The AT-PS/2 Keyboard Interface

Introduction:

This article tries to cover every aspect of the 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 interfacing your PC's keyboard controller. As of right now, all code samples involving the keyboard interface are written in assembly for Microchip's PIC microcontrollers. All code samples for interfacing the PC's keyboard controller are written in x86 assembly.

I should mention that all of the information in this article comes from my own experiences and from other sources that may or may not be accurate. I did not consult any official documentation of "AT" or "PS/2" keyboard standards since none has been available to me. Therefore, I provide the following disclaimer:

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.

You may click here to goto my main page. There, you will find other articles, code, projects, and links related to the computer keyboard. I am also available for private contracting--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.

A History Lesson:

The most popular keyboards in use today include:

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

IBM introduced a new keyboard with each of its major desktop computer models. The original IBM PC, and later the IBM XT, used what we call the "XT keyboard." These are obsolete and differ significantly from modern keyboards; the XT keyboard is not covered in this article. Next came the IBM AT system and later the IBM PS/2. They introduced the keyboards we use today, and are the topic of this article. AT keyboards and PS/2 keyboards were very similar devices, but the PS/2 device used a smaller connector and supported a few additional features. Nonetheless, it remained backward

compatible with AT systems and few of the additional features ever caught on (since software also wanted to remain backward compatible.) Below is a summary of IBM's three major keyboards.

IBM PC/XT Keyboard (1981):

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

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

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

IBM PS/2 Keyboard (1987) - Backward compatible with AT systems; NOT compatible with XT systems.

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

The PS/2 keyboard was originally an extension of the AT device. It supported a few additional host-to-keyboard commands and featured a smaller connector. These were the only differences between the two devices. However, computer hardware has never been about standards so much as compatibility. For this reason, any keyboard you buy today will be *compatible* with PS/2 and AT systems, but it may not fully support all the features of the original devices.

Today, "AT keyboard" and "PS/2 keyboard" refers only to their connector size. Which settings/commands any given keyboard does or does not support is anyone's guess. For example, the keyboard I'm using right now has a PS/2-style connector but only fully supports seven commands, partially supports two, and merely "acknowledges" the rest. In contrast, my "Test" keyboard has an AT-style connector but supports every feature/command of the original PS/2 device (plus a few extra.) It's important you treat modern keyboards as compatible, not standard. If your project relies on non-general features, it may work on some systems, but not on others...

Modern AT-PS/2 compatible keyboards

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

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

General Description:

Keyboards consist of a large matrix of keys, all of which are monitored by an on-board processor (called the "keyboard encoder".) The specific processor 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.

Footnote 1) Originally, IBM used the Intel 8048 microcontroller as its keyboard encoder. The following is a short list of modern keyboard encoders:

- *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.

Electrical Interface / Protocol:

The AT and PS/2 keyboards use the same protocol as the PS/2 mouse.

Scan Codes:

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

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

three scan code sets:

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

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

Make Codes, Break Codes, and Typematic Repeat:

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

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

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

Key	(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.

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

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

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

Reset:

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

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

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

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.

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.

Command Set:

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.

- 0xFF (Reset) - Causes keyboard to enter "Reset" mode. (See "Reset" section.)
- 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.
- *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.
- *0xFC (Set Key Type Make/Break) - Similar to "Set Key Type Make", but both make codes and break codes are enabled (typematic is disabled.)
- *0xFB (Set Key Type Typematic) - Similar to previous two commands, except make and typematic is enabled; break codes are disabled.
- *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.)
- *0xF9 (Set All Keys Make) - Causes only make codes to be sent; break codes and typematic repeat are disabled for all keys.
- *0xF8 (Set All Keys Make/Break) - Similar to previous two commands, except only typematic repeat is disabled.
- *0xF7 (Set All Keys Typematic) - Similar to previous three commands, except only break codes are disabled. Make codes and typematic repeat are enabled.
- 0xF6 (Set Default) - Load default typematic rate/delay (10.9cps / 500ms), key types (all keys typematic/make/break), and scan code set (2).
- 0xF5 (Disable) - Keyboard stops scanning, loads default values (see "Set Default" command), and waits further instructions.
- 0xF4 (Enable) - Re-enables keyboard after disabled using previous command.
- 0xF3 (Set Typematic Rate/Delay) - Host follows this command with one argument byte that defines the typematic rate and delay as follows:

  .

Repeat Rate

Bits 0-4	Rate(cps)	Bits 0-4	Rate(cps)	Bits 0-4	Rate(cps)	Bits 0-4	Rate(cps)
00h	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.

- *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.
- 0xEE (Echo) - The keyboard responds with "Echo" (0xEE).
- 0xED (Set/Reset LEDs) - The host follows this command with one argument byte, that specifies the state of the keyboard's Num Lock, Caps Lock, and Scroll Lock LEDs. This argument byte is defined as follows:

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.

