PS/2 Keyboard Interface

- Source: http://www.Computer-Engineering.org
- Author: Adam Chapweske

navigation

- Main Page
- Community portal
- Current events
- Recent changes
- Random page
- Help
- Donations

search

Go Search

toolbox

- What links here
- Related changes
- Upload file
- Special pages
- Printable version
- Permanent link

Contents [hide]

Legal Information

All information within this article is provided "as is" and without any express or implied warranties, including, without limitation, the implied warranties of merchantibility and fitness for a particular purpose.

This article is protected under copyright law. This document may be copied only if the source, author, date, and legal information is included.

Abstract

This article tries to cover every aspect of AT and PS/2 keyboards. It includes information on the low-level signals and protocol, scan codes, the command set, initialization, compatibility issues, and other miscellaneous information. Since it's closely related, I've also included information on the PC keyboard controller. All code samples involving the keyboard encoder are written in assembly for Microchip's PIC microcontrollers. All code samples related to the keyboard controller are written in x86 assembly

A History Lesson

The most popular keyboards in use today include:

- USB keyboard - Latest keyboard supported by all new computers (Macintosh and IBM/compatible). These are relatively complicated to interface and are not covered in this article.
- IBM/Compatible keyboards - Also known as "AT keyboards" or "PS/2 keyboards", all modern PCs support this device. They're the easiest to interface, and are the subject of this article.
- ADB keyboards - Connect to the Apple Desktop Bus of older Macintosh systems. These are not covered in this article

IBM introduced a new keyboard with each of its major desktop computer models. The original IBM PC, and later the IBM XT, used what we call the "XT keyboard." These are obsolete and differ significantly from modern keyboards; the XT keyboard is not covered in this article. Next came the IBM AT system and later the IBM PS/2. They introduced the keyboards we use today, and are the topic of this article. AT keyboards and PS/2 keyboards were very similar devices, but the PS/2 device used a smaller connector and supported a few additional features. Nonetheless, it remained backward compatible with AT systems and few of the additional features ever caught on (since software also wanted to remain backward compatible.) Below is a summary of IBM's three major keyboards.

IBM PC/XT Keyboard (1981):

- 83 keys
- 5-pin DIN connector
- Simple uni-directional serial protocol
- Uses what we now refer to as scan code set 1
- No host-to-keyboard commands

IBM AT Keyboard (1984) - Not backward compatible with XT systems(1).

- 84 -101 keys
- 5-pin DIN connector
- Bi-directional serial protocol
- Uses what we now refer to as scan code set 2
- Eight host-to-keyboard commands

IBM PS/2 Keyboard (1987) - Compatible with AT systems, not compatible with XT systems(1).

- 84 - 101 keys
- 6-pin mini-DIN connector
- Bi-direction serial protocol
- Offers optional scan code set 3
- 17 host-to-keyboard commands

The PS/2 keyboard was originally an extension of the AT device. It supported a few additional host-to-keyboard commands and featured a smaller connector. These were the only differences between the two devices. However, computer hardware has never been about standards as much as compatibility. For this reason, any keyboard you buy today will be compatible with PS/2 and AT systems, but it may not fully support all the features of the original devices. Today, "AT keyboard" and "PS/2 keyboard" refers only to their connector size. Which settings/commands any given keyboard does or does not support is anyone's guess. For example, the keyboard I'm using right now has a PS/2-style connector but only fully supports seven commands, partially supports two, and merely "acknowledges" the rest. In contrast, my "Test" keyboard has an AT-style connector but supports every feature/command of the original PS/2 device (plus a few extra.) It's important you treat modern keyboards as compatible, not standard. If your design a keyboard-related device that relies on non-general features, it may work on some systems, but not on others...

Modern PS/2 (AT) compatible keyboards:

- Any number of keys (usually 101 or 104)
- 5-pin or 6-pin connector; adaptor usually included
- Bi-directional serial protocol
- Only scan code set 2 guaranteed.
- Acknowledges all commands; may not act on all of them.

