第一章　PS/2 鼠标、键盘协议

Introduction:

引言：

The PS/2 device interface, used by many modern mice and keyboards, was developed by IBM and originally appeared in the IBM Technical Reference Manual. However, this document has not been printed for many years and as far as I know, there is currently no official publication of this information. I have not had access to the IBM Technical Reference Manual, so all information on this page comes from my own experiences as well as help from the references listed at the bottom of this page.
PS/2 设备接口用于许多现代的鼠标和键盘，它是由 IBM 开发并且最初出现在 IBM 技术参考手册里。但是，当我知道的时候这篇文件就已经很多年没有印刷了，因此关于这个内容现在没有官方的出版物。我无法访问 IBM 的技术参考手册，所以本网页中的所有信息都来自于我自己的经验及本页最下面列出的参考的帮助。
（译者注：这些参考的条目在本章的结尾处。）

This document descibes the interface used by the PS/2 mouse, PS/2 keyboard, and AT keyboard. I'll cover the physical and electrical interface, as well as the protocol. If you need higher-level information, such as commands, data packet formats, or other information specific to the keyboard or mouse, I have written separate documents for the two devices:
这个文件描述了用于 PS/2 鼠标、PS/2 键盘及 AT 键盘的接口。我将论及物理和电气接口也包括协议。如果你需要更高级的信息，诸如命令、数据包的格式或者其他关于键盘鼠标的特别细节，那么我对这两种设备写了独立的文件：

The PS/2 (AT) Keyboard Interface
The PS/2 Mouse Interface
（译者注：这两篇文章已经包含到这篇译文中来了，是第二章和第三章。）

I also encourage you to check out my homepage for more information related to this topic, including projects, code, and links related to the mouse and keyboard.
我同样鼓励你在我的主页上校对更多与这个话题相关的信息，包括工程、代码和与鼠标键盘有关的链接。

The Connector:

连接器：

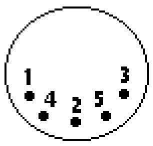
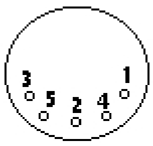
The physical PS/2 port is one of two styles of connectors: The 5-pin DIN or the 6-pin mini-DIN. Both connectors are completely (electrically) similar; the only practical difference between the two is the arrangement of pins. This means the two types of connectors can easily be changed with simple hard-wired adaptors. These cost about $6 each or you can make your own by matching the pins on any two connectors. The DIN standard was created by the German Standardization Organization (Deutsches Institut fuer Norm) . Their website is at http://www.din.de/ (this site is in German, but most of their pages are also available in English.)
物理上的 PS/2 端口是两类连接器中的一种：5 脚的 DIN 或 6 脚的 mini-DIN。这两种连接器（在电气特性上）是十分类似的，实际上两者只有一点不同那就是管脚的排列。这就意味着这两类连接器可以很容易用一种简单的硬件连线的适配器来转换。这种适配器大约每个值 6 美元，或者你可以根据任意两种连接器的对应管脚关系做你自己的适配器。DIN 标准是由德国标准化组织(Deutsches Institut fuer Norm)建立的。他们的网站在 http://www.din.de/（这个站点是德文的，但他们的很多网页同样可用于英文）。

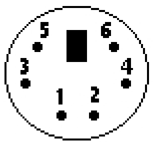
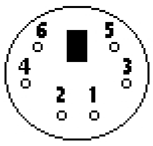
PC keyboards can have either a 6-pin mini-DIN or a 5-pin DIN connector. If your keyboard has a 6-pin mini-DIN and your computer has a 5-pin DIN (or visa versa), the two can be made compatible with the adaptors described above. Keyboards with the 6-pin mini-DIN are often referred to as "PS/2" keyboards, while those with the 5-pin DIN are called "AT" devices ("XT" keyboards also used the 5-pin DIN, but they are quite old and haven't been made for many years.) All modern keyboards built for the PC are either PS/2, AT, or USB. This document does not apply to USB devices, which use a completely different interface.
PC 键盘可以有 6 脚的 mini-DIN 或 5 脚的 DIN 连接器。如果你的键盘是 6 脚的 mini-DIN 而你的计算机是 5 脚的 DIN（或者相反），这两类连接器可以用上面提到的适配器来兼容。具有 6 脚 mini-DIN 的键盘通常被叫做“PS/2”键盘，而那些有 5 脚 DIN 叫做“AT”设备（“XT”键盘也使用 5 脚 DIN，但它们非常古老并且多年前就不生产了）。所有现代的为 PC 建造的键盘不是 PS/2,AT 就是 USB 的。这篇文章不适用于 USB 设备，它们使用了一种完全不同的接口。

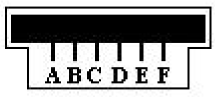
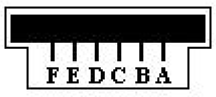
Mice come in a number of shapes and sizes (and interfaces.) The most popular type is probably the PS/2 mouse, with USB mice gaining popularity. Serial mice are also quite popular, but the computer industry is abandoning them in support of USB and PS/2 devices. This document applies only to PS/2 mice. If you want to interface a serial mouse, check out Microchip's appnote #519, "Implementing a Simple Serial Mouse Controller."
鼠标流行着大量的形状和大小（和接口），最流行的类型可能算是 PS/2 鼠标，现在 USB 鼠标渐渐开始流行起来了。串行鼠标同样非常流行，但计算机工业放弃了它们转而支持 USB 和 PS/2 设备。这篇文章仅适用于 PS/2 鼠标。如果你要知道一个串行鼠标的接口，请查验 Microchip 的 519 号应用“实现一个简单的串行鼠标控制器”。

As a side note, there is one other type of connector you may run into on keyboards. While most keyboard cables are hard-wired to the keyboard, there are some whose cable is not permanently attached and come as a separate component. These cables have a DIN connector on one end (the end that connects to the computer) and a SDL (Sheilded Data Link) connector on the keyboard end. SDL was created by a company called "AMP." This connector is somewhat similar to a telephone connector in that it has wires and springs rather than pins, and a clip holds it in place. If you need more information on this connector, you might be able to find it on AMP's website at http://www.connect.amp.com/. I have only seen this type of connector on (old) XT keyboards, although there may be AT keyboards that also use the SDL. Don't confuse the SDL connector with the USB connector--they probably both look similar in my diagram below, but they are actually very different. Keep in mind that the SDL connector has springs and moving parts, while the USB connector does not.
作为边注，还有另外一种类型的连接器你可以在键盘上碰到。虽然多数键盘电缆都是固定连接到键盘的，但还有一些键盘电缆不是永久挂接的，而是作为一个单独的部件。这种电缆在一端有一个 DIN 的连接器（这端连接到计算机）而连接到键盘的那一端是一个 SDL（屏蔽的数据连接器）连接器。SDL 是由一个叫 AMP 的公司建立的。这种连接器有点象电话连接器，它有金属丝和弹簧代替管脚，用一个卡子保持它在适当的位置上。如果你需要知道关于这种连接器的更多信息，你可以在 AMP 的网站找到它，AMP 的网站在 http://www.connect.amp.com/。尽管可能 AT 键盘同样可以使用 SDL，但我只在一个很老的 XT 键盘上看见过这种连接器。不要把 SDL 连接器和 USB 连接器混淆起来，这两个可能在我下面画的示意图中看起来有点类似，但是它们实际上是非常不同的。记住 SDL 有弹簧和活动部件，而 USB 连接器没有。

The pinouts for each connector are shown below:
每种连接器的引脚定义如下所示：

Male 公的	Female 母的	5-pin DIN (AT/XT):	5 脚 DIN(AT/XT)
		1 - Clock	1－时钟
		2 - Data	2－数据
		3 - Not Implemented	3－未实现，保留
		4 - Ground	4－电源地
(Plug) 插头	(Socket) 插座	5 - +5v	5－电源+5V

Male 公的	Female 母的	6-pin Mini-DIN (PS/2):	6 脚 Mini-DIN(PS/2)
		1 - Data	1－数据
		2 - Not Implemented	2－未实现，保留
		3 - Ground	3－电源地
		4 - +5v	4－电源+5V
(Plug) 插头	(Socket) 插座	5 - Clock	5－时钟
		6 - Not Implemented	6－未实现，保留

		6-pin SDL:	6 脚 SDL
		A - Not Implemented	A－未实现，保留
		B - Data	B－数据
		C - Ground	C－电源地
		D - Clock	D－时钟
		E - +5v	E－电源+5V
		F - Not Implemented	F－未实现，保留

General Description:
一般性描述：

(Note: Throughout this document, I may use the more general term "host" to refer to the computer--or whatever the keyboard/mouse is connected to-- and the term "device" will refer to the keyboard/mouse.)
（注意：遍及这篇文章，我使用了更普通的术语“host”使之计算机或者键盘/鼠标连接到的单元，而术语“device”是指键盘/鼠标）
（译者注：下面的译文中 host 可能被翻译成主机。）

There are four interesting pins on the connectors just described: Ground, +5v, Data, and Clock. The +5V is supplied by the host (computer) and the keyboard/mouse's ground is connected to the host's electrical ground. Data and Clock are both open collector, which means they are normally held at a high logic level but can easily be pulled down to ground (logic 0.) Any device you connect to a PS/2 mouse, keyboard, or host should have large pull-up resistors on the Clock and Data lines. You apply a "0" by pulling the line low and you apply a "1" by letting the line float high. Refer to Figure 1 for a general interface to Data and Clock. (Note: if you are going to use a microcontroller such as the PIC, where I/O is bidirectional, you may skip the transistors and buffers and use the same pin for both input and output. With this configuration, a "1" is asserted by setting the pin to input and let the resistor pull the line high. A "0" is then asserted by changing the pin to output and write a "0" to that pin, which will pull the line to ground.)
在刚才提到连接器上有四个有趣的管脚：电源地、＋5V、数据和时钟。host（计算机）提供＋5V，并且键盘/鼠标的地连接到 host 的电源地上。数据和时钟都是集电极开路的，这就意味着它们通常保持高电平而且很容易下拉到地（逻辑 0）。任何你连接到 PS/2 鼠标、键盘或 host 的设备在时钟和数据线上要

有一个大的上拉电阻。置“0”就把线拉低，置“1”就让线上浮成高电平。参考图 1 中数据和时钟线的一般接口结构。(注意：如果你打算使用象 PIC 这样的微控制器，由于它们的 I/O 管脚是双向的，你可以跳过晶体管和缓冲门，并且通用同一个管脚进行输入和输出。在这种组态情况下，要设置管脚为输入就写入 1 使得电阻上拉线上的电平，要改变管脚为输出就写入 0 到那个管脚，把线路下拉到地。)

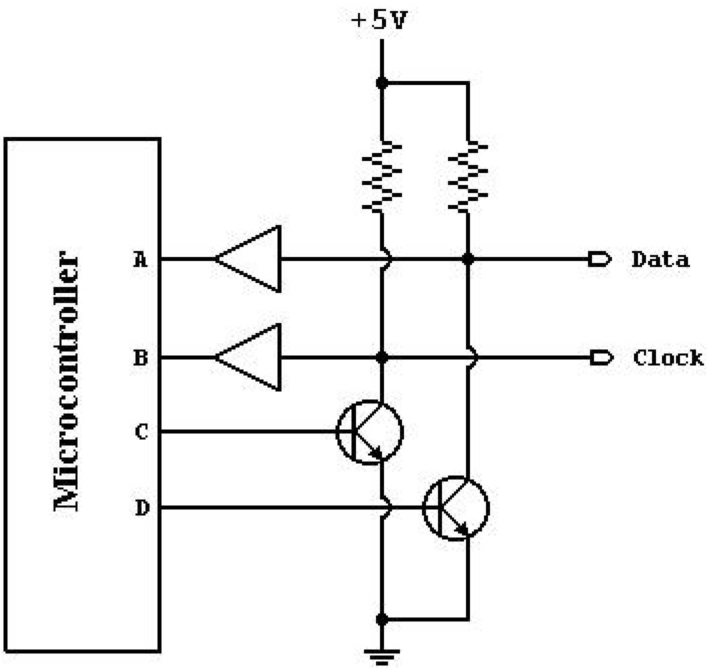


Figure 1: Open-collector interface to Data and Clock. Data and Clock are read on the microcontroller's port A and B, respectively. Both lines are normally held at +5V, but can be pulled to ground by asserting logic 1 on C and D. As a result, Data equals D, inverted, and Clock equals C, inverted.

图 1：数据线和时钟线的集电极开路接口。数据和时钟分别由微控制器的 A 端口和 B 端口读入。这两条线通常保持+5V，但可以往端口 C 和 D 写入 1 来拉到地。结果是，数据是 D 的反相，时钟是 C 的反相。

The PS/2 mouse and keyboard implement a bidirectional synchronous serial protocol. In other words, Data is sent one bit at a time on the Data line and is read on each time Clock is pulsed. The keyboard/mouse can send data to the host and the host can send data to the device, but the host always has priority over the bus and can inhibit communication from the keyboard/mouse at any time by holding Clock low.
PS/2 鼠标和键盘履行一种双向同步串行协议。换句话说，每次数据线上发送一位数据并且每在时钟线上发一个脉冲就被读入。键盘/鼠标可以发送数据到主机，而主机也可以发送数据到设备，但主机总是在总线上有优先权，它可以在任何时候抑制来自于键盘/鼠标的通讯，只要把时钟拉低即可。

Data sent from the keyboard/mouse to the host is read on the falling edge of the clock signal (when Clock goes from high to low); data sent from the host to the keyboard/mouse is read on the rising edge (when Clock goes from low to high.) Regardless of the direction of communication, the keyboard/mouse always generates the clock signal. If the host wants to send data, it must first tell the device to start generating a clock signal (that process is described in the next section.) The maximum clock frequency is 33 kHz and most devices operate within 10-20kHz. If you want to build a PS/2 device, I would recommend keeping this frequency around 15 kHz. This means Clock should be high for about 40 microseconds and low for 40 microseconds.
从键盘/鼠标发送到主机的数据在时钟信号的下降沿（当时钟从高变到低的时候）被读取；从主机发送到键盘/鼠标的数据在上升沿（当时钟从低变到高的时候）被读取。不管通讯的方向怎样，键盘/鼠标总是产生时钟信号。如果主机要发送数据，它必须首先告诉设备开始产生时钟信号（这个过程在下一章节中被描述）。最大的时钟频率是 33kHz，而且大多数设备工作在 10－20kHz。如果你要制作一个 PS/2 设备，我推荐你把频率控制在 15kHz 左右。这就意味着时钟应该是高 40 微秒低 40 微秒。

All data is arranged in bytes with each byte sent in a frame consisting of 11-12 bits. These bits are:
所有数据安排在字节中，每个字节为一帧，包含了 11－12 个位。这些位的含义如下：

1 start bit. This is always 0.	1 个起始位，总是为 0
8 data bits, least significant bit first.	8 个数据位，低位在前

1 parity bit (odd parity).	1 个校验位，奇校验
1 stop bit. This is always 1.	1 个停止位，总是为 1
1 acknowledge bit (Host-to-device communication only)	1 个应答位（仅在主机对设备的通讯中）

The parity bit is set if there is an even number of 1's in the data bits and reset (0) if there is an odd number of 1's in the data bits. The number of 1's in the data bits plus the parity bit always add up to an odd number (odd parity.) This is used for error detection.
如果数据位中包含偶数个 1，校验位就会置 1；如果数据位中包含奇数个 1，校验位就会置 0。数据位中 1 的个数加上校验位总为奇数（这就是奇校验）。这是用来错误检测。

When the host is sending data to the keyboard/mouse, a handshaking bit is sent from the device to acknowledge the packet was received. This bit is not present when the device sends data to the host.
当主机发送数据给键盘/鼠标时，设备回送一个握手信号来应答数据包已经收到。这个位不会出现在设备发送数据到主机的过程中。

Device-to-Host Communication:
设备到主机的通讯过程：

The Data and Clock lines are both open collector (normally held at a high logic level.) When the keyboard or mouse wants to send information, it first checks Clock to make sure it's at a high logic level. If it's not, the host is inhibiting communication and the device must buffer any to-be-sent data until it regains control of the bus (the keyboard has a 16-byte buffer and the mouse's buffer stores only the last packet sent.) If the Clock line is high, the device can begin to transmit its data.
数据和时钟线都是集电极开路结构（正常保持高电平）。当键盘或鼠标等待发送数据时，它首先检查时钟以确认它是否是高电平。如果不是，那么是主机抑制了通讯，设备必须缓冲任何要发送的数据直到重新获得总线的控制权（键盘有 16 字节的缓冲区，而鼠标的缓冲区仅存储最后一个要发送的数据包）。如果时钟线是高电平，设备就可以开始传送数据。

As I mentioned in the previous section, the keyboard and mouse use a serial protocol consisting of 11-bit frames. These bits are:
如我在上一节提及的，键盘和鼠标使用一种每帧包含 11 位的串行协议。这些位含义是：

1 start bit. This is always 0.	1 个起始位，总是为 0
8 data bits, least significant bit first.	8 个数据位，低位在前
1 parity bit (odd parity).	1 个校验位，奇校验
1 stop bit. This is always 1.	1 个停止位，总是为 1

Each bit is read by the host on the falling edge of the clock, as is illustrated in Figures 2 & 3.
每位在时钟的下降沿被主机读入，如图 2 和 3 所示：

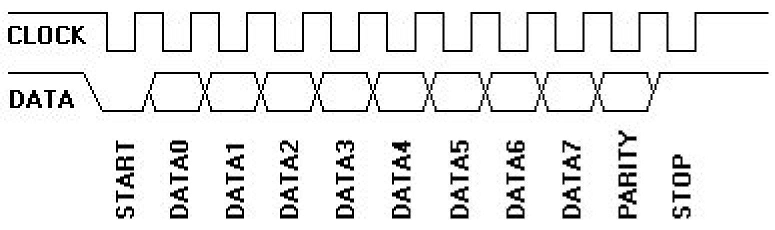


Figure 2: Device-to-host communication. The Data line changes state when Clock is high and that data is latched on the falling edge of the clock signal.

图 2：设备到主机的通讯。当时钟为高，数据线改变状态，在时钟信号的下降沿数据被锁存。

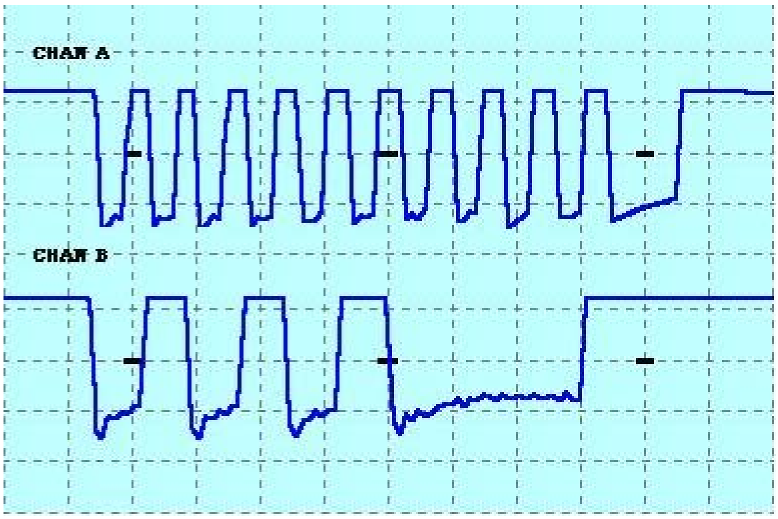


Figure 3: Scan code for the "Q" key (15h) being sent from a keyboard to the computer. Channel A is the Clock signal; channel B is the Data signal.
图 3："Q"键的扫描码从键盘发送到计算机。通道 A 是时钟信号，通道 B 是数据信号。

The clock frequency is 10-16.7kHz. The time from the rising edge of a clock pulse to a Data transition should be at least 5 microseconds. The time from a data transition to the falling edge of a clock pulse should be at least 5 microseconds and no greater than 25 microseconds. This timing is very important--you should follow it exactly. The host may pull the line low before the 11th clock pulse (stop bit), causing the device to abort sending the current byte (this is very rare.) After the stop bit is transmitted, the device should wait at least 50 microseconds before sending the next packet. This gives the host time to inhibit transmission while it processes the received byte (the host will usually automatically do this after each packet is received.) The device should wait at least 50 microseconds after the host releases an inhibit before sending any data.
时钟频率为 10－16.7kHz。从时钟脉冲的上升沿到一个数据转变的时间至少要有 5 微秒。数据变化到时钟脉冲的下降沿的时间至少要有 5 微秒并且不大于 25 微秒。这个定时非常重要－你应该严格遵循它。主机可以在第 11 个时钟脉冲（停止位）之前把线拉低，导致设备放弃发送当前字节（这是非常罕见的）。在停止位发送后，设备在发送下个包前至少应该等待 50 毫秒。这将给主机时间当它处理接收到的字节时抑制发送（主机在收到每个包时，通常自动做这个）。在主机释放抑制后，设备至少应该在发送任何数据前等 50 毫秒。

I would recommend the following process for sending a single byte from an emulated keyboard/mouse to the host:
我推荐下面的过程发送一个单一字节从仿真键盘/鼠标到主机：

```
1)  Wait for Clock = high.
2)  Delay 50 microseconds.
3)  Clock still = high?
        No--goto step 1
4)  Data = high?
        No--Abort (and read byte from host)
5)  Delay 20 microseconds (=40 microseconds to the time Clock is pulled low in sending the start bit.)
6)  Output Start bit (0)     \   After sending each of these bits, test
7)  Output 8 data bits        > Clock to make sure host hasn't pulled it
8)  Output Parity bit        /       low (which would abort this transmission.)
9)  Output Stop bit (1)
10)  Delay 30 microseconds (=50 microseconds from the time Clock is released in sending the stop bit)
1)  等待 Clock = high。
2)  延时  50  微秒。
```

3) Clock s 仍旧为 high?
   No—到第 1 步
4) Data = high?
   No—放弃 (并且从主机读取字节)
5) 延迟 20 毫秒 (=40 微秒 to the time Clock is pulled low in sending the start bit.)
6) 输出起始位 (0) \ 在发送所有这些位的每一位后，
7) 输出 8 个数据位 > 测试时钟确认主机是否把它拉低了
8) 输出校验位 / （这说明主机要放弃这次传送）
9) 输出停止位 (1)
10) 延迟 30 毫秒 (=50 微秒 from the time Clock is released in sending the stop bit)

