The PC Keyboard

El teclado de la PC es el dispositivo de entrada humana primario del sistema. Aunque parece más que mundano, el teclado es el dispositivo de entrada primario de la mayoría del software, así que aprender cómo programar el teclado apropiadamente es muy importante para los desarrolladores de aplicación.

IBM e innumerables fabricantes de teclados han producido numerosos teclados para PC y compatibles. La mayoría de los teclados modernos proporcionan al menos 101 teclas diferentes y son razonablemente compatibles con el teclado mejorado de 101 teclas de la IBM PC/AT (IBM PC/AT 101 key Enhanced Keyboard). Los teclados que proporcionan más teclas generalmente programan esas teclas para emitir una secuencia de otras pulsaciones (keystrokes) o permitir al usuario programar una secuencia de pulsaciones sobre las teclas extra. Dado que el teclado de 101 teclas es omnipresente, se supondrá que el teclado en uso es un teclado de 101 teclas.

Cuando IBM desarrolló la PC, usaron una interface muy simple entre el tecado y la computadora. Cuando IBM introdujo la PC/AT, rediseñaron completamente la interface del teclado. Desde la introducción de la PC/AT, casi todos los teclados han sido conformes al estándar PC/AT. Aun cuando IBM introdujo los sistemas PS/2, los cambios a la interface del teclado fueron menores y compatibles con el diseño PC/AT.

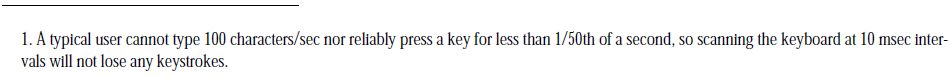
Hay cinco componentes principales para el teclado –información del teclado básica, la interface DOS, la interface BIOS, la rutina de servicio de interrupción del teclado int 9 y la interface hardware con el teclado. La última sección discutirá cómo hacer fake keyboard input en una aplicación.

Keyboard Basics

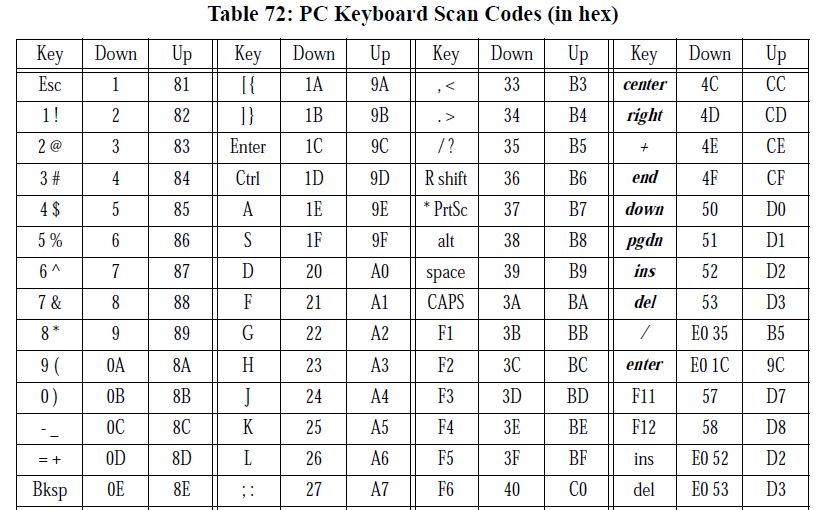
El teclado de una PC es un sistema de computadora en sí mismo. Incrustado dentro del teclado está un chip microcontrolador 8042 que constantemente escanea los switches sobre el teclado para ver si algunas teclas están pulsadas. Este procesamiento se hace en paralelo con las actividades normales de la PC, por lo tanto el teclado nunca pierde una pulsación porque el 80x86 esté ocupado.

