Wout Mertens' Guide To Keyboard Programming v1.1 Complete

Download a ZIP of this document and it's accompanying source code.

0. Legal Info

0.1. Preface

I wrote this document because I needed info, and thought I could get it this way. Boy was I wrong! I ended up finding it all by myself. Anyway, I hope you can use it. It is meant for people who know what interrupts are and that 0ah equals 10. Enjoy.

Oh, almost forgot. I didn't give this text any page formatting (aside from spaces before and room after for ease of reading) because:

- I read ALL my documents on-line
- People have differing page sizes and then it would look like shit for some people and too short for others.

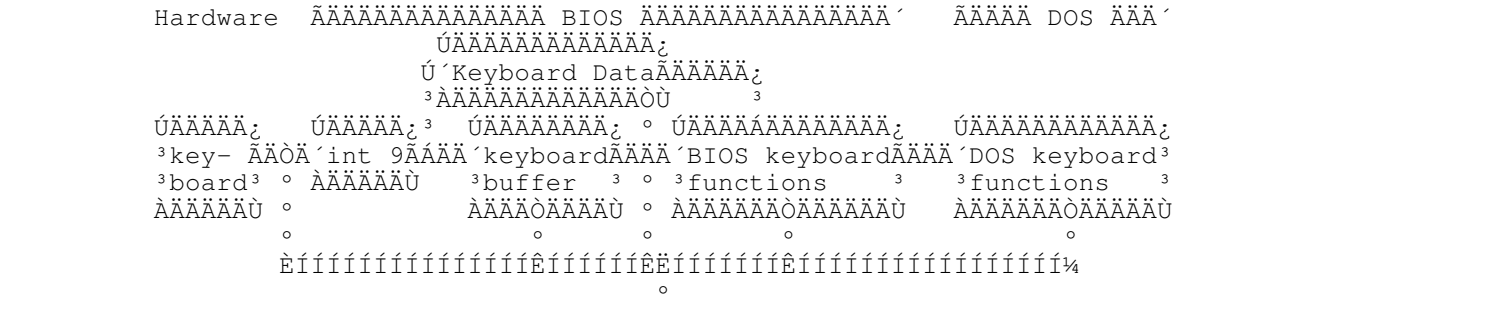
If you want to print this, well, go ahead and format it, BUT DON'T EVEN *THINK* OF SPREADING IT !!! (Except when you ask my permission)

Everytime you see something like d9h or 65h, it is a hexadecimal number. No trailing 0 was added for ease of typing.

1. Overall Information

On the IBM, there are three ways, all alike, to access the keyboard. Via the operating system, via BIOS or via low-level access. Which way you use depends very much on the application you are writing. Games do not use DOS functions, for example. And a file-compressor is really not interested wether you are actually pressing 'Y' or not. Or how long.

This is the way it works:



Possible tap points

The keyboard triggers IRQ 1 (Interrupt Request), also known as int 9. Int 9 then translates the keyboard codes into ASCII, or when necessary, extended ASCII, and places it into the keyboard buffer. Also, the shift and lock states are saved in the BIOS Data Area (seg 40h). The keyboard buffer is then used by the BIOS functions to interface with programs. The DOS functions use the BIOS keyboard functions to interface with programs as well, but on a higher and more protected (Ctrl-Brk etc) level.

1.1. Extended ASCII

Extended ASCII is IBM's way of letting non-ASCII keys be recognized by programs. The BIOS will first send 0 and then the extended ASCII code.

Here is the table:

Key	Hex	Dec	Key	Hex	Dec	Key	Hex	Dec	Key	Hex	Dec
F1	3B	59	Shift-F1	54	84	Ctrl-F1	5E	94	Alt-F1	68	104
F2	3C	60	Shift-F2	55	85	Ctrl-F2	5F	95	Alt-F2	69	105
F3	3D	61	Shift-F3	56	86	Ctrl-F3	60	96	Alt-F3	6A	106
F4	3E	62	Shift-F4	57	87	Ctrl-F4	61	97	Alt-F4	6B	107

Key	Hex	Dec	Key	Hex	Dec	Key	Hex	Dec	Key	Hex	Dec
F5	3F	63	Shift-F5	58	88	Ctrl-F5	62	98	Alt-F5	6C	108
F6	40	64	Shift-F6	59	89	Ctrl-F6	63	99	Alt-F6	6D	109
F7	41	65	Shift-F7	5A	90	Ctrl-F7	64	100	Alt-F7	6E	110
F8	42	66	Shift-F8	5B	91	Ctrl-F8	65	101	Alt-F8	6F	111
F9	43	67	Shift-F9	5C	92	Ctrl-F9	66	102	Alt-F9	70	112
F10	44	68	Shift-F10	5D	93	Ctrl-F10	67	103	Alt-F10	71	113

Key	Hex	Dec	Key	Hex	Dec	Key	Hex	Dec	Key	Hex	Dec
Alt-A	1E	30	Alt-P	19	25	Alt-3	7A	122	down	50	80
Alt-B	30	48	Alt-Q	10	16	Alt-4	7B	123	left	4B	75
Alt-C	2E	46	Alt-R	13	19	Alt-5	7C	124	right	4D	77
Alt-D	20	32	Alt-S	1F	31	Alt-6	7D	125	up	48	72
Alt-E	12	18	Alt-T	14	20	Alt-7	7E	126	End	4F	79
Alt-F	21	33	Alt-U	16	22	Alt-8	7F	127	Home	47	71
Alt-G	22	34	Alt-V	2F	47	Alt-9	80	128	PgDn	51	81
Alt-H	23	35	Alt-W	11	17	Alt--	82	130	PgUp	49	73
Alt-I	17	23	Alt-X	2D	45	Alt-=	83	131			
Alt-J	24	36	Alt-Y	15	21				^left	73	115
Alt-K	25	37	Alt-Z	2C	44	NUL	03	3	^right	74	116
Alt-L	26	38				Shift-Tab	0F	15	^End	75	117
Alt-M	32	50	Alt-0	81	129	Ins	52	82	^Home	77	119
Alt-N	31	49	Alt-1	78	120	Del	53	83	^PgDn	76	118
Alt-O	18	24	Alt-2	79	121	^PrtSc	72	114	^PgUp	84	132