The process for sending a single bit should then be as follows:
按如下的过程发送单个位：

1) Set/Reset Data
2) Delay 20 microseconds
3) Bring Clock low
4) Delay 40 microseconds
5) Release Clock
6) Delay 20 microseconds
1) 设置/复位数据
2) 延迟 20 微秒
3) 把时钟拉低
4) 延迟 40 微秒
5) 释放时钟
6) 延迟 20 微秒

Here is some sample code written for the PIC16F84 that follows the above algorithms to send a byte to the host. "Delay" is a self-explanitory macro; "CLOCK" and "DATA" are the bits connected to the Clock and Data lines; "TEMP0", "PARITY", and "COUNTER" are all general purpose registers. Note that in the "PS2outBit" routine, the Data and Clock lines are brought low by setting the appropriate I/O pin to output (it's assumed their output was set to "0" at the beginning of the program.) And they are allowed to float (high) by setting the I/O pin to input (and allow a pull-up resistor to pull the line high.) This was written for a PIC running at 4.61 MHz +/- 25% (RC oscillator: 5k/20pF). This is very important for timing considerations.
这里有一些采用上面算法来发送一个字节到主机的 PIC16F84 的简单样例代码。“Delay”是一自说明宏；“CLOCK”和“DATA”是连接到时钟和数据线上的位单元；“TEMP0”、“PARITY”和“COUNTER”都是一般目的的寄存器。注意在“PS2outBit”例程中，数据线和时钟线通过设置适当的 I/O 脚为输出态来拉低电平（在程序的开始，它们的输出被置为 0），通过置 I/O 脚为输出，它们就允许上浮为高电平（允许上来电阻拉线路为高电平）。下面给 PIC 用的代码运行于 4.61MHz +/-25%（RC 振荡器：5k/20pF），这个值对定时条件非常重要。

```
ByteOut        movwf    TEMP0           ;Save to-be-sent byte
InhibitLoop    btfss    CLOCK           ;Check for inhibit
               goto     InhibitLoop
               Delay    50              ;Delay 50 microseconds
```

```
                btfss   CLOCK           ;Check again for inhibit
                goto    InhibitLoop
                btfss   DATA            ;Check for request-to-send
                retlw   0xFF
                clrf    PARITY          ;Init reg for parity calc
                movlw   0x08
                movwf   COUNTER
                movlw   0x00
                call    BitOut          ;Output Start bit (0)
                btfss   CLOCK           ;Test for inhibit
                goto    ByteOutEnd
                Delay   4
ByteOutLoop     movf    TEMP0, w
                xorwf   PARITY, f       ;Calculate parity
                call    BitOut          ;Output Data bits
                btfss   CLOCK           ;Test for inhibit
                goto    ByteOutEnd
                rrf     TEMP0, f
                decfsz  COUNTER, f
                goto    ByteOutLoop
                Delay   2
                comf    PARITY, w
                call    BitOut          ;Output Parity bit
                btfss   CLOCK           ;Test for inhibit
                goto    ByteOutEnd
                Delay   5
                movlw   0xFF
                call    BitOut          ;Output Stop bit (1)
                Delay   48
                retlw   0x00
ByteOutEnd      bsf     STATUS, RP0     ;Host has aborted
                bsf     DATA            ;DATA=1
                bsf     CLOCK           ;CLOCK=1
                bcf     STATUS, RP0
                retlw   0xFE

BitOut          bsf     STATUS, RP0
                andlw   0x01
                btfss   STATUS, Z
                bsf     DATA
                btfsc   STATUS, Z
                bcf     DATA
                Delay   21
                bcf     CLOCK
                Delay   45
                bsf     CLOCK
```

```
bcf     STATUS, RP0
Delay   5
return
```

Host to Device Communication:
主机到设备的通讯：

The packet is sent a little differently in host-to-device communication...
被发送的包有点不同于主机到设备通讯过程。

First of all, the PS/2 device always generates the clock signal. If the host wants to send data, it must first put the Clock and Data lines in a "Request-to-send" state as follows:
首先，PS/2 设备总是产生时钟信号。如果主机要发送数据，它必须首先把时钟和数据线设置为“请求发送”状态，如下示：

Inhibit communication by pulling Clock low for at least 100 microseconds.
通过下拉时钟线至少 100 微秒来抑制通讯。
Apply "Request-to-send" by pulling Data low, then release Clock.
通过下拉数据线来应用“请求发送”，然后释放时钟。

The device should check for this state at intervals not to exceed 10 milliseconds. When the device detects this state, it will begin generating Clock signals and clock in eight data bits and one stop bit. The host changes the Data line only when the Clock line is low, and data is latched on the rising edge of the clock pulse. This is opposite of what occours in device-to-host communication.
设备应该在不超过 10 毫秒的间隔内就要检查这个状态。当设备检测到这个状态，它将开始产生时钟信号，并且时钟脉冲标记下输入八个数据位和一个停止位。主机仅当时钟线为低的时候改变数据线，而数据在时钟脉冲的上升沿被锁存。这在发生在设备到主机通讯的过程中正好相反。

After the stop bit is sent, the device will acknowledge the received byte by bringing the Data line low and generating one last clock pulse. If the host does not release the Data line after the 11th clock pulse, the device will continue to generate clock pulses until the the Data line is released (the device will then generate an error.)
在停止位发送后，设备要应答接收到的字节，就把数据线拉低并产生最后一个时钟脉冲。如果主机在第 11 个时钟脉冲后不释放数据线，设备将继续产生时钟脉冲直到数据线被释放（然后设备将产生一个错误）。

The Host may abort transmission at time before the 11th clock pulse (acknowledge bit) by holding Clock low for at least 100 microseconds.
主机可以在第 11 个时钟脉冲（应答位）前中止一次传送，只要下拉时钟线至少 100 微秒。

To make this process a little easier to understand, here's the steps the host must follow to send data to a PS/2 device:
要使得这个过程易于理解，主机必须按下面的步骤发送数据到 PS/2 设备：

1) Bring the Clock line low for at least 100 microseconds.
2) Bring the Data line low.
3) Release the Clock line.
4) Wait for the device to bring the Clock line low.

5) Set/reset the Data line to send the first data bit
6) Wait for the device to bring Clock high.
7) Wait for the device to bring Clock low.
8) Repeat steps 5-7 for the other seven data bits and the parity bit
9) Release the Data line.
10) Wait for the device to bring Data low.
11) Wait for the device to bring Clock low.
12) Wait for the device to release Data and Clock
1) 把时钟线拉低至少 100 微秒。
2) 把数据线拉低。
3) 释放数据线。
4) 等待设备把时钟线拉低。
5) 设置/复位数据线发送第一个数据位。
6) 等待设备把时钟拉高。
7) 等待设备把时钟拉低。
8) 重复 5-7 步 发送剩下的 7 个数据位和校验位。
9) 释放数据线。
10) 等待设备把数据线拉低。
11) 等待设备把时钟线拉低。
12) 等待设备释放数据线和时钟线。

Figure 3 shows this graphically and Figure 4 separates the timing to show which signals are generated by the host, and which are generated by the PS/2 device. Notice the change in timing for the Ack bit--the data transition occours when the Clock line is high (rather than when it is low as is the case for the other 11 bits.)
图 3 用图形表示图 4 以单独的时序表示了由主机产生的信号及由 PS/2 设备产生的信号。注意应答位时序的改变－数据改变发生在时钟线为高的时候（不同于其它 11 位是当它为低的时候）。

Figure 3: Host-to-Device Communication.
图 3：主机到设备的通讯

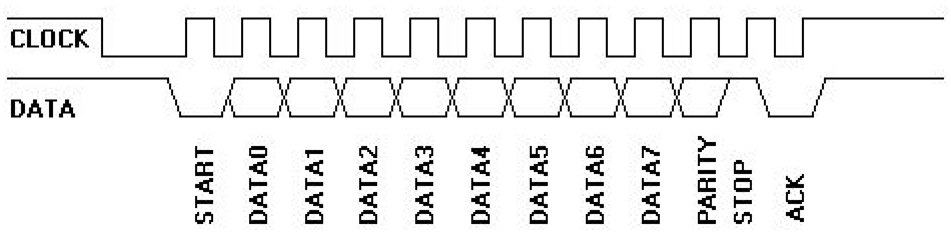


Figure 4: Detailed host-to-device communication.
图 4：主机到设备通讯的详细过程

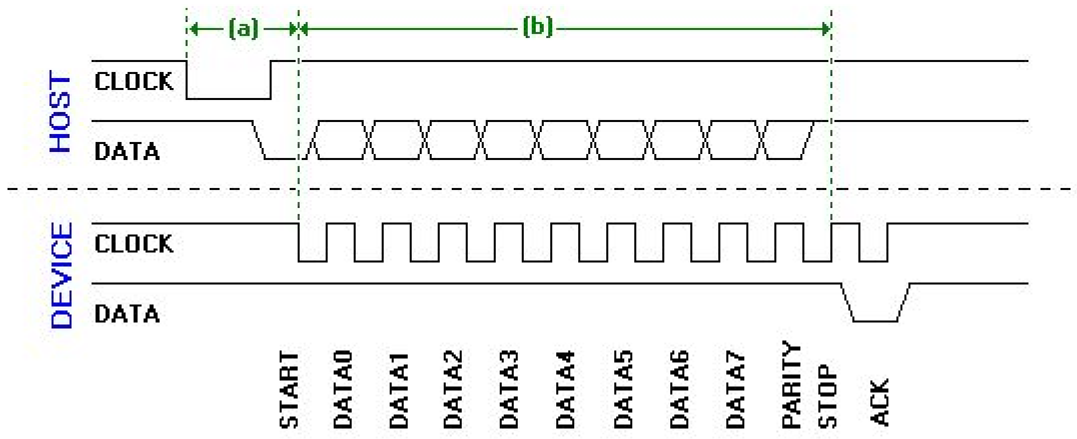


Figure 4 shows two important timing considerations: (a), and (b). (a), the time it takes the device to begin generating clock pulses after the host initially takes the Clock line low, must be no greater than 15ms; (b), the time it takes for the packet to be sent, must be no greater than 2ms. If either of these time limits is not met, the host will generate an error. Immediately after the packet is received, the host may bring the Clock line low to inhibit communication while it processes data. If the command sent by the host requires a response, that response must be received no later than 20ms after the host releases the Clock line. If this does not happen, the host generates an error. As was the case with Device-to-host communication, no Data transition may occur with 5 microseconds of a Clock transition.
图 4 描述了两个重要的定时条件：（a）和（b）。（a）在主机最初把书记现拉低后，设备开始产生时钟脉冲的时间，必须步大于 15ms；（b）数据包被发送的时间必须不大于 2ms。如果这两个条件不满足，主机将产生一个错误。在包收到后，主机为了处理数据立刻把时钟线拉低来抑制通讯。如果主机发送的命令要求有一个回应，这个回应必须在书籍释放时钟线后 20ms 之内被收到。如果没有收到，则主机产生一个错误。在设备到主机通讯的情况中，时钟改变后的 5 微秒内不应该发生数据改变的情况。

If you want to emulate a mouse or keyboard, I would recommend reading data from the host as follows:
如果呢要仿真一个鼠标或键盘，我推荐你按如下的过程从主机读入数据：

In your main program, check for Data=low at least every 10 milliseconds.
If Data has been brought low by the host, read one byte from the host
在你的主程序中，至少每 10 毫秒检测数据线是否为低。
如果数据线已被主机拉低，则从主机读取一个字节。

```
1)  Wait for Clock=high
2)  Is Data still low?
        No--An error occurred; Abort.
3)  Read 8 data bits       \   After reading each of these bits, test
4)  Read parity bit         >     Clock to make sure host hasn't pulled it
5)  Read stop bit          /        low (which would abort this transmission.)
6)  Data still equals 0?
        Yes--Keep clocking until Data=1 then generate an error
7)  Output Acknowledge bit
8)  Check Parity bit.
        Generate an error if parity bit is incorrect
```

9) Delay 45 microseconds (to give host time to inhibit next transmission.)

1) 等待时钟线为高
2) 数据线仍然为低吗？
   不－有错误发生；放弃。
3) 读入 8 个数据位 \ 在读入这些位后，
4) 读入校验位 > 测试时钟线数否被主机拉低
5) 读入停止位 / （这就意味着放弃这次传送）
6) 数据线仍旧为 0 吗？
   是－保持时钟直到数据＝1，然后产生一个错误
7) 输出应答位
8) 检查校验位
   如果校验位不正确则产生一个错误
9) 延迟 45 微秒（给主机时间抑制下次的传送）

Read each bit (8 data bits, parity bit, and stop bit) as follows:

1) Delay 20 microseconds
2) Bring Clock low
3) Delay 40 microseconds
4) Release Clock
5) Delay 20 microsecond
6) Read Data line

按如下次序读取每位（8 个数据位、检验位和停止位）：

1) 延迟 20 微秒
2) 把时钟拉低
3) 延迟 40 微秒
4) 释放时钟
5) 延迟 20 微秒
7) 读数据线

Send the acknowledge bit as follows:

1) Delay 15 microseconds
2) Bring Data low
3) Delay 5 microseconds
4) Bring Clock low
5) Delay 40 microseconds
6) Release Clock
7) Delay 5 microseconds
8) Release Data

按如下次序发送应答位：

1) 延迟 15 微秒
2) 把数据线拉低
3) 延迟 5 微秒
4) 把时钟线拉低
5) 延迟 40 微秒
6) 释放时钟线
7) 延迟 5 微秒

8) 释放数据线

Here is some sample code written for the PIC16F84 that implements the above algorithms to read data from a PS/2 host. "Delay" is a self-explanitory macro; "CLOCK" and "DATA" are the port bits connected to the Clock and Data lines; "TEMP0", "PARITY", and "COUNTER" are all general purpose registers. Note that in the "PS2inBit" routine, Clock is brought low by setting the appropriate I/O pin to output (it's assumed they were set to "0" at the beginning of the program.) And it is allowed to float (high) by setting the I/O pin to input (and allow a pull-up resistor to pull the line high.) Timing was worked out for a PIC running at 4.61 MHz +/- 25% (RC oscillator with values 5k/20 pF). Will work for any oscillator between 3.50 MHz - 5.76 MHz.
这里有写给 PIC16F84 的样例实现了上述从 PS/2 主机读取数据的算法。“Delay”是一个自扩展宏；“CLOCK”和“DATA”是连接到时钟线和数据线上的位端口；“TEMP0”、“PARITY”和“COUNTER”都是一般用途的寄存器。注意在“PS2inBit”例程中，置适当的 I/O 脚为输出来把时钟拉低（在程序的开始处它们被设置为 0）。置 I/O 脚为输入就是让它们上浮成高电平（上拉电阻把管线拉成高电平）。为达到设计的定时时序，PIC 应运行于 4.61MHz +/-25%（RC 振荡器的值是 5k/20pF）。当然也可以工作于任何 3.50MHz～5.76MHz 的振荡器。

```
ByteIn          btfss   CLOCK           ;Wait for start bit
                goto    ByteIn
                btfsc   DATA
                goto    ByteIn
                movlw   0x08
                movwf   COUNTER
                clrf    PARITY          ;Init reg for parity calc
                Delay   28
ByteInLoop      call    BitIn           ;Clock in Data bits
                btfss   CLOCK           ;Test for inhibit
                retlw   0xFE
                bcf     STATUS, C
                rrf     RECEIVE, f
                iorwf   RECEIVE, f
                xorwf   PARITY,f
                decfsz  COUNTER, f
                goto    ByteInLoop
                Delay   1
                call    BitIn           ;Clock in Parity bit
                btfss   CLOCK           ;Test for inhibit
                retlw   0xFE
                xorwf   PARITY, f
                Delay   5
ByteInLoop1     Delay   1
                call    BitIn           ;Clock in Stop bit
                btfss   CLOCK           ;Test for inhibit
                retlw   0xFE
                xorlw   0x00
                btfsc   STATUS, Z       ;Stop bit = 1?
                clrf     PARITY                  No--cause an error condition.
               btfsc    STATUS, Z          ;Stop bit = 1?
                goto     ByteInLoop1         ;   No--keep clocking.
                bsf      STATUS, RP0         ;Acknowledge
                bcf      DATA
                Delay    11
                bcf      CLOCK
                Delay    45
                bsf      CLOCK
                Delay    7
                bsf      DATA
                bcf      STATUS, RP0

                btfss    PARITY, 7           ;Parity correct?
                retlw    0xFF                ;   No--return error

                Delay    45
                retlw    0x00

BitIn          Delay    8
                bsf      STATUS, RP0
                bcf      CLOCK
                Delay    45
                bsf      CLOCK
                bcf      STATUS, RP0
                Delay    21
                btfsc    DATA
                retlw    0x80
                retlw    0x00
```

Other Sources / References:
其它资源和参考：

Adam's micro-Resources Home - Many pages/links to related information.
The AT Keyboard - My page on AT keyboards
The PS/2 Mouse - My page on the PS/2 mouse
Synaptics Touchpad Interfacing Guide -Very informative!
PS/2 Keyboard and Mouse Protocols - Timing diagrams.
Holtek - Informative datasheets on many different PS/2 mice (and other peripherals).
More Links - Many more links to related resources.

第二章 AT-PS/2 键盘接口

Introduction:

引言：

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 interface are written in assembly for Microchip's PIC microcontrollers. All code samples related to the keyboard controller are written in x86 assembly.
这篇文章试着囊括 AT 和 PS/2 键盘各方面的问题。它包含了如低级别信号和协议、扫描码、命令集、初始化、兼容性问题和其他各种信息。我还包含了关于 PC 键盘控制器的信息，这是由于它们非常相关。所有键盘接口的代码样例是用 Microchip 的 PIC 微控制器的汇编语言书写。所有与键盘控制器相关的代码样例是用 x86 汇编书写的。

I should mention that all of the information in this article comes from my own experiences and other sources that may or may not be accurate. I did not consult any official documentation of since none has been available to me. Therefore, I provide the following disclaimer:
应该说明的是，在这篇文章里提到的信息来自我自己的经验和其他资源，因此可能不正确。我没有参考任何官方的文件，因为没有我能用到的。因而我提出如下的弃权：

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. I DO NOT GUARANTEE ANY INFORMATION IN THIS ARTICLE IS ACCURATE, AND IT SHOULD BE USED FOR ABSTRACT EDUCATIONAL PURPOSES ONLY.
（译者注：大伙们自己看原文吧，这个申明对要使用该文的人来说很重要！）

You may click here to goto my main page. There, you will find other articles, code, projects, and links related to the computer keyboard. Click here for more information about me, including my resume. If you find any errors in this article or have any questions, feel free to send me an email. I don't have time to respond to them all, but I will read them all and keep your questions/comments in mind when updating this page.
点击这里你可以到我的主页。在那里，你将能发现其他关于计算机键盘的别的文章、代码、工程和链接。点击这里可以获得我的信息，包括我的简历。如果你在文章中发现有错误或者你有疑问，请给我写封 email。我可能没有时间给所有的都回信，但我会都读它们，并在更新本网页时记住你的问题或注解。

A History Lesson:

相关的历史：

The most popular keyboards in use today include:
现今仍在使用中的绝大多数流行的键盘包括：

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

USB 键盘 － 最后出现的键盘，被所有新式的计算机支持（Macintosh 和 IBM/及其兼容机）。它们有自己相关的复杂接口并且不包含这篇文章中。

IBM 机器兼容键盘 － 也叫做“AT 键盘”或“PS/2 键盘”，所有现代的 PC 都支持这个设备。它们时最容易使用的接口，也是本文的主题。

ADB 键盘 － 连接到老式 Macintosh 系统的 Apple 桌面总线，不包含在这篇文章中。

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 引入了一种新型的键盘作为它每种主要桌面计算机型号的配备。最早的 IBM PC 和后来的 IBM XT 使用的我们称之为“XT 键盘”。它们很古老并和现代的键盘一点都不相同；关于 XT 键盘没有在本文中论及。后来出现了 IBM AT 系统，再后来出现 IBM PS/2。他们引进的键盘我们至今还在使用，也是本文的主题。AT 键盘和 PS/2 键盘时十分相似的设备，但是 PS/2 使用了更小的连接器并且支持少量附加的特征。虽然如此，它仍保留了与 AT 系统向后兼容，以及一些曾经流行的附加特征（因为软件总要保持向后兼容）。下面似 IBM 三种主要键盘的概要：

IBM PC/XT Keyboard (1981): IBM PC/XT 键盘（1981）：	
83 keys	83 键
5-pin DIN connector	5 脚 DIN 连接器
Simple uni-directional serial protocol	简单的单向串行协议
Uses what we now refer to as scan code set 1	采用我们现在提及的作为第一套扫描码集
No host-to-keyboard commands	没有主机到键盘的命令

IBM AT Keyboard (1984) - Not backward compatible with XT systems.	
84 -101 keys	84－101 键
5-pin DIN connector	5 脚 DIN 连接器
Bi-directional serial protocol	双向串行协议
Uses what we now refer to as scan code set 2	采用我们现在提及的作为第二套扫描码集
Eight host-to-keyboard commands	八个主机到键盘的命令

IBM PS/2 Keyboard (1987) - Compatible with AT systems, not compatible with XT systems.	
84 - 101 keys	84－101 键
6-pin mini-DIN connector	6 脚 mini-DIN 连接器
Bi-direction serial protocol	双向串行协议
Offers optional scan code set 3	提供可选的第三套扫描码集

17 host-to-keyboard commands	17 个主机到键盘的命令

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.
PS/2 最初似 AT 设备的扩展。它支持少量附加的主机到键盘的命令并以小型连接器为特征。在这两种设备之间只有这两个区别。但是计算机硬件决不会有象兼容性这样多的标准。由于这种原因，你今天买到的任何键盘都和 PS/2 和 AT 系统兼容，但它可能不完全支持原始键盘的所有特征。

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 project relies on non-general features, it may work on some systems, but not on others...
今天，“AT 键盘”和“PS/2 键盘”仅涉及它们的连接器大小。任何给定的键盘支持或不支持哪些设置及命令是每个人猜测。例如，我现在使用的键盘有一个 PS/2 风格的连接器，但它仅完全支持七个命令，部分支持两个命令，对其他的命令只是“应答”。作为对照，我做测试的键盘有一个 AT 风格的连接器但是支持原始 PS/2 设备的每个特征/命令（还加上少量额外的命令）。这很重要就是你使用现代的键盘是兼容性而不是标准。如果你的工程依赖于某些非一般的特征，它可能在一些系统上工作，而在另一些上却不能。