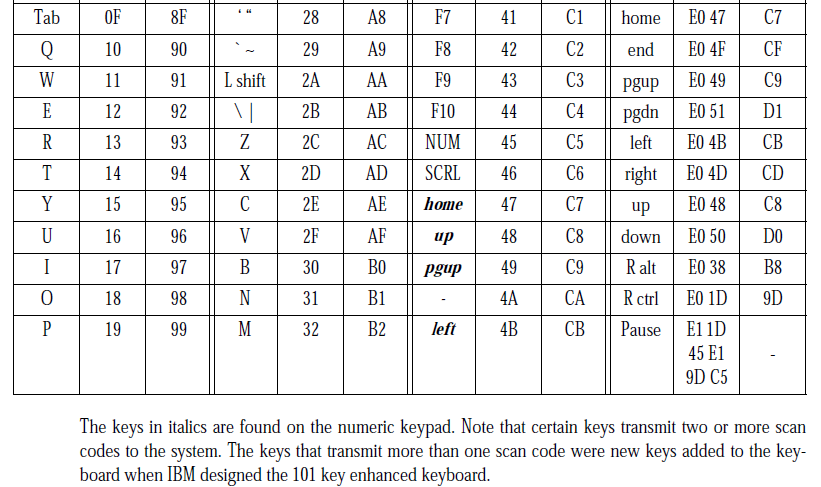
Una pulsación típica empieza con el usuario sobre el teclado. Esto cierra un contacto eléctrico en el switch así que el microcontrolador sensa que usted ha presionado el switch. Los switches (siendo estos, cosas mecánicas) no siempre cierran (hacer contacto) tan limpiamente. Frecuentemente, los contactos rebotan unas cuantas veces antes de descansar haciendo un contacto sólido. Si el chip microcontrolador lee el switch constantemente, estos contactos de rebote parecerán como una serie muy rápida de presionar y liberar una tecla. Esto podría generar múltiples a las computadoras principales, un fenómeno conocido como keybounce, común a muchos teclado baratos y antiguos. Pero aun en los teclados más caros y nuevos, el keybounce es un problema si usted mira el switch millones de veces por segundo; los switches mecánicos simplemente no pueden estabilizarse tan rápido. La mayoría de los algoritmos de escaneo de teclado, por lo tanto, controlan qué tan frecuentemente escanean el teclado. Una tecla no cara típica se estabilizará dentro de cinco milisegundos, así que si el software de escaneo de teclado solamente mira la tecla cada diez milisegundos, o algo así, el controlador efectivamente no verá el keybounce![%FontSize=14
%TeXFontSize=14
\documentclass{article}
\pagestyle{empty}
\begin{document}
\[
^{1}
\]
\end{document}](data:image/png;base64,iVBORw0KGgoAAAANSUhEUgAAABgAAAA5BAMAAADeqfZYAAAALVBMVEX///8AAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAADAOrOgAAAADnRSTlMAVHbdzUSrEGa7IpnvMqLTjT0AAAAJcEhZcwAADsQAAA7EAZUrDhsAAAAJcEhZcwAAXEYAAFxGARSUQ0EAAAB1SURBVCgVY2CAACEoDaKEnyA4wu8QHOF3CE7IOwQHxIbJgNlQTsgT1TC4TFIBA4MfXBnQAj1kTh8yR26UwzBywgAl6oGJ4jE88c179+4djMMOZL8DJiEw4ARxEkBM9qrocyDO462rChjugVgQ8IyB9y4c3AIArve/o+pt5YcAAAAASUVORK5CYII=).

Simplemente el notar que una tecla está presionada no es razón suficiente para generar un key code. Un usuario podría mantener una tecla presionada por varais decenas de milisegundos antes de liberarla. El controlador del teclado no debe generar una nueva secuencia de tecla cada vez que escanea el teclado y encuentra una tecla presionada. En lugar de ello, debe generar un solo valor key code cuando la tecla va de una up position a la down position (una operación down key). Cuando detecta una pulsación down key (down key stroke), el microcontrolador envía un scan code de teclado a la PC. El scan code no está relacionado al código ASCII para esa tecla, es un valor arbitrario que IBM escogió cuando por primera vez desarrollaron el teclado de la PC.



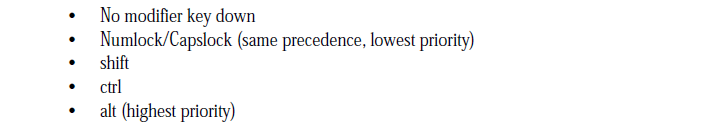
El teclado de la PC en realidad genera dos scan codes por cada tecla que usted presiona. Genera un **down code** cuando usted presiona la tecla y un **up code** cuando usted libera la tecla. El chip microcontrolador 8042 transmite esos scan codes a la PC donde son procesados por la rutina de servicio de interrupción de teclado. Tener códigos up y down separados es importante porque ciertas teclas (como shift, control, y alt) son significativas solamente cuando son mantenidas presionadas. Generando códigos up para todas las teclas, el teclado asegura que la rutina de servicio de interrupción sabe cuáles teclas son presionadas mientras el usuario mantiene presionadas una de esas teclas modificadoras (modifier keys). La tabla 72 lista los scan codes que el microcontrolador del teclado transmite a la PC.





Cuando el scan code llega a la PC, un segundo chip microcontrolador recibe el scan code, hace una conversión sobre el scan code%FontSize=14
%TeXFontSize=14
\documentclass{article}
\pagestyle{empty}
\begin{document}
\[
^{2}
\]
\end{document}, hace que el scan code esté disponible en el puerto I/O 60h, y entonces interrumpe a procesador y deja que la ISR del teclado obtenga e scan code del puerto I/O.

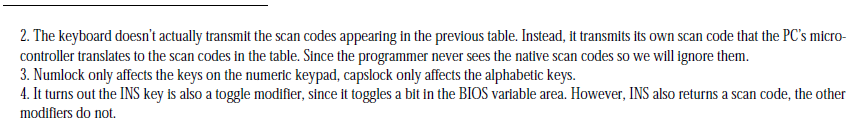
La rutina de servicio de interrupción (int 9) lee el scan code del puerto de entrada del teclado y procesa el scan code como lo considere apropiado. Note que el scan code que el sistema recibe del microcontrolador del teclado es un solo valor, aunque algunas teclas sobre el teclado representan hasta cuatro valores diferentes. Por ejemplo, la tecla “A” sobre el teclado puede producir A, a, ctrl-A, o alt-A. El código real que el sistema produce depende del estado actual de las teclas modificadoras (shift, ctrl, alt, capslock, y numlock). Por ejemplo, si un scan code de la tecla A llega (1Eh) y la tecla shift está presionada, el sistema produce el código ASCII para una A mayúscula. Si el usuario está presionando múltiples teclas modificadoras el sistema las prioriza de baja prioridad a alta prioridad como sigue:



