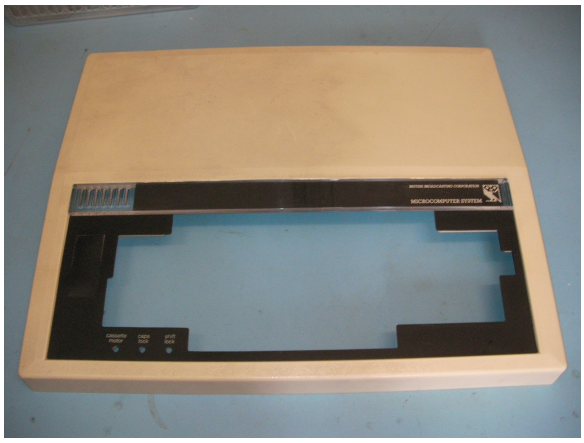
BBC Micro to Windows Computer

Introduction

I have an old BBC micro computer stored in my Attic. I get it out on the odd occasion and power it up. I have fond memories of the BBC micro and thought that it would be a great shame if it was left in my attic gathering dust. I am going to removed the circuit boards from the case, leaving just the case and the keyboard. I will then install a micro ITX motherboard in the case. I need to convert the BBC keyboard to enable it to connect to the ITX motherboard as well. I will get a motherboard with USB, PS2, parallel and serial interfaces, which will enable me to use the computer for interfacing with future and past projects.

Case removal & cleaning



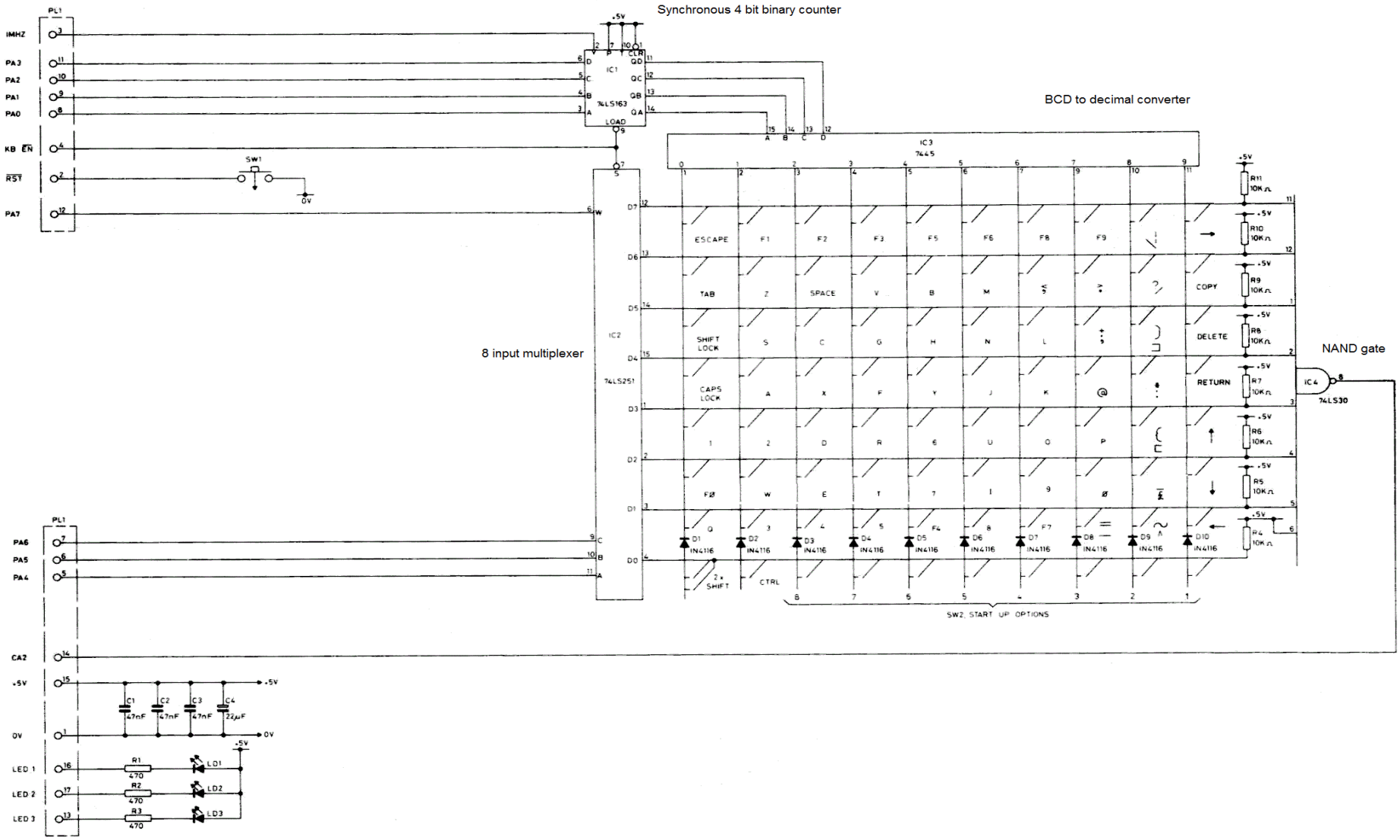
The case had yellowed slightly and there were a few scratches. I started off by removing the case and putting the base to one side.