Footnote 1) XT keyboards use a completely different protocol than that used by AT and PS/2 systems, making it incompatible with the newer PCs. However, there was a transition period where some keyboard controllers supported both XT and AT (PS/2) keyboards (through a switch, jumper, or auto-sense.) Also, some keyboards were made to work on both types of systems (again, through the use of a switch or auto-sensing.) If you've owned such a PC or keyboard, don't let it fool you--XT keyboards are NOT compatible with modern computers.

General Description

Keyboards consist of a large matrix of keys, all of which are monitored by an on-board processor (called the "keyboard encoder".) The specific processor(1) varies from keyboard-to-keyboard but they all basically do the same thing: Monitor which key(s) are being pressed/released and send the appropriate data to the host. This processor takes care of all the debouncing and buffers any data in its 16-byte buffer, if needed. Your motherboard contains a "keyboard controller"(2) that is in charge of decoding all of the data received from the keyboard and informing your software of what's going on. All communication between the host and the keyboard uses an IBM protocol.

Footnote 1) Originally, IBM used the Intel 8048 microcontroller as its keyboard encoder. There are now a wide variety of keyboard encoder chips available from many different manufacturers.

Footnote 2) Originally, IBM used the Intel 8042 microcontroller as its keyboard controller. This has since been replaces with compatible devices integrated in motherboards' chipsets. The keyboard controller is covered later in this article.

Electrical Interface / Protocol

The AT and PS/2 keyboards use the same protocol as the PS/2 mouse. Click here for detailed information on this protocol.

Scan Codes

Your keyboard's processor spends most of its time "scanning", or monitoring, the matrix of keys. If it finds that any key is being pressed, released, or held down, the keyboard will send a packet of information known as a "scan code" to your computer. There are two different types of scan codes: "make codes" and "break codes". A make code is sent when a key is pressed or held down. A break code is sent when a key is released. Every key is assigned its own unique make code and break code so the host can determine exactly what happened to which key by looking at a single scan code. The set of make and break codes for every key comprises a "scan code set". There are three standard scan code sets, named one, two, and three. All modern keyboards default to set two.(1)

So how do you figure out what the scan codes are for each key? Unfortunately, there's no simple formula for calculating this. If you want to know what the make code or break code is for a specific key, you'll have to look it up in a table. I've composed tables for all make codes and break codes in all three scan code sets:

- Scan Code Set 1 - Original XT scan code set; supported by some modern keyboards
- Scan Code Set 2 - Default scan code set for all modern keyboards
- Scan Code Set 3 - Optional PS/2 scan code set, rarely used

Footnote 1) Originally, the AT keyboard only supported set two, and the PS/2 keyboard would default to set two but supported all three. Most modern keyboards behave like the PS/2 device, but I have come across a few that didn't support set one, set three, or both. Also, if you've ever done any low-level PC programming, you've probably notice the keyboard controller supplies set ONE scan codes by default. This is because the keyboard controller converts all incomming scan codes to set one (this stems from retaining compatibility with software written for XT systems.) However, it's still set two scan codes being sent down the keyboard's serial line.

Make Codes, Break Codes, and Typematic Repeat

Whenever a key is pressed, that key's make code is sent to the computer. Keep in mind that a make code only represents a key on a keyboard--it does not represent the character printed on that key. This means that there is no defined relationship between a make code and an ASCII code. It's up to the host to translate scan codes to characters or commands.

Although most set two make codes are only one-byte wide, there are a handfull of "extended keys" whose make codes are two or four bytes wide. These make codes can be identified by the fact that their first byte is E0h.

Just as a make code is sent to the computer whenever a key is pressed, a break code is sent whenever a key is released. In addition to every key having its own unique make code, they all have their own unique break code(1). Fortunately, however, you won't always have to use lookup tables to figure out a key's break code--certain relationships do exist between make codes and break codes. Most set two break codes are two bytes long where the first byte is F0h and the second byte is the make code for that key. Break codes for extended keys are usually three bytes long where the first two bytes are E0h, F0h, and the last byte is the last byte of that key's make code. As an example, I have listed below a the set two make codes and break codes for a few keys:

Key	Make Code	Break Code
"A"	1C	F0,1C
"5"	2E	F0,2E
"F10"	09	F0,09
Right Arrow	E0,74	E0,F0,74
Right Ctrl	E0,14	E0,F0,14

Example: What sequence of make codes and break codes should be sent to your computer for the character "G" to appear in a word processor? Since this is an upper-case letter, the sequence of events that need to take place are: press the "Shift" key, press the "G" key, release the "G" key, release the "Shift" key. The scan codes associated with these events are the following: make code for the "Shift" key (12h), make code for the "G" key (34h), break code for the "G" key(F0h,34h), break code for the "Shift" key (F0h,12h). Therefore, the data sent to your computer would be: 12h, 34h, F0h, 34h, F0h, 12h.