Numlock y capslock afectan diferentes conjuntos de teclas%FontSize=14
%TeXFontSize=14
\documentclass{article}
\pagestyle{empty}
\begin{document}
\[
^{3}
\]
\end{document}, así que no hay ambigüedad a partir del hecho de que tienen la misma precedencia. Si el usuario está presionando dos teclas modificadoras al mismo tiempo, el sistema solamente reconoce la tecla modificadora con la prioridad más alta. Por ejemplo, si el usuario está presionando las teclas ctrl y alt al mismo tiempo, el sistema solo reconoce la tecla alt. Las teclas numlock, capslock, y shift son un caso especial. Si numlock o capslock está activa, presionar la tecla shift la hace inactiva. De la misma forma, si numlock o capslock está inactiva, presionar la tecla shift efectivamente “activa” esos modificadores.

No todos los modificadores son legales para cada tecla. Por ejemplo, ctrl-8 no es una combinación legal. La rutina de servicio de interrupción de teclado ignora todas las teclas presionadas combinadas con teclas modificadoras ilegales. Por alguna razón desconocida, IBM decidió hacer ciertas combinaciones de teclas legales y otras ilegales. Por ejemplo, ctrl-left y ctrl-right son legales, pero ctrl-up y ctrl-down no lo son. Se verá como corregir este problema un poco más tarde.

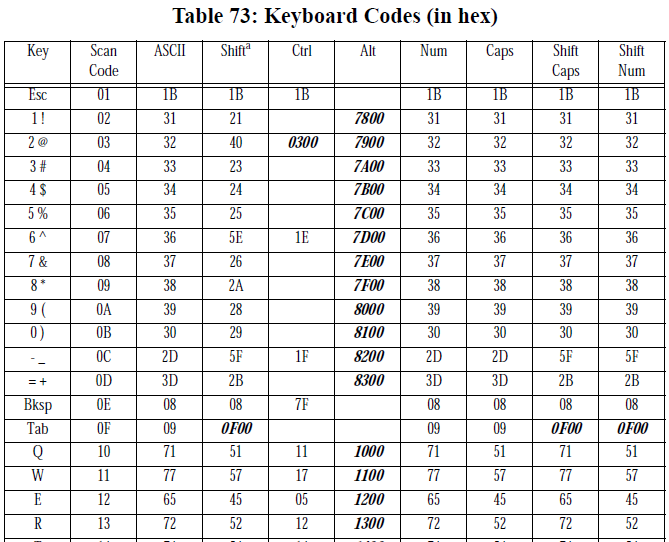
Las teclas shift, ctrl, y alt son modificadores activos. Esto es, la modificación a una tecla presionada ocurre solamente mientras el usuario mantiene presionada una de esas teclas modificadoras. La ISR del teclado mantiene la pista de si estas teclas están abajo o arriba estableciendo un bit asociado al recibir el down code y limpia ese bit al recibir el up code por shift, ctrl, o alt. En contraste, las teclas numlock, scroll lock, y capslock son modificadores toggle%FontSize=14
%TeXFontSize=14
\documentclass{article}
\pagestyle{empty}
\begin{document}
\[
^{4}
\]
\end{document}. La ISR del teclado invierte un bit asociado cada vez que ve un down code seguido de un up code para esas teclas.