I cleaned the case using some car polish, which was very effective at removing the top layer of dirt from the plastic. It also lightened the case slightly, probably nearer to it's original colour. The polish is slightly abrasive, but didn't have any effect on the original texture of the plastic.

Cleaning in progress. This was a rather lengthy process, but eventually I was pleased with the result. Here is a picture of the case after cleaning.

The fully cleaned case.

BBC Micro Keyboard Circuit Diagram



Circuit Description

The keyboard matrix consists of ten columns by eight rows of normally open switches mounted on a metal plate. Connections to the gold contacts of these switches are made on a printed circuit board.

Initially the four-bit synchronous binary counter (IC1) increments at a rate of 1MHz as set by the system clock, in what's know as free-running mode. The BBC micro has no control over the keyboard at this point.

In free running mode the counter is clocked at 1MHz and it's output is applied to a BCD to decimal converter (IC3). In this way each one of the 10 columns of the keyboard is pulse low then high in turn, producing a “walking zeros” pattern. Once all the columns have been scanned, the counter will return to zero and then continue to increment, scanning the columns again.

Any key presses short a column line to a row line and this event is detected by IC4 an 8 input NAND gate, pulsing high as the walking zero passes the column to which that key is connected. This causes an interrupt to be sent to the computer.

The BBC micro recognises this interrupt and the computer executes the keyboard reading routine to discover which key was depressed.
First KB $\overline{\text{EN}}$ is set low. This will set a low level at the 'load' input to the binary counter, disabling the counter and causing it's outputs to agree with the inputs after the next clock pulse. This stops the keyboard operating in 'free running' mode. It also enables the multiplexer by setting it's 'strobe' input low.

The computer can now scan each column in turn to determine the column in which the key was pressed.

This is achieved by setting the inputs to the counter to zero. At the next clock pulse the inputs will appear on the outputs. The four outputs of the counter will now be latched onto the four inputs of the BCD to Decimal converter. By setting the inputs to the counter and incrementing them, each column can be interrogated in turn.

At the same time the computer loads a 3 bit code into the inputs of the multiplexer (IC2) . This code will increment until all 8 rows have been interrogated. The logic level on a particular row appears at the output of the multiplexer when selected.

In this way, the keyboard matrix is scanned for the intersection between row and column when a key has been depressed.

This scanning method enables any number of key presses to be detected simultaneously. Also the computer doesn't have to monitor the keyboard for key presses as the keyboard reading routine will only be active when a key press is detected and an interrupt is sent.

Column Scanning

74LS163 binary counter

The binary counter in conjunction with the 7445 BCD to decimal converter enables each column of the matrix to be scanned in turn.