Modern AT-PS/2 compatible keyboards
现代 AT-PS/2 兼容键盘

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.
任意数目的按键（通常是 101 或 104）
5 脚或 6 脚连接器，通常包括了适配器
双向串行协议
只有第二套扫描码集是保证的
应答所有的命令；但可能不是所有的都起作用

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.
脚注 1）XT 键盘使用了一套于 AT 和 PS/2 系统中用的完全不同的协议，因此它不和较新的 PC 兼容。但

是在某些键盘控制器中有一转换过程可以既支持 XT 又支持 AT-PS/2 键盘（通过开关、跳线或自适应）。同样这些键盘在两类系统都可以工作（再次重申，是通过开关或者自适应）。如果你有这样的 PC 或键盘，不要被它愚弄了—XT 键盘并不兼容与现代的计算机。

General Description:
一般性描述：

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 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" 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.
键盘上包含了一个大型的按键矩阵，它们是由安装在电路板上的处理器（叫做“键盘编码器”）来监视的。具体的处理器在键盘与键盘之间是多样化的，大拿好似它们基本上都做着同样的事情：监视哪些按键被按下或释放了，并传送适当的时候到主机。如果有必要，处理器处理所有的去抖动并在它的 16 字节缓冲区里缓冲数据。你的主板包含了一个“键盘控制器”负责解码所有来自键盘的数据，并告诉你的软件什么事件发生了。在主机和键盘之间的通讯使用 IBM 的协议。

Footnote 1) Originally, IBM used the Intel 8048 microcontroller as its keyboard encoder. The following is a short list of modern keyboard encoders:
脚注 1）最初，IBM 使用 Intel8048 微处理器作为它的键盘编码器。如下是现代键盘编码器的短清单：

Holtek: HT82K28A, HT82K628A, HT82K68A, HT82K68E
EMC: EM83050, EM83050H, EM83052H, EM83053H,
Intel: 8048, 8049
Motorola: 6868, 68HC11, 6805
Zilog: Z8602, Z8614, Z8615, Z86C15, Z86E23

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.
脚注 2）最初 IBM 使用 Intel 的 8042 微控制器作为它的键盘控制器。现在已经被兼容设备取代，并整合到主板的新品组中。键盘控制器是本文稍后论及的内容。

Electrical Interface / Protocol:
电气接口/协议：

The AT and PS/2 keyboards use the same protocol as the PS/2 mouse. Click here for detailed information about that protocol.
AT 和 PS/2 键盘使用了与 PS/2 鼠标一样的协议。点击这里可以获得关于这个协议的细节。
（译者注：对于本文的中文读者来说，就是本文第一章所讲述的内容。）

Scan Codes:
扫描码：

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)
键盘的处理器花费很多的时间来扫描或监视按键矩阵。如果它发现有键被按下、释放或按住，键盘将发送“扫描码”的信息包到计算机。扫描码有两种不同的类型：“通码”和“断码”。当一个键被按下或按住就发送通码；当一个键被释放就发送断码。每个按键被分配了唯一的通码和断码，这样主机通过查找唯一的扫描码就可以测定是哪个按键。每个键一整套的通断码组成了“扫描码集”。有三套标准的扫描码集分别是第一套、第二套和第三套。所有现代的键盘默认使用第二套扫描码。
（译者注：通码和断码是国内的描述，如果你知道更好的翻译，请告诉我。）

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:
那么你能计算出每个按键的扫描码吗？不幸的是，没有一个简单的公式可以计算扫描码。如果你要知道某特定按键的通码和断码，你将不得不查表获得。我已经把所有三套扫描码集中所有的通码和断码做成了表格。

Scan Code Set 1 - Original XT scan code set; supported by some modern keyboards
第一套扫描码集 — 原始的 XT 扫描码集；某些现代的键盘还支持。
Scan Code Set 2 - Default scan code set for all modern keyboards
第二套扫描码集 — 所有现代键盘默认的扫描码集。
Scan Code Set 3 - Optional PS/2 scan code set--rarely used
第三套扫描码集 — 可选的 PS/2 扫描码集（很少使用）
（译者注：这是我见过的收录最全的扫描码集，所有这三个扫描码表列在文本附录 1、2、3 中。感谢 Adam Chapweske 所做的工作！）

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.
*脚注 1）最初，AT 键盘只支持第二套，PS/2 键盘默认使用第二套且支持所有这三套。许多现代键盘行为象 PS/2 设备，但我遇到少数不支持第一套、第三套或这两套都不支持的。同样，如果你曾经做过低级 PC 编程，你可能注意到键盘控制器缺省支持**第一套**扫描码。这是因为键盘控制器转换所有进来的扫描码到第一套（这是为了和 XT 系统的软件保持兼容）。但是，它仍旧下发第二套扫描码到键盘的串行线。*

Make Codes, Break Codes, and Typematic Repeat:
通码、断码和机打重复率：

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.
只要一个键被按下，这个键的通码就被发送到计算机。记住通码只表示键盘上的一个按键，它不表示印刷在按键上的那个字符。这就意味着在通码和 ASCII 码之间没有已定义的关联。直到主机把扫描码翻译成一个字符或命令。

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.
虽然多数第二套通码都只有一个字节宽，但也有少数“扩展按键”的通码是两字节或四字节宽。这类的通码第一个字节总是为 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:
正如键按下通码就被发往计算机一样，只要键一释放，断码就会被发送。每个键都有它自己唯一的通码，它们也都有唯一的断码。幸运的是，你不用总是通过查表来找出按键的断码－在通码和断码之间存在着必然的联系。多数第二套断码有两字节长，它们的第一个字节是 F0h，第二个字节是这个键的通码。扩展按键的断码通常有三个字节，它们前两个字节是 E0h,F0h，最后一个字节是这个按键通码的最后一个字节。作为一个例子，我在下面列出了几个按键的第二套通码和断码：

Key	(Set 2)Make Code	(Set 2) 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.
例如：通码和断码是以什么样的序列发送到你的计算机使得字符“G”出现在你的字处理软件里呢？因为这是一个大写字母，需要发生这样的事件次序：按下“Shift”键，按下“G”键，释放“G”键，释放“Shift”键。与这些时间相关的扫描码如下：“Shift”键的通码（12h），“G”键的通码（34h），“G”键的断码（F0h，34h），“Shift”键的断码（F0h，12h）。因此，发送到

你的计算机的数据应该是：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.
如果你按了一个键，这个键的通码被发送到计算机。当你按下并按住这个键，则这个键就变成了机打，着就意味着键盘将一直发送这个键的通码直到它被释放或者其他键被按下。要想证实这点，只要打开一个文本编辑器并按下“A”键。当你首先按下这个键，字符“a”立刻出现在你的屏幕上。在一个短暂的延迟后，接着出现一整串的“a”，直到你释放“A”键。这里有两个重要的参数：机打延时，是第一个和第二个“a”之间的延迟；机打速率是在机打延时后每秒有多少字符出现你的屏幕上。机打延时的范围可以从 0.25 秒到 1.00 秒；机打速率的范围可以从 2.0cps(字符每秒)到 30.0cps。你可以用“Set Typematic Rate/Delay”（0xF3）命令来改变机打速率和延时。
（译者注：typematic 本文中翻译成“机打”。按字面上说是非人工的机械的按键打字的意思。类似的词还有 automatic。如果有更好的翻译请告诉我。）

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.
机打的数据不被键盘所缓冲。在多个键被按下的情况下，只有最后一个按下的键变成机打。当这个键被释放时，机打重复就停止了，甚至于其他的键依然还按着。

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.
脚注 1）实际上，在第一第二套扫描码里没有“Pause/Break”键的断码。当这个键按下时发送它的通码，当它释放时，什么都没有被发送。

Reset:
复位：

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:
在上电或软件复位（见“Reset”命令）后，键盘执行诊断自检叫做 BAT（基本保证测试）并载入如下的缺省值：

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

*Variable in some keyboards, hard-coded in others.

- 机打延迟为 500ms。
- 机打速率为 10.9cps。
- *第二套扫描码集
- *置所有按键为机打/通码/断码

“*”所指的项在某些键盘上时可变的，而在其他键盘上时硬件编码的（不可变）。

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.
当进入 BAT，键盘点亮它的三个 LED 指示器，并在完成 BAT 后关闭它们。此时，BAT 完成代码要么 0xAA（BAT 成功）或 0xFC（有错误）被发送到主机。

Most keyboards 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
多数键盘忽略它们的时钟和数据线直到 BAT 完成代码发送后。所以，“抑制”条件（时钟线拉低）可能不能防止键盘发送它们的 BAT 完成代码。

Command Set:
命令集：

Every byte sent to the keyboard gets a response of 0xFA ("acknowledge") from the keyboard. The only exceptions to this are the keyboard's response to the "Resend" and "Echo" commands. The host should wait for an "acknowledge" before sending the next byte to the keyboard. The keyboard clears its output buffer in response to any command. The following is a list of all commands that may be sent to the keyboard.
每个发送到键盘的字节都从键盘获得一个 0xFA（“应答”）的回应。唯一例外的是键盘对“Resend”和“Echo”命令的回应。在发送下一个字节给键盘之前，主机要等待“应答”。键盘应答任何命令后清除自己的输出缓冲区。下面列出了所有可能被发给键盘的命令：

- 0xFF (Reset) - Causes keyboard to enter "Reset" mode. (See "Reset" section.)
- 0xFF (Reset)－引起键盘进入“Reset”模式。（见“复位”部分。）
- 0xFE (Resend) - This is used to indicate an error in reception. Keyboard responds by resending the last scan code or command response sent to the host. However, 0xFE is never sent in response to a "Resend" command.
- 0xFE (Resend)－用于只是在接收中出现的错误。键盘的响应就是重发送最后的扫描码或者命令回应给主机。但是 0xFE 绝不会作为“Resend”命令的回应而被发送。
- *0xFD (Set Key Type Make) - Allows the host to specify a key that is to send only make codes. This key will not send break codes or typematic repeat. This key is specified by its set 3 scan code.
- *0xFD (Set Key Type Make)－允许主机指定一个按键只发送通码。这个按键不发送断码或进行机打重复。指定的按键采用它的第三套扫描码。
- *0xFC (Set Key Type Make/Break) - Similar to "Set Key Type Make", but both make codes and break codes are enabled (typematic is disabled.)
- *0xFC (Set Key Type Make/Break)－类似于“Set Key Type Make”，只有通码和断码是使能的（机打被禁止了）。
- *0xFB (Set Key Type Typematic) - Similar to previous two commands, except make and typematic is enabled; break codes are disabled.

- *0xFB (Set Key Type Typematic)－类似于前两条命令，通码和机打是使能的，而断码被禁止。
- *0xFA (Set All Keys Typematic/Make/Break) - Default setting. Make codes, break codes, and typematic repeat enabled for all keys (except "Print Screen" key, which has no break code in sets 1 and 2.)
- *0xFA (Set All Keys Typematic/Make/Break)－缺省设置。所有键的通码、断码和机打重复都使能（除了“Print Screen”键，它在第一套和第二套中没有断码。）
- *0xF9 (Set All Keys Make) - Causes only make codes to be sent; break codes and typematic repeat are disabled for all keys.
- *0xF9 (Set All Keys Make)－所有键都只发送通码；断码和机打重复被禁止。
- *0xF8 (Set All Keys Make/Break) - Similar to previous two commands, except only typematic repeat is disabled.
- *0xF8 (Set All Keys Make/Break)－类似于前两条命令，除了只是机打重复被禁止外。
- *0xF7 (Set All Keys Typematic) - Similar to previous three commands, except only break codes are disabled. Make codes and typematic repeat are enabled.
- *0xF7 (Set All Keys Typematic)－类似于前三条命令，仅断码被禁止，通码和机打重复是使能的。
- 0xF6 (Set Default) - Load default typematic rate/delay (10.9cps / 500ms), key types (all keys typematic/make/break), and scan code set (2).
- 0xF6 (Set Default)－载入缺省的机打速率/延时（10.9cps/500ms），按键类型(所有按键都使能机打/通码/断码)，以及第二套扫描码集。
- 0xF5 (Disable) - Keyboard stops scanning, loads default values (see "Set Default" command), and waits further instructions.
- 0xF5 (Disable)－键盘停止扫描，载入缺省值（键“Set Default”命令），等待进一步指令。
- 0xF4 (Enable) - Re-enables keyboard after disabled using previous command.
- 0xF4 (Enable)－在用上一条命令禁止键盘后，重新使能键盘。
- 0xF3 (Set Typematic Rate/Delay) - Host follows this command with one argument byte that defines the typematic rate and delay as follows:
- 0xF3 (Set Typematic Rate/Delay)－主机在这条命令后会发送一个字节的参数来定义机打速率和延时，具体含义如下：

Repeat Rate

Bits 0-4	Rate(cps)	Bits 0-4	Rate(cps)	Bits 0-4	Rate(cps)	Bits 0-4	Rate(cps)
00h	2.0	08h	4.0	10h	8.0	18h	16.0
01h	2.1	09h	4.3	11h	8.6	19h	17.1
02h	2.3	0Ah	4.6	12h	9.2	1Ah	18.5
03h	2.5	0Bh	5.0	13h	10.0	1Bh	20.0
04h	2.7	0Ch	5.5	14h	10.9	1Ch	21.8
05h	3.0	0Dh	6.0	15h	12.0	1Dh	24.0
06h	3.3	0Eh	6.7	16h	13.3	1Eh	26.7
07h	3.7	0Fh	7.5	17h	15.0	1Fh	30.0

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.
- *0xF2 (Read ID)－键盘回应两个字节的设备 ID，0xAB、0x83。
- *0xF0 (Set Scan Code Set) - Host follows this command with one argument byte, that specifies which scan code set the keyboard should use. This argument byte may be 0x01, 0x02, or 0x03 to select scan code set 1, 2, or 3, respectively. You can get the current scan code set from the keyboard by sending this command with 0x00 as the argument byte.
- *0xF0 (Set Scan Code Set)－主机在这个命令后发送一个字节的参数，是定键盘使用哪套扫描码集。参数字节可以是 0x01、0x02 或 0x03 分别选择扫描码集第一套、第二套或第三套。如果要获得当前正在使用的扫描码集，只要发送带 0x00 参数的本命令即可。
- 0xEE (Echo) - The keyboard responds with "Echo" (0xEE).
- 0xEE (Echo)－键盘用“Echo”（0xEE）回应。
- 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:
- 0xED (Set/Reset LEDs)－主机在本命令后跟随一个参数字节，用于指示键盘上 Num Lock, Caps Lock, and Scroll Lock LED 的状态。这个参数字节的定义如下：

MSB LSB

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)

*Originally available in PS/2 keyboards only.
*只是最初可用于 PS/2 键盘。

Emulation:
仿真：

Click here for keyboard/mouse routines. Source in MPASM for PIC microcontrollers.
点击这里可以看到键盘/鼠标的程序。源代码是用 PIC 微控制器的 MPASM 书写的。
（译者注：程序已经收录到本文的附录 4 中。）

The i8042 Keyboard Controller:
i8042 键盘控制器：

Up to this point in the article, all information has been presented from a hardware point-of-view. However, if you're writing low-level keyboard-related software for the host PC, you won't be communicating directly with the keyboard. Instead, a keyboard controller provides an interface between the keyboard and the peripheral bus. This controller takes care of all the signal-level and protocol details, as well as providing some conversion, interpretation, and handling of scan codes and commands.
写到文章的这里，从硬件的观点来说所有的信息都已经被提出。但是如果你打算给 host PC 写一低级的与键盘相关的程序，你却不能直接和键盘通讯。而键盘控制器提供了键盘和周边总线之间的接口。控制器不但封装了所有信号级和协议的细节，又提供了相关转换、解释和扫描码及命令的句柄。

An Intel 8042/compatible microcontroller is used as the PC's keyboard controller. In modern computers, this microcontroller is hidden within the motherboard's chipset, which integrates many controllers in a single package. Nonetheless, this device is still there, and the keyboard controller is still commonly referred to as "the 8042".
Intel 8042 或兼容微控制器被用作 PC 键盘的控制器。在现代的计算机中这个微控制器被隐藏到了主板的芯片组中，主板的芯片组在一个单一封装中整合了很多的控制器。虽然如此，但设备仍旧是存在的，键盘控制器一般仍按“8042”来论及。

Depending on the motherboard, the keyboard controller may operate in one of two modes: "AT-compatible" mode, or "PS/2-compatible" mode. The latter is used if a PS/2 mouse is supported by the motherboard. If this is the case, the 8042 acts as the keyboard controller and the mouse controller. The keyboard controller auto-detects which mode it is to use according to how it's wired to the keyboard port.
依赖于主板的不同，键盘控制器可以工作于两个模式之一：“AT 兼容”模式或“PS/2 兼容”模式。如果主板支持 PS/2 鼠标就工作在后一种模式下。在这种情况下，8042 的作用是键盘控制器和鼠标控制器。键盘控制器根据键盘端口的连线情况自动检测它应该工作在何种模式下。

The 8042 contains the following registers:
8042 包含了如下的寄存器：

- A one-byte input buffer - contains byte read from keyboard; read-only
- 一个字节的输入缓冲区－包含从键盘读入的字节；只读
- A one-byte output buffer - contains byte to-be-written to keyboard; write-only
- 一个字节的输出缓冲区－包含要写到键盘的字节；只写
- A one-byte status register - 8 status flags; read-only
- 一个字节的状态寄存器－8 个状态标志；只读
- A one-byte control register - 7 control flags; read/write
- 一个字节的控制寄存器－7 个控制标志；读写

The first three registers (input, output, status) are directly accessible via ports 0x60 and 0x64. The last register (control) is read using the "Read Command Byte" command, and written using the "Write Command Byte" command. The following table shows how the peripheral ports are used to interface the 8042:
前三个寄存器（输入、输出、状态）可以通过 0x60 和 0x64 端口直接存取。最后那个寄存器（控制）要使用“Read Command Byte”命令读，使用“Write Command Byte”命令写。下表示出周边端口示如何于 8042 接口的：

Port	Read / Write	Function	功能
0x60	Read	Read Input Buffer	读输入缓冲区
0x60	Write	Write Output Buffer	写输出缓冲区
0x64	Read	Read Status Register	读状态寄存器
0x64	Write	Send Command	发送命令

Writing to port 0x64 doesn't write to any specific register, but sends a command for the 8042 to interpret. If the command accepts a parameter, this parameter is sent to port 0x60. Likewise, any results returned by the command may be read from port 0x60.
写 0x64 端口不会写入到任何特定的寄存器中，但是解释为发送命令给 8042。如果命令接收一个参数，则参数被发往 0x60 端口。同样，命令的任何返回结构可以从 0x60 端口读出。

When describing the 8042, I may occasionally refer to its physical I/O pins. These pins are defined below:
在描述 8042 时，我偶尔提及它的物理 I/O 管脚，这些管脚定义如下：

AT-compatible mode

Port 1 (Input Port):			Port 2 (Output Port):			Port 3 (Test Port):		
Pin	Name	Function	Pin	Name	Function	Pin	Name	Function
0	P10	Undefined	0	P20	System Reset 1: Normal 0: Reset computer	0	T0	Keyboard Clock (Input)
1	P11	Undefined	1	P21	Gate A20	1	T1	Keyboard Data (Input)
2	P12	Undefined	2	P22	Undefined	2	--	Undefined
3	P13	Undefined	3	P23	Undefined	3	--	Undefined
4	P14	External RAM 1: Enable external RAM 0: Disable external RAM	4	P24	Input Buffer Full	4	--	Undefined
5	P15	Manufacturing Setting 1: Setting enabled 0: Setting disabled	5	P25	Output Buffer Empty	5	--	Undefined
6	P16	Display Type Switch 1: Color display 0: Monochrome	6	P26	Keyboard Clock 1: Pull Clock low 0: High-Z	6	--	Undefined
7	P17	Keyboard Inhibit Switch 1: Keyboard enabled 0: Keyboard inhibited	7	P27	Keyboard Data: 1: Pull Data low 0: High-Z	7	--	Undefined

PS/2-compatible mode

Port 1 (Input Port):			Port 2 (Output Port):			Port 3 (Test Port):		
Pin	Name	Function	Pin	Name	Function	Pin	Name	Function
0	P10	Keyboard Data (Input)	0	P20	System Reset 1: Normal 0: Reset computer	0	T0	Keyboard Clock (Input)
1	P11	Mouse Data (Input)	1	P21	Gate A20	1	T1	Mouse Clock (Input)
2	P12	Undefined	2	P22	Mouse Data: 1: Pull Data low 0: High-Z	2	--	Undefined
3	P13	Undefined	3	P23	Mouse Clock: 1: Pull Clock low	3	--	Undefined

					0: High-Z			
4	P14	External RAM 1: Enable external RAM 0: Disable external RAM	4	P24	Keyboard IBF interrupt: 1: Assert IRQ 1 0: De-assert IRQ 1	4	--	Undefined
5	P15	Manufacturing Setting 1: Setting enabled 0: Setting disabled	5	P25	Mouse IBF interrupt: 1: Assert IRQ 12 0: De-assert IRQ 12	5	--	Undefined
6	P16	Display Type Switch 1: Color display 0: Monochrome	6	P26	Keyboard Clock 1: Pull Clock low 0: High-Z	6	--	Undefined
7	P17	Keyboard Inhibit Switch 1: Keyboard enabled 0: Keyboard inhibited	7	P27	Keyboard Data: 1: Pull Data low 0: High-Z	7	--	Undefined

(Note: Reading keyboard controller datasheets can be confusing--it will refer to the "input buffer" as the "output buffer" and vice versa. This makes sense from the point-of-view of someone writing firmware for the controller, but for somebody used to interfacing the controller, this can cause problems. Throughout this document, I only refer to the "input buffer" as the one containing input from the keyboard, and the "output buffer" as the one that contains output to be sent to the keyboard.)
（注意：读键盘控制器数据表可能会迷惑你，其中提到的“输入缓冲区”是指“输出缓冲区”，反之亦然。这是取决于你看问题的观点，是从控制器的固件，还是从控制器的接口。在本文中，我所提到的“输入缓冲区”指放置从键盘获得的输入的地方，“输出缓冲器”指放置把输出发送到键盘的数据的地方。）

Status Register:
状态寄存器：

The 8042's status flags are read from port 0x64. They contain error information, status information, and indicate whether or not data is present in the input and output buffers. The flags are defined as follows:
8042 的状态标志是从 0x64 端口读出的。它们包含了错误信息、状态信息和输入输出缓冲区里有无数据的指示。这些标志的定义如下：

	MSB							LSB
AT-compatible mode:	PERR	RxTO	TxTO	INH	A2	SYS	IBF	OBF
PS/2-compatible mode:	PERR	TO	MOBF	INH	A2	SYS	IBF	OBF

●OBF (Output Buffer Full) - Indicates when it's okay to write to output buffer.
0: Output buffer empty - Okay to write to port 0x60
1: Output buffer full - Don't write to port 0x60
●IBF (Input Buffer Full) - Indicates when input is available in the input buffer.
0: Input buffer empty - No unread input at port 0x60
1: Input buffer full - New input can be read from port 0x60
●SYS (System flag) - Post reads this to determine if power-on reset, or software reset.
0: Power-up value - System is in power-on reset.