101-key Keyboard Extensions Supported by BIOS

Key	Hex	Dec	Key	Hex	Dec	Key	Hex	Dec
F11	85	133	Alt-Bksp	0E	14	Alt - K /	A4	164
F12	86	134	Alt-Enter	1C	28	Alt - K *	37	55
Shft-F11	87	135	Alt-Esc	01	1	Alt - K -	4A	74
Shft-F12	88	136	Alt-Tab	A5	165	Alt - K +	4E	78
Ctrl-F11	89	137	Ctrl-Tab	94	148	Alt - K Enter	A6	166
Ctrl-F12	8A	138						
Alt-F11	8B	139	Alt-up	98	152	Ctrl- K /	95	149
Alt-F12	8C	140	Alt-down	A0	160	Ctrl- K *	96	150
Alt-[	1A	26	Alt-left	9B	155	Ctrl- K -	8E	142
Alt-]	1B	27	Alt-right	9D	157	Ctrl- K +	90	144
Alt-;	27	39						
Alt-'	28	40	Alt-Delete	A3	163	Ctrl- K Up [8]	8D	141
Alt-`	29	41	Alt-End	9F	159	Ctrl- K Cn [5]	8F	143
Alt-\	2B	43	Alt-Home	97	151	Ctrl- K Dw [2]	91	145
Alt-,	33	51	Alt-Insert	A2	162	Ctrl- K Ins[0]	92	146
Alt-.	34	52	Alt-PageUp	99	153	Ctrl- K Del[.]	93	147
Alt-/	35	53	Alt-PageDown	A1	161			

K indicates a key on the numeric keypad (when not in NumLock mode)

1.2. Special Functions

There are a few functions and interrupts invoked by int 9:

```
int 5   - Print Screen Handler

int 15h
fns 4fh - Check Scancode
          (See int 9)

    85h - System Request
          Normally IRET

int 23h - Ctrl-Break handler
```

Feel free to revector any of them.

2. DOS Interfacing

One of the ways to use the keyboard is to let DOS handle it.

Pro:
- The keyboard lay-out is unimportant
- You can even do strings
- The user doesn't actually have to type

Contra:
- You don't know if you are actually accessing the keyboard (like in "Really format drive C: ? Y/N" :-)
- The functions are quite slow

2.1. Functions

DOS provides a set of 7 functions to handle the keyboard:

01h Keyboard Input
06h Console I/O
07h No Echo Unfiltered Input
08h No Echo Filtered Input
0Ah Buffered Input
0Bh Input Status
0Ch Clear Keyboard Buffer & Input

They all expect the keyboard to be file handle 0. If you want to let a program think you are typing something, you can replace this handle with a file containing the keystrokes it must read. This is what happens when you 'pipe' something in DOS. (Don't forget to change the handle back to the old one!)

This also means you can use:

3Fh Read bytes from handle

Fn 01h: Keyboard Input

Expects: AH 01h

Returns: AL Character fetched from the Standard Input

Description: Reads (waits for) a character from the Standard Input Device. Echoes that character to the Standard Output Device. If Ctrl-Break is detected, INT 23h is executed.

Notes: Extended ASCII keystrokes (ie, F1-F12, PgUp, cursor, etc) will require two calls to this function. The first call will return AL=0. The second will return AL with the extended ASCII code.

Fn 06h: Console I/O

Expects: AH 06h
DL 0 to 0FEh Character to send to the Standard Output
0FFh Request for input from the Standard Input

Returns: ZF Clear (NZ) if character is ready \ on input requests
AL Character read, if ZF is clear / (when DL=0FFh)

Description: If DL is 0FFh, this performs a "no wait" console input, returning the Zero Flag (ZF) set (ZR) if there is no character ready. If a character is ready, returns ZF cleared (NZ) with the character that was read in AL.

```
               If DL  is anything  but 0FFh,  DL is  sent to  the Standard
               Output.

Notes:         Does not  check for  Ctrl-Break.   Call twice  for Extended
               ASCII.

Fn 07h: No Echo Unfiltered Console Input
----------------------------------------
Expects: AH  07h

Returns: AL  Character fetched from the Standard Input

Description: Reads  (waits  for)  a  character  from  the Standard Input
             Device, returning that character in AL.

             Unfiltered: Does not detect Ctrl-Break, backspace, etc.

Notes:       Call twice for Extended ASCII character input.
             Use Fn 0Bh to check status  (if you don't want to wait  for
             a key).

Fn 08h: No Echo Console Input
-----------------------------
Expects: AH  08h

Returns: AL  Character fetched from the Standard Input

Description: Reads  (waits  for)  a  character  from  the Standard Input
             Device, returning that character in AL.

             If Ctrl-Break is detected, INT 23h is executed.

Notes:       Call twice for Extended ASCII character input.

Fn 0Ah: Buffered String Input
-----------------------------
Expects: AH  0Ah
         DS:DX Address of an input buffer (see below)

Returns:     Buffer contains input terminated with CR (ASCII 13h)

Description: On entry, the buffer at DS:DX must be set up as:
             ÚÄÄÄÂÄÄÄÂÄÄÄÂÄÄÄÂÄÄÄÂÄÄÄÂÄ Ä Ä
             ³max³ ? ³ ?   ?   ?   ?   ?    max is maximum acceptable
             ÀÄÄÄÁÄÄÄÁÄÄÄÁÄÄÄÁÄÄÄÁÄÄÄÁ Ä  Ä  input (range: 1 to 254)
             On exit, the buffer is filled:
             ÚÄÄÄÂÄÄÄÂÄÄÄÂÄÄÄÂÄÄÄÂÄÄÄÂÄ Ä Ä  len is actual length of
             ³max³len³ T   E   X   T   0Dh  input, less the termina-
             ÀÄÄÄÁÄÄÄÁÄÄÄÁÄÄÄÁÄÄÄÁÄÄÄÁ Ä  Ä  ting CR (eg, 4).

             Characters are  read from  the Standard  Input up  to a  CR
             (ASCII  13)  or  up  to  the  value  of max-1.  If max-1 is
             reached, the console bell rings (beeps) for each  character
             until Enter (CR) is read.

             The second  byte of  the buffer  is filled  with the actual
             length of the  input, less the  terminating CR.   The final
             character in the buffer is always CR (which is not  counted
             in the length byte).

             The characters  in the  buffer (including  the len)  before
             the call are used as a "template" and the DOS editing  keys
             are in effect:  [Esc]  displays "\" and restarts the  edit,
             [F3] displays  to the  end of  the template,  [F5] displays
             "@"  and  stores  the  current  line  as the template, etc.
             Most Extended ASCII keystrokes are ignored.

             If  Ctrl-Break  is  detected,  INT  23h is executed and the
```