The counter simply takes 4 inputs (A,B,C,D) and will set the corresponding output (QA,QB,QC,QD) after the next clock pulse. The clock rate received from the BBC micro is 1MHz, however I can run this at a slower rate if required.

7445 BCD to Decimal Converter

The four outputs from the binary counter are applied to the inputs of this device. By incrementing the BCD inputs to the 7445 from 0000 to 1001, each column in turn will be pulsed from a high state to a low state, back to a high state, producing the 'walking zeros pattern'.

BCD Inputs				**Decimal Outputs**										
A	**B**	**C**	**D**	**0**	**1**	**2**	**3**	**4**	**5**	**6**	**7**	**8**	**9**	**Matrix Column Selected**
0	0	0	0	**0**	1	1	1	1	1	1	1	1	1	0
0	0	0	1	1	**0**	1	1	1	1	1	1	1	1	1
0	0	1	0	1	1	**0**	1	1	1	1	1	1	1	2
0	0	1	1	1	1	1	**0**	1	1	1	1	1	1	3
0	1	0	0	1	1	1	1	**0**	1	1	1	1	1	4
0	1	0	1	1	1	1	1	1	**0**	1	1	1	1	5
0	1	1	0	1	1	1	1	1	1	**0**	1	1	1	6
0	1	1	1	1	1	1	1	1	1	1	**0**	1	1	7
1	0	0	0	1	1	1	1	1	1	1	1	**0**	1	8
1	0	0	1	1	1	1	1	1	1	1	1	1	**0**	9

The counter needs to be clocked for each time the input changes. At this point I am not actually testing for any keys pressed. I do however know by the Inputs A,B,C,D what column is currently being tested for a key press.

Row Scanning

74LS251 8 Input multiplexer

The multiplexer takes 3 inputs A,B,C.

Input			**Output (Matrix Row Selected)**
C	**B**	**A**	**W**
0	0	0	$\overline{D0}$
0	0	1	$\overline{D1}$
0	1	0	$\overline{D2}$
0	1	1	$\overline{D3}$
1	0	0	$\overline{D4}$
1	0	1	$\overline{D5}$
1	1	0	$\overline{D6}$
1	1	1	$\overline{D7}$

Each row is selected in turn by the three bit code present on the input to the multiplexer. The state of the selected row (low or high) is transferred to the output of the multiplexer W.

In summary, if we know which column has been selected from the input to the BCD to Decimal Converter and we know which row is currently being scanned by setting the input to the multiplexer, it will be possible to detect a key press at that position. By scanning each column and row it will be possible to detect multiple key presses.

BBC Micro Keyboard Connector.

Pin No	Name	Description
1	0V	0V DC Supply line
2	$\overline{\text{RST}}$	Keyboard reset (break key)
3	1MHz	System clock. Goes to clock input of binary counter
4	KB $\overline{\text{EN}}$	Keyboard enable. High = binary counter free running, Low = binary counter follows inpu
5	PA4	A
6	PA5	B Multiplexer Data inputs (Keyboard matrix row Select)
7	PA6	C
8	PA0	A
9	PA1	B Binary counter data inputs (Keyboard matrix Column select)
10	PA2	C
11	PA3	D
12	PA7	W Output of Multiplexer – Detects key pressed at intersection of current row/column
13	LED3	CASSETTE MOTOR
14	CA2	Pulses high when a key is pressed.
15	+5V	+5V DC supply
16	LED1	SHIFT LOCK
17	LED2	CAPS LOCK

Testing the Keyboard

First I will apply 5V to the keyboard and make sure nothing goes bang. The connector atached to the end of the ribbon cable is a 0.1” pitch female connector. I mounted this connector onto my breadboard and applied 5v across pins 15 and 1 as per the circuit diagram. I monitored the current by using a power supply with a voltage and current display. The measured current was 63mA.

If either pins 13,16 or 17 are connected to 0v, it can be seen from the circuit diagram that the corresponding LED will light up. This was verified, see picture. The current increased to 70mA.