1: BAT code received - System has already beed initialized.

●A2 (Address line A2) - Used internally by the keyboard controller

0: A2 = 0 - Port 0x60 was last written to

1: A2 = 1 - Port 0x64 was last written to

●INH (Inhibit flag) - Indicates whether or not keyboard communication is inhibited.

0: Keyboard Clock = 0 - Keyboard is inhibited

1: Keyboard Clock = 1 - Keyboard is not inhibited

●TxTO (Transmit Timeout) - Indicates keyboard isn't accepting input (kbd may not be plugged in).

0: No Error - Keyboard accepted the last byte written to it.

1: Timeout error - Keyboard didn't generate clock signals within 15 ms of "request-to-send".

●RxTO (Receive Timeout) - Indicates keyboard didn't respond to a command (kbd probably broke)

0: No Error - Keyboard responded to last byte.

1: Timeout error - Keyboard didn't generate clock signals within 20 ms of command reception.

●PERR (Parity Error) - Indicates communication error with keyboard (possibly noisy/loose connection)

0: No Error - Odd parity received and proper command response recieved.

1: Parity Error - Even parity received or 0xFE received as command response.

●MOBF (Mouse Output Buffer Full) - Similar to OBF, except for PS/2 mouse.

0: Output buffer empty - Okay to write to auxillary device's output buffer

1: Output buffer full - Don't write to port auxillary device's output buffer

●TO (General Timout) - Indicates timeout during command write or response. (Same as TxTO + RxTO.)

0: No Error - Keyboard received and responded to last command.

1: Timeout Error - See TxTO and RxTO for more information.

●OBF （输出缓冲区满）－ 指示是否成功写入输出缓冲区。

0：输出缓冲区空－写入到 0x60 端口成功

1：输出缓冲区满－不能写到 0x60 端口

●IBF （输入缓冲区满）－ 指示是否可以从输入缓冲区中读入。

0：输入缓冲区空－不能从 0x60 端口读入数据

1：输入缓冲区满－有新的数据输入了，可以从 0x60 端口读入

●SYS（系统标志）－ Post 读取这个标记测定是否上电复位还是软件复位。

0：上电－系统处于上电自检中。

1：BAT 代码完成－系统已经完成了初始化。

●A2 （地址线 A2）－ 键盘控制器内部使用。

0：A2 = 0－最后写入的是端口 0x60 。

1：A2 = 1－最后写入的是端口 0x64 。

●INH（禁止标志）－ 指示键盘通讯是否被禁止。

0：键盘时钟 = 0－键盘被禁止。

1：键盘时钟 = 1－键盘没有被禁止。

●TxTO （发送超时）－ 指示键盘不可接受输入（比如键盘没有插入）。

0：无错误－键盘接收了最后一个给它的字节。

1：超时错－键盘在 15ms 的“请求发送”时间内没有产生时钟信号。

●RxTO （接收超时）－ 指示键盘不能回应命令(键盘可能是坏的)

0：无错误－键盘回应了最后一个字节。

1：超时错－键盘在 20ms 的命令接收期内没有产生时钟信号。

●PERR（校验错误）－ 指示和键盘的通讯有错误（可能有干扰或者连接松掉了）。

0：无错误－接收到了奇校验并且收到了适当的命令回应。

1：校验错－接收到偶校验或者 0xFE 作为命令回应被收到了。

●MOBF（鼠标输出缓冲区满）－ 类似于 OBF 但不包括 PS/2 鼠标。
0：输出缓冲区空－写到扩展设备的输出缓冲区成功。
1：输出缓冲区满－不可以写到辅助设备的输出缓冲区端口。
●TO （一般性超时）－ 指示在命令写入或回应中有超时（和 TxTO + RxTO 一样）
0：无错误－键盘收到并回应了最后一条命令。
1：超时错－参考 TxTO 和 RxTO ，获得更多信息。

[EG: On my PC, the normal value of the 8042's "Status" register is 14h = 00010100b. This indicates keyboard communication is not inhibited, and the 8042 has already completed its self-test ("BAT"). The "Status" register is accessed by reading from port 64h ("IN AL, 64h")]
『例如：在我的 PC 上，8042 转台寄存器的正常值是 14h＝00010100b。这个值说明键盘通讯没有被禁止，8042 已经完成了自检（“BAT”）。状态寄存器可以从 64h 端口读取（“IN AL,64h”）』

Reading keyboard input:
读键盘的输入：

When the 8042 recieves a valid scan code from the keyboard, it is converted to its set 1 equivalent. The converted scan code is then placed in the input buffer, the IBF (Input Buffer Full) flag is set, and IRQ 1 is asserted. Furthermore, when any byte is received from the keyboard, the 8042 inhibits further reception (by pulling the "Clock" line low), so no other scan codes will be received until the input buffer is emptied.
当 8042 从键盘收到有效的扫描码，就把它转换成第一套相等的扫描码。转换后的扫描码放置在输入缓冲区，IBF（输入缓冲区满）标志被设置，产生 IRQ1。而且，此时如果有其他来自键盘的字节，8042 将抑制更多的接收（通过把“时钟线拉低”），这样就不会收到更多的扫描码一直到输入缓冲区被清空。

If enabled, IRQ 1 will activate the keyboard driver, pointed to by interrupt vector 0x09. The driver reads the scan code from port 0x60, which causes the 8042 to de-assert IRQ 1 and reset the IBF flag. The scan code is then processed by the driver, which responds to special key combinations and updates an area of the system RAM reserved for keyboard input.
如果中断是使能的，IRQ1 将激活键盘驱动程序，这是指向 0x09 号中断向量的。驱动程序从 0x60 端口读取扫描码，这个动作会释放 IRQ1 并复位 IBF 标志。接着扫描码被驱动程序处理，如回应特殊键的组合更新系统 RAM 保留给键盘输入的区域里的数据等。

If you don't want to patch into interrupt 0x09, you may poll the keyboard controller for input. This is accomplished by disabling the 8042's IBF Interrupt and polling the IBF flag. This flag is set (1) when data is available in the input buffer, and is cleared (0) when data is read from the input buffer. Reading the input buffer is accomplished by reading from port 0x60, and the IBF flag is at port 0x64, bit 1. The following assembly code illustrates this:
如果你不打算在中断 0x09 中嵌入个补丁，你可以轮询键盘控制器来输入数据。这是通过禁止 8042IBF 中断再轮询 IBF 标志来实现的。数据如果再输入缓冲区是可用的，标志就会被置 1，如果数据从输入缓冲区里被读出了，标志就会被清 0。读输入缓冲区就是读 0x60 端口中的数据，IBF 标志在 0x64 端口的第 1 位。汇编代码举例说明如下：

```
kbRead:
WaitLoop:     in       al, 64h       ; Read Status byte
              and      al, 10b       ; Test IBF flag (Status<1>)
              jz       WaitLoop      ; Wait for IBF = 1
```

```
        in      al, 60h      ; Read input buffer
```

Writing to keyboard:
往键盘写数据：

When you write to the 8042's output buffer (via port 0x60), the controller sets the OBF ("Output Buffer Full") flag and processes the data. The 8042 will send this data to the keyboard and wait for a response. If the keyboard does not accept or generate a response within a given amount of time, the appropriate timeout flag will be set (see Status register definition for more info.) If an incorrect parity bit is read, the 8042 will send the "Resend" (0xFE) command to the keyboard. If the keyboard continues to send erroneous bytes, the "Parity Error" flag is set in the Status register. If no errors occur, the response byte is placed in the input buffer, the IBF ("Input Buffer Full") flag is set, and IRQ 1 is activated, signaling the keyboard driver.
当你写数据到 8042 的输出缓冲区（通过 0x60 端口），控制器设置 OBF（“输出缓冲区满”）标志并处理数据。8042 将发送这个数据到键盘并等待一个回应。如果键盘没有接收或者在指定时间内没有回应，相应的超时标志就会被设置（见状态寄存器的定义给获得更多信息）。如果键盘继续发送错误的字节，状态寄存器里的“校验错”标志就会被置位。如果没有错误发生，回应字节就会放到输入缓冲区中，IBF（“输入缓冲区满”）标志被置位，IRQ1 被激活，发信号给键盘驱动程序。

The following assembly code shows how to write to the output buffer. (Remember, after you write to the output buffer, you should use int 9h or poll port 64h to get the keyboard's response.)
下面的汇编代码说明如何写入输出缓冲区。（记住，在你往输出缓冲区中写入数据后，你必须使用中断 9h 或者轮询 64h 端口来获得键盘的回应。）

```
kbWrite:
WaitLoop:   in      al, 64h     ; Read Status byte
            and     al, 01b     ; Test OBF flag (Status<0>)
            jnz     WaitLoop    ; Wait for OBF = 0
            out     60h, cl     ; Write data to output buffer
```

Keyboard Controller Commands:
键盘控制器命令：

Commands are sent to the keyboard controller by writing to port 0x64. Command parameters are written to port 0x60 after teh command is sent. Results are returned on port 0x60. Always test the OBF ("Output Buffer Full") flag before writing commands or parameters to the 8042.
范颂命令给键盘控制器就是写 0x64 端口。在命令发送后，命令参数写到 0x60 端口。结构也返回到 0x60 端口。在写命令或参数到 8042 之前总是要测试 OBF（“输出缓冲区满”）标志的。

●0x20 (Read Command Byte) - Returns command byte. (See "Write Command Byte" below).
●0x60 (Write Command Byte) - Stores parameter as command byte. Command byte defined as follows:
●0x20（读命令字节）－返回命令字节。（参考下面的“写命令字节”）。
●0x60（写命令字节）－存储参数作为命令字节。命令字节定义如下：

	MSB							LSB
AT-compatible mode:	--	XLAT	PC	_EN	OVR	SYS	--	INT
PS/2-compatible mode:	--	XLAT	_EN2	_EN	--	SYS	INT2	INT

◆INT (Input Buffer Full Interrupt) - When set, IRQ 1 is generated when data is available in the input buffer.

0: IBF Interrupt Disabled - You must poll STATUS<IBF> to read input.

1: IBF Interrupt Enabled - Keyboard driver at software int 0x09 handles input.

◆SYS (System Flag) - Used to manually set/clear SYS flag in Status register.

0: Power-on value - Tells POST to perform power-on tests/initialization.

1: BAT code received - Tells POST to perform "warm boot" tests/initiailization.

◆OVR (Inhibit Override) - Overrides keyboard's "inhibit" switch on older motherboards.

0: Inhibit switch enabled - Keyboard inhibited if pin P17 is high.

1: Inhibit switch disabled - Keyboard not inhibited even if P17 = high.

◆_EN (Disable keyboard) - Disables/enables keyboard interface.

0: Enable - Keyboard interface enabled.

1: Disable - All keyboard communication is disabled.

◆PC ("PC Mode") - ???Enables keyboard interface somehow???

0: Disable - ???

1: Enable - ???

◆XLAT (Translate Scan Codes) - Enables/disables translation to set 1 scan codes.

0: Translation disabled - Data appears at input buffer exactly as read from keyboard

1: Translation enabled - Scan codes translated to set 1 before put in input buffer

◆INT2 (Mouse Input Buffer Full Interrupt) - When set, IRQ 12 is generated when mouse data is available.

0: Auxillary IBF Interrupt Disabled -

1: Auxillary IBF Interrupt Enabled -

◆_EN2 (Disable Mouse) - Disables/enables mouse interface.

0: Enable - Auxillary PS/2 device interface enabled

1: Disable - Auxillary PS/2 device interface disabled

◆INT（输入缓冲区满中断）－如果设置了，当输入缓冲区有有效数据时产生 IRQ1。

0：IBF 中断禁止－你必须轮询状态寄存器的<IBF>来读取输入。

1：IBF 中断使能－在中断 0x19 中的键盘驱动软件俘获输入。

◆SYS（系统标志）－常用于手动设置/清除状态寄存器中的 SYS 标志。

0：上电－告诉 POST 执行上电自检及初始化。

1：BAT 代码收到－告诉 POST 执行“热启动”测试及初始化。

◆OVR（忽略禁止）－在某些老主板上忽略键盘的“禁用”开关。

0：禁用开关使能－如果 P17 脚为高则禁用键盘。

1：禁用开关禁止－键盘不被禁用甚至于 P17 为高。

◆_EN（禁止键盘）－禁止/使能键盘接口。

0：使能－键盘接口使能。

1：禁止－所有键盘通讯被禁止。

◆PC（“PC 模式”）－???在某种情况下使能键盘接口???

0：禁止－???

1：使能－???

（译者注：这个命令显然作者不知道是如何工作的，所以我也不知道。）

◆XLAT（翻译扫描码）－使能/禁止翻译成第一套扫描码。

0：翻译禁止－从键盘读入的数据直接放入输入缓冲区。

1：翻译使能－扫描码在放入输入缓冲区之前先翻译成第一套的对应值。

◆INT2（鼠标输入缓冲区满中断）－当设置后，如果鼠标数据有效则产生 IRQ 12。

0：辅助 IBF 中断禁止－

1：辅助 IBF 中断使能－

◆_EN2（禁止鼠标）－禁止/使能鼠标接口。

0：使能－辅助的 PS/2 设备接口被使能。

1：禁止－辅助的 PS/2 设备接口被禁止。

●?0x90-0x9F (Write to output port) - Writes command's lower nibble to lower nibble of output port (see Output Port definition.)

●?0xA1 (Get version number) - Returns firmware version number.

●?0xA4 (Get password) - Returns 0xFA if password exists; otherwise, 0xF1.

●?0xA5 (Set password) - Set the new password by sending a null-terminated string of scan codes as this command's parameter.

●?0xA6 (Check password) - Compares keyboard input with current password.

●0xA7 (Disable mouse interface) - PS/2 mode only. Similar to "Disable keyboard interface" (0xAD) command.

●0xA8 (Enable mouse interface) - PS/2 mode only. Similar to "Enable keyboard interface" (0xAE) command.

●0xA9 (Mouse interface test) - Returns 0x00 if okay, 0x01 if Clock line stuck low, 0x02 if clock line stuck high, 0x03 if data line stuck low, and 0x04 if data line stuck high.

●0xAA (Controller self-test) - Returns 0x55 if okay.

●0xAB (Keyboard interface test) - Returns 0x00 if okay, 0x01 if Clock line stuck low, 0x02 if clock line stuck high, 0x03 if data line stuck low, and 0x04 if data line stuck high.

●0xAD (Disable keyboard interface) - Sets bit 4 of command byte and disables all communication with keyboard.

●0xAE (Enable keyboard interface) - Clears bit 4 of command byte and re-enables communication with keyboard.

●0xAF (Get version)

●0xC0 (Read input port) - Returns values on input port (see Input Port definition.)

●0xC1 (Copy input port LSn) - PS/2 mode only. Copy input port's low nibble to Status register (see Input Port definition)

●0xC2 (Copy input port MSn) - PS/2 mode only. Copy input port's high nibble to Status register (see Input Port definition.)

●0xD0 (Read output port) - Returns values on output port (see Output Port definition.)

●0xD1 (Write output port) - Write parameter to output port (see Output Port definition.)

●0xD2 (Write keyboard buffer) - Parameter written to input buffer as if received from keyboard.

●0xD3 (Write mouse buffer) - Parameter written to input buffer as if received from mouse.

●0xD4 (Write mouse Device) - Sends parameter to the auxillary PS/2 device.

●0xE0 (Read test port) - Returns values on test port (see Test Port definition.)

●0xF0-0xFF (Pulse output port) - Pulses command's lower nibble onto lower nibble of output port (see Output Port definition.)

●?0x90-0x9F（写到输出端口）－写命令的低 4 位到输出端口的低 4 位（见输出端口的定义）。

●?0xA1（获得版本号）－返回固件的版本号

●?0xA4（获得密码）－如果密码存在则返回 0xFA，否则返回 0xF1。

●?0xA5（设置密码）－设置新的密码，发送一个以空字符结束的扫描码字串作为命令的参数。

●?0xA6（比较密码）－把键盘输入和当前的密码相比较。

●0xA7（禁止鼠标接口）－仅用于 PS/2 模式。类似于“禁止键盘接口”（0xAD）命令。
●0xA8（使能鼠标接口）－仅用于 PS/2 模式。类似于“使能键盘接口”（0xAE）命令。
●0xA9（鼠标接口测试）－若通过则返回 0x00，若时钟线保持低不变则返回 0x01，若时钟线保持高不变则返回 0x02，若数据线保持低不变则返回 0x03，若数据线保持高不变则返回 0x04。
●0xAA（控制器自检）－若成功则返回 0x55。
●0xAB（键盘接口测试）－若通过则返回 0x00，若时钟线保持低不变则为 0x01，若时钟线保持高不变则为 0x02，若数据线保持低不变则为 0x03，若数据线保持高不变则为 0x04。
●0xAD（禁止键盘接口）－设置命令字节的第 4 位并禁止所有与键盘间的通讯。
●0xAE（使能键盘接口）－清除命令字节的第 4 位并重新使能与键盘的通讯。
●0xAF（获得版本）
●0xC0（读输入端口）－返回输入端口中的值（参考输入端口的定义）。
●0xC1（拷贝输入端口的 LSn）－仅用于 PS/2 模式。拷贝输入端口的低四位到状态寄存器（参考输入端口定义）。
●0xC2（拷贝输入端口的 MSn）－仅用于 PS/2 模式。拷贝输入端口的高四位到状态寄存器（参考输入端口定义）。
●0xD0（读输出端口）－返回输出端口的值（参考输出端口定义）。
●0xD1（写输出端口）－写参数到输出端口（参考输出端口定义）。
●0xD2（写键盘缓冲区）－把参数写到输入缓冲区就像时从键盘接收到的一样。
●0xD3（写鼠标缓冲区）－把参数写到输入缓冲区就像时从鼠标接收到的一样。
●0xD4（写鼠标设备）－发送参数给辅助的 PS/2 设备。
●0xE0（读测试端口）—返回测试端口的值（参考测试端口的定义）。
●0xF0-0xFF（脉冲输出端口）－在脉冲下输出命令的低四位到输出端口的低四位（参考输出端口定义）。

Modern Keyboard Controllers:
现代键盘控制器：

So far, I've only discussed the 8042 keyboard controller. Although modern keyboard controllers remain compatible with the original device, compatibility is their only requirement (and their goal.)
到目前为止，我只讨论了 8042 键盘控制器。虽然现代的键盘控制器保持了和原始设备的兼容性，但兼容是他们唯一的需求（和目标）。

My motherboard's keyboard controller is a great example of this. I connected a microcontroller+LCD in parallel to my keyboard to see what data is sent by the keyboard controller. At power-up, the keyboard controller sent the "Set LED state" command to turn off all LEDs, then reads the keyboard's ID. When I tried writing data to the output buffer, I found the keyboard controller only forwards the "Set LED state" command and "Set Typematic Rate/Delay" command. It does not allow any other commands to be sent to the keyboard. However, it does emulate the keyboard's response by placing "acknowledge" (0xFA) in the input buffer when appropriate (or 0xEE in response to the "Echo" command.) Furthermore, if the keyboard sends it an erroneous byte, the keyboard controller takes care of error handling (sends the "Retry" command; if byte still erroneous; sends error code to keyboard and places error code in input buffer.)
我的主板上的键盘控制器是一个重要的例子。我把一个微控制器和 LCD 并接到我的键盘上来观察键盘控制器发送了哪些数据。上电后，键盘控制器发送了“设置 LED 状态”关闭了所有的 LED，然后读取键盘的 ID。当我试着写数据到输出缓冲区，我发现键盘控制器只转发了“设置 LED 状态”命令和“设置机打速率/延时”命令。它不允许任何其他命令发往键盘。但是，在适当时候它通过放置“应答”到输入缓冲区来仿真键盘的应答（和 0xEE 回应“Echo”命令）。此外，如果键盘发送了一个错误的字节，键盘控制器将拿走错误俘获（发送“Retry”命令；如果字节仍旧是错的；发送错误代码给键盘并将错

误代码放入输入缓冲区）。

Once again, keep in mind chipset designers are more interested in compatibility than standardization.
再说一次，记住芯片组的设计者对兼容比标准更有兴趣。

Initialization:
初始化：

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.
如下的通讯过程发生在我的计算机和键盘之间，当计算机启动后。我相信嵌三个命令是初始化键盘控制器，后一条命令（使能 Numlock LED）是由 BIOS 发送的，剩下来的命令是由我的 OS（Win98SE）发送的。记住，在我计算机上这么结果是明确的，但是它指示给你一个一般性的概念告诉你启动时发生了什么。

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	
Host:	ED	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	

Other Sources / References:
其他资源/参考：

Adam's micro-Resources Home - This site's homepage.
Keyboard Scan Codes - My collection of scan code sets, verified in hardware.
PS/2 Mouse/Keyboard Protocol - Protocol used by AT and PS/2 keyboards.

Keyboard Code/Projects - My keyboard projects and source code.

National Semiconductor - "Super I/O" chipset datasheets.
IBM Archives - Non-technical historical information.
Samtech, Holtech - Keyboard encoder datasheets.
Sci.Electronics.Repair - PC Keyboard FAQ.

Adam Chapweske's Homepage - Information about me.
Email me - Questions/comments?

第三章 PS/2 鼠标接口

This document is under construction... I'll post more information as I have time...