buffer is left unchanged.

Fn 0Bh: Check Input Status

Expects: AH 0Bh

Returns: AL 0FFh if a character is available from the Standard Input
0 if no character is available

Description: Checks the status of the Standard Input.

If Ctrl-Break is detected, INT 23h is executed.

Notes: Use before Fns 01h, 07h and 08h to avoid having DOS wait for a key.

This is a simple, non-destructive way to check for Ctrl-Break during long calculations or other processing that does not normally look for input. It lets the user abort from such a sequence.

Fn 0Ch: Clear & Input

Expects: AH 0Ch
AL DOS input function number (01h, 06h, 07h, 08h, or 0Ah)

Returns: none

Description: Clears the Standard Input type-ahead buffer then invokes the DOS input function specified by AL. This forces the system to wait for a character to be typed.

These values are allowed for AL:
- 01h Keyboard Input
- 06h Console I/O
- 07h No Echo Unfiltered Input
- 08h No Echo Filtered Input
- 0Ah Buffered Input

In addition to these functions, it is also possible to read a selected amount of characters from the keyboard, using DOS's File Handle functions, as the Keyboard, aka Standard Input, has a pre-set handle of 0000h:

Fn 3Fh: Read from keyboard via Handle

Expects: AH 3Fh
BX 0000h - Handle for Standard Input (Keyboard)
DS:DX Address of buffer to receive data
CX Number of bytes to read

Returns: AX Error code if CF is set to CY
AX Number of bytes actually read

Description: CX bytes of data are read from the keyboard. The data is placed into the caller's buffer pointed to by DS:DX.

Notes: It is handy to use this function for reading default handles such as the Standard I/O handles, instead of the buffered input or character-by-character input functions.

When you read from a device, AX returns the length of the line up to and including the termination CR (ASCII 13h).

3. BIOS Interfacing

```
Pro:
    - You get to know all the statusses and such
    - It's a tad bit faster than DOS
    - You can only read the keyboard
    - It's easier than the really hardcore low level, and the keys
      are translated

Contra:
    - It is still to slow for games or demos
    - You don't have bulk access, like strings

The BIOS has 3 different ways of reading (parts of) the keyboard:
    - functions
    - keyboard flags
    - keyboard buffer

This part describes all of them.


3.1. Functions
ÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄ

These functions can be accessed through int 16h.

Fn 00h: Read (wait for) next keystroke
--------------------------------------
Expects: AH  0

Returns: AL  ASCII  character (if  AL=0, AH  is an Extended  ASCII  key-
             stroke)
         AH  Scan Code or Extended ASCII keystroke

Fn 01h: Check if a keystroke is ready (and preview it if so)
------------------------------------------------------------
Expects: AH  1

Returns: ZF  ZR or 1 if no key is ready
         ZF  NZ or 0 if a key is ready.
         AX  is  set  as  for  Fn  00h  (but  the keystroke has not been
             removed from the queue).

Fn 02h: Read the shift-key status
---------------------------------
Expects: AH  2

Returns: AL  shift key and 'lock' status as in 83-keyboard flags

Description: Determine which shift keys are currently being pressed  and
             whether the keyboard is in NumLock state, etc.

Fn 03h Set keyboard typeamatic rate and delay. (11/15/85 BIOS)
--------------------------------------------------------------
Expects: AH  3
         AL  05h (eg, AX = 0305h)
         BL  Typeamatic Rate
             0: 30 keys/sec  10: 10
             1: 26.7         13: 9
             2: 24           16: 7.5
             4: 20           20: 5
             8: 15           31: 2
         BH  Delay: 0=250ms 1=500ms 2=750ms 3=1 second)

Returns: none

Description: when a key is pressed, the keyboard will wait during Delay
             before it starts repeating at Typematic Rate.

Fn 05h Place a keystroke into the keyboard buffer. (11/15/85 BIOS)
------------------------------------------------------------------
```

```
Expects: AH  5
         CL  ASCII character.
         CH  Scan Code  byte (or 0 if you don't care)

Returns: AL  Status: 0=success; 1=buffer full

Fn 10h Read (wait for) a keystroke; 101-keyboard only (11/15/85 BIOS)
-----------------------------------------------------------------------
Expects: AH  10h

Returns: AL  ASCII  character  (if  AL=0,  AH is an  Extended ASCII key-
             stroke)
         AH  Scan Code or Extended ASCII keystroke

Fn 11h Preview keystroke; same as 01; 101-keyboard only (11/15/85 BIOS)
-----------------------------------------------------------------------
Expects: AH  11h

Returns: ZF  ZR or 1 if no key is ready
         ZF  NZ or 0 if a key is ready.
         AX  set as for Fn 10 but keystroke is still in the buffer.

12h Read shift-key status; same as 02; 101-keyboard only (11/15/85 BIOS)
-----------------------------------------------------------------------
Expects: AH  12H
Returns: AL  shift key and 'lock' status as in 101-keyboard flags


3.2. Keyboard Flags
ÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄ

The keyboard flags are found in the BIOS Data Area: segment 40h.

17h: 83-keyboard flags      0=Off, 1=On
---------------------------------------
bit 0: Right shift
    1: Left shift
    2: Ctrl, either side
    3: Alt, either side
    4: Scroll Lock
    5: Num Lock
    6: Caps Lock
    7: Insert state

Do NOT just change one of these and then hope the keyboard follows.  The
LEDs will definitely get out of sync.

18h: 101-keyboard flags     0=Off, 1=On
---------------------------------------
bit 0¿                  : Left ctrl
    1ÃÄAt keyb. only     : Left Alt
    2Ù                  : Sys Req
    3: Pause state
    4: Scroll Lock      ¿
    5: Num Lock         ÃÄBeing pressed
    6: Caps Lock        ³
    7: Insert           Ù

Do NOT just change one of these and then hope the keyboard follows.  The
LEDs will definitely get out of sync.

19h: Pseudokey value
--------------------
This is the  accumulating value of  the key being  made with Alt+numeric
keypad.  Normally 0

71h: Ctrl-break flag        0=Off, 1=On
---------------------------------------
bit 7: Ctrl-Break was pressed. Never gets reset, unless you do.
```

```
96h: AT only - keyboard ?   0=Off, 1=On
---------------------------------------
bit 4: 101/102 keyboard is attached

97h: AT only - lock LEDs    0=Off, 1=On
---------------------------------------
bit 0: ScrollLock ¿
    1: NumLock    ÃÄ keyboard LED is turned on
    2: CapsLock   Ù
```