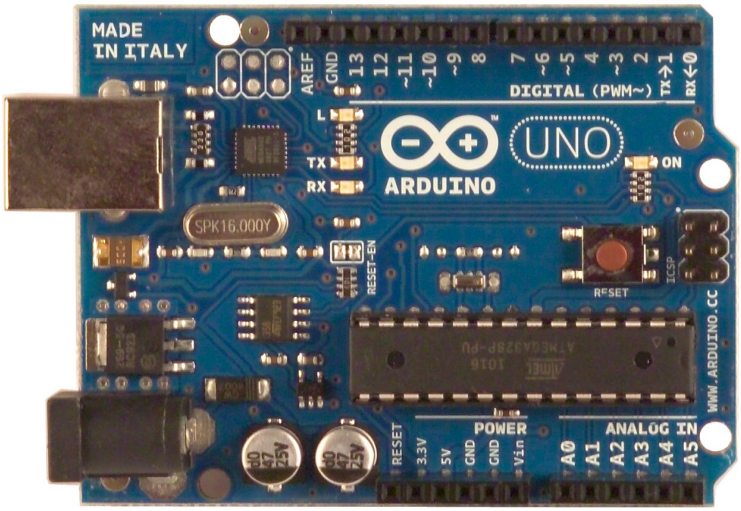
The current limiting resistor for each LED is 470 Ohms. I=5v / 470 Ohms = 10.6mA.

So, if all three LED’s are illuminated the current will increase by approx 30mA.

As I can't connect the keyboard directly to the ITX motherboard, I will need an interface.

Arduino

"*Arduino is an open-source electronics prototyping platform based on flexible, easy-to-use hardware and software. It's intended for artists, designers, hobbyists, and anyone interested in creating interactive objects or environments*"



I will be connecting the BBC's keyboard circuit to the Arduino board which will simulate the I/O lines from the BBC micro to the keyboard circuit.

I will not be running the keyboard in 'free-running' mode, but will use the micro controller to scan the keyboard continually. I will then be sending any keys detected to the ITX motherboard using the PS2 interface.

I chose the Arduino UNO for a number of reasons:

Easy to program – Free software

Plenty of I/O lines

Cost

Lots of support

The Arduino is programmed using the Arduino programming language which is very similar to C++, except that it has various commands that can be used to utilise the various I/O functions on the board. Short programs can be written on a PC and sent over the USB interface to the devices flash memory.

Summary

Micro-controller ATmega328

Operating Voltage 5V.

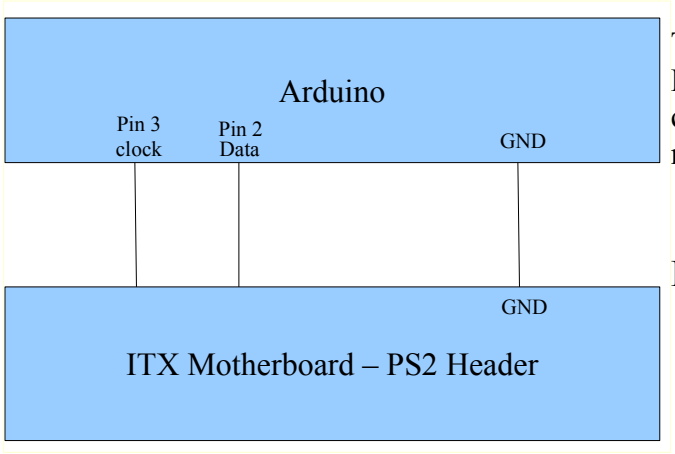
Digital I/O Pins 14. Analogue Input Pins 6

Flash Memory 32 KB (ATmega328) of which 0.5 KB used by boot loader

Clock Speed 16 MHz

Interfacing the Arduino with the PS2 port on the ITX Motherboard

As well as controlling the keyboard, the Arduino will communicate with the PS2 port on the ITX motherboard.
This only requires 3 connections.



The motherboard I am using doesn't have a PS2 socket on the back panel, so I will be connecting straight to the PS2 header on the motherboard.

