any non-ASCII keystrokes since it would not be a good thing to insert zero bytes into the middle of a zero terminated string.