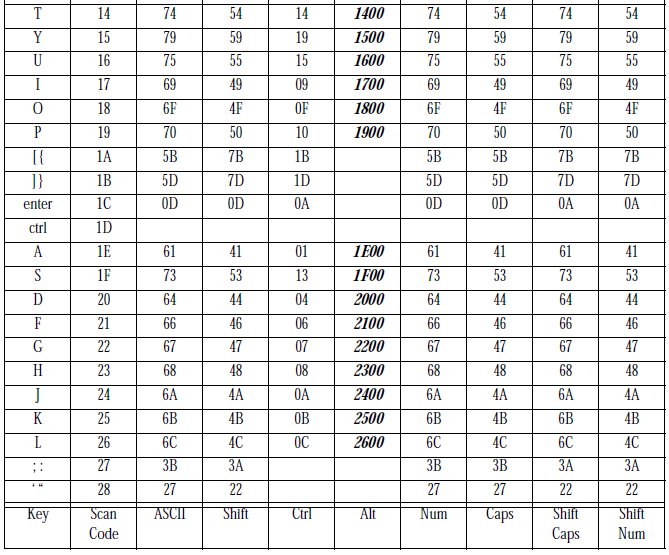


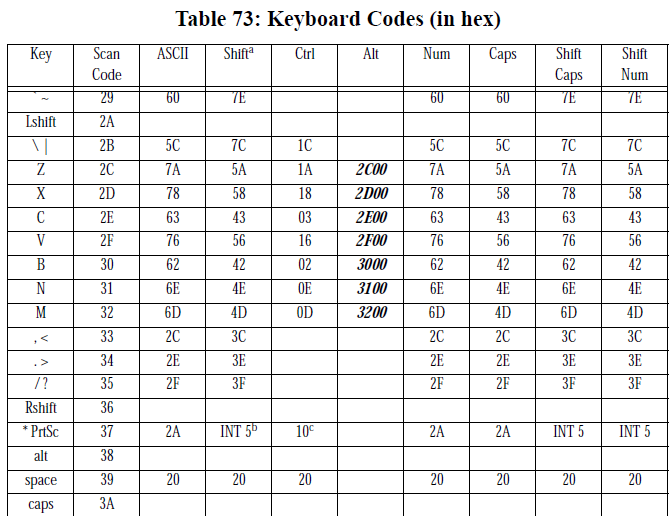
La mayoría de las teclas en el teclado de la PC corresponden a caracteres ASCII. Cuando la ISR del teclado encuentra tal carácter, lo traduce a un valor de 16 bits cuyo byte de L.O. es el código ASCII y el byte de H.O. es el scan code de la tecla. Por ejemplo, presionando la tecla “A” sin modificador, con shift, y con control produce 1E61h, 1E41h, y 1E01h, respectivamente (“a”, “A”, y ctrl-A). Muchas secuencias de teclas no tienen códigos ASCII correspondientes. Por ejemplo, las teclas función, las teclas de control del cursor, y las secuencias de teclas alt no tienen códigos ASCII correspondientes. Para esos códigos extendidos especiales, la ISR del teclado almacena un cero en el byte de L.O. (donde típicamente va el código ASCII) y el código extendido va en el byte de H.O. El código extendido es usualmente, aunque ciertamente no siempre, el scan code para esa tecla.

El único problema con este enfoque de código extendido es que el valor cero es un carácter ASCII legal (el carácter NUL). Por lo tanto, usted no puede introducir directamente caracteres NUL en una aplicación. Si una aplicación debe introducir caracteres NUL, IBM ha establecido aparte el código extendido 0300h (ctrl-3) para este propósito. Su aplicación debe convertir explícitamente este código extendido al carácter NUL (realmente, solamente necesita reconocer el valor de H.O. 03, dado que el byte de L.O. ya es el carácter NUL). Afortunadamente, muy pocos programas necesitan permitir la entrada del carácter NUL desde el teclado, así que este problema raramente es un tema.

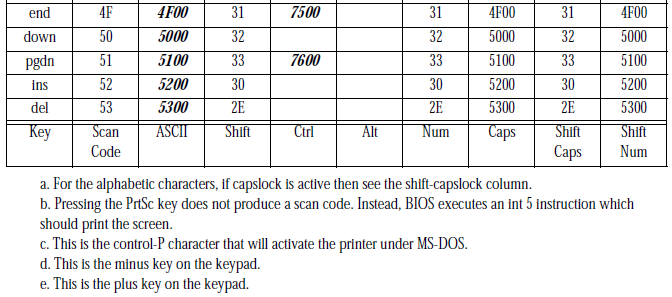
La siguiente tabla (Tabla 73) lista los códigos scan y extendidos que la ISR del teclado genera para las aplicaciones en respuesta a teclas presionadas con varios modificadores. Los códigos extendidos están en itálicas. Todos los otros valores (excepto la columna de los scan code) representa los ocho bits de L.O. del código de 16 bits. El byte de H.O. viene de la columna scan code.











Los teclados de 101 teclas generalmente proporcionan una tecla enter y una tecka “/” sobre el keypad numérico. A menos que usted escriba su propia ISR de teclado int 9, usted no será capaz de diferenciar esas teclas de las del teclado principal. El pad de control de cursor separado también genera los mismos códigos extendidos que el keypad numérico, excepto que nunca genera códigos ASCII numéricos. En cualquier otro caso, usted no puede diferenciar esas tecas de las teclas equivalentes sobre el keypad numérico (suponiendo que numlock está apagado, por supuesto).

La ISR del teclado proporciona una facilidad especial que permite introducir el código ASCII para una pulsación de tecla directamente desde el teclado. Para hacer esto, mantenga presionada la tecla alt y teclee el código ASCII decimal (0..255) para un carácter desde el teclado numérico (numeric keypad). La ISR del teclado convertirá esas pulsaciones de tecla a un valor de 8 bits, agregará en el byte de H.O. de cero al carácter, y usará eso como el código de carácter.

La ISR del teclado inserta el valor de 16 bits en el **type ahead buffer** de la PC. El type ahead buffer del sistema es una cola circular que usa las siguientes variables



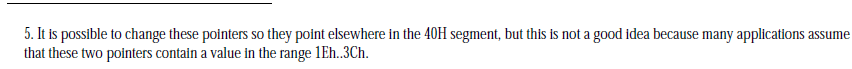
La ISR del teclado inserta el dato en la ubicación a la que apunta TailPtr. La función de teclado del BIOS remueve caracteres de la ubicación a la que apunta la variable HeadPtr. Esos dos apuntadores casi siempre contienen un offset en el arreglo

Buffer%FontSize=14
%TeXFontSize=14
\documentclass{article}
\pagestyle{empty}
\begin{document}
\[
^{5}
\]
\end{document}. Si esos dos apuntadores son iguales, el type ahead buffer está vacío. Si el valor en HeadPtr es tal que el valor en TailPtr es dos veces más grande (o HeadPtr es 1Eh y TailPtr es 3Ch [Nótese que 1E<<1 = 3C]), entonces el buffer está lleno y la ISR del teclado rechazará cualesquiera pulsaciones de teclas adicionales.