Emulation:

Click here for keyboard/mouse routines. Source in MPASM for PIC microcontrollers.

The i8042 Keyboard Controller:

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.

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".

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.

The 8042 contains the following registers:

- A one-byte input buffer - contains byte read from keyboard; read-only
- A one-byte output buffer - contains byte to-be-written to keyboard; write-only
- A one-byte status register - 8 status flags; read-only
- A one-byte control register - 7 control flags; read/write

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:

Port	Read / Write	Function
0x60	Read	Read Input Buffer
0x60	Write	Write Output Buffer
0x64	Read	Read Status Register
0x64	Write	Send Command

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.

When describing the 8042, I may occasionally refer to its physical I/O pins. These pins are defined below:

AT-compatible mode

Port 1 (Input Port):

Pin	Name	Function
0	P10	Undefined .
1	P11	Undefined .
2	P12	Undefined .
3	P13	Undefined .
4	P14	External RAM 1: Enable external RAM 0: Disable external RAM
5	P15	Manufacturing Setting 1: Setting enabled 0: Setting disabled
6	P16	Display Type Switch 1: Color display 0: Monochrome

Port 2 (Output Port):

Pin	Name	Function
0	P20	System Reset 1: Normal 0: Reset computer
1	P21	Gate A20 .
2	P22	Undefined .
3	P23	Undefined .
4	P24	Input Buffer Full .
5	P25	Output Buffer Empty .
6	P26	Keyboard Clock 1: Pull Clock low 0: High-Z

Port 3 (Test Port):

Pin	Name	Function
0	T0	Keyboard Clock (Input) .
1	T1	Keyboard Data (Input) .
2	--	Undefined .
3	--	Undefined .
4	--	Undefined .
5	--	Undefined .
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):

Pin	Name	Function
0	P10	Keyboard Data (Input) .
1	P11	Mouse Data (Input) .
2	P12	Undefined .
3	P13	Undefined .
4	P14	External RAM 1: Enable external RAM 0: Disable external RAM
5	P15	Manufacturing Setting 1: Setting enabled 0: Setting disabled
6	P16	Display Type Switch 1: Color display 0: Monochrome
7	P17	Keyboard Inhibit Switch 1: Keyboard enabled 0: Keyboard disabled

Port 2 (Output Port):

Pin	Name	Function
0	P20	System Reset 1: Normal 0: Reset computer
1	P21	Gate A20 .
2	P22	Mouse Data: 1: Pull Data low 0: High-Z
3	P23	Mouse Clock: 1: Pull Clock low 0: High-Z
4	P24	Keyboard IBF interrupt: 1: Assert IRQ 1 0: De-assert IRQ 1
5	P25	Mouse IBF interrupt: 1: Assert IRQ 12 0: De-assert IRQ 12
6	P26	Keyboard Clock: 1: Pull Clock low 0: High-Z
7	P27	Keyboard Data: 1: Pull Data low 0: High-Z

Port 3 (Test Port):

Pin	Name	Function
0	T0	Keyboard Clock (Input) .
1	T1	Mouse Clock (Input) .
2	--	Undefined .
3	--	Undefined .
4	--	Undefined .
5	--	Undefined .
6	--	Undefined .
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.)

Status Register:

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:

	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.

[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")]

Reading keyboard input:

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.

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.

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:

```
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:

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.

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.)

```
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:

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.

- 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:

	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

- ?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.)

Modern Keyboard Controllers:

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.)

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.)

Once again, keep in mind chipset designers are more interested in compatibility than standardization.

Initialization:

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

```
Keyboard: AA  Self-test passed                ;Keyboard controller init
Host:     ED  Set/Reset Status Indicators
Keyboard: FA  Acknowledge
Host:     00  Turn off all LEDs
Keyboard: FA  Acknowledge
Host:     F2  Read ID
```

```
Keyboard: FA  Acknowledge
Keyboard: AB  First byte of ID
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
```

Keyboard Scan Codes: Set 1

*All values are in hexadecimal

101-, 102-, and 104-key keyboards:

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 Scan Codes:

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

Windows Multimedia Scan Codes:

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

Keyboard Scan Codes: Set 2

*All values are in hexadecimal

101-, 102-, and 104-key keyboards:

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 Scan Codes:

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

Windows Multimedia Scan Codes:

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

AT Keyboard Scan Codes (Set 3)

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
	3D	F0,3D		SCROLL	5F	F0,5F		/	4A	F0,4A
8	3E	F0,3E		PAUSE	62	F0,62				

PS/2 Device Routines:

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:

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-16.7kH

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

;       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
```