Do NOT just change one of these and then hope the keyboard follows. The LEDs will definitely get out of sync.

3.3. Keyboard Buffer

The keyboard buffer is a circular data area. This means that when a pointer in the buffer gets one larger than the buffer, it is wrapped around to the beginning.

The keyboard buffer is fed by int 9 and function 5 of int 16h. It is found at the BIOS data segment, 40h. It is pointed to by 4 variables in the BDA: The head (1ah), the tail (1ch), the Beginning (80h) and the End (82h). They are all words, pointing at locations in the BDA.

The latter two are only available on ATs and PSs. They are used to enlarge the keyboard buffer by mapping it to another spot in the BIOS data area. Normally, that spot is 32 bytes long starting from 1eh.

The head is the pointer to the next word. The tail is the pointer to the next available word. Each code is two bytes, the scan code and the ASCII value.

The buffer is empty if the Head = the Tail and it is full if the Tail is two smaller than the Head, both counted circularly. This means that the storage space equals (length buffer/2)-1.

4. Low-Level Interfacing

Pro:
- Fast
- Complete control

Contra:
- Hard to code
- Totally NO functions at all. It's Handyman work here...

The interfacing is split in two items:
- Just changing something, such as the LED's
- Reading out codes: int 9

4.1. Interfacing And Configuring

The computer and the AT or MF II interface through I/O ports 60h and 64h, controlled by a programmable Intel 8042 (old ATs), 8741 or 8742 (newer, allow two input devices (like the PS/2 mouse)) microprocessor or compatible, which allows typematic rate programming, LEDs lighting and some other stuff. It also has a +-20 byte output buffer for smooth operation and long scancodes.

The old XT keyboard has a 8048, which is in essence just a very primitive one-way serial interface, so all used is port 61h, to disable and reenable the keyboard on every scancode.

```
Port 60h: Input & output
------------------------
Read: Scancodes and keyboarddata
--------------------------------
This port gives the following output codes:

    00h: Keyboard error, too many keys are being pressed at once
    aah: Basic Assurance Test (BAT) end
    abh 41h: The result of requesting keyboard ID on a MF II keyboard
    eeh: The result of the echo command
    fah: ACK(noledge). Sent by every command, except eeh and feh
    fch: BAT failed
    feh: Resend your data please
    ffh: Keyboard error

All the rest are make (press) and break (release) codes of the keys.

Write: Command data
-------------------
This is the  place where command  data has to  be sent.   If the command
consists of two bytes, you must  wait until the outputbuffer is sent  to
the keyboard.   Check on  it via  bit 1  of port  64h. When  you send  a
command, the outputbuffer is cleared,  so pending results may not  come.
During transmission of a two-byte command, the keyboard stops  scanning.
When you  send something  out of  range or  so, the  keyboard will react
with feh  (resend).   All commands,  except echo  (eeh) and resend (feh)
result in ACK (fah) to be sent.

Commands:

edh: Set keyboard LEDs
     Send a second byte with:

         bit 0 = Scroll Lock        0=Off 1=On
             1 = Num Lock
             2 = Caps Lock
          rest = 0

     Do make an effort to keep the BIOS keyboard flags in sync.

eeh: Great fun. Send it, and get 0eeh right back! :-]
     (Diagnostics)

f0h: Select scancode set.

         0: return current set number: 1:'C', 2:'A', 3:'?'
         1: set scancode set no 1
         2: set scancode set no 2 -> standard
         3: set scancode set no 3

f2h: Identify keyboard
         XT: nothing (that is, time-out error :-) (see port 64h)
         AT: ACK
         MF II: ACK abh 41h

f3h: Typematic rate programming
     Send a second byte with:

         bit 0 -> 4: rate. Timings:

         0: 30 keys/sec   10: 10
         1: 26.7          13: 9
         2: 24            16: 7.5
         4: 20            20: 5
         8: 15            31: 2

         bit 5 & 6: pause before repeat:
```

```
         0: 250 ms
         1: 500
         2: 750
         4: 1000

         bit 7: Always 0
```

The next three are doubtfull, since one of my sources say they don't exist and another says they do. I leave it up to you :)

```
f4h: Enable keyboard. It clears its buffer and starts scanning.
f5h: Reset keyboard, disable scanning
f6h: Reset keyboard, enable scanning

feh: Resend last transmission. I really don't know what it does, since
     it sends something incomprehesible.

ffh: Internal diagnostics: Sends aah if successfull.  Warning! The
     keyboard reacts with ACK and then you have to set the data and
     clock pins high, DURING AT LEAST 500 SECONDS!. Do this via the
     outputport (see 64h). After that, the BAT (Basic Assurance Test)
     starts. This sends aah on success and fch on failure.
```