If you press a key, its make code is sent to the computer. When you press and hold down a key, that key becomes typematic, which means the keyboard will keep sending that key's make code until the key is released or another key is pressed. To verify this, open a text editor and hold down the "A" key. When you first press the key, the character "a" immediately appears on your screen. After a short delay, another "a" will appear followed by a whole stream of "a"s until you release the "A" key. There are two important parameters here: the typematic delay, which is the short delay between the first and second "a", and the typematic rate, which is how many characters per second will appear on your screen after the typematic delay. The typematic delay can range from 0.25 seconds to 1.00 second and the typematic rate can range from 2.0 cps (characters per second) to 30.0 cps. You may change the typematic rate and delay using the "Set Typematic Rate/Delay" (0xF3) command.

Typematic data is not buffered within the keyboard. In the case where more than one key is held down, only the last key pressed becomes typematic. Typematic repeat then stops when that key is released, even though other keys may be held down.

Footnote 1) Actually, the "Pause/Break" key does not have a break code in scan code sets one and two. When this key is pressed, its make code is sent; when it's released, it doesn't send anything. So how do you tell when this key has been released? You can't.

Reset

At power-on or software reset (see the "Reset" command) the keyboard performs a diagnostic self-test referred to as BAT (Basic Assurance Test) and loads the following default values:

- Typematic delay 500 ms.
- Typematic rate 10.9 cps.
- Scan code set 2.
- Set all keys typematic/make/break.

When entering BAT, the keyboard enables its three LED indicators, and turns them off when BAT has completed. At this time, a BAT completion code of either 0xAA (BAT successful) or 0xFC (Error) is sent to the host. This BAT completion code must be sent 500~750 milliseconds after power-on.

Many of the keyboards I've tested ignore their CLOCK and DATA lines until after the BAT completion code has been sent. Therefore, an "Inhibit" condition (CLOCK line low) may not prevent the keyboard from sending its BAT completion code.

Command Set

A few notes regarding commands the host can issue to the keyboard:

- The keyboard clears its output buffer when it recieves any command.
- If the keyboard receives an invalid command or argument, it must respond with "resend" (0xFE).
- The keyboard must not send any scancodes while processing a command.
- If the keyboard is waiting for an argument byte and it instead receives a command, it should discard the previous command and process this new one.

Below are all the commands the host may send to the keyboard:

- 0xFF (Reset) - Keyboard responds with "ack" (0xFA), then enters "Reset" mode. (See "Reset" section.)
- 0xFE (Resend) - Keyboard responds by resending the last-sent byte. The exception to this is if the last-sent byte was "resend" (0xFE). If this is the case, the keyboard resends the last non-0xFE byte. This command is used by the host to indicate an error in reception.

The next six commands can be issued when the keyboard is in any mode, but it only affects the behavior of the keyboard when in "mode 3" (ie, set to scan code set 3.)

- 0xFD (Set Key Type Make) - Disable break codes and typematic repeat for specified keys. Keyboard responds with "ack" (0xFA), then disables scanning (if enabled) and reads a list of keys from the host. These keys are specified by their set 3 make codes. Keyboard responds to each make code with "ack". Host terminates this list by sending an invalid set 3 make code (eg, a valid command.) The keyboard then re-enables scanning (if previously disabled).
- 0xFC (Set Key Type Make/Break) - Similar to previous command, except this one only disables typematic repeat.
- 0xFB (Set Key Type Typematic) - Similar to previous two, except this one only disables break codes.
- 0xFA (Set All Keys Typematic/Make/Break) - Keyboard responds with "ack" (0xFA). Sets all keys to their normal setting (generate scan codes on make, break, and typematic repeat)
- 0xF9 (Set All Keys Make) - Keyboard responds with "ack" (0xFA). Similar to 0xFD, except applies to all keys.
- 0xF8 (Set All Keys Make/Break) - Keyboard responds with "ack" (0xFA). Similar to 0xFC, except applies to all keys.
- 0xF7 (Set All Keys Typematic) - Keyboard responds with "ack" (0xFA). Similar to 0xFB, except applies to all keys.
- 0xF6 (Set Default) - Load default typematic rate/delay (10.9cps / 500ms), key types (all keys typematic/make/break), and scan code set (2).
- 0xF5 (Disable) - Keyboard stops scanning, loads default values (see "Set Default" command), and waits further instructions.
- 0xF4 (Enable) - Re-enables keyboard after disabled using previous command.
- 0xF3 (Set Typematic Rate/Delay) - Host follows this command with one argument byte that defines the typematic rate and delay as follows:

