Keyboard Controller 介绍

Daway 整理

来源：BIOS 论坛 BIOS 工程师的伊甸园 » BIOS 入门

http://www.biosren.com/thread-40-1-1.html

2009-11-06

目　　录

1 Keyboard Controller 简介

主板的键盘有一块专用的接口芯片，一般是采用一块单片微处理器 8042（现在大多已集成在南桥或 SIO 里）。它控制整个键盘的工作，包括加电自检、键盘扫描码的缓冲以及与主板的通讯。INT 09H 是 H/W 中断，对应 IRQ1，INT 16H 是一个 S/W 中断。当键盘的一个键被按下时，键盘接口芯片根据被按下的位置，INT 09H 负责把键值转换成 INT16H 认识的值，返回给 INT 16H。INT 16H 再把该值根据 OS 所选定的不同语系键盘而转换成相应的二进制字符传给 OS 或应用程序。当用户敲击键盘速度过快，使主 CPU 来不及处理时，则先将所键入的内容送往住存储器的键盘缓冲区，等 CPU 能处理时，便从缓冲区中取出，送入 CPU 进行分析和执行。一般在 PC 机的内存中安排了大约 20 个字符的键盘缓冲区。

8042 分输入缓冲和输出缓冲，它的数据传输在 I/O 口 60H 和 64H 进行。基本上，I/O 64H 是命令和状态口，I/O 60H 是数据口，它们同时可做读写动作，在读和写时有着不同的意义。I/O 64H 的 bit 0、1 置位分别代表输出/输入缓冲满。如果发现输入缓冲满（即判断出 I/O 64H[1]=1），要从 I/O 60H 将数据读完。BIOS 在自检时如果确定输入/输出缓冲都没有问题，会发“AAH”给 I/O 64H，让它自测试。等到输入缓冲空（说明上一个命令已执行完），输出缓冲满（KB 控制器对自测试命令有反应），再读 I/O 60H 是否为“55H”（IBM PC/AT 规范）。如果是，则表示 KB 没有问题，若等不到输出缓冲满，说明有问题。

在写命令之前，必须对 I/O 64H 口送一个 60H 的值，并等到输入缓冲空，再操作 I/O 60H。同样，在读状态之前，也必须对 I/O 64H 口送一个 20H 的值，并等到输出缓冲满（表示有状态输出），再操作 I/O 60H。这时，我们可以把 64H 看作索引口，而 60H 看作数据口。

键盘接口芯片除了接受来自键盘的信息外，还要负责 A20 地址线的切换，因为当 CPU 从实模式切换到保护模式时便是通过 A20 地址线的切换完成的。平常 A20 为“0”时，CPU 工作于 DOS 的实模式；当 A20 切换为“1”时，便可进入保护模式。但由于键盘接口芯片切换 A20 地址线的速度不够快，目前多由主板上的芯片组以模拟方式取代，这样也就省去了一块键盘接口芯片。

2 Overview

键盘是计算机系统的重要输入设备，所有的 IBM PC 及其兼容机都有一个键盘。所以键盘驱动是一个面向 IBM PC 机 OS 的必不可少的部分。

当 IBM 从 1981 年开始，每次推出其新的 PC 机架构，同时也推出其新的键盘设计—最早的“IBM PC”，到稍后的“IBM XT”，所使用的键盘被称作"XT Keyboard"，现在这种键盘已经完全过时，我们现在写键盘驱动程序时可以完全不用考虑它。随后 1984 年 IBM 推出了“IBM AT”，它所使用的键盘被称作“AT Keyboard”；1987 年 IBM 推出的“IBM PS/2”使用的键盘被称作“PS/2 Keyboard”。“AT Keyboard”和“PS/2 Keyboard”大同小异，被称作 IBM 兼容键盘，所有的现代 IBM PC/兼容机都支持他，它的接口相对简单，是本部分内容的重点。而当今最新的 PC 上都支持 USB 接口的键盘，但他的接口相对复杂的多，并且也不向后兼容，所以本部分内容不涉及它。

3 History

IBM 从 1981 年发布它的第一款个人计算机“IBM PC”以来，它所使用的键盘也在不断的更新。如下表所示：

机型	IBM PC/XT	IBM AT	IBM PS/2
发布年份	1981	1984	1987
按键数	81	83-101	83-101
串行协议	单向	双向	双向
Scan code set	1	2	2(增加了可选的 Scan code set 3)
接口芯片	8255	8042	8042
主机到键盘的命令数	无	8	17

"PS/2 Keyboard"最初只是对“AT Keyboard”作了一些扩展，它完全兼容“AT Keyboard”。但对于数量众多的键盘生产商来说，他们所生产的键盘并非完全遵照"PS/2 Keyboard"或“AT Keyboard”的标准，而是只需要做到对它们兼容即可。比如，某些使用 PS/2 接口的键盘却只实现了 7 个主要的“主机到键盘的命令”，对于剩下了 10 个只是简单的回馈 ACK。而某些使用 AT 接口的键盘却完全实现了“PS/2 Keyboard”的 17 个命令。所以，你所拥有的键盘即使是 IBM 兼容键盘，而未必会完全实现了标准所规定的所有功能。