1E = 16 + 14 =30,

3C = 48 + 12 =60,

60 – 30 = 30. Desde la dirección 1E hasta la dirección 3C se pueden colocar 16 palabras de 16 bits (de dos bytes)

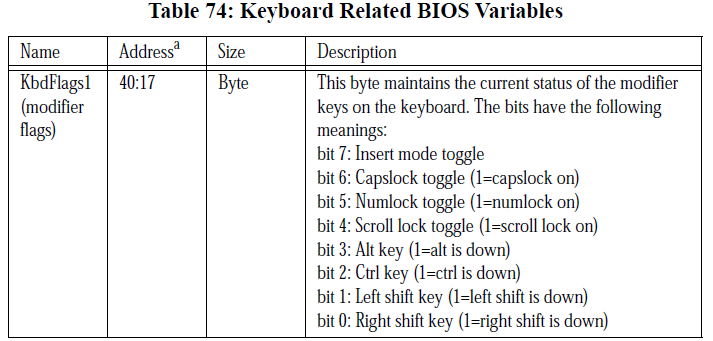


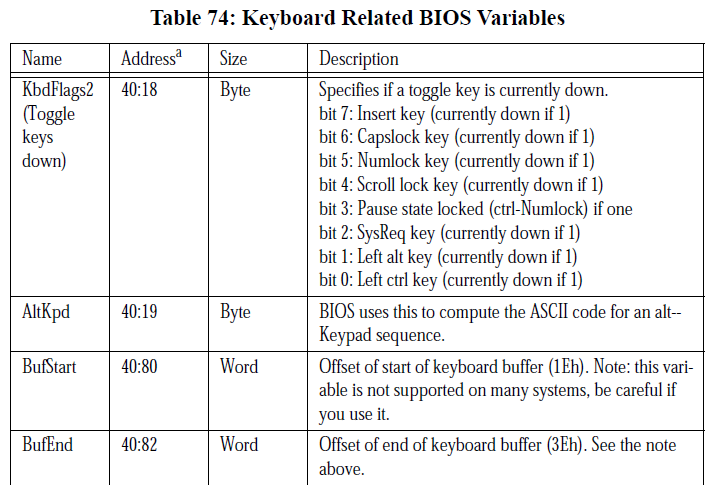


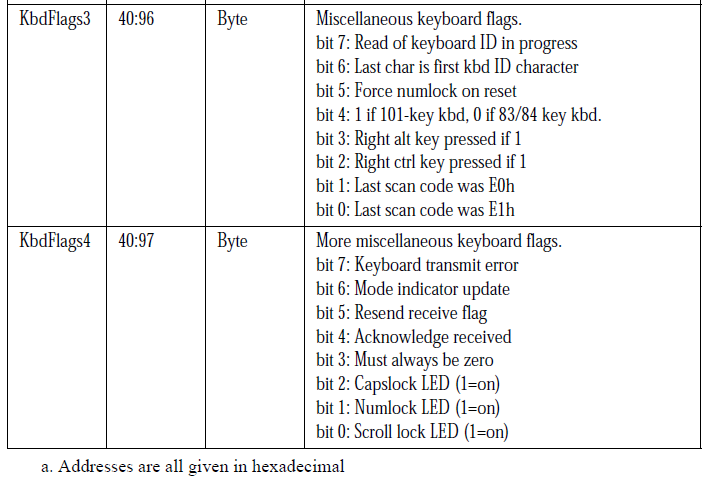
El type ahead buffer

Note que la variable TailPtr siempre apunta a la siguiente ubicación disponible en el type ahead buffer. Dado que no hay una variable “count” que proporcione el número de entradas en el buffer, siempre debemos dejar una entrada libre en el área de buffer; esto significa que el type ahead buffer puede contener solamente 15 keystrokes, no 16.

Además del type ahead buffer, el BIOS mantiene otras variables relacionadas con el tecladoen el segmento 40h. La siguiente tabla lista esas variables y sus contenidos:



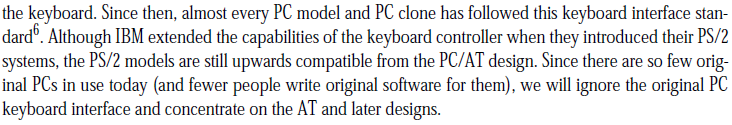


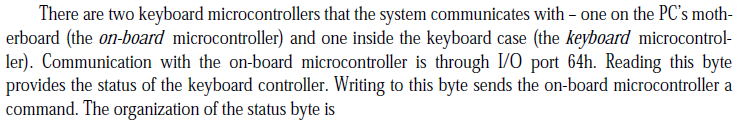


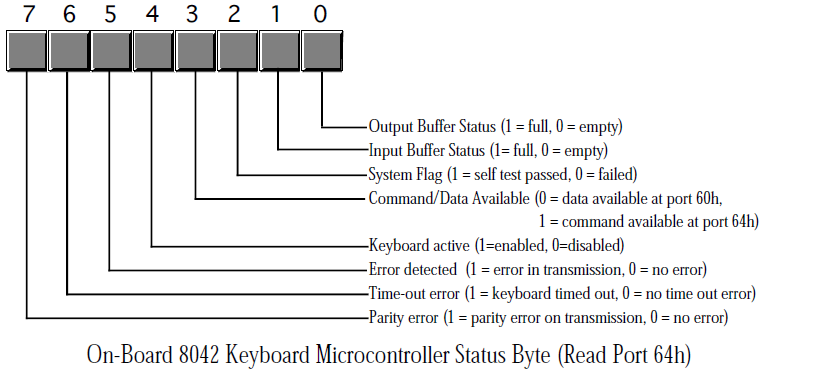
Un comentario acerca de KbdFlags1 y KbdFlags4. Los bits 0 al 2 de la variable KbdFlags4 es el ajuste actual de BIOS para los leds sobre el teclado. Periodicamente, el BIOS compara los valores para capslock, numlock, y scroll lock en KbdFlags1 contra estos tres bits en KbdFlags4. Si no coinciden, BIOS enviará un comando apropiado al teclado para actualizar los leds y cambiará los valores en la variable KbdFlag4 de tal manera que el sistema sea consistente. Por lo tanto, si usted enmascara nuevos valores para numlock, scroll lock, o caps lock, el BIOS automáticamente ajustará KbdFlags4 y ajustará los leds de acuerdo con esos bits (los 3 bits del bit 0 al bit 2 de KbdFlags4).