Electrical Interface / Protocol:
电气接口/协议：

The PS/2 mouse uses the same protocol as the PS/2 (AT) keyboard. This standard originally appeared in the IBM technical reference manual, but I am not aware of any current official publication of this standard. However, you may click here for the (detailed) information I have gathered about that protocol.
PS/2 鼠标使用和 PS/2 键盘一样的协议。这个标准最初出现在 IBM 技术参考手册里，但我没有知道当前关于这个标准的任何官方文件。不过，你可以点击这里获得我收集的关于这个协议的细节。
（译者注：别点击了，就是本译文的第一章。）

Inputs, Resolution, and Scaling:
输入、分辨率和缩放比例：

The standard PS/2 mouse supports the following inputs: X (right/left) movement, Y (up/down) movement, left button, middle button, and right button. The mouse reads these inputs at a regular freqency and updates various counters and flags to reflect movement and button states. There are many PS/2 pointing devices that have additional inputs and may report data differently than described in this document. One popular extension I cover later in this document is the Microsoft Intellimouse, which includes support for the standard inputs as well as a scrolling wheel and two additional buttons.
标准的 PS/2 鼠标支持下面的输入：X（左右）位移，Y（上下）位移，左键，中键和右键。鼠标以一个固定的频率读取这些输入并更新不同的计数器然后标记出反映的移动和按键状态。有很多 PS/2 指示设备具有额外的输入并可以报告不同于本文描述的数据。一个受欢迎的扩充是我在文章后面介绍的 Microsoft 的 Intellimouse，它既支持标准输入也支持滚轮和两个附加的按键。

The standard mouse has two counters that keep track of movement: the X-movement counter and the Y-movement counter. These are 9-bit 2's complement values and each has an associated overflow flag. Their contents, along with the state of the three mouse buttons, are sent to the host in the form of a 3-byte movement data packet (as described in the next section.) The movement counters represent the amount of movement that has occurred since the last movment data packet was sent to the host.
标准的鼠标有两个计数器保持位移的跟踪：X 位移计数器和 Y 位移计数器。可存放 9 位的 2 进制补码并且每个计数器都有相关的溢出标志。它们的内容连同三个鼠标按钮的状态一起以三字节移动数据包的形式发送给主机（描述见下一部分）。位移计数器表示从最后一次位移数据包被送往主机后有位移量发生。

When the mouse reads its inputs, it records the current state of its buttons, then checks for movement. If movement has occurred, it increments (for +X or +Y movement) or decrements (for -X or -Y movement) its X and/or Y movement counters. If either of the counters has overflowed, it sets the appropriate overflow flag.
当鼠标读取它的输入的时候，它记录按键的当前状态，然后检查位移。如果位移发生，它就增加（对正位移）或减少（对负位移）X 和/或 Y 位移计数器的值。如果有一个计数器溢出了，就设置相应的溢出标志。

The parameter that determines the amount by which the movement counters are incremented/decremented is the resolution. The default resolution is 4 counts/mm and the host may change that value using the "Set Resolution" (0xE8) command.
决定位移计数器增减数量的参数叫分辨率。缺省的分辨率为 4 计数单位/毫米，主机可以用“设置分辨率”（0xE8）命令改变这个值。

There is a parameter that does not effect the movement counters, but does effect the reported(1) value of these counters. This parameter is scaling. By default, the mouse uses 1:1 scaling, which has no effect on the reported mouse movement. However, the host may select 1:2 scaling by sending the "Set Scaling 2:1" (0xE7) command. If 2:1 scaling is enabled, the mouse will apply the following algorithm to the counters before sending their contents to the host:
有一个参数不影响位移计数器的值，但是影响这些计数器报告的值（见本章的脚注 1）。这个参数就是缩放比例。缺省情况下，鼠标使用 1：1 比例，因此对报告的鼠标位移没有影响。但是主机可以用“设置比例 2：1”（0xE7）命令选择 2：1 比例。如果启用了 2：1 比例，鼠标在发数据给主机前采用如下的算法运算计数器内容：

Movement Counter	Reported Movement
0	0
1	1
2	1
3	3
4	6
5	9
N>5	2*N

Movement Data Packet:
位移数据包：

The standard PS/2 mouse sends movement (and button) information to the host using the following 3-byte packet (4):
标准的 PS/2 鼠标发送位移和按键信息给主机采用如下的 3 字节数据包格式（见本章脚注 4）：

	Bit 7	Bit 6	Bit 5	Bit 4	Bit 3	Bit 2	Bit 1	Bit 0
Byte 1	Y overflow	X overflow	Y sign bit	X sign bit	Always 1	Middle Btn	Right Btn	Left Btn
Byte 2	X Movement							
Byte 3	Y Movement							

The movement counters are 9-bit 2's complement integers, where the most significant bit appears as a sign bit in Byte 1 of the movement data packet. These counters are updated when the mouse reads its input and finds movement has occurred. Their value is the amount of movement that has occurred since the last movement data packet was sent to the host (ie, after a packet is sent to the host, the movement counters are reset.) The range of values that can be expressed by the movement counters is -255 to +255. If this range is exceeded, the appropriate overflow bit is set and the counter is not incremented/decremented until it is reset.
位移计数器是一个 9 位 2 的补码整数。它的最高位作为符号位出现在位移数据包的第一个字节里。这些计数器在鼠标读取输入发现有位移时被更新。这些值是自从最后一次发送位移数据包给主机后位移的累计量（即最后一次包发给主机后，位移计数器被复位）。位移计数器可表示的值的范围是-255 到+255。

如果超过了范围，相应的溢出位就被设置，并且在复位前，计数器不会增减。

As I mentioned earlier, the movement counters are reset whenever a movement data packet is successfully sent to the host. They are also reset after the mouse receives any command from the host other than the "Resend" (0xFE) command.
正如我前面提及的，一旦位移数据包成功地发送给主机，位移计数器就会复位。同样鼠标在收到主机不是“Resend”（0xFE）命令外的其他命令，计数器也会复位。

Modes of Operation:
操作模式：

Data reporting is handled according to the mode in which the mouse is operating. There are four standard modes of operation:
根据鼠标工作的模式来处理的数据报告。有四种标准的工作模式：

●Reset - The mouse enters Reset mode at power-up or after receiving the "Reset" (0xFF) command.
●Stream - This is the default mode (after Reset finishes executing) and is the mode in which most software uses the mouse. If the host has previously set the mouse to Remote mode, it may re-enter Stream mode by sending the "Set Stream Mode" (0xEA) command to the mouse.
●Remote - Remote mode is useful in some situations and may be entered by sending the "Set Remote Mode" (0xF0) command to the mouse.
●Wrap - This mode isn't particularly useful except for testing the connection between the mouse and its host. Wrap mode may be entered by sending the "Set Wrap Mode" (0xEE) command to the mouse. To exit Wrap mode, the host must issue the "Reset" (0xFF) command or "Reset Wrap Mode" (0xEC) command. If the "Reset" (0xFF) command is recieved, the mouse will enter Reset mode. If the "Reset Wrap Mode" (0xEC) command is received, the mouse will enter the mode it was in prior to Wrap Mode.
●Reset－鼠标在上电或收到“Reset”（0xFF）命令后进入 Reset 模式。
●Stream－这是缺省模式（在 Reset 执行完成后），也是多数软件使用鼠标的模式。如果主机先前吧鼠标设置到了 Remote 模式，那它可以发送“Set Stream Mode”（0xEA）命令给鼠标让鼠标重新进入 Stream 模式。
●Remote－在某些情况下 Remote 模式很有用，可以通过发送“Set Remote Mode”（0xF0）命令进入。
●Wrap－除了为测试鼠标和它的主机之间的连接外，这个模式不是特别地有用。Wrap 模式可以通过发送“Set Wrap Mode”（0xEE）命令给鼠标来进入。要退出 Wrap 模式，主机必须发布“Reset”（0xFF）命令或“Reset Wrap Mode”（0xEC）命令。如果“Reset”（0xFF）命令收到了，鼠标将进入 Reset 模式。如果收到的是“Reset Wrap Mode”（0xEC）命令，鼠标将进入 Wrap 模式前的那个模式。

(Note: The mouse may also enter "extended" modes of operation, as described later in this document. However, this is not a feature of the standard PS/2 mouse.)
（注意：鼠标同样可以进入“extended”操作模式，正如本文后面所述。但是，这不是标准 PS/2 鼠标的特征。）

Reset Mode:
Reset 模式：

The mouse enters reset mode at power-on or in response to the "Reset" (0xFF) command. After entring this mode, the mouse performs a diagnostic self-test referred to as BAT (Basic Assurance Test) and sets the

follwing default values:
鼠标在上电后或应答“Reset”（0xFF）命令就进入 reset 模式。进入这个模式后，鼠标执行象前面提到的 BAT（基本保证测试）一样的自检并设置如下的缺省值：

●Sample Rate - 100 samples/sec
●Resolution - 4 counts/mm
●Scaling - 1:1
●Data Reporting Disabled
●采样速率－100 采样点/秒
●分辨率－4 个计数值/毫米
●缩放比例－1:1
●数据报告被禁止

It then sends a BAT completion code of either 0xAA (BAT successful) or 0xFC (Error). If the host receives a response other than 0xAA, it may cycle the mouse's power supply, causing the mouse to reset and re-execute its BAT.
然后发送 BAT 完成代码，这个代码不是 0xAA（BAT 成功）就是 0xFC（错误）。如果主机收到了不是 0xAA 的回应，它可能重新给鼠标供电，这样来引起鼠标复位并重新执行 BAT。

Following the BAT completion code (0xAA or 0xFC), the mouse sends its device ID of 0x00. This distinguishes it from a keyboard, or a mouse in an extended mode. I have read documents saything the host is not supposed to transmit any data until it receives a device ID. However I've found that some BIOS's will send the "Reset" (0xFF) command immediately following the 0xAA received after a power-on reset.
接着 BAT 完成代码（0xAA 或 0xFC）的后面，鼠标发送它的设备 ID 0x00。这个 ID 用来区别设备是键盘还是处于扩展模式中的鼠标。我读到的文件中说，主机在没收到设备 ID 前不会假定发送任何数据。但是我发现有些 BIOS 在上电复位并收到 0xAA 后立刻发送“Reset”（0xFF）命令。

After the mouse has sent its device ID to the host, it will enter Stream Mode. Note that one of the default values set by the mouse is "Data Reporting Disabled". This means the mouse will not send any movement data packets to the host until the "Enable Data Reporting" (0xF4) command is received.
鼠标发送自己的设备 ID 给主机后，它就进入了 Stream 模式。注意鼠标设置的一个缺省值之一是“数据报告被禁止”。这就意味着鼠标在没收到“使能数据报告”（0xF4）命令之前不会发送任何位移数据包给主机。

Stream Mode:
Stream 模式：

In stream mode, the mouse sends movement data when it detects movement or a change in state of one or more mouse buttons. The maximum rate at which this data reporting may occur is known as the sample rate. This parameter ranges from 10 samples/sec to 200 samples/sec. Its default value is 100 samples/sec and the host may change that value by using the "Set Sample Rate" (0xF3) command. Stream mode is the default mode of operation.
在 Stream 模式中，一旦鼠标检测到位移或发现一个或多个鼠标键的状态改变了，就发送位移数据包。数据报告的最大速率被认为是采样速率。参数的范围从 10 采样点/秒到 200 采样点/秒。这个参数的缺省值是 100 采样点/秒，主机可以用“设置采样速率”（0xF3）命令来改变它。Stream 模式是操作的缺省模式。

Remote Mode:
Remote 模式：

In this mode, the mouse reads its inputs and updates its counters/flags at the current sampling rate, but it only notifies the host of movement (and change in button state) when that information is requested by the host. The host does this by issuing the "Read Data" (0xEB) command. After receiveing this command, the mouse will send a movement data packet, and reset its movement counters.
在这个模式下，鼠标以当前的采样速率读取输入并更新它的计数器和标志，但是它只在主机请求数据的时候才报告给主机位移（和按键状态）。主机通过“读数据”（0xEB）命令来获得数据。在收到命令后，鼠标发送位移数据包并复位它的位移计数器。

Wrap Mode:
Wrap 模式：

This is an "echoing" mode in which every byte received by the mouse is sent back to the host. Even if the byte represents a valid command, the mouse will not respond to that command--it will only echo that byte back to the host. There are two exceptions to this: the "Reset" (0xFF) command and "Reset Wrap Mode" (0xEC) command. The mouse treats these as valid commands and does not echo them back to the host.
这是一个“回声”模式，鼠标收到的每个字节都会被发回主机。甚至收到的是一个有效的命令，鼠标都不会应答这条命令－它只把这个字节回送给主机。但是有两个例外：“Reset”（0xff）命令和“Reset Wrap Mode”（0xEC）命令。鼠标认为这两条命令是一有效的命令并且不会回送它们到主机。

Intellimouse Extensions:
Intellimouse 的扩展：

A popular extension to the standard PS/2 mouse is the Microsoft Intellimouse. This includes support for a total of five mouse buttons and three axises of movement (right-left, up-down, and a scrolling wheel). These additional features require the use of a 4-byte movement data packet rather than the standard 3-byte packet. Since standard PS/2 mouse drivers cannot recognize this packet format, the Microsoft Intellimouse is required to operate exactly like a standard PS/2 mouse unless it knows the drivers support the extended packet format. This way, if a Microsoft Intellimouse is used on a computer which only supports the standard PS/2 mouse, the Microsoft Intellimouse will still function, except for its scrolling wheel and 4th and 5th buttons.
对标准的 PS/2 鼠标的一个流行的扩展是微软的 Intellimouse。它包括支持五个鼠标按键和三个位移轴（左右、上下和滚轮）。这些附加特征要求使用 4 字节的位移数据包而不是标准 3 字节包。因为标准 PS/2 鼠标

The Microsoft Intellimouse operates just like a standard PS/2 mouse (ie, it uses a 3-byte movement data packet, responds to all commands in the same way as a standard PS/2 mouse, and reports a device ID of 0x00.) To enter scrolling wheel mode, the host sends the following command sequence:
微软的 Intellimouse 工作起来象标准的 PS/2 鼠标（也就是，使用 3 字节位移数据包，和标准 PS/2 鼠标一样回应所有命令，报告设备 ID0x00）。要进入滚轮模式，主机应该发送如下的命令序列：

Set sample rate 200
Set sample rate 100
Set sample rate 80

The host then issues the "Get device ID" command (0xF2) and waits for a response. If a standard PS/2 mouse (ie, non-Intellimouse) is attached, it will respond with a device ID of 0x00. In this case, the host will recognize the fact that the mouse does have a scrolling wheel and will continue to treat it as a standard PS/2 mouse. However, if a Microsoft Intellimouse is attached, it will respond with an ID of 0x03. This tells the host that the attached pointing device has a scrolling wheel and the host will then expect the mouse to use the following 4-byte movement data packet:
主机然后应该发布“获得设备 ID”命令（0xF2）并等待回应。如果安装的是是标准 PS/2 鼠标（非 Intellimouse），它回应设备 ID0x00。在这种情况下，主机回辨认出实际这个阿成边没有滚轮并继续把它当作是标准 PS/2 鼠标。但是，如果安装的是微软的 Intellimouse，它返回的 ID 是 0x03。这就告诉主机挂接的定点设备有滚轮并且主机认为鼠标使用 4 字节的位移数据包：

	Bit 7	Bit 6	Bit 5	Bit 4	Bit 3	Bit 2	Bit 1	Bit 0
Byte 1	Y overflow	X overflow	Y sign bit	X sign bit	Always 1	Middle Btn	Right Btn	Left Btn
Byte 2	X Movement							
Byte 3	Y Movement							
Byte 4	Z Movement							

Z Movement is a 2's complement number that represents the scrolling wheel's movement since the last data report. Valid values are in the range of -8 to +7. This means the number is actually represented only by the least significant four bits; the upper four bits act only as sign extension bits.
Z 位移是 2 的补码表示滚轮的自上次数据报告以来的位移。有效值的范围在-8 到+7。这意味着数值实际只有低四位；高四位仅用作符号扩展位。

To enter scrolling wheel + 5 button mode, the host sends the following command sequence:
要进入滚轮+5 键模式，主机要发送如下命令序列：

Set sample rate 200
Set sample rate 200
Set sample rate 80

The host then issues the "Get device ID" command (0xF2) and waits for a response. A Microsoft Intellimouse will respond with a device ID of 0x04, then use the following 4-byte movement data packet:
主机接着发布“获得设备 ID”命令（0xF2）并等待回应。微软的 Intellimouse 用 0x04 这样设备 ID 应答，并且使用如下的 4 字节位移数据包：

	Bit 7	Bit 6	Bit 5	Bit 4	Bit 3	Bit 2	Bit 1	Bit 0
Byte 1	Y overflow	X overflow	Y sign bit	X sign bit	Always 1	Middle Btn	Right Btn	Left Btn
Byte 2	X Movement							
Byte 3	Y Movement							
Byte 4	Always 0	Always 0	5th Btn	4th Btn	Z3	Z2	Z1	Z0

Z0-Z3 is a 2's complement number which represents the amount of movement that has occurred since the last data report. Valid values range from -8 to +7.
4th Btn: 1 = 4th mouse button is pressed; 0 = 4th mouse button is not pressed.
5th Btn: 1 = 5th mouse button is pressed; 0 = 5th mouse button is not pressed.
Z0-Z3 是 2 的补码用于表示从上次数据报告以来滚轮的位移量，有效范围从-8 到+7。
第 4 键：1＝第 4 键按下了；0＝第 4 键没有按下。

第 5 键：1＝第 5 键按下了；0＝第 5 键没有按下。

You may have seen mice with two scrolling wheels--one vertical and the other horizontal. These mice use the Microsoft Intellimouse data packet format as described above. If the vertical wheel is scrolled upward, the Z-counter is incremented by one and if that wheel is scrolled down, the Z-counter is decremented by one. This is normal operation for a scrolling wheel. However, if the horizontal wheel is scrolled right, the Z-counter is incremented by two and if it is scrolled left, the Z-counter is decremented by two. This seems like an odd way to implement the second scrolling wheel, but it works since the placement of the two wheels make it impossible to use both of them at the same time (and if you try to trick the software and use both at the same time, it will ignore the horizontal wheel.)
你也许见过有两个滚轮的鼠标：一个是垂直的一个是水平的。这种鼠标使用上面介绍的微软 Intellimouse 数据包格式。如果垂直的滚轮向上滚动，Z 计数器加 1；如果这个滚轮向下滚动，Z 计数器减 1。这是滚轮的正常操作。但是如果水平的滚轮向右滚动，Z 计数器增加 2；向左滚动，Z 计数器减少 2。看上去象用一种临时的途径实现了第二个滚轮，但是由于放置了两个滚轮所以不可能同时使用这两个滚轮（如果你试着欺骗软件，同时使用它们，你会发现软件忽略了水平的那个滚轮）。

Command Set:
命令集：

The following are the only commands that may be sent to the mouse... If the mouse is in Stream mode, the host should disable data reporting (command 0xF5) before sending any other commands...
下面列出的是仅可发送给鼠标的命令。如果鼠标工作在 Stream 模式，主机在发送任何其他命令之前要线禁止数据报告（命令 0xF5）。

- 0xFF (Reset) - The mouse responds to this command with "acknowledge" (0xFA) then enters Reset Mode.
- 0xFF（Reset）－鼠标用“应答”（0xFA）回应这条命令并进入 Reset 模式。
- 0xFE (Resend) - The host sends this command whenever it receives invalid data from the mouse. The mouse responds by resending the last(2) packet(3) it sent to the host. If the mouse responds to the "Resend" command with another invalid packet, the host may either issue another "Resend" command, issue an "Error" command, cycle the mouse's power supply to reset the mouse, or it may inhibit communication (by bringing the Clock line low). The action taken depends on the host.
- 0xFE（Resend）－只要从鼠标收到无效数据主机就发送这条命令。鼠标的回应是重新发送它最后发给主机的数据包（见本章脚注 2、3）。如果鼠标用了另外一个非法的包来回应，主机要么发布另一条“Resend”命令，要么发布“Error”命令，要么让鼠标重建上电来复位它，或者禁止通讯（把时钟线拉低）。采取什么样的动作取决于主机。
- 0xF6 (Set Defaults) - The mouse responds with "acknowledge" (0xFA) then loads the following values: Sampling rate = 100, Resolution = 4 counts/mm, Scaling = 1:1, Disable Data Reporting. The mouse then resets its movement counters and enters stream mode.
- 0xF6（Set Defaults）－鼠标用“应答”（0xFA）来回应，然后载入如下的值：采样率＝100，分辨率＝4 个值/毫米，比例＝1：1，禁止数据报告。接着鼠标清空它所有的位移计数器并进入 stream 模式。
- 0xF5 (Disable Data Reporting) - The mouse responds with "acknowledge" (0xFA) then disables data reporting and resets its movement counters. This only effects data reporting in Stream mode and does not disable sampling. Disabled stream mode funcions the same as remote mode.
- 0xF5（Disable Data Reporting）－鼠标用“应答”（0xFA）回应命令，然后禁止数据报告并复位它的位移计数器。这仅对 Stream 模式下的数据报告有效并且它不能禁止采样。禁止的 stream 模式功能与 remote 模式相同。