Example: Set the keyboard LEDs

```
start:
      in  al, 64h                                      \It would be good
      and al, 02h   ;Test if command buffer is empty   |to put this in a
      jnz start                                        /macro...

      mov al, edh
      out 60h, al   ;Write outputport

wait:
      in  al, 64h
      and al, 02h   ;Test if command came through
      jnz wait

      mov al, 0111b
      out 60h, al   ;Set all LED's to ON.
```

Port 61h

This port is used to acknoledge the receival of a scancode, by disabling the keyboard and immediately reenabling it. This also means that you can read a scancode as many times as you like, until you acknoledge the receival.

```
bit 0 -> 5: Nothing to do with keyboard, but with the Programmable
            Peripheral Interface (PPI) -> save them!
bit 6: Hold keyboard clock low -> Keyboard can't send any data.
bit 7: 0=Enable keyboard; 1=Disable keyboard
```

Example:

```
      in  al, 61h
      mov ah, al    ;Save keyboard status
      or  al, 80h   ;Disable
      out 61h, al
      mov al, ah    ;Enable (If it was disabled at first, you wouldn't
      out 61h, al   ; be doing this anyway :-)
```

Port 64h: Interface: data and control

Read: Statusport

```
    bit 0: 1: Keyboard data is in buffer
```

```
            0: Output buffer empty -> use it to check for results
         1: 1: User data is in buffer
            0: Command buffer is empty -> time to send a command
         2: 1: Selftest successful
            0: Reset (?)
         3: 1: 64h was last accessed port
            0: 60h was last accessed port
         4: 1: Keyboard enabled
            0: Keyboard locked
         5: PS/2: Mouse interface
         6: 1: Time-out error occurred: Keyboard or PS/2 mouse didn't
               react. Use the Resend command to retry fetching the data
               byte. This could happen when trying to get a XT keyboard
               to do something :).
         7: 1: Last transmission had a parity error

Write: Control register
-----------------------
This is the control room  of the keyboard interface. If  additional data
is required,  send it  to port  60h after  writing the  command to  64h.
Also, check 61h bit 2 before sending anything.

Commands:

aah: Keyboard self test. Sends 55h if successfull.

abh: Test interface. Sends:

          00h: No error
          01h: Clock low
          02h: Clock high
          03h: Data low
          04h: Data high
          ffh: Total Error

adh: Deactivate keyboard

aeh: Activate keyboard

c0h: Read inputport. This is some highly specialized stuff and I wonder
     why I am typing this. Ok. The inputport is that what the keyboard
     is sending and some more. Layout:

         bit 0: Keyboard data in pin
             1: PS/2 mouse in pin
             2->5: reserved
             6: Wether you have a color or mono screen
             7: 1: Keyboard not locked
                0: Keyboard locked

     When you issue this command, the inputport is put on the
     outputbuffer, so you have the great priviledge of reading it at
     port 60h.

c1h: Puts the low nibble of the input port over bits 4-7 of the
     statusport, so you can read them out continuously. This lasts
     until bit 2 of the statusport gets set, meaning you are sending
     data to the keyboard.

c2h: Ditto, but it puts the high nibble over bits 0-3 of the
     statusport. Lifespan is the same.

d0h: Puts the outputport on the buffer. Layout:

         bit 0: 1: Reset processor
             1: 1: A20 gate enable
             2: PS/2 mouse data out
             3: PS/2 mouse clock signal
             4: 1: Output buffer full
```

```
              5: 1: Output buffer PS/2 mouse full
              6: Keyboard clock signal
              7: Keyboard data out

     Bit 0 and 1 are quite important for high memory and
     286-extended-memory access.

d1h: Write the following data byte to the outputport

d2h: Write the following data byte to the keyboardbuffer. This is VERY
     handy for TSRs that need to read codes that start with e0h. This
     way, they don't have to pass through the e0h, unless they know for
     sure it isn't their code, which results in correct functioning
     shift keys etc. At least, if it does what I think it
     does... [UNTESTED]

d3h: Ditto, for PS/2 mouse.

d4h: Write byte to PS/2 mouse.

e0h: Reads the keyboards testinputs, T0 and T1. T0 goes to bit 0 and T1
     to bit 1 of the byte that is put on the outputbuffer.

fxh: I think it sends x to the low nibble of the output port. It does
     reset my computer when I send feh, but that doesn't mean anything
     :-). The official explanation says that it keeps the corresponding
     bits in the output port low for 6ms...

Example: Send something to the outputport

start:
      in  al, 64h                                    \It would be good
      and al, 02h   ;Test if command buffer is empty|to put this in a
      jnz start                                      /macro...

      mov al, d1h
      out 64h, al   ;Write outputport

wait:
      in  al, 64h
      and al, 02h   ;Test if command came through
      jnz wait

      mov al, 01h
      out 60h, al
```

4.2. Lay-Out

The keyboard first consisted of 83 keys, which is now known as the XT keyboard. Then came along the AT-keyboard, which has 84 keys, a slightly different layout and an extra SysReq key. The next keyboard is the MF II keyboard. This one has 101 or 102 keys, and this is the one this section will be babbling about.