PS/2 Keyboard Port Header

Pin	Signal Name
1	Data
2	Key (no pin)
3	Ground
4	+5V
5	Clock
6	Key (no pin)

Communication between the Arduino and the motherboard

The first step is to get the Arduino communicating with the PS2 port and make the motherboard think it has a keyboard attached. See Appendix 1 for more details on PS2 keyboard communication protocols.

With the Arduino connected to the PS2 header as shown above, I now needed some software to enable the correct communication protocols to be used. After a lot of searching I found the following library:

PS2 Device Library

Rather than try to re-invent the wheel I found the following links on the Arduino forums:

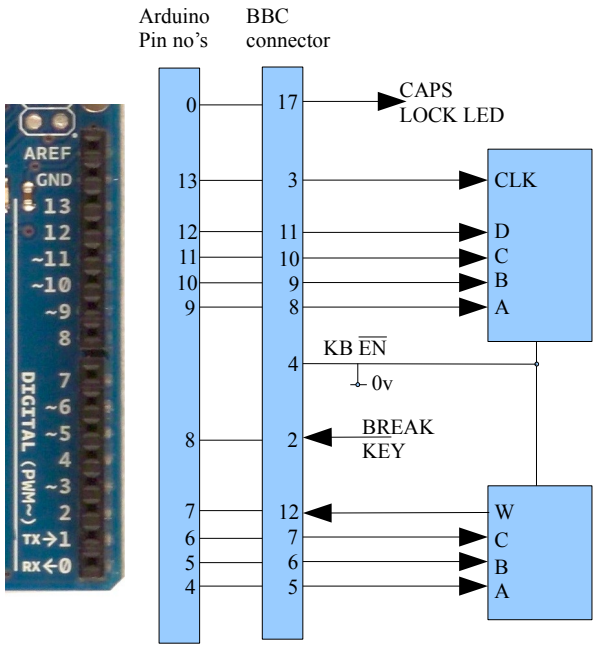
PS2 keyboard emulator
http://arduino.cc/forum/

I made some slight modifications to the code to get my Arduino communicating with the motherboard. ***Click here to download the files – HOST ON PLUSNET***
unpack ps2dev.h & ps2dev.cpp into the Arduino Libraries folder.. The sketch file should be placed wherever you want to keep it.

Connecting the BBC Keyboard to the Arduino

BBC Keyboard		**Arduino**		
Pin No	**Name**	**Pin No**	**Name**	**Description**
1	0V	NC		
2	$\overline{\text{RST}}$	8	Digital Input	This is the BREAK key, which is wired separately to the keybo
3	1MHz	13	Digital Output	Clock – Won't need to run at 1MHz, will need to experiment
4	KB $\overline{\text{EN}}$	NC		Keep Low to enable Arduino control of keyboard
5	PA4	4	Digital Output	Input A to Multiplexer
6	PA5	5	Digital Output	Input B to Multiplexer
7	PA6	6	Digital Output	Input C to Multiplexer
8	PA0	8	Digital Output	Input A to Binary counter
9	PA1	10	Digital Output	Input B to Binary counter
10	PA2	11	Digital Output	Input C to Binary counter
11	PA3	12	Digital Output	Input D to Binary counter
12	PA7	7	Digital Input	W Output of Multiplexer – Detects key pressed at intersection
13	LED3	NC		Connect to Power LED on motherboard
14	CA2	NC		This is used to detect a key press when the keyboard is 'free ru
15	+5V	NC		
16	LED1	NC		Connect to HDD LED on motherboard
17	LED2	0	Digital Output	CAPS LOCK LED

Revised Circuit diagram



Appendix 1. PS/2 Keyboard protocol

The PS/2 keyboard communicates using a bi-directional serial line. When the line is idle, the keyboard can send data consisting of 11 bits of data, 8 bits of this representing the code of the key pressed or a keyboard-to-host command. Additionally, there is a set of commands that can be exchanged between the host and the keyboard. The host has priority over the communication line (i.e., command may be sent at any time and there will be no key codes transmitted from the keyboard until the command has been sent).