- 0xF4 (Enable Data Reporting) - The mouse responds with "acknowledge" (0xFA) then enables data reporting and resets its movement counters. This command may be issued while the mouse is in Remote Mode (or Stream mode), but it will only effect data reporting in Stream mode.
- 0xF4（Enable Data Reporting）－鼠标用“应答”（0xFA）回应命令，然后使能数据报告并复位它的位移计数器。这条命令可以对在 Remote 模式（或 Stream 模式）下的鼠标发布，但只对 Stream 模式下的数据报告有效。
- 0xF3 (Set Sample Rate) - The mouse responds with "acknowledge" (0xFA) then reads one more byte from the host. The mouse saves this byte as the new sample rate. After receiving the sample rate, the mouse again responds with "acknowledge" (0xFA) and resets its movement counters. Valid sample rates are 10, 20, 40, 60, 80, 100, and 200 samples/sec.
- 0xF3（Set Sample Rate）－鼠标用“应答”（0xFA）回应命令，然后从主机读入一个或更多字节。鼠标保留这个字节作为新的采样速率。在收到采样速率后，鼠标再次用“应答”（0xFA）回应并复位它的位移计数器。有效的采样速率是 10、20、40、60、80、100 和 200 采样点/秒。
- 0xF2 (Get Device ID) - The mouse responds with "acknowledge" (0xFA) followed by its device ID (0x00 for the standard PS/2 mouse.) The mouse should also reset its movement counters.
- 0xF2（Get Device ID）－鼠标用“应答”（0xFA）回应命令后面跟着它的设备 ID（对标准 PS/2 鼠标来说是 0x00）。鼠标同样会复位它的位移计数器。
- 0xF0 (Set Remote Mode) - The mouse responds with "acknowledge" (0xFA) then resets its movement counters and enters remote mode.
- 0xF0（Set Remote Mode）－鼠标用“应答”（0xFA）回应，然后复位它的位移计数器，并进入 Remote 模式。
- 0xEE (Set Wrap Mode) - The mouse responds with "acknowledge" (0xFA) then resets its movement counters and enters wrap mode.
- 0xEE（Set Wrap Mode）－鼠标用“应答”（0xFA）回应，然后复位它的位移计数器，并进入 wrap 模式。
- 0xEC (Reset Wrap Mode) - The mouse responds with "acknowledge" (0xFA) then resets its movement counters and enters the mode it was in prior to wrap mode (Stream Mode or Remote Mode.)
- 0xEC（Reset Wrap Mode）－鼠标用“应答”（0xFA）回应，然后复位它的位移计数器，并进入 wrap 模式之前的那个模式（stream 模式或 remote 模式）。
- 0xEB (Read Data) - The mouse responds with acknowledge (0xFA) then sends a movement data packet. This is the only way to read data in Remote Mode. After the data packets has been successfully sent, it resets its movement counters.
- 0xEB（Read Data）－鼠标用“应答”（0xFA）回应，然后发送位移数据包。这是在 remote 模式中读数据的位移方法。在数据包成功地被发送后，鼠标将复位它的位移计数器。
- 0xEA (Set Stream Mode) - The mouse responds with "acknowledge" then resets its movement counters and enters steram mode.
- 0xEA（Set Stream Mode）－鼠标用“应答”（0xFA）回应，然后复位它的位移计数器，并进入 stream 模式。
- 0xE9 (Status Request) - The mouse responds with "acknowledge" then sends the following 3-byte status packet (then resets its movement counters.):
- 0xE9（Status Request）－鼠标用“应答”（0xFA）回应，然后发送如下 3 个字节的状态包（然后复位它的位移计数器）：

	Bit 7	Bit 6	Bit 5	Bit 4	Bit 3	Bit 2	Bit 1	Bit 0
Byte 1	Always 0	Mode	Enable	Scaling	Always 0	Left Btn	Middle Btn	Right Btn
Byte 2	Resolution							

Byte 3	Sample Rate

Right, Middle, Left Btn = 1 if button pressed; 0 if button is not pressed.
Scaling = 1 if scaling is 2:1; 0 if scaling is 1:1. (See commands 0xE7 and 0xE6)
Enable = 1 if data reporting is enabled; 0 if data reporting is disabled. (See commands 0xF5 and 0xF4)
Mode = 1 if Remote Mode is enabled; 0 if Stream mode is enabled. (See commands 0xF0 and 0xEA)
右键、中键、左键＝1 表示键被按下；＝0 表示键没有按下。
Scaling = 1 缩放比例位 2：1；＝0 比例为 1：1。（见命令 0xE7 和 0xE6）
Enable = 1 表示数据报告被使能；＝0 表示数据报告被禁止。（见命令 0xF5 和 0xF4）
Mode = 1 表示 remote 模式被使能；＝0 表示 stream 模式被使能。（见命令 0xF0 和 0xEA）

- 0xE8 (Set Resolution) - The mouse responds with acknowledge (0xFA) then reads one byte from the host and again responds with acknowledge (0xFA) then resets its movement counters. The byte read from the host determines the resolution as follows:
- 0xE8（Set Resolution）－鼠标用“应答”（0xFA）回应，然后从主机读取一个字节，并再次用“应答”（0xFA）回应，然后复位它的位移计数器。从主机读入的字节决定了分辨率，如下示：

Byte Read from Host	Resolution
0x00	1 count/mm
0x01	2 count/mm
0x02	4 count/mm
0x03	8 count/mm

- 0xE7 (Set Scaling 2:1) - The mouse responds with acknowledge (0xFA) then enables 2:1 scaling (discussed earlier in this document.)
- 0xE7（Set Scaling 2:1）－鼠标用“应答”（0xFA）回应，然后使能 2：1 比例（在本文前面讨论过）。
- 0xE6 (Set Scaling 1:1) - The mouse responds with acknowledge (0xFA) then enables 1:1 scaling (discussed earlier in this document.)
- 0xE6（Set Scaling 1:1）－鼠标用“应答”（0xFA）回应，然后使能 1：1 比例（在本文前面讨论过）

The only commands the standard PS/2 mouse will send to the host are the "Resend" (0xFE) and "Error" (0xFC) commands. They both work the same as they do as host-to-device commands.
对于标准鼠标而言只有“Resend”（0xFE）和“Error”（0xFC）命令会发送给主机。这两条命令的工作情况和主机到设备间的命令一样。

Initialization:
初始化：

The PS/2 mouse is normally detected/initialized only when the computer is booting up. That is, the mouse is not hot-pluggable and you must restart your computer whenever you add/remove a PS/2 mouse (furthermore, some motherboards may be damaged if you add/remove a PS/2 mouse while the computer is running.)
正常情况下，PS/2 鼠标仅在计算机启动的守候被检测和初始化。因此，鼠标不能热插拔，只要你增加和移走 PS/2 鼠标，你就要重新启动你的机器（此外，如果你在计算机开着的时候增加/移走 PS/2 鼠标，可能会损坏某些主板）。

The initial detection of the PS/2 mouse occurrs during POST. If a mouse is detected, the BIOS will allow the

operating system to configure/enable the mouse. Otherwise, it will inhibit communication on the mouse's bus. If you boot the computer with a mouse attached, then detach/reattach the mouse while in Windows, the OS may be able to detect the mouse was reattached. Microsoft tried to support this, but it only works about 50% of the time.
PS/2 鼠标的初始检测发生在 POST 期间。如果鼠标检测到了，BIOS 将允许操作系统配置/使能鼠标。否则 OS 将禁止在鼠标总线上的通讯。如果你启动一台插右鼠标的计算机，然后在 windows 里拔掉又重新插上鼠标，OS 也许能检测到鼠标又重新插上了。Microsoft 显然想试着支持这样的操作，但是只可能大约 50％的情况下，它是工作的。

The following is the communication between my computer (running Win98SE) and mouse when it boots up with a standard PS/2 mouse attached. It is fairly typical of how a PS/2 mouse is initialized and if you want to emulate a PS/2 mouse it must (at minimum) be able to support the following sequence of commands...
下面是在我的计算机（运行 Win98SE）和鼠标之间的通讯，启动的时候插有一个 PS/2 鼠标。PS/2 鼠标的初始化过程相当的典型。如果你要仿真一个 PS/2 鼠标，必须（至少）能支持如下的命令序列：

Power-on Reset:
上电复位：

```
Mouse: AA   Self-test passed
Mouse: 00   Mouse ID
Host:  FF   Reset command
Mouse: FA   Acknowledge
Mouse: AA   Self-test passed
Mouse: 00   Mouse ID
Host:  FF   Reset command
Mouse: FA   Acknowledge
Mouse: AA   Self-test passed
Mouse: 00   Mouse ID
Host:  FF   Reset command
Mouse: FA   Acknowledge
Mouse: AA   Self-test passed
Mouse: 00   Mouse ID
Host:  F3   Set Sample Rate    : Attempt to Enter Microsoft
Mouse: FA   Acknowledge        : Scrolling Mouse mode
Host:  C8   decimal 200        :
Mouse: FA   Acknowledge        :
Host:  F3   Set Sample Rate    :
Mouse: FA   Acknowledge        :
Host:  64   decimal 100        :
Mouse: FA   Acknowledge        :
Host:  F3   Set Sample Rate    :
Mouse: FA   Acknowledge        :
Host:  50   decimal 80         :
Mouse: FA   Acknowledge        :
Host:  F2   Read Device Type   :
Mouse: FA   Acknowledge        :
Mouse: 00   Mouse ID           : Response 03 if microsoft scrolling mouse
```

```
Host:  F3  Set Sample Rate
Mouse: FA  Acknowledge
Host:  0A  decimal 10
Mouse: FA  Acknowledge
Host:  F2  Read Device Type
Mouse: FA  Acknowledge
Mouse: 00  Mouse ID
Host:  E8  Set resolution
Mouse: FA  Acknowledge
Host:  03  8 Counts/mm
Mouse: FA  Acknowledge
Host:  E6  Set Scaling 1:1
Mouse: FA  Acknowledge
Host:  F3  Set Sample Rate
Mouse: FA  Acknowledge
Host:  28  decimal 40
Mouse: FA  Acknowledge
Host:  F4  Enable
Mouse: FA  Acknowledge
Initialization complete...
```

初始化完成。

If I then press the Left Button...
如果我按左键：

```
Mouse: 09 1 1 00001001; bit0 = Left button state; bit3 = always 1
Mouse: 00 1 1 No X-movement
Mouse: 00 1 1 No Y-movement
```

... and release the Left Button:
然后释放左键：

```
Mouse: 08 0 1 00001000 bit0 = Left button state; bit3 = always 1
Mouse: 00 1 1 No X-movement
Mouse: 00 1 1 No Y-movement
```

The following is the communication between my computer (running Win98SE) and mouse when it boots up with an (emulated) Intellimouse...
下面还是我的计算机和鼠标间的通讯，但这次启动时插有一个（仿真的）Intellimouse。

Power-on Reset:
上电复位：

```
Mouse: AA  Self-test passed
Mouse: 00  Mouse ID
Host:  FF  Reset command
Mouse: FA  Acknowledge
Mouse: AA  Self-test passed
Mouse: 00  Mouse ID
```

```
Host:    FF    Reset command
Mouse: FA    Acknowledge
Mouse: AA    Self-test passed
Mouse: 00    Mouse ID
Host:    FF    Reset command
Mouse: FA    Acknowledge
Mouse: AA    Self-test passed
Mouse: 00    Mouse ID
Host:    F3    Set Sample Rate      : Attempt to Enter Microsoft
Mouse: FA    Acknowledge             : Scrolling Mouse mode
Host:    C8    decimal 200             :
Mouse: FA    Acknowledge             :
Host:    F3    Set Sample Rate      :
Mouse: FA    Acknowledge             :
Host:    64    decimal 100             :
Mouse: FA    Acknowledge             :
Host:    F3    Set Sample Rate      :
Mouse: FA    Acknowledge             :
Host:    50    decimal 80               :
Mouse: FA    Acknowledge             :
Host:    F2    Read Device Type    :
Mouse: FA    Acknowledge             :
Mouse: 03    Mouse ID                  : Response 03 if microsoft scrolling mouse
Host:    E8    Set Resolution
Mouse: FA    Acknowledge
Host:    03    8 counts/mm
Mouse: FA    Acknowledge
Host:    E6    Set scaling 1:1
Dev:       FA    Acknowledge
Host:    F3    Set Sample Rate
Mouse: FA    Acknowledge
Host:    28    decimal 40
Mouse: FA    Acknowledge
Host:    F4    Enable device
Mouse: FA    Acknowledge
```

If I then press the left mouse button:
如果我按了一个鼠标左键：

```
Mouse: 09    00001001 bit0 = Left button state; bit3 = always 1
Mouse: 00    No X-movement
Mouse: 00    No Y-movement
Mouse: 00    No Z-movement
```

...and then release the left mouse button button:
然后释放鼠标左键：

```
Mouse: 08    00001000 bit0 = Left button state; bit3 = always 1
```

```
Mouse: 00    No X-movement
Mouse: 00    No Y-movement
Mouse: 00    No Z-movement
```

After I downloaded/installed the Microsoft's Intellimouse drivers with support for the 4th and 5th buttons, the following sequence was found:
在我下载并安装了微软支持第 4 和第 5 键的 Intellimouse 的驱动后，发现了如下序列：

```
... (starts same as before) ...
（开始的部分和前面相同）
Host:   F3   Set Sample Rate     : Attempt to Enter Microsoft
Mouse: FA    Acknowledge          : Scrolling Mouse mode.
Host:   C8   decimal 200          :
Mouse: FA    Acknowledge          :
Host:   F3   Set Sample Rate     :
Mouse: FA    Acknowledge          :
Host:   64   decimal 100          :
Mouse: FA    Acknowledge          :
Host:   F3   Set Sample Rate     :
Mouse: FA    Acknowledge          :
Host:   50   decimal 80           :
Mouse: FA    Acknowledge          :
Host:   F2   Read Device Type    :
Mouse: FA    Acknowledge          :
Mouse: 03    Mouse ID             : Response 03 if microsoft scrolling mouse.
Host:   F3   Set Sample Rate     : Attempt to Enter Microsoft 5-button
Mouse: FA    Acknowledge          : Scrolling Mouse mode.
Host:   C8   decimal 200          :
Mouse: FA    Acknowledge          :
Host:   F3   Set Sample Rate     :
Mouse: FA    Acknowledge          :
Host:   C8   decimal 200          :
Mouse: FA    Acknowledge          :
Host:   F3   Set Sample Rate     :
Mouse: FA    Acknowledge          :
Host:   50   decimal 80           :
Mouse: FA    Acknowledge          :
Host:   F2   Read Device Type    :
Mouse: FA    Acknowledge          :
Mouse: 04    Mouse ID            : Response 04 if 5-button scrolling mouse.
... rest of initialization same as before ...
```

初始化的剩余部分和前面相同。

Emulation/Interfacing:
仿真/接口：

- Click here for routines that emulate a PS/2 mouse or keyboard
- 点击这里获得仿真 PS/2 鼠标或键盘的例程
- Click here for routines that emulate a PS/2 host (ie, interface a mouse/keyboard)
- 点击这里获得仿真 PS/2 主机的例程（即，和鼠标/键盘的接口）
- Click here for a fully-functional PS/2 mouse written for the PIC16F84.
- 点击这里获得使用 PIC16F84 的全功能 PS/2 鼠标

（译者注：所有这些例程全部收录在文章的附录 4、5、6 中。呵呵！）

If you want to build a truely fully-implemented mouse or host, you should implement all of the features described in this document (except for, of course, the Microsoft Intellimouse extensions, which are optional). However, at an absolute minimum, your device should operate as follows:
如果你要建立一个真正的完全实现的鼠标或主机，你应该实现本文描述的所有特征（当然除了微软的 Intellimouse 的扩展，因为它是可选的）。但最骑马你的设备要可以象下面讲的那样操作：

To Emulate a Mouse:
要想仿真一个鼠标：

- Never send data when the "Clock" line low. If the host pulls the "Data" line low, prepare to read a byte from the host.
- 绝对不要在“时钟”线为低的时候发送数据。如果主机拉低时钟线，就准备从主机读取数据。
- ~500 milliseconds after powerup, transmit "0xAA, 0x00".
- 上电后 500 毫秒左右，发送“0xAA、0x00”。
- Wait for the host to send the enable (0xF4) command before sending any movement/button data.
- 在发送任何位移/按键数据前等待主机发送使能（0xF4）命令。
- Emulate the various mouse functions as follows:
- 要仿真的各种鼠标功能如下：

Emulated Action	**Data sent to host**
Move up one	0x08,0x00,0x01
Move down one	0x28,0x00,0xFF
Move right one	0x08,0x01,0x00
Move left one	0x18,0xFF,0x00
Press left button	0x09,0x00,0x00
Release left button	0x08,0x00,0x00
Press middle button	0x0C,0x00,0x00
Release middle button	0x08,0x00,0x00
Press right button	0x0A,0x00,0x00
Release right button	0x08,0x00,0x00

- Respond to the "Reset" (0xFF) command with "0xFA" then goto the beginning of your program. (ie, send 0xAA, 0x00, then wait for the enable command before sending any movement/button data.)
- 用“0xFA”回应“Reset”（0xFF）命令，然后跳转到你程序的开始处。(即，发送 0xAA、0x00，在发送位移数据前等待使能命令）
- Respond to the "Get Device ID" (0xF2) command with "0xFA, 0x00".
- 用“0xFA、0x00”回应“获得设备 ID”（0xF2）命令。
- Respond to the "Status Request" (0xE9) command with "0xFA, 0x00, 0x02, 0x64".

- 用“0xFA、0x00、0x02、0x64”回应“状态请求”（0xE9）命令。
- Respond to all other commands with acknowledge (0xFA).
- 用“应答”（0xFA）回应所有其他的命令。

To Interface a Mouse:
要和鼠标接口：

- Wait for the mouse to send "0xAA", then send the "Enable" (0xF4) command.
- 等待鼠标发送“0xAA”后，然后发送“使能”（0xF4）命令。
- The mouse will then send a 3-byte movement packets as described earlier in this document.
- 然后鼠标将发送本文前面描述的那种 3 字节的位移数据包。

Footnotes:
脚注：

1) 2:1 scaling only applies to the automatic data reporting in Stream mode. It does not effect the reported data sent in response to the "Read Data" (0xEB) command.
2：1 比例仅适用于 Stream 模式的自动数据报告中，对于回应“Read Data”（0xEB）命令的报告数据是无效的。

2) The mouse and host do not buffer "Resend" (0xFF) commands. This means "0xFE" will never be sent in response to the "Resend" command.
鼠标和主机不缓冲“Resend”（0xFF）命令。这意味着“0xFE”绝不会作为“Resend”命令的回应来发送。

3) A "packet" may be a 3-byte movement data packet, a 4-byte movement data packet (for the Intellimouse), a 3-byte status packet (see "Status Request" [0xE9] command) a 2-byte completion-code-ID packet (0xAA,0x00 or 0xFC,0x00), or a 1-byte response to a command.
一个“数据包”可以是 3 字节的位移数据包，或 4 字节的位移数据包（Intellimouse 的），或 3 字节的状态包（键“Status Request”0xE9 命令），或 2 字节的完成代码 ID 包（0xAA、0x00 或 0xFC、0x00），或 1 字节的命令回应。

4) A little advice from my own experience... Even though bit 3 of the first byte in a movement data packet is supposed to be set, some drivers (such as the standard PS/2 mouse driver included with Windows 98SE) don't care and just ignore that bit. However, other drivers do check that bit and if it is not set, it is considered an error. I mention this so that, if you're designing a mouse, you double-check that this bit is set in every movement data packet sent by your mouse. If it is not, your mouse may work properly when you test it on your computer, but it may not work on other computers that use different mouse drivers.
我自己经验的一点小建议：即使位移数据包中首字节的第 3 位被设置了，某些驱动（诸如标准 PS/2 鼠标驱动，包括 windows98SE 的）并不关心且指示忽略掉这一位。但是其他驱动可能会检查这一位是否设置了，如果没设置则认为是一个错误。我提及这点的意思是，如果你设计一个鼠标，你要检查由你的鼠标发送出来的每个位移数据包中这一位是否被设置了。如果没有，那么你的鼠标在你计算机上测试时可能工作得很好，而到别的使用不同鼠标驱动的计算机上就不工作了。

For example, if using MS Intellimouse drivers and bit 3 of the first byte in a movement data packet is not set, the driver will discard that packet, then send the "Disable Data Reporting" (0xF5) command, followed by the

"Set Defaults" (0xF6) command, then it will reinitialize the mouse using the same command sequence as it does when Windows boots up (see the "Initialization" section above.)
例如，如果使用 MS Intellimouse 驱动，位移数据包中首字节的第 3 位没有设置，驱动程序将丢弃这个包，然后发送“Disable Data Reporting”（0xF5）命令，跟着是“Set Defaults”（0xF6）命令。然后使用 windows 启动时相同的命令序列重新初始化鼠标（见上面的“初始化”部分）。

Other Sources / References:
其他资源/参考：

Holtek - Informative datasheets on many different PS/2 mice (and other peripherals).
EMC - More inormative datasheets on many different PS/2 mice (and an ADB mouse).
Synaptics Touchpad Interfacing Guide -Very informative!
More links - Many more links to related information.