Repeat Rate

Bits 0-4	Rate (cps)	-	Bits 0-4	Rate (cps)	-	Bits 0-4	Rate (cps)	-	Bits 0-4	Rate (cps)
00h	30.0		08h	15.0		10h	7.5		18h	3.7
01h	26.7		09h	13.3		11h	6.7		19h	3.3
02h	24.0		0Ah	12.0		12h	6.0		1Ah	3.0
03h	21.8		0Bh	10.9		13h	5.5		1Bh	2.7
04h	20.7		0Ch	10.0		14h	5.0		1Ch	2.5
05h	18.5		0Dh	9.2		15h	4.6		1Dh	2.3
06h	17.1		0Eh	8.6		16h	4.3		1Eh	2.1
07h	16.0		0Fh	8.0		17h	4.0		1Fh	2.0

Delay

Bits 5-6	Delay (seconds)
00b	0.25
01b	0.50
10b	0.75
11b	1.00

- 0xF2 (Read ID) - The keyboard responds by sending a two-byte device ID of 0xAB, 0x83. (0xAB is sent first, followed by 0x83.)
- 0xF0 (Set Scan Code Set) - Keyboard responds with "ack", then reads argument byte from the host. This argument byte may be 0x01, 0x02, or 0x03 to select scan code set 1, 2, or 3, respectively. The keyboard responds to this argument byte with "ack". If the argument byte is 0x00, the keyboard responds with "ack" followed by the current scan code set.
- 0xEE (Echo) - The keyboard responds with "Echo" (0xEE).
- 0xED (Set/Reset LEDs) - The host follows this command with one argument byte, that specifies the state of the keyboard's Num Lock, Caps Lock, and Scroll Lock LEDs. This argument byte is defined as follows:

Always 0	Always 0	Always 0	Always 0	Always 0	Caps Lock	Num Lock	Scroll Lock

- "Scroll Lock" - Scroll Lock LED off(0)/on(1)
- "Num Lock" - Num Lock LED off(0)/on(1)
- "Caps Lock" - Caps Lock LED off(0)/on(1)

Initialization

The following is the communication between my computer and keyboard when it boots-up. I beleive the first three commands were initiated by the keyboad controller, the next command (which enables Num lock LED) was sent by the BIOS, then the rest of the commands were sent my the OS (Win98SE). Remember, these results are specific to my computer, but it should give you a general idea as to what happens at startup.

```
Keyboard: AA  Self-test passed              ;keyboard controller init
Host:     ED  Set/Reset Status Indicators
Keyboard: FA  Acknowledge
Host:     00  Turn off all LEDs
Keyboard: FA  Acknowledge
Host:     F2  Read ID
Keyboard: FA  Acknowledge
Keyboard: AB  First byte of ID
Keyboard: 83  Set/Reset Status Indicators    ;BIOS init
Keyboard: FA  Acknowledge
Host:     02  Turn on Num Lock LED
Keyboard: FA  Acknowledge
Host:     F3  Set Typematic Rate/Delay       ;Windows init
Keyboard: FA  Acknowledge
Host:     20  500 ms / 30.0 reports/sec
Keyboard: FA  Acknowledge
Host:     F4  Enable
Keyboard: FA  Acknowledge
Host:     F3  Set Typematic Rate/delay
Keyboard: FA  Acknowledge
Host:     00  250 ms / 30.0 reports/sec
Keyboard: FA  Acknowledge
```