Although the keyboard will operate properly without any configuration, the user may wish to change certain parameters (e.g., instruct the keyboard to turn on/off the status LEDs or change the key repeat rate). This can be performed by sending the appropriate command to the keyboard, as described in the section called “Host-to-Keyboard communication”.

Keyboard-to-Host communication

The keyboard will send a code whenever a key is pressed, held down, or released. A "Make" code is sent when the key is pressed down, and repeated periodically if the key is held down. The "break" code is sent when the key is released. e.g. If key ‘A’ is pressed and then released the following key codes will be sent from the keyboard to the host :

1C key code for ‘A’ (to indicate that ‘A’ has been pressed)
F0 break code
1C key code for ‘A’ (to indicate that ‘A’ has been released

In addition to the key codes, the keyboard might also send commands to the host. The most common commands are listed below:

Keyboard-to-host commands

Code (hexadecimal)	Command
FA	Acknowledge
AA	Self Test Passed
EE	Echo response
FE	Resend request
00	Error
FF	Error

Keyboard serial data format

Bit	Function
11	Stop bit (always 1)
10	Parity bit (odd parity)
9	Data bit 7 (MSB)
8	Data bit 6
7	Data bit 5
6	Data bit 4
5	Data bit 3
4	Data bit 2
3	Data bit 1
2	Data bit 0 (LSB)
1	Start bit (always 0)

At power on the keyboard will perform a diagnostic self-test, referred to as BAT (basic assurance test). When the test is complete a BAT completion code oF either AA (success) or FC (Error) is sent. The code must be sent 500-700mS after power-on or the keyboard may not be detected by the host.

Host-to-Keyboard communication

The host can send commands to the keyboard to control its behaviour. The most common commands are described below.

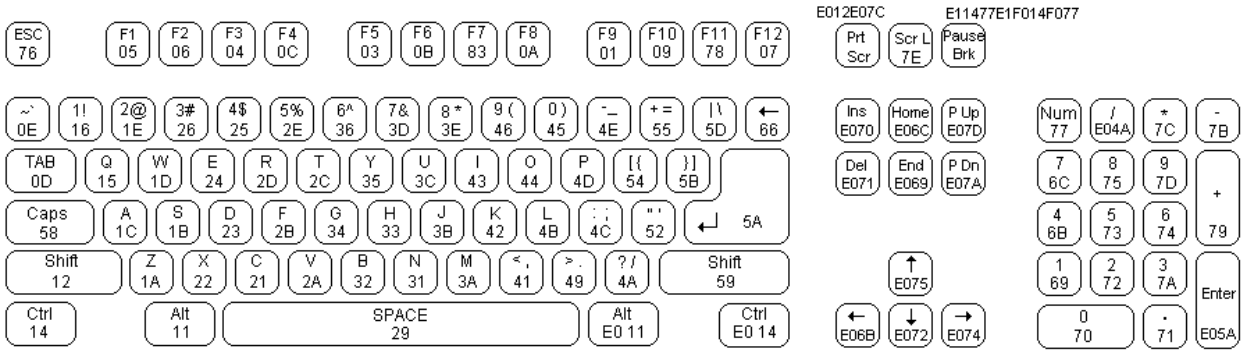
Host commands have priority over keyboard commands and will inhibit the transmission of any key codes from the keyboard. Instead, the keyboard will transmit an acknowledge code upon receiving the command.