The keycaps change, but the most popular settings are QWERTY and AZERTY. Also popular is the Dvorak lay-out, made by what's-his-name Dvorak, who made the lay-out so that both hands did not have to move that much, resulting in fast (up to double) typing speed. This is it, should you be interested (slight modifications by me, because it actually requires a 12x4 keyboard):

```
        101 - key                          102 - key

~ ! @ # $ % ^ & * ( ) [ +          ü ! @ # $ % ^ & * ( ) [ +
` 1 2 3 4 5 6 7 8 9 0 ] =          ý 1 2 3 4 5 6 7 8 9 0 ] =

  " , . P Y F G C R L ? { |          " , . P Y F G C R L ? {
```

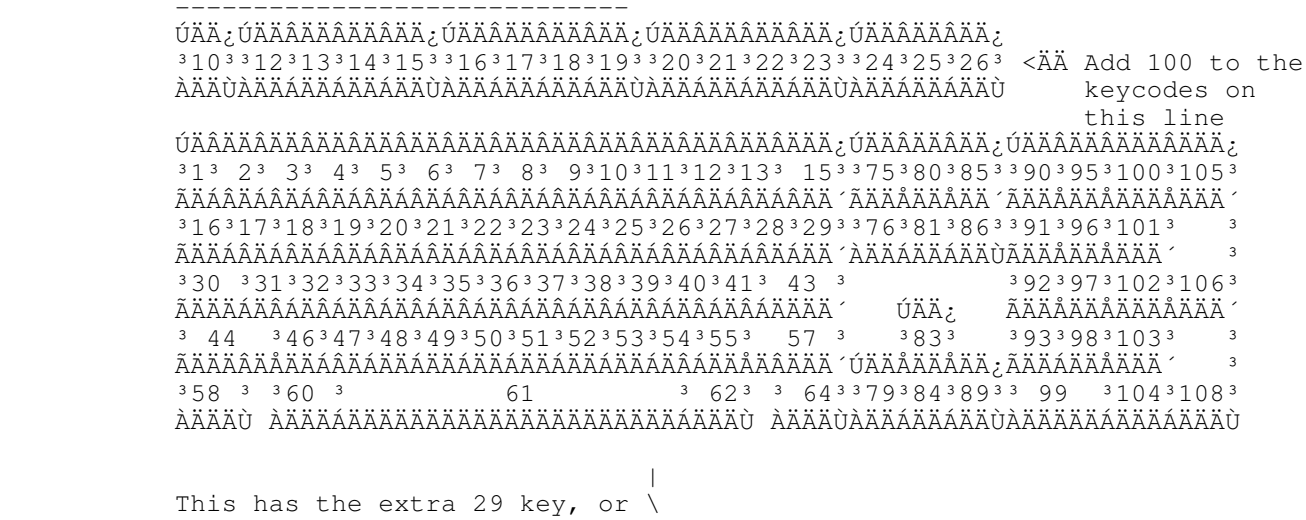
```
  ' , . p y f g c r l / } \          ' , . p y f g c r l / }

  A O E U I D H T N S _ <            A O E U I D H T N S _ ~
  a o e u i d h t n s - >            a o e u i d h t n s - `

  : Q J K X B M W V Z                > : Q J K X B M W V Z
  ; q j k x b m w v z                < ; q j k x b m w v z
```

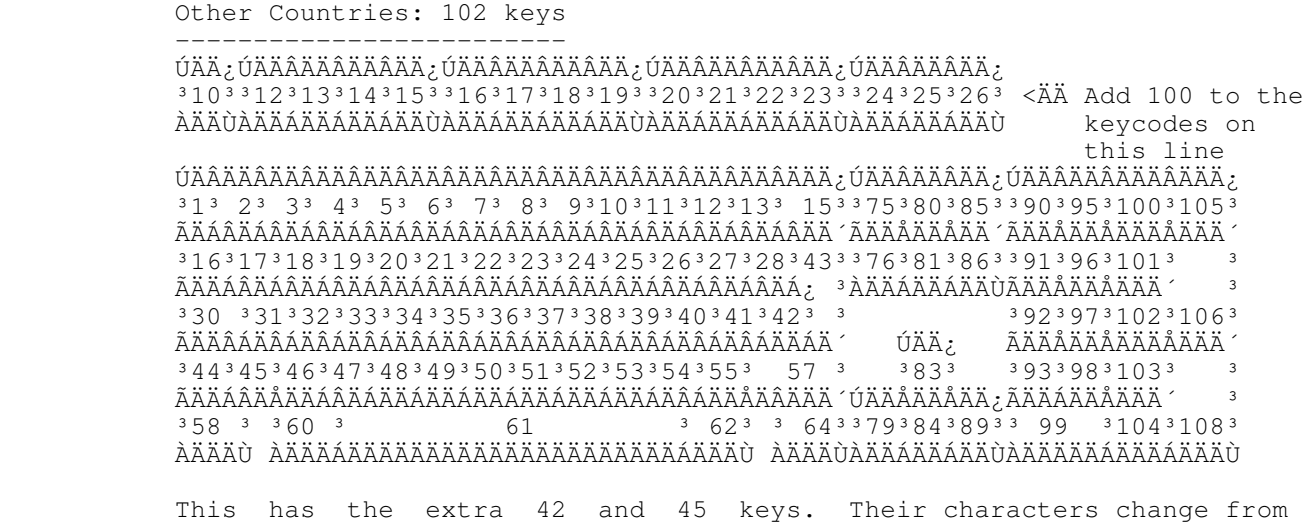
The key lay-out is as follows: (The numbers are internal to the keyboard)

US-English Keyboard: 101 keys



This has the extra 29 key, or \

Other Countries: 102 keys



This has the extra 42 and 45 keys. Their characters change from country to country.

4.3. Scancodes

The AT-keyboard has 3 separate scancode settings: One as we know it, (83 key-mapping, and added codes have an extra e0h added), one (almost) sequential and one with ONE byte codes! Problem with the latter is that only for lshift, caps, lctrl and lalt breakcodes are sent :-(. The keyboard starts up in set 2, the set can be changed via port 64h (see above).

In set 1 and 2, there are special codes, namely e0h and e1h. They are used for keys that have the same function. An example: 1dh for the left control key and e0h 1dh for the right one. This is done for lowlevel compatibility with XT programs. Notice that the only time e1h

is used, is when it represents a temporary control key, which also has a e0h version.

e0h 2ah is a temporary shift function, used by for example PrtScr, which is in reality shift-numkeypad-*, like on the XT keyboard. See below for further information.