The Keyboard Hardware Interface

IBM utilizó un diseño hardware muy simple para el puerto del teclado sobre las máquinas PC original y PC/XT. Cuando introdujeron la PC/AT, IBM rediseñó completamente la interface entre la PC y el teclado. Desde entonces, casi cada modelo de PC y de clone de PC ha seguido este estándar de interface de teclado%FontSize=14
%TeXFontSize=14
\documentclass{article}
\pagestyle{empty}
\begin{document}
\[
^{6}
\]
\end{document}.



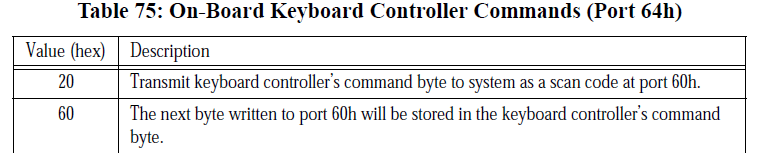


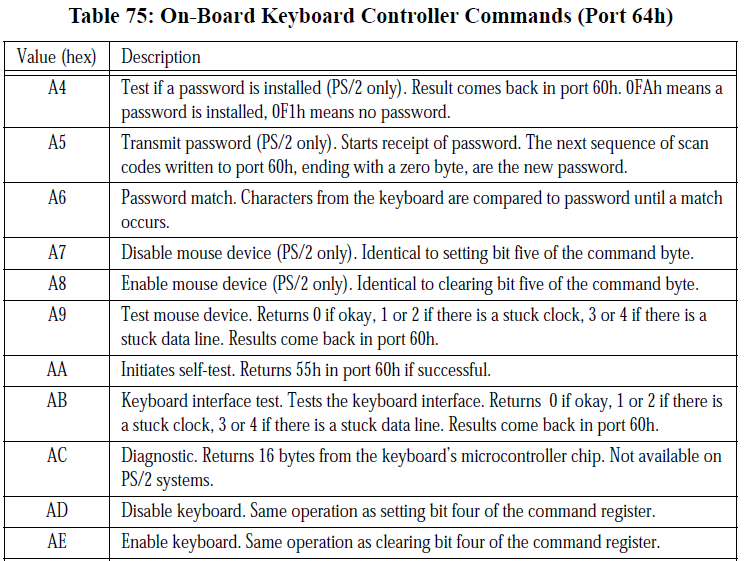


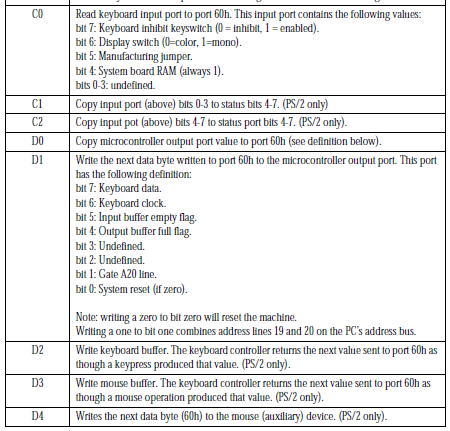


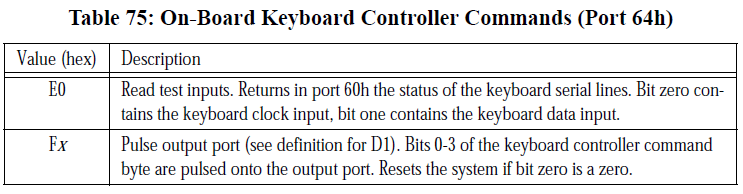
La comunicación al microcontrolador en el teclado es a través de los bytes en la dirección de I/O 60h y 64h. Los bits cero y uno en el byte de status en el puerto 64h proporcionan el control **handshaking** necesario para esos puertos. Antes de escribir algún dato a esos puertos, el bit cero del puerto 64h debe ser cero; en el puerto 60h hay datos disponibles para lectura cuando en el bit uno del puerto 64h contiene un uno. Los bits enable y disable en el byte de comando (puerto 64h) determina si el teclado está activo y si el teclado interrumpirá el sistema cuando el usuario presiona (o libera) una tecla, etc.