Host-to-keyboard commands

Code (hexadecimal)	Command
ED	Set status LEDs - This command can be used to turn on and off the Num Lock, Caps Lock and Scroll Lock LEDs. After receiving this command, the keyboard will reply with an ACK (FA) and wait for another byte which determines the status of the LEDs. Bit 0 controls the Scroll Lock, bit 1 controls the Num Lock and Bit 2 controls the Caps lock. Bits 3 to 7 are ignored.
EE	Echo - The keyboard should reply with an Echo (EE) upon receiving this command.
F0	Set scan code set - Upon receiving F0, the keyboard will reply with an ACK (FA) and wait for another byte. This byte can be in the range 01 to 03, and it determines the scan code set to be used. Sending 00 as the second byte will return the scan code set currently in use. Default is normally set 2.
F3	Set repeat rate - The keyboard will acknowledge the command with an ACK (FA) and wait for the second byte which determines the repeat rate.
F4	Keyboard Enable - Clears the output buffer, enables the keyboard (i.e., key codes will be transmitted) and returns an ACK.
F5	Keyboard Disable - Resets the keyboard, disables the keyboard (key codes will not be transmitted) and returns an ACK.
FE	Resend - Upon receipt of the resend command the keyboard will retransmit the last byte sent.
FF	Reset - Resets the keyboard.

Graphic keys-to-keycodes map

Key names are on top, with the hexadecimal make code below.



Keycode tables

Standard PS/2 Keycodes, default set (set 2)

KEY	MAKE	BREAK	KEY	MAKE	BREAK
A	1C	F0,1C	R ALT	E0,11	E0,F0,11
B	32	F0,32	APPS	E0,2F	E0,F0,2F
C	21	F0,21	ENTER	5A	F0,5A
D	23	F0,23	ESC	76	F0,76
E	24	F0,24	F1	05	F0,05
F	2B	F0,2B	F2	06	F0,06
G	34	F0,34	F3	04	F0,04
H	33	F0,33	F4	0C	F0,0C
I	43	F0,43	F5	03	F0,03
J	3B	F0,3B	F6	0B	F0,0B
K	42	F0,42	F7	83	F0,83
L	4B	F0,4B	F8	0A	F0,0A
M	3A	F0,3A	F9	01	F0,01
N	31	F0,31	F10	09	F0,09
O	44	F0,44	F11	78	F0,78
P	4D	F0,4D	F12	07	F0,07
Q	15	F0,15	SCROLL	7E	F0,7E
R	2D	F0,2D	[	54	FO,54
S	1B	F0,1B	INSERT	E0,70	E0,F0,70
T	2C	F0,2C	HOME	E0,6C	E0,F0,6C
U	3C	F0,3C	PG UP	E0,7D	E0,F0,7D
V	2A	F0,2A	DELETE	E0,71	E0,F0,71
W	1D	F0,1D	END	E0,69	E0,F0,69
X	22	F0,22	PG DN	E0,7A	E0,F0,7A
Y	35	F0,35	UP	E0,75	E0,F0,75
Z	1A	F0,1A	LEFT	E0,6B	E0,F0,6B
0	45	F0,45	DOWN	E0,72	E0,F0,72
1	16	F0,16	RIGHT	E0,74	E0,F0,74
2	1E	F0,1E	NUM	77	F0,77
3	26	F0,26	KP /	E0,4A	E0,F0,4A
4	25	F0,25	KP *	7C	F0,7C
5	2E	F0,2E	KP -	7B	F0,7B
6	36	F0,36	KP +	79	F0,79
7	3D	F0,3D	KP EN	E0,5A	E0,F0,5A
8	3E	F0,3E	KP .	71	F0,71
9	46	F0,46	KP 0	70	F0,70
`	0E	F0,0E	KP 1	69	F0,69
-	4E	F0,4E	KP 2	72	F0,72
=	55	FO,55	KP 3	7A	F0,7A
\	5D	F0,5D	KP 4	6B	F0,6B
BKSP	66	F0,66	KP 5	73	F0,73
SPACE	29	F0,29	KP 6	74	F0,74
TAB	0D	F0,0D	KP 7	6C	F0,6C
CAPS	58	F0,58	KP 8	75	F0,75
L SHFT	12	FO,12	KP 9	7D	F0,7D
L CTRL	14	FO,14	]	5B	F0,5B
L GUI	E0,1F	E0,F0,1F	;	4C	F0,4C
L ALT	11	F0,11	'	52	F0,52
R SHFT	59	F0,59	,	41	F0,41
R CTRL	E0,14	E0,F0,14	.	49	F0,49
R GUI	E0,27	E0,F0,27	/	4A	F0,4A