The code will be sent as shown further. The codes listed are the make codes. They are sent when a key is pressed. Upon release, the keyboard sends a break code. It is the make code, but ORed with 80h. The only exception to this are the codes e0h and e1h, which remain the same. So for example pressing and releasing the right ctrl key would give e0h 1dh and e0h 9dh. I only give the codes for set 2 because the rest would be too much work and stupid. If you want them, look them up yourself. Modify any of the accompanying source codes or so...

ø Only on US-English keyboards
øø Only on other country versions

Scancodes are in hex.

Key no.	Scan code	Key no.	Scan code	Key no.	Scan code	Key no.	Scan code	Key no.	Scan code	Key no.	Scan code
1	³29	°19	³12	°36	³23	°53	³33	°86	³e0 51	°106	³4e
2	³02	°20	³13	°37	³24	°54	³34	°89	³e0 4d	°108	³e0 1c
3	³03	°21	³14	°38	³25	°55	³35	°90	³45	°110	³01
4	³04	°22	³15	°39	³26	°57	³36	°91	³47	°112	³3b
5	³05	°23	³16	°40	³27	°58	³1d	°92	³4b	°113	³3c
6	³06	°24	³17	°41	³28	°60	³38	°93	³4f	°114	³3d
7	³07	°25	³18	°42øø	³2b	°61	³39	°95	³e0 35	°115	³3e
8	³08	°26	³19	°43	³1c	°62	³e0 38	°96	³48	°116	³3f
9	³09	°27	³1a	°44	³2a	°64	³e0 1d	°97	³4c	°117	³40
10	³0a	°28	³1b	°45øø	³56	°75	³e0 52	°98	³50	°118	³41
11	³0b	°29ø	³2b	°46	³2c	°76	³e0 53	°99	³52	°119	³42
12	³0c	°30	³3a	°47	³2d	°79	³e0 4b	°100	³37	°120	³43
13	³0d	°31	³1e	°48	³2e	°80	³e0 47	°101	³49	°121	³44
15	³0e	°32	³1f	°49	³2f	°81	³e0 4f	°102	³4d	°122	³57
16	³0f	°33	³20	°50	³30	°83	³e0 48	°103	³51	°123	³58
17	³10	°34	³21	°51	³31	°84	³e0 50	°104	³53	°124	³(*)
18	³11	°35	³22	°52	³32	°85	³e0 49	°105	³4a	°125	³46
										°126	³(*)

(*)
Key 124, AKA PrtScr/SysRq, is both. When pressed normally, it will send (hex) e0 2a e0 37. This is in fact a special shift-*, or the original place of that code on the XT keyboard.

Used in conjuction with:

Normal: e0 2a e0 37
Shift : e0 37
Ctrl : e0 37
Alt : e0 54

Key 126: Pause/Break. On the XT keyboard, this used to be ctrl-NumLock and ctrl-ScrollLock. Now guess the codes... Very special is that the break codes are sent immediately after the make codes. I think that is because the codes have odd length.

Normal: e1 1d 45 (e0 1d is already used by rightctrl)
Ctrl : e0 46 (46 is the code for ScrollLock...)

4.4. Int 9

When a key is pressed or released, or when the 8042 sends an ACK or NAK the keyboard triggers IRQ1, or int 9. This can be masked by setting

bit 1 on port 21h, the interrupt controller. Int 9 gets the scancode, translates it and puts it in the keyboard buffer.

BEWARE: When a scancode consists of more than 1 byte, it should be read
------ one byte per call. (Took me quite long to find out...)

Translating:

Int 9 will first call int 15h, subfunction 4fh, with the scancode in al. If the scancode is legitimate, the carry flag is set, and al contains the scancode. If not, the carry flag is reset, and int 9 stops. (The carries are picked so that if int 9 thinks the BIOS supports the call and it doesn't, the carry is set by the BIOS, and the scancode can always be used). This allows the keyboard to be redefined, by taking over the function and replacing scancodes.

If you want to take over int 9, you must remember to let the interrupt controller know when you are finished, by writing 20h to port 20h, since this is a hardware interrupt. You should also disable and then reenable the keyboard (see port 61h), so the keyboard knows you got the code. The codes come in one byte per IRQ, so save e0hs.

The key will be translated by int 9, with the following special cases:

00h:
    User is pressing too many keys at once: beep or something

aah, bah 41h, eeh, fah, feh:
    Ignore it. Someone is playing with port 60h

fch, ffh:
    Ditto, but now you know the keyboard is screwed up :)

e0h 2ah:
    Well. If you let the keyboard decide what the NumLock state is (nothing to do with the LED), use it to see how the code must be translated (e0h 2ah: NumLock is on). Else, ignore. (Most Smart)

Ctrl-NumLock or Pause:
    Place system in a tight wait loop until next key pressed. It would be friendly to allow hardware IRQ's... (clock, comms etc) (I think)

Ctrl-Break:
    Clear keyboard buffer (=Equal Head and Tail), place word 0000h in buffer, invoke int 23h, and set flag at 0040h:0071h (bit 7=1).

Shift-PrtScr:
    Invoke int 5

Ctrl-PrtScr:
    redirect CON to PRN. (Teletype mode)
    Never used it, perhaps never will. Doesn't work on my keyboard driver... (DOS) Don't know how to stop it. Perhaps rehitting Ctrl-PrtScr... This is from hearsay.

SysRq:
    Invoke int 15 subfunction 85h. al->0 when pressed, 1 when released.

Ctrl-Alt-Del:
    Reboot. Here's a sample of how to reboot:

```
    mov ah,0Dh                  ; Disk Reset
    int 21h                     ; causes SmartDrv 4.x to write cache
    mov ax, 40h                 ; set up segment addressing
    mov ds,ax
    or  byte ptr ds:[17h],0Ch   ; equivalent of pressing CTRL+ALT
    mov ax,4F53h                ; Issue a "DEL" (53h = DEL scan code)
    int 15h                     ; EMM386 sees this & shuts down
    mov word ptr ds:[72h],1234h ; Set REBOOT flag to Warm-Boot (0=cold)
```

```
    db 0EAh,0h,0h,0FFh,0FFh      ; JMP FFFF:0000
```

Of course, the int 15h call should already have been done by the handler. It is also used by other caches to flush.