Los bytes escritos al puerto 60h son enviados al microcontrolador del teclado y los bytes escritos al puerto 64h son enviados al microcontrolador sobre la tarjeta madre (on-board). Los bytes leídos del puerto 60h generalmente vienen desde el teclado, aunque usted puede programar el microcontrolador on-board para retornar ciertos valores en este puerto, también. Las siguientes tablas listan los comandos enviados al microcontrolador de teclado y los valores que usted puede esperar de regreso. La siguiente tabla lista los comandos permitidos que usted puede escribir al puerto 64h:

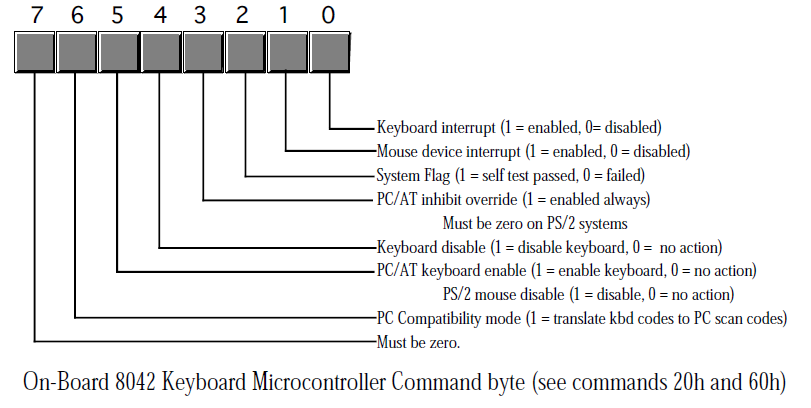




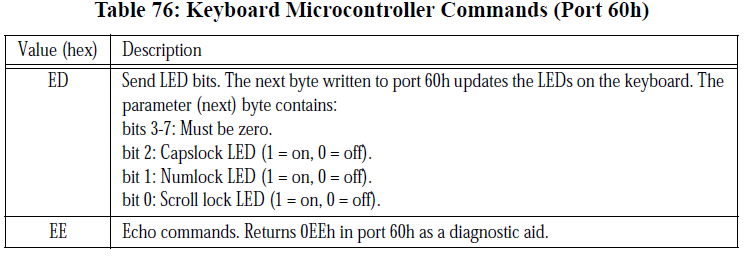


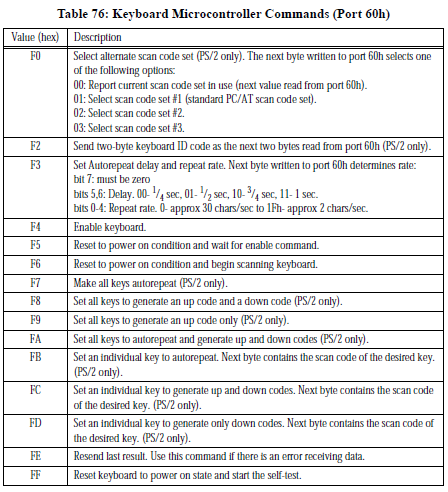


Los comandos 20h y 60h le permiten a usted leer y escribir el byte comando del controlador de teclado. Este byte es interno al microcontrolador on-board y tiene el siguiente layout:



El sistema transmite los bytes escritos al puerto de I/O 60h directamente al microcontrolador de teclado. El bit cero del registro de estado debe contener un cero antes de escribir algún dato a este puerto. Los comandos que el teclado reconoce son





El siguiente programa corto demuestra como enviar comandos al controlador de teclado. Esta pequeña utilidad TSR programa un “light show” sobre los LEDs del teclado.

; LEDSHOW.ASM

;

; This short TSR creates a light show on the keyboard’s LEDs. For space

; reasons, this code does not implement a multiplex handler nor can you

; remove this TSR once installed. See the chapter on resident programs

; for details on how to do this.

;

; 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

; 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 al, 0ADh ;Disable kbd for now.

call SetCmd

cli ;Disable ints while accessing HW.

; 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

mov al, 0AEh ;Reenable keyboard.

call SetCmd

sti ;Allow interrupts now.

pop cx

pop bx

pop ds

ret

SendCmd endp

; SetLEDs- Writes the value in AL to the LEDs on the keyboard.

; Bits 0..2 correspond to scroll, num, and caps lock,

; respectively.

SetLEDs proc near

push ax

push cx

mov ah, al ;Save LED bits.

mov al, 0EDh ;8042 set LEDs cmd.

call SendCmd ;Send the command to 8042.

mov al, ah ;Get parameter byte

call SendCmd ;Send parameter to the 8042.

pop cx

pop ax

ret

SetLEDs endp

; MyInt1C- Every 1/4 seconds (every 4th call) this routine

; rotates the LEDs to produce an interesting light show.