附录一 第一套键盘扫描码

*所有的值都是十六进制的。

101、102 和 104 键的键盘：

KEY	MAKE	BREAK	KEY	MAKE	BREAK	KEY	MAKE	BREAK
A	1E	9E	9	0A	8A	[	1A	9A
B	30	B0	`	29	89	INSERT	E0,52	E0,D2
C	2E	AE	-	0C	8C	HOME	E0,47	E0,97
D	20	A0	=	0D	8D	PG UP	E0,49	E0,C9
E	12	92	\	2B	AB	DELETE	E0,53	E0,D3
F	21	A1	BKSP	0E	8E	END	E0,4F	E0,CF
G	22	A2	SPACE	39	B9	PG DN	E0,51	E0,D1
H	23	A3	TAB	0F	8F	U ARROW	E0,48	E0,C8
I	17	97	CAPS	3A	BA	L ARROW	E0,4B	E0,CB
J	24	A4	L SHFT	2A	AA	D ARROW	E0,50	E0,D0
K	25	A5	L CTRL	1D	9D	R ARROW	E0,4D	E0,CD
L	26	A6	L GUI	E0,5B	E0,DB	NUM	45	C5
M	32	B2	L ALT	38	B8	KP /	E0,35	E0,B5
N	31	B1	R SHFT	36	B6	KP *	37	B7
O	18	98	R CTRL	E0,1D	E0,9D	KP -	4A	CA
P	19	99	R GUI	E0,5C	E0,DC	KP +	4E	CE
Q	10	19	R ALT	E0,38	E0,B8	KP EN	E0,1C	E0,9C
R	13	93	APPS	E0,5D	E0,DD	KP .	53	D3
S	1F	9F	ENTER	1C	9C	KP 0	52	D2
T	14	94	ESC	01	81	KP 1	4F	CF
U	16	96	F1	3B	BB	KP 2	50	D0
V	2F	AF	F2	3C	BC	KP 3	51	D1
W	11	91	F3	3D	BD	KP 4	4B	CB
X	2D	AD	F4	3E	BE	KP 5	4C	CC
Y	15	95	F5	3F	BF	KP 6	4D	CD
Z	2C	AC	F6	40	C0	KP 7	47	C7
0	0B	8B	F7	41	C1	KP 8	48	C8
1	02	82	F8	42	C2	KP 9	49	C9
2	03	83	F9	43	C3	]	1B	9B
3	04	84	F10	44	C4	;	27	A7
4	05	85	F11	57	D7	'	28	A8
5	06	86	F12	58	D8	,	33	B3
6	07	87	PRNT SCRN	E0,2A, E0,37	E0,B7, E0,AA	.	34	B4
7	08	88	SCROLL	46	C6	/	35	B5
8	09	89	PAUSE	E1,1D,45 E1,9D,C5	-NONE-			

ACPI 扫描码

Key	**Make Code**	**Break Code**
Power	E0, 5E	E0, DE
Sleep	E0, 5F	E0, DF
Wake	E0, 63	E0, E3

Windows 多媒体扫描码：

Key	Make Code	Break Code
Next Track	E0, 19	E0, 99
Previous Track	E0, 10	E0, 90
Stop	E0, 24	E0, A4
Play/Pause	E0, 22	E0, A2
Mute	E0, 20	E0, A0
Volume Up	E0, 30	E0, B0
Volume Down	E0, 2E	E0, AE
Media Select	E0, 6D	E0, ED
E-Mail	E0, 6C	E0, EC
Calculator	E0, 21	E0, A1
My Computer	E0, 6B	E0, EB
WWW Search	E0, 65	E0, E5
WWW Home	E0, 32	E0, B2
WWW Back	E0, 6A	E0, EA
WWW Forward	E0, 69	E0, E9
WWW Stop	E0, 68	E0, E8
WWW Refresh	E0, 67	E0, E7
WWW Favorites	E0, 66	E0, E6

附录二 第二套键盘扫描码

*所有的值都是十六进制的。

101、102 和 104 键的键盘：

KEY	MAKE	BREAK	KEY	MAKE	BREAK	KEY	MAKE	BREAK
A	1C	F0,1C	9	46	F0,46	[	54	FO,54
B	32	F0,32	`	0E	F0,0E	INSERT	E0,70	E0,F0,70
C	21	F0,21	-	4E	F0,4E	HOME	E0,6C	E0,F0,6C
D	23	F0,23	=	55	FO,55	PG UP	E0,7D	E0,F0,7D
E	24	F0,24	\	5D	F0,5D	DELETE	E0,71	E0,F0,71
F	2B	F0,2B	BKSP	66	F0,66	END	E0,69	E0,F0,69
G	34	F0,34	SPACE	29	F0,29	PG DN	E0,7A	E0,F0,7A
H	33	F0,33	TAB	0D	F0,0D	U ARROW	E0,75	E0,F0,75
I	43	F0,43	CAPS	58	F0,58	L ARROW	E0,6B	E0,F0,6B
J	3B	F0,3B	L SHFT	12	FO,12	D ARROW	E0,72	E0,F0,72
K	42	F0,42	L CTRL	14	FO,14	R ARROW	E0,74	E0,F0,74
L	4B	F0,4B	L GUI	E0,1F	E0,F0,1F	NUM	77	F0,77
M	3A	F0,3A	L ALT	11	F0,11	KP /	E0,4A	E0,F0,4A
N	31	F0,31	R SHFT	59	F0,59	KP *	7C	F0,7C
O	44	F0,44	R CTRL	E0,14	E0,F0,14	KP -	7B	F0,7B
P	4D	F0,4D	R GUI	E0,27	E0,F0,27	KP +	79	F0,79
Q	15	F0,15	R ALT	E0,11	E0,F0,11	KP EN	E0,5A	E0,F0,5A
R	2D	F0,2D	APPS	E0,2F	E0,F0,2F	KP .	71	F0,71
S	1B	F0,1B	ENTER	5A	F0,5A	KP 0	70	F0,70
T	2C	F0,2C	ESC	76	F0,76	KP 1	69	F0,69
U	3C	F0,3C	F1	05	F0,05	KP 2	72	F0,72
V	2A	F0,2A	F2	06	F0,06	KP 3	7A	F0,7A
W	1D	F0,1D	F3	04	F0,04	KP 4	6B	F0,6B
X	22	F0,22	F4	0C	F0,0C	KP 5	73	F0,73
Y	35	F0,35	F5	03	F0,03	KP 6	74	F0,74
Z	1A	F0,1A	F6	0B	F0,0B	KP 7	6C	F0,6C
0	45	F0,45	F7	83	F0,83	KP 8	75	F0,75
1	16	F0,16	F8	0A	F0,0A	KP 9	7D	F0,7D
2	1E	F0,1E	F9	01	F0,01	]	5B	F0,5B
3	26	F0,26	F10	09	F0,09	;	4C	F0,4C
4	25	F0,25	F11	78	F0,78	'	52	F0,52
5	2E	F0,2E	F12	07	F0,07	,	41	F0,41
6	36	F0,36	PRNT SCRN	E0,12, E0,7C	E0,F0, 7C,E0, F0,12	.	49	F0,49
7	3D	F0,3D	SCROLL	7E	F0,7E	/	4A	F0,4A
8	3E	F0,3E	PAUSE	E1,14,77, E1,F0,14, F0,77	-NONE-			

ACPI 扫描码：

Key	Make Code	Break Code
Power	E0, 37	E0, F0, 37
Sleep	E0, 3F	E0, F0, 3F
Wake	E0, 5E	E0, F0, 5E

Windows 多媒体扫描码：

Key	Make Code	Break Code
Next Track	E0, 4D	E0, F0, 4D
Previous Track	E0, 15	E0, F0, 15
Stop	E0, 3B	E0, F0, 3B
Play/Pause	E0, 34	E0, F0, 34
Mute	E0, 23	E0, F0, 23
Volume Up	E0, 32	E0, F0, 32
Volume Down	E0, 21	E0, F0, 21
Media Select	E0, 50	E0, F0, 50
E-Mail	E0, 48	E0, F0, 48
Calculator	E0, 2B	E0, F0, 2B
My Computer	E0, 40	E0, F0, 40
WWW Search	E0, 10	E0, F0, 10
WWW Home	E0, 3A	E0, F0, 3A
WWW Back	E0, 38	E0, F0, 38
WWW Forward	E0, 30	E0, F0, 30
WWW Stop	E0, 28	E0, F0, 28
WWW Refresh	E0, 20	E0, F0, 20
WWW Favorites	E0, 18	E0, F0, 18

附录三 第三套键盘扫描码

KEY	MAKE	BREAK	KEY	MAKE	BREAK	KEY	MAKE	BREAK
A	1C	F0,1C	9	46	F0,46	[	54	F0,54
B	32	F0,32	`	0E	F0,0E	INSERT	67	F0,67
C	21	F0,21	-	4E	F0,4E	HOME	6E	F0,6E
D	23	F0,23	=	55	F0,55	PG UP	6F	F0,6F
E	24	F0,24	\	5C	F0,5C	DELETE	64	F0,64
F	2B	F0,2B	BKSP	66	F0,66	END	65	F0,65
G	34	F0,34	SPACE	29	F0,29	PG DN	6D	F0,6D
H	33	F0,33	TAB	0D	F0,0D	U ARROW	63	F0,63
I	43	F0,48	CAPS	14	F0,14	L ARROW	61	F0,61
J	3B	F0,3B	L SHFT	12	F0,12	D ARROW	60	F0,60
K	42	F0,42	L CTRL	11	F0,11	R ARROW	6A	F0,6A
L	4B	F0,4B	L WIN	8B	F0,8B	NUM	76	F0,76
M	3A	F0,3A	L ALT	19	F0,19	KP /	4A	F0,4A
N	31	F0,31	R SHFT	59	F0,59	KP *	7E	F0,7E
O	44	F0,44	R CTRL	58	F0,58	KP -	4E	F0,4E
P	4D	F0,4D	R WIN	8C	F0,8C	KP +	7C	F0,7C
Q	15	F0,15	R ALT	39	F0,39	KP EN	79	F0,79
R	2D	F0,2D	APPS	8D	F0,8D	KP .	71	F0,71
S	1B	F0,1B	ENTER	5A	F0,5A	KP 0	70	F0,70
T	2C	F0,2C	ESC	08	F0,08	KP 1	69	F0,69
U	3C	F0,3C	F1	07	F0,07	KP 2	72	F0,72
V	2A	F0,2A	F2	0F	F0,0F	KP 3	7A	F0,7A
W	1D	F0,1D	F3	17	F0,17	KP 4	6B	F0,6B
X	22	F0,22	F4	1F	F0,1F	KP 5	73	F0,73
Y	35	F0,35	F5	27	F0,27	KP 6	74	F0,74
Z	1A	F0,1A	F6	2F	F0,2F	KP 7	6C	F0,6C
0	45	F0,45	F7	37	F0,37	KP 8	75	F0,75
1	16	F0,16	F8	3F	F0,3F	KP 9	7D	F0,7D
2	1E	F0,1E	F9	47	F0,47	]	5B	F0,5B
3	26	F0,26	F10	4F	F0,4F	;	4C	F0,4C
4	25	F0,25	F11	56	F0,56	'	52	F0,52
5	2E	F0,2E	F12	5E	F0,5E	,	41	F0,41
6	36	F0,36	PRNT SCRN	57	F0,57	.	49	F0,49
7	3D	F0,3D	SCROLL	5F	F0,5F	/	4A	F0,4A
8	3E	F0,3E	PAUSE	62	F0,62			

附录四 PS/2 设备例程

These routines can be used to emulate a PS/2 mouse or keyboard. They were written for a PIC16F84 @ 4.61 MHz +/- 25% (perfect for a 5k/20pF RC oscillator). For more information about the PS/2 mouse, keyboard, and their protocol, check out one of the folowing links:
这个例程可用于仿真 PS/2 鼠标或键盘。适用于 PIC16F84，振荡频率为 4.61MHz +/-25%（最好用 5k/20pF RC 振荡器）。关于 PS/2 鼠标、键盘及它们的协议，检查下列的连接：

The AT Keyboard Interface
The PS/2 Mouse Interface
PS/2 Mouse/Keyboard Protocol
（译者注：中文译文的第二章、第三章和第一章。向上翻！）

Header:

```
;---------------------------------------------------------------------------------------------
; CLOCK/TIMING INFORMATION:
;---------------------------------------------------------------------------------------------
;
;     PS/2 bus clock low time =    40 us +/- 25% (30 us - 50 us)
;     PS/2 bus clock high time = 40 us +/- 25% (30 us - 50 us)
;     RC osc @ 20pF/5k = 4.61 MHz +/- 25% (3.50 MHz - 5.76 MHz)
;     1 instruction cycle @ 4.61 MHz (RC) = 0.87 us +/- 25% (0.65 us - 1.09 us)
;     Optimum PS/2 bus clock low time @4.61MHz = 45.97 instruction cycles
;     Actual PS/2 bus clock low time = 46 instruction cycles
;     Actual PS/2 bus clock low time @4.61MHz (RC) = 40.0us +/- 25% (30us-50us)
;     Actual PS/2 bus clock frequency @461MHz (RC) = 12.5 kHz +/- 25% (10.0kHz-16.7kHz)
;---------------------------------------------------------------------------------------------
;            HEADER:
;---------------------------------------------------------------------------------------------
   TITLE            "PS/2 Device Routines"
   SUBTITLE          "By Adam Chapweske"
   LIST             P=16F84
   INCLUDE           "p16f84.inc"
   RADIX             DEC
   ERRORLEVEL    -224, 1
   __CONFIG        _CP_OFF & _WDT_OFF & _RC_OSC



;---------------------------------------------------------------------------------------------
;          DEFINES:
;---------------------------------------------------------------------------------------------
#DEFINE DATA     PORTB, 7
#DEFINE CLOCK PORTB, 6

;---------------------------------------------------------------------------------------------
; RAM ALLOCATION:
```

```
;---------------------------------------------------------------------------------------------
   cblock
            TEMP0
            RECEIVE
            PARITY
            COUNTER
   endc
```

Required Routines & Macros:

```
;---------------------------------------------------------------------------------------------
;           MACROS:
;---------------------------------------------------------------------------------------------
Delay     macro     Time                    ;Delay "Cycles" instruction cycles
   if (Time==1)
            nop
            exitm
   endif
   if (Time==2)
            goto $ + 1
            exitm
   endif
   if (Time==3)
            nop
            goto $ + 1
            exitm
   endif
   if (Time==4)
            goto $ + 1
            goto $ + 1
            exitm
   endif
   if (Time==5)
            goto $ + 1
            goto $ + 1
            nop
            exitm
   endif
   if (Time==6)
            goto $ + 1
            goto $ + 1
            goto $ + 1
            exitm
   endif
   if (Time==7)
```

```
            goto $ + 1
            goto $ + 1
            goto $ + 1
            nop
            exitm
    endif
    if (Time%4==0)
            movlw (Time-4)/4
            call Delay_Routine
            exitm
    endif
    if (Time%4==1)
            movlw (Time-5)/4
            call Delay_Routine
            nop
            exitm
    endif
    if (Time%4==2)
            movlw (Time-6)/4
            call Delay_Routine
            goto $ + 1
            exitm
    endif
    if (Time%4==3)
            movlw (Time-7)/4
            call Delay_Routine
            goto $ + 1
            nop
            exitm
    endif
    endm

;---------------------------------------------------------------------------------------------
; DELAY:
;---------------------------------------------------------------------------------------------
;Delays 4w+4 cycles (including call,return, and movlw) (0=256)
Delay_Routine     addlw     -1                   ;Precise delays used in I/O
                   btfss     STATUS, Z
                   goto      Delay_Routine
                   return
```

ByteOut:

Sends a byte in w to the host. Returns 0xFE if inhibited during transmission. Returns 0xFF if host interrupts to send its own data. Returns 0x00 if byte sent successfully.

```
;-------------------------------------------------------------------------------------------
; OUTPUT ONE BYTE:    - TIMING IS CRITICAL!!!
;-------------------------------------------------------------------------------------------
ByteOut         movwf   TEMP0
InhibitLoop     btfss   CLOCK           ;Test for inhibit
                goto    InhibitLoop
                Delay   50
                btfss   CLOCK
                goto    InhibitLoop
                btfss   DATA            ;Check for request-to-send
                retlw   0xFF
                clrf    PARITY
                movlw   0x08
                movwf   COUNTER
                movlw   0x00
                call    BitOut          ;Start bit (0)
                btfss   CLOCK           ;Test for inhibit
                goto    ByteOutEnd
                Delay   4
ByteOutLoop     movf    TEMP0, w
                xorwf   PARITY, f
                call    BitOut          ;Data bits
                btfss   CLOCK           ;Test for inhibit
                goto    ByteOutEnd
                rrf     TEMP0, f
                decfsz  COUNTER, f
                goto    ByteOutLoop
                Delay   2
                comf    PARITY, w
                call    BitOut          ;Parity bit
                btfss   CLOCK           ;Test for inhibit
                goto    ByteOutEnd
                Delay   5
                movlw   0xFF
                call    BitOut          ;Stop bit (1)
                Delay   48
                retlw   0x00

ByteOutEnd      bsf     STATUS, RP0
                bsf     DATA
                bsf     CLOCK
                bcf     STATUS, RP0
                retlw   0xFE

BitOut          bsf     STATUS, RP0
                andlw   0x01
                btfss   STATUS, Z
                bsf     DATA
                btfsc   STATUS, Z
                bcf     DATA
                Delay   21
                bcf     CLOCK
                Delay   45
                bsf     CLOCK
                bcf     STATUS, RP0
                Delay   5
                return
```

ByteIn:

Reads a byte from the host.　Result in "RECEIVE" register.　Returns 0xFE in w if host aborts transmission. Returns 0xFF in w if framing/parity error detected.　Returns 0x00 in w if byte received successfully.

```
;-------------------------------------------------------------------------------------------
; READ ONE BYTE: - TIMING IS CRITICAL!!!
;-------------------------------------------------------------------------------------------
ByteIn          btfss   CLOCK           ;Wait for start bit
                goto    ByteIn
                btfsc   DATA
                goto    ByteIn
                movlw   0x08
                movwf   COUNTER
                clrf    PARITY
                Delay   28
ByteInLoop      call    BitIn           ;Data bits
                btfss   CLOCK           ;Test for inhibit
                retlw   0xFE
                bcf     STATUS, C
                rrf     RECEIVE, f
                iorwf   RECEIVE, f
                xorwf   PARITY,f
                decfsz  COUNTER, f
                goto    ByteInLoop
                Delay   1
                call    BitIn           ;Parity bit
                btfss   CLOCK           ;Test for inhibit
                retlw   0xFE
                xorwf   PARITY, f
                Delay   5
ByteInLoop1     Delay   1
                call    BitIn           ;Stop bit
                btfss   CLOCK           ;Test for inhibit
                retlw   0xFE
```

```
                  xorlw     0x00
                  btfsc     STATUS, Z
                  clrf      PARITY
                  btfsc     STATUS, Z              ;Stop bit = 1?
                  goto      ByteInLoop1            ;    No--keep clocking.

                  bsf       STATUS, RP0            ;Acknowledge
                  bcf       DATA
                  Delay     11
                  bcf       CLOCK
                  Delay     45
                  bsf       CLOCK
                  Delay     7
                  bsf       DATA
                  bcf       STATUS, RP0

                  btfss     PARITY, 7              ;Parity correct?
                  retlw     0xFF                   ;    No--return error

                  Delay     45
                  retlw     0x00

BitIn             Delay     8
                  bsf       STATUS, RP0
                  bcf       CLOCK
                  Delay     45
                  bsf       CLOCK
                  bcf       STATUS, RP0
                  Delay     21
                  btfsc     DATA
                  retlw     0x80
                  retlw     0x00
```

附录五 **PS/2** 主机例程

These routines can be used to interface a PS/2 mouse or keyboard.
这些例程用于和 PS/2 鼠标或键盘进行接口。

PS2get:
This routine reads a byte from the PS/2 device (keyboard or mouse).　Result in w.

```
PS2get       call     PS2getBit        ;Get/ignore the start bit
             movlw    0x08             ;Load Counter
             movwf    COUNTER
PS2getLoop   bcf      STATUS, C
             rrf      TEMP0, f
             call     PS2getBit        ;Read a data bit from the keyboard/mouse
             iorwf    TEMP0, f
             decfsz   COUNTER, f       ;Read 8 data bits yet?
             goto     PS2getLoop
             call     PS2getBit        ;Get/ignore parity bit.
             call     PS2getBit        ;Get/ignore stop bit
             movf     TEMP0, w         ;Result in w.
             return
PS2getBit    btfss    CLOCK            ;Make sure clock is high.
             goto     $ - 1
             btfsc    CLOCK
             goto     $ - 1
             goto     $ + 1
             btfss    DATA             ;Read data.
             retlw    0x00
             retlw    0x80
```

PS2cmd:
This routine sends a byte in w to a PS/2 mouse or keyboard.　TEMP0, TEMP1, and TEMP2 are general purpose registers.　CLOCK and DATA are assigned to port bits, and "Delay" is a self-explainatory macro. DATA and CLOCK are held high by setting their I/O pin to input and allowing an external pullup resistor to pull the line high.　The lines are brought low by setting the I/O pin to output and writing a "0" to the pin.

```
PS2cmd:
        movwf   TEMP0           ;Store to-be-sent byte
        movlw   0x08            ;Initialize a counter
        movwf   TEMP1
        clrf    TEMP2           ;Used for parity calc
```

```
            bsf       STATUS, RP0
            bcf       CLOCK
            bcf       STATUS, RP0
            bcf       CLOCK                ;Inhibit communication
            Delay     100                 ;for at least 100 microseconds
            bsf       STATUS, RP0
            bcf       DATA
            bcf       STATUS, RP0
            bcf       DATA                 ;Pull DATA low
            Delay     5
            bsf       STATUS, RP0
            bsf       CLOCK                ;Release CLOCK
            bcf       STATUS, RP0
PS2cmdLoop:
            movf      TEMP0, w
            xorwf     TEMP2, f            ;Parity calc
            call      PS2cmdBit          ;Output 8 data bits
            rrf       TEMP0, f
            decfsz    TEMP1, f
            goto      PS2cmdLoop
            comf      TEMP2, w
            call      PS2cmdBit          ;Output parity bit
            movlw     0x01
            call      PS2cmdBit          ;Output stop bit (1)
            btfsc     CLOCK                ;Wait for acknowledge
            goto      $ - 1
            btfss     CLOCK
            goto      $ - 1
            return

PS2cmdBit:
            btfsc     CLOCK                ;Wait for CLOCK=low
            goto      $ - 1
            bsf       STATUS, RP0
            andlw     0x01
            btfss     STATUS, Z          ;Set/Reset DATA line
            bsf       DATA
            btfsc     STATUS, Z
            bcf       DATA
            bcf       STATUS, RP0
            btfss     CLOCK                ;Wait for CLOCK=high
            goto      $ - 1
            return
```

附录六 PS/2 "Access" Mouse

This is a fully-functional PS/2 mouse written for the PIC16F84 microcontroller. It can be adapted to virtually any inputs, which gives the user a lot of flexability in how he/she controls the computer. It was developed to give computer access to people with physical disabilities, but I'm sure you can find many additional uses for this project.
这是一个使用 PIC16F84 微控制器的全功能的 PS/2 鼠标。它实际上可以适合任何输入，这就给了用户巨大的灵活性来控制他们的计算机。开发它出来让有残障的人们能访问计算机，但我确信你可以为这个工程找到更多额外的用途。

Feel free to use the code for non-commercial purposes only. You may distribute the code only if it is unmodified from its orginal form. I do not imply any warrenties or guarantees with this code. Use at your own risk. Enjoy!

Click on the following links to get the files:

- Access Mouse v1.50 - MPASM source code
- Access Mouse v1.50 - Schematic diagram (jpg)
- Access Mouse v1.51 - MPASM source code
- Access Mouse v1.51 - Schematic diagram (jpg)

VERSION 1.51:
- All inputs are active low.
- Speed is controlled in software by adjusting "PERIOD" and "DISTANCE" constants.
- All inputs, including "Clock" and "Data" may be assigned to any I/O pin.

VERSION 1.50:
- Internal PORTB pullups are enabled and all inputs are active low.
- All movement/button inputs may be assigned to any pin on PORTB.
- Speed is controlled by adjusting a variable resistor.
- An LED indicates the mouse's status.
- Potientiometer and LED may be assigned to any pin on PORTA.