Shift-numkeypad:
Temporarily reverse the NumLock state, e.g. 8 becomes arrow up and vice versa.

Alt+numkeypad:
Make the pseudokey in BDA byte 19h until the alt is released, then put it in the keyboard buffer. How? Well, everytime an extra number comes in, multiply BDA:19h with 10 and add the new number.

Alt release:
See above. Just making sure it is implemented ;-)

Ctrl+a->z:
Send bytes 1 through 27

Foreign keyboards:
Some keys are accents, to be placed on the next key.

Right alt:
Some keys have three keys on it. To access them, the right alt is pressed. So remember to send the right ASCII code...

NumLock:
Switch numkeypad on/off and light/switch off LED

CapsLock:
Translate normal letters to caps or vice versa and light/switch off LED. A nice touch would be to have a distinction between CapsLock and Ctrl-CapsLock. The former would shift alfabetic keys only and the latter all keys...

ScrollLock:
Light/sw. off LED

Int 9 also has to adjust the BDA flags and keyboard buffer. In addition, the driver should warn when the keyboard buffer is full.

5. Tech Stuff

Interface:
Bidirectional, serial synchronous. The keyboard communicates via clock and data line with the system. The data comes in 11 bit packets, namely start-data-parity-stop. Parity is uneven.

```
                  =1   8bit  1bit   =0
```

Also, see further.

Data Format:
Data transfer to and from the keyboard in IBM-compatible format:
AT-, PS/2-mode: Idle state - "Data & Clock" high.
PC mode: Idle state - "Data" low, "Clock" high.

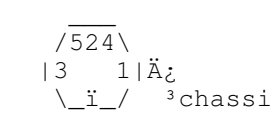
Data Output:
Open drain.

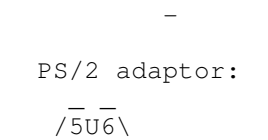
Keyboard Sequence:
Alpha-N-key-rollover.

Automatic repeat function:
All keys have auto repeat. Delay and repeat sequence can be modified through the system, but is fixed for PC-mode. (10Hz after 500ms)

Keyboard Self-diagnostic test:
After "Power-On" or upon request, the keyboard carries out a self-diagnostic test. After positive test, the keyboard sends AAh. Any other is a failure.

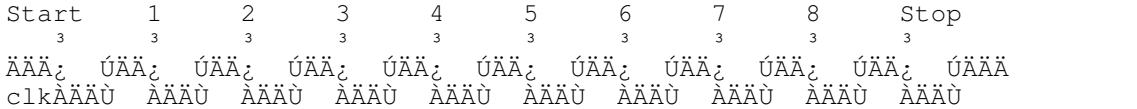
Pin assignment:





The PC-XT keyboard communication protocol

Below is a drawing of the timing of the data, send to the PC. The upper line shows the clock line, the lower the data line. The text above indicates the position of the start, data and stop bits (clocked on the negative edge of the clock line).



The communication abides to the following rules:

- On power up or reset, the PC pulls the clock line (normally high) low for at least 20 ms. When it is released (goes high again), the keyboard should send the code 0AAh to the PC to indicate its existance.

- The data is clocked in on the negative edge of the clock signal. The clock line must normally be high, the data line can be anything between transmissions. The clock line is delayed two PC clock cycles in the PC, so data changes and the negative clock edge may take place at the same time. It is safer, however, to build in a bigger delay.

- A transmission starts with a start bit (high). Then follow eight data bits, of witch bit 7 (MSB) indicates the release of the key. After that normally follows a stop bit (low), but that may be left out. In fact, due to the shift register hardware inside the PC, any number of stop bits could be send, as long as they are low. Not 100% hardware compatibles, however, may get confused then.

- After a transmission, the PC pulls the data line low until it is ready processing the data. The keyboard should wait with sending any more data until the PC releases the data line again.

A. Acknowledgments

This text was made by Wout Mertens, with the help of Tech Help of Flambeaux software, the tech spex of Cherry and some texts I found.

Information on the PC-XT keyboard communication protocol and the kbfunc.c file by Gertjan Klein (Floating somewhere in cyberspace).

Cherry is the registered trademark of Cherry Microschalter Gmbh etc.
IBM is the registered trademark of the IBM corporation
Mertens is the registered trademark of the Mertens Family ;-)

Improved copyright notice, thanks & greetings to:
    Emil Gilliam (Floating in cyberspace as well)
    Kip Cooley at the Diamond Bar BBS (909) 923-1031 (1:218/101).
    Ian Remmler at the DownTown BBS (210) 625-4479 (1:387/1001).

B. How To Contact Me

Please let me know if something is inaccurate or missing etc. Also, I would like some feedback on the quality and usability of this text. (Keeps me writing...) If you think this text is very usefull, you can always send me a nice postcard from where you live to thank me... I will then try to notify you when a new version arrives.

I am usually reachable through the Fido 80XXX echo, but you can also reach me at the following addresses:

    Fido 2:292/805.1
    SBC 14:1900/457
    DGI 68:320/1.3
    CDN 94:810/1104
    CIN 112:913/101.4

    SnailMail: Wout Mertens
               Jozef de Bomstr 62
               2018 Antwerp
               Belgium - Europe

C. The Answer To Life, The Universe And All The Rest

42.

PS: Could anyone tell me the Question?

(With thanks to Douglas Adams :-)

D. History

20 feb 94: release of v1.0
5 apr 94: changed info on rebooting in int 9
           added info about XT protocol, by Gertjan Klein
           added C program to interface with keyboard, by Gertjan Klein
           removed a bug in the copyright: Emil Gilliam was not to be
           held liable :-):-):-)
           removed some general typing errors
9 apr 94: removed bug in keyboard buffer info: Head and Tail do not
           point to a location relative to Beginning, but instead
           directly to the keyboardbuffer
23 may 94: v1.1