CallsPerIter equ 4

CallCnt byte CallsPerIter

LEDIndex word LEDTable

LEDTable byte 111b, 110b, 101b, 011b,111b, 110b, 101b, 011b

byte 111b, 110b, 101b, 011b,111b, 110b, 101b, 011b

byte 111b, 110b, 101b, 011b,111b, 110b, 101b, 011b

byte 111b, 110b, 101b, 011b,111b, 110b, 101b, 011b

byte 000b, 100b, 010b, 001b, 000b, 100b, 010b, 001b

byte 000b, 100b, 010b, 001b, 000b, 100b, 010b, 001b

byte 000b, 100b, 010b, 001b, 000b, 100b, 010b, 001b

byte 000b, 100b, 010b, 001b, 000b, 100b, 010b, 001b

byte 000b, 001b, 010b, 100b, 000b, 001b, 010b, 100b

byte 000b, 001b, 010b, 100b, 000b, 001b, 010b, 100b

byte 000b, 001b, 010b, 100b, 000b, 001b, 010b, 100b

byte 000b, 001b, 010b, 100b, 000b, 001b, 010b, 100b

byte 010b, 001b, 010b, 100b, 010b, 001b, 010b, 100b

byte 010b, 001b, 010b, 100b, 010b, 001b, 010b, 100b

byte 010b, 001b, 010b, 100b, 010b, 001b, 010b, 100b

byte 010b, 001b, 010b, 100b, 010b, 001b, 010b, 100b

byte 000b, 111b, 000b, 111b, 000b, 111b, 000b, 111b

byte 000b, 111b, 000b, 111b, 000b, 111b, 000b, 111b

byte 000b, 111b, 000b, 111b, 000b, 111b, 000b, 111b

byte 000b, 111b, 000b, 111b, 000b, 111b, 000b, 111b

TableEnd equ this byte

OldInt1C dword ?

MyInt1C proc far

assume ds:cseg

push ds

push ax

push bx

mov ax, cs

mov ds, ax

dec CallCnt

jne NotYet

mov CallCnt, CallsPerIter ;Reset call count.

mov bx, LEDIndex

mov al, [bx]

call SetLEDs

inc bx

cmp bx, offset TableEnd

jne SetTbl

lea bx, LEDTable

SetTbl: mov LEDIndex, bx

NotYet: pop bx

pop ax

pop ds

jmp cs:OldInt1C

MyInt1C endp

Main proc

mov ax, cseg

mov ds, ax

print

byte “LED Light Show”,cr,lf

byte “Installing....”,cr,lf,0

; Patch into the INT 1Ch interrupt vector. Note that the

; statements above have made cseg the current data segment,

; so we can store the old INT 1Ch values directly into

; the OldInt1C variable.

cli ;Turn off interrupts!

mov ax, 0

mov es, ax

mov ax, es:[1Ch\*4]

mov word ptr OldInt1C, ax

mov ax, es:[1Ch\*4 + 2]

mov word ptr OldInt1C+2, ax

mov es:[1Ch\*4], offset MyInt1C

mov es:[1Ch\*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

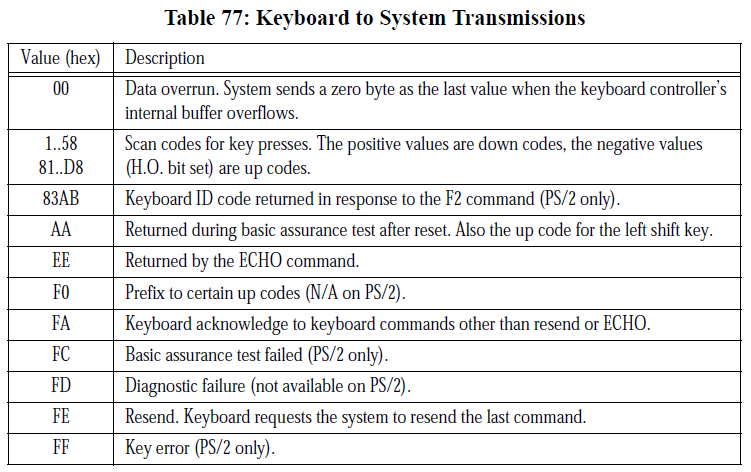
zzzzzzseg segment para public ‘zzzzzz’

LastBytes db 16 dup (?)

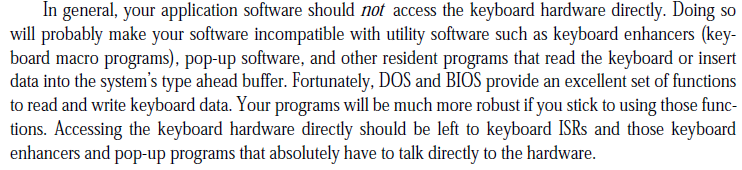
zzzzzzseg ends

end Main

El microcontrolador de teclado también envía datos al microcontrolador on-board para procesar y liberar al sistema a través del puerto 60h. La mayoría de esos valores son scan codes de tecla presionada (up o down codes), pero el teclado transmite otros valores también. Una rutina de servicio de interrupción de teclado bien diseñada debe ser capaz de manejar (o al menos ignorar) los valores que no son scan codes. Cualquier particular, cualquier programaque envía comandos al teclado necesita ser capaz de manejar los comandos de reenvío y de acknowledge que el microcontrolador de teclado devuelve en el puerto 60h. El microcontrolador de teclado envía los siguientes valores al sistema:



Suponiendo que usted no ha deshabilitado las interrupciones del teclado (véase el byte de comando del controlador de teclado), cualquier valor que el microcontrolador de teclado envía al sistema a través del puerto 60h generará una interrupción sobre la línea IRQ uno (int 9). Por lo tanto, la rutina de servicio de interrupción del teclado normalmente maneja todos los códigos de arriba. Si usted está modificando la ISR de la int 9, no olvide enviar una señal end of interrupt (EOI) al PIC 8259A al final de su código de la ISR. También no olvide que usted puede habilitar o deshabilitar la interrupción de teclado en el 8259A.



REFERENCIA

The Art of Assembly Language Programming, chapter 20 The PC Keyboard. CH20.PDF