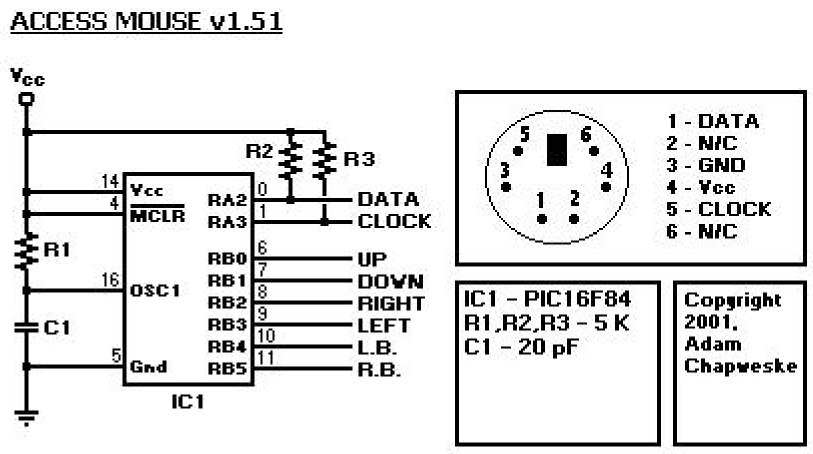
Just to give you an example of how these may be used, if you were to connect a condensor microphone element to a 339 Quad Comparator, then connect the output of the comparator to the left mouse button input pin on the PIC, you will be able to emulate a mouse click by blowing on the microphone.
If you find any bugs or have any comments, please send me an email. You may also email me your questions, but I won't have time to respond to most of them. This is a work under progress. Check back every few weeks for updated code and additions to this page.

For more information related to this project, try the following links:

- The PS/2 Mouse Interface
- PS/2 Mouse/Keyboard Protocol
- PIC Code/Projects
- Adam's Micro-Resources Home

- More Links

Access Mouse 1.51 版的电路图：



Access Mouse 1.51 版的软件：（比较长的样子，我没有核对，小心！）

```
;   PS/2 Mouse Emulator by Adam Chapweske (chap0179@tc.umn.edu)                    v1.51
;   http://panda.cs.ndsu.nodak.edu/~achapwes/
;   This was written for the PIC16F84 with an RC oscillator @ 20pF/5kohm
;   (will work with any oscillator between 3.50 MHz - 5.76 MHz)
;
;   FEEL FREE TO USE THIS CODE FOR NON-COMMERCIAL PURPOSES ONLY.   YOU MAY DISTRIBUTE
;   THIS CODE ONLY IF IT IS UNMODIFIED FROM ITS ORIGINAL FORM AND IT MUST CONTAIN THIS
;   HEADING.   I DO NOT IMPLY ANY WARANTEES OR GUARANTEES.   USE AT YOUR OWN RISK.   ENJOY!!!
;
;   Copyright 2001, Adam Chapweske

;---------------------------------------------------------------------------------------
; CLOCK/TIMING INFORMATION:
;---------------------------------------------------------------------------------------
;
;   PS/2 bus clock low time =    40 us +/- 25% (30 us - 50 us)
;   PS/2 bus clock high time = 40 us +/- 25% (30 us - 50 us)
;   RC osc @ 20pF/5k = 4.61 MHz +/- 25% (3.50 MHz - 5.76 MHz)
;   1 instruction cycle @ 4.61 MHz (RC) = 0.87 us +/- 25% (0.65 us - 1.09 us)
;   Optimum PS/2 bus clock low time @4.61MHz = 45.97 instruction cycles
;   Actual PS/2 bus clock low time = 46 instruction cycles
;   Actual PS/2 bus clock low time @4.61MHz (RC) = 40.0us +/- 25% (30us-50us)
;   Actual PS/2 bus clock frequency @461MHz (RC) = 12.5 kHz +/- 25% (10.0kHz-16.7kHz)

;---------------------------------------------------------------------------------------
;          HEADER:
;---------------------------------------------------------------------------------------
          TITLE          "PS/2 Mouse Emulator"
          SUBTITLE       "Copyright 2001, Adam Chapweske"
```

```
            LIST        P=16F84
            INCLUDE         "p16f84.inc"
                    RADIX             DEC
            ERRORLEVEL    -224, 1
                    __CONFIG        _CP_OFF & _WDT_OFF & _XT_OSC

;---------------------------------------------------------------------------------------------
;       DEFINES:
;---------------------------------------------------------------------------------------------
#DEFINE   DATA        PORTA, 2  ;May be assigned to any I/O pin
#DEFINE   CLOCK       PORTA, 3  ;May be assigned to any I/O pin
#DEFINE PS2_Yp        PORTB, 0  ;May be assigned to any I/O pin
#DEFINE   PS2_Yn      PORTB, 1  ;May be assigned to any I/O pin
#DEFINE   PS2_Xp      PORTB, 2  ;May be assigned to any I/O pin
#DEFINE   PS2_Xn      PORTB, 3  ;May be assigned to any I/O pin
#DEFINE   PS2_Bl      PORTB, 4  ;May be assigned to any I/O pin
#DEFINE   PS2_Br      PORTB, 5  ;May be assigned to any I/O pin



#DEFINE   PERIOD    20            ;Time between reading of inputs.    Min=(osc frequency)/204800
#DEFINE   DISTANCE 2              ;Amount by which X/Y counters are incremented/decremented

;---------------------------------------------------------------------------------------------
;       RAM ALLOCATION:
;---------------------------------------------------------------------------------------------

   cblock 0x0C

       TEMP0, TEMP1
       RECEIVE, PARITY, COUNTER   ;Used in I/O routines
       REPORT_RATE, RESOLUTION          ;Used for responses to status requests
       FLAGS, XY_FLAGS
       dBUTTONS                   ;"Delta Button States"
       X_COUNTER
       Y_COUNTER
   endc

;---------------------------------------------------------------------------------------------
;FLAGS:
; bit 7 -- Always 0
; bit 6 -- Stream(0)/Remote(1) mode
; bit 5 -- Disable(0)/Enable(1) reporting
; bit 4 -- 1:1(0)/2:1(1) Scaling
; bit 3 -- Always 0
; bit 2 -- Always 0
; bit 1 -- Always 0
; bit 0 -- Always 0

MODE      equ   6
ENABLE    equ   5
SCALE     equ   4

;-------------------------------------------------------------------------------------------
;XY_FLAGS:
; bit 7 -- Y Counter overflow
; bit 6 -- X Counter overflow
; bit 5 -- Y counter sign bit
; bit 4 -- X counter sign bit
; bit 3 -- Always 1
; bit 2 -- Always 0 (middle button)
; bit 1 -- Previous right button state
; bit 0 -- Previous left button state

yOVF      equ   7
xOVF      equ   6
ySIGN     equ   5
xSIGN     equ   4

;dBUTTONS
; bit 7 -- Always 0
; bit 6 -- Always 0
; bit 5 -- Always 0
; bit 4 -- Always 0
; bit 3 -- Always 0
; bit 2 -- Always 0
; bit 1 -- Change in right buton state
; bit 0 -- Change in left button state

;-------------------------------------------------------------------------------------------

  cblock          ;Contains to-be-sent packet and last packet sent
      LENGTH
      SEND1
      SEND2
      SEND3
  endc

;-------------------------------------------------------------------------------------------
;           MACROS:
;-------------------------------------------------------------------------------------------

;Delay "Cycles" instruction cycles
```

```
Delay         macro     Time
   if (Time==1)
             nop
             exitm
   endif
   if (Time==2)
             goto  $ + 1
             exitm
   endif
   if (Time==3)
             nop
             goto  $ + 1
             exitm
   endif
   if (Time==4)
             goto  $ + 1
             goto  $ + 1
             exitm
   endif
   if (Time==5)
             goto  $ + 1
             goto  $ + 1
             nop
             exitm
   endif
   if (Time==6)
             goto  $ + 1
             goto  $ + 1
             goto  $ + 1
             exitm
   endif
   if (Time==7)
             goto  $ + 1
             goto  $ + 1
             goto  $ + 1
             nop
             exitm
   endif
   if (Time%4==0)
             movlw      (Time-4)/4
             call   Delay_us
             exitm
   endif
   if (Time%4==1)
             movlw      (Time-5)/4
             call   Delay_us
```

```
            nop
            exitm
    endif
    if (Time%4==2)
            movlw       (Time-6)/4
            call    Delay_us
            goto    $ + 1
            exitm
    endif
    if (Time%4==3)
            movlw       (Time-7)/4
            call    Delay_us
            goto    $ + 1
            nop
            exitm
    endif
            endm
;---------------------------------------------------------------------------------------------
;           ORG 0x000:
;---------------------------------------------------------------------------------------------

            org     0x000
            goto    Start

;---------------------------------------------------------------------------------------------
;       HANDLE COMMAND:
;---------------------------------------------------------------------------------------------

    if (high Table1End != 0)
            ERROR       "Command handler table must be in low memory page"
    endif

Command         movlw       0x04        ;Test for a resolution value
            subwf       RECEIVE, w
            bnc     SetResolution
            movlw       0xC8        ;Test for report rate value
            subwf       RECEIVE, w
            bnc     SetReportRate
            movlw       0xE6        ;0xE6 is lowest code
            subwf       RECEIVE, w
            bnc     MainLoop
HandlerTable    addwf       PCL, f              ;Add offset
            goto    Mouse_E6 ;0xE6 - Set Scaling 1:1
            goto    Mouse_E7 ;0xE7 - Set Scaling 2:1
            goto    MainLoop ;0xE8 - Set Resolution
            goto    Mouse_E9 ;0xE9 - Status Request
            goto  Mouse_EA ;0xEA - Set Stream Mode
            goto  Report         ;0xEB - Read Data
            goto  MainLoop  ;0xEC - Reset Wrap Mode
            goto  MainLoop  ;0xED -
            goto  WrapMode ;0xEE - Set Wrap Mode
            goto  MainLoop  ;0xEF
            goto  Mouse_F0  ;0xF0 - Set Remote Mode
            goto  MainLoop  ;0xF1
            goto  Mouse_F2  ;0xF2 - Read Device Type
            goto  MainLoop  ;0xF3 - Set Report Rate
            goto  Mouse_F4  ;0xF4 - Enable
            goto  Mouse_F5  ;0xF5 - Disable
            goto  Mouse_F6  ;0xF6 - Set Default
            goto  MainLoop  ;0xF7
            goto  MainLoop  ;0xF8
            goto  MainLoop  ;0xF9
            goto  MainLoop  ;0xFA
            goto  MainLoop  ;0xFB
            goto  MainLoop  ;0xFC
            goto  MainLoop  ;0xFD
            goto  PacketOut  ;0xFE - Resend
Table1End goto  Reset       ;0xFF - Reset


;---------------------------------------------------------------------------------------------
;      START:
;---------------------------------------------------------------------------------------------

Start       clrf   PORTA
            clrf   PORTB
            bsf    STATUS, RP0    ;(TRISA=TRISB=0xFF by default)
            movlw       0x57         ;Timer mode, assign max. prescaler, enable pullups
            movwf       OPTION_REG
            bcf    STATUS, RP0
            movlw       0x08         ;Bit 3 always = 1, clear previous button states
            movwf       XY_FLAGS
;           goto  Reset

;---------------------------------------------------------------------------------------------
;           Reset Mode:
;---------------------------------------------------------------------------------------------
Reset       movlw       0xAA
            movwf       SEND1              ;Load BAT completion code
            call   LoadDefaults
            clrf   SEND2             ;Load Device ID (0x00)
            movlw       0x02
            movwf     LENGTH
            call   BATdelay
            goto  PacketOut  ;Output 2-byte "completion-code, device ID" packet

;---------------------------------------------------------------------------------------------
;           Stream/Remote Mode:
;---------------------------------------------------------------------------------------------
MainLoop  clrf   X_COUNTER    ;Clear movement counters
            clrf   Y_COUNTER

MainLoop1         btfss  DATA              ;Check for host request-to-send
            goto  PacketIn
            movlw      PERIOD            ;Report period
            subwf      TMR0, w
            btfss  STATUS, C      ;TMR0=report period?
            goto  MainLoop1       ;   No--loop
            clrf   TMR0           ;   Yes--reset TMR0, then read inputs...
            call   ReadInputs
            btfsc  FLAGS, MODE  ;Stream(0)/Remote(1) mode
            goto  MainLoop1
            btfss  FLAGS, ENABLE      ;Disable(0)/Enable(1) reporting
            goto  MainLoop1
            movf X_COUNTER, w       ;Test for X-movement
            iorwf Y_COUNTER, w      ;Test for Y-movement
            iorwf dBUTTONS, w   ;Test for change in button states
            bz     MainLoop1
;           goto  Report

;---------------------------------------------------------------------------------------------
;      REPORT:
;---------------------------------------------------------------------------------------------

Report            movf dBUTTONS, w
            xorwf      XY_FLAGS, f   ;Find current button state
            movf XY_FLAGS, w
            movwf      SEND1
            movf X_COUNTER, w
            movwf      SEND2
            movf Y_COUNTER, w
            movwf      SEND3
            movlw      0x03        ;Movement data report length
            movwf      LENGTH
;           goto  PacketOut

;---------------------------------------------------------------------------------------------
;      OUTPUT PACKET
```

```
;---------------------------------------------------------------------------------------------

PacketOut  movlw      SEND1            ;First byte of packet
           movwf      FSR
           movf LENGTH, w        ;Length of packet
           movwf      TEMP1
PacketOutLoop   movf INDF, w            ;Get data byte
           call   ByteOut            ;   Output that byte
           xorlw0xFF          ;Test for RTS error
           bz     PacketIn
           xorlw0xFE ^ 0xFF       ;Test for inhibit error
           bz     PacketOut
           incf   FSR, f             ;Point to next byte
           decfsz     TEMP1, f
           goto  PacketOutLoop
           goto  MainLoop

;---------------------------------------------------------------------------------------------
;      READ PACKET
;---------------------------------------------------------------------------------------------

PacketIn   call   ByteIn
           xorlw0xFF          ;Test for parity/framing error
           bz     Mouse_ERR
           xorlw0xFE ^ 0xFF       ;Test for inhibit error
           bz     MainLoop1
           movlw      0xFE       ;Test for "Resend" command
           xorwf      RECEIVE, w
           bz     PacketOut
Acknowledge     movlw      0xFA        ;Acknowledge
           call   ByteOut
           goto  Command

;---------------------------------------------------------------------------------------------
;      READ INPUTS:
;---------------------------------------------------------------------------------------------

ReadInputs movlw      DISTANCE

           btfss  PS2_Xp             ;Read inputs
           addwf      X_COUNTER, f
           btfss  PS2_Yp
           addwf      Y_COUNTER, f
           btfss  PS2_Xn
           subwf      X_COUNTER, f
           btfss  PS2_Yn
```

```
		subwf	Y_COUNTER, f

		bcf	XY_FLAGS, xSIGN
		btfsc	X_COUNTER, 7
		bsf	XY_FLAGS, xSIGN
		bcf	XY_FLAGS, ySIGN
		btfsc	Y_COUNTER, 7
		bsf	XY_FLAGS, ySIGN

		movf	XY_FLAGS, w	;Get previous button states
		andlw	b'00000111'
		btfss	PS2_Bl		;Find changes in button states
		xorlw	b'00000001'
		btfss	PS2_Br
		xorlw	b'00000010'
		movwf	dBUTTONS	;Save *change* in button state
		retlw	0x00

;---------------------------------------------------------------------------------------------
;	WRAP MODE:
;---------------------------------------------------------------------------------------------

WrapMode	btfsc	DATA		;Wait for RTS
		goto	WrapMode
		call	ByteIn		;Read one byte from host
		xorlw	0xFE	;Test for aborted transmission
		bz	WrapMode
		movf	RECEIVE, w
		xorlw	0xFF	;Test for "Reset" command
		bz	Acknowledge
		xorlw	0xFF^0xEC	;Test for "Reset Wrap Mode" command
		bz	Acknowledge
		xorlw	0xEC
		call	ByteOut		;Else, echo
		goto	WrapMode

;---------------------------------------------------------------------------------------------
;	LOAD DEFAULT VALUES:
;---------------------------------------------------------------------------------------------

LoadDefaults	movlw	100		;Default report rate
		movwf	REPORT_RATE
		movlw	0x02		;Default resolution
		movwf	RESOLUTION
		clrf	FLAGS		;Stream mode, 1:1 scaling, disabled
		retlw	0x00
```

```
;------------------------------------------------------------------------------------------
;       EMULATE BAT:
;------------------------------------------------------------------------------------------
BATdelay  clrf   TEMP0           ;Used for a 400 ms delay at power-on
          clrf   TEMP1
DelayLoop Delay         6
          decfsz        TEMP0, f
          goto   DelayLoop
          decfsz        TEMP1, f
          goto   DelayLoop
          retlw 0x00

;------------------------------------------------------------------------------------------
;       HANDLE COMMANDS:
;------------------------------------------------------------------------------------------

SetResolution     movf RECEIVE, w
          movwf         RESOLUTION
          goto   MainLoop

SetReportRate     movf RECEIVE, w
          movwf         REPORT_RATE
          goto   MainLoop

;0xE6 - Set Scaling 1:1
Mouse_E6 bcf     FLAGS, SCALE
          goto   MainLoop

;0xE7 - Set Scaling 2:1
Mouse_E7 bsf     FLAGS, SCALE
          goto   MainLoop

;0xE9 - Status Request
Mouse_E9 movf FLAGS, w
          btfss  PS2_Bl
          iorlw 0x04
          btfss  PS2_Br
          iorlw 0x01
          movwf         SEND1
          movf RESOLUTION, w
          movwf         SEND2
          movf REPORT_RATE, w
          movwf         SEND3
          movlw         0x03
          movwf         LENGTH
```

```
                goto  PacketOut

;0xEA - Set Stream Mode
Mouse_EA bcf    FLAGS, MODE
                goto  MainLoop

;0xF0 - Set Remote Mode
Mouse_F0 bsf    FLAGS, MODE
                goto  MainLoop

;0xF2 - Get Device ID
Mouse_F2 clrf   SEND1
                movlw       0x01
                movwf       LENGTH
                goto  PacketOut

;0xF4 - Enable Reporting
Mouse_F4 bsf    FLAGS, ENABLE
                goto  MainLoop

;0xF5 - Disable Reporting
Mouse_F5 bcf    FLAGS, ENABLE
                goto  MainLoop

;0xF6 - Set Default
Mouse_F6 call   LoadDefaults
                goto  MainLoop

;Invalid command
Mouse_ERR       movlw       0xFE
                call  ByteOut
                goto  MainLoop

;---------------------------------------------------------------------------------------------
;       OUTPUT ONE BYTE:    - TIMING IS CRITICAL!!!
;---------------------------------------------------------------------------------------------

ByteOut         movwf       TEMP0
InhibitLoop     btfss  CLOCK            ;Test for inhibit
                goto  InhibitLoop
                Delay       100         ;(50 microsec = 58 clock cycles, min)
                btfss  CLOCK
                goto  InhibitLoop
                btfss  DATA             ;Check for request-to-send
                retlw 0xFF
                clrf   PARITY
```

```
          movlw     0x08
          movwf     COUNTER
          movlw     0x00
          call  BitOut         ;Start bit (0)
          btfss CLOCK          ;Test for inhibit
          goto  ByteOutEnd
          Delay     4
ByteOutLoop    movf TEMP0, w
          xorwf     PARITY, f
          call  BitOut         ;Data bits
          btfss CLOCK          ;Test for inhibit
          goto  ByteOutEnd
          rrf   TEMP0, f
          decfsz    COUNTER, f
          goto  ByteOutLoop
          Delay     2
          comf PARITY, w
          call  BitOut         ;Parity bit
          btfss CLOCK          ;Test for inhibit
          goto  ByteOutEnd
          Delay     5
          movlw     0xFF
          call  BitOut         ;Stop bit (1)
          Delay     48
          retlw 0x00
ByteOutEnd     bsf  STATUS, RP0
          bsf   DATA
          bsf   CLOCK
          bcf   STATUS, RP0
          retlw 0xFE

BitOut         bsf  STATUS, RP0
          andlw     0x01
          btfss STATUS, Z
          bsf   DATA
          btfsc STATUS, Z
          bcf   DATA
          Delay     21
          bcf   CLOCK
          Delay     45
          bsf   CLOCK
          bcf   STATUS, RP0
          Delay     5
          return

;-------------------------------------------------------------------------------------------
```

```
;       READ ONE BYTE: (Takes about 1ms) - TIMING IS CRITICAL!!!
;-------------------------------------------------------------------------------------------

ByteIn          btfss  CLOCK            ;Test for Request-to-send
        retlw 0xFE
        btfsc  DATA
        retlw 0xFE
        movlw       0x08
        movwf       COUNTER
        clrf   PARITY
        Delay       28
ByteInLoop      call   BitIn        ;Data bits
        btfss  CLOCK            ;Test for inhibit
        retlw 0xFE
        bcf    STATUS, C
        rrf    RECEIVE, f
        iorwf RECEIVE, f
        xorwf       PARITY,f
        decfsz      COUNTER, f
        goto  ByteInLoop
        Delay       1
        call   BitIn        ;Parity bit
        btfss  CLOCK            ;Test for inhibit
        retlw 0xFE
        xorwf       PARITY, f
        Delay       5
ByteInLoop1     Delay       1
        call   BitIn        ;Stop bit
        btfss  CLOCK            ;Test for inhibit
        retlw 0xFE
        xorlw0x00
        btfsc  STATUS, Z
        clrf   PARITY
        btfsc  STATUS, Z        ;Stop bit = 1?
        goto  ByteInLoop1      ;   No--keep clocking.

        bsf    STATUS, RP0   ;Acknowledge
        bcf    DATA
        Delay       11
        bcf    CLOCK
        Delay       45
        bsf    CLOCK
        Delay       7
        bsf    DATA
        bcf    STATUS, RP0
```

```
          btfss  PARITY, 7 ;Parity correct?
          retlw 0xFF        ;    No--return error

          Delay      45
          retlw 0x00

BitIn     Delay      8
          bsf    STATUS, RP0
          bcf    CLOCK
          Delay      45
          bsf    CLOCK
          bcf    STATUS, RP0
          Delay      21
          btfsc  DATA
          retlw 0x80
          retlw 0x00

;---------------------------------------------------------------------------------------------
;       DELAY:
;---------------------------------------------------------------------------------------------
;Delays 4w+4 cycles (including call,return, and movlw) (0=256)
Delay_us  addlw      -1            ;Precise delays used in I/O
          btfss  STATUS, Z
          goto   Delay_us
          return

          end
```