所以，现在的 IBM 兼容键盘有如下特征：

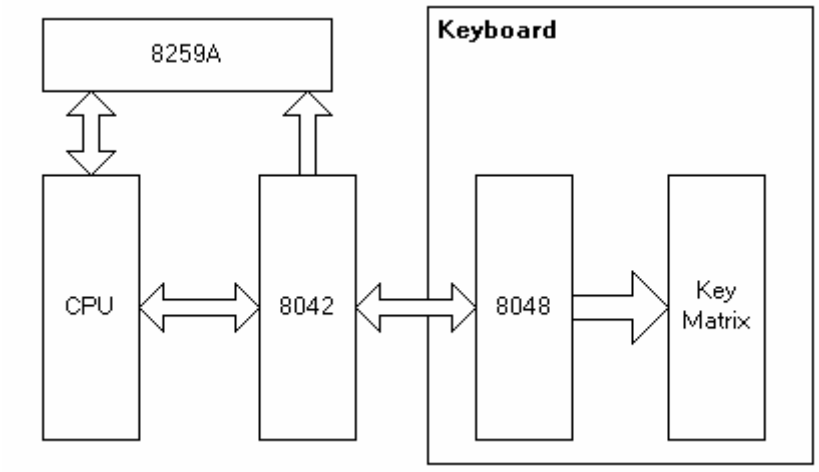
- 任意数量的按键（通常是 101 到 104）；
- 双向串行协议；
- 仅仅保证支持 Scan code set 2；
- 接口芯片可能不是 8042，但保证都和 8042 兼容。
- 对所有的 17 个命令都会回复“ACK”，但未必会完全实现它们。

“XT Keyboard”使用的协议和“AT-PS/2 Keyboard”完全不同，所以它们之间完全不兼容。但在过渡时期，有一些键盘生产厂商生产的"AT-PS/2 Keyboard"，可以通过设置跳线，使其兼容"XT Keyboard"。这只不过在一个键盘里实现了两种方式，并不意味着“XT Keyboard”和"AT-PS/2 Keyboard"是兼容的。

由于“XT Keyboard”已经完全过时，我们下面不会再介绍“XT Keyboard”相关的细节，我们的内容定位于“AT-PS/2 Keyboard”。

4 Architechure

在 IBM AT 和 IBM PS/2 键盘系统中，CPU 并不直接和 Keyboard 进行通信，而是通过一个 8042 芯片或者其它与之兼容的芯片。增加这么一个中间层，就可以屏蔽掉不同键盘之间实现的差别，并可以增加新的特性。如下图所示：



CPU 直接和 8042 芯片进行通信，以实现对整个键盘的控制；键盘从外界输入得到的数据也可以通过 8042 芯片通知给 CPU，然后 CPU 可以通过 8042 芯片读取这些数据。另外，CPU 也直接向 8042 芯片发送命令，以使用 8042 芯片自身所提供的功能。

键盘自身也有自己的芯片（Intel 8048 及其兼容芯片），此芯片的功能主要是检索来自于 Key Matrix 的外界输入（击键（Press key）或释放键（Release Key））所产生的 Scan code，并将这些 Scan code 存放于键盘自身的内部缓冲；还负责和外部系统（i8042）之间的通信，以及自身的控制（Self Test, Reset, etc）等等。

5 Mechanism

对于 PC 机的操作用户来说，与键盘的接口就是键盘上的按键。操作键盘的方式就是敲击这些按键。对于键盘系统而言，操作用户对键盘的敲击分为两种动作：Press key 和 Release key。这两个动作之间，

还有一个时间段，被称为 Press key delay。我们将这 2 个动作和 1 个时间段称为一个“击键过程”。

你可以设想一下这个动作—按下一个键，保持一段时间，再松开这个键—这就是你在大多数情况下快速击键动作的一个慢镜头，它明确的出现了 2 个动作和 1 个时间段。但无论你的击键动作有多快，都是上述 2 个动作和 1 个时间段的组合。

对于除了 Pause 键之外的所有键而言，键盘针对 Press Key 和 Release Key 两个动作会分别产生两个 Scan code，被称作 Make Code 和 Break Code。在这两个动作之间的时间段里，会按照一定的频率产生 Repeat code。在大多数情况下，由于你的击键速度非常快，所以不会产生 Repeat code；但在任何情况下，肯定会产生 Make code 和 Break Code。

键盘都有一个 Repeat code 的“产生延迟”设置，这个“产生延迟”指的是两次产生“Repeat code”之间的时间间隔，比如,如果"产生延迟"被设置为 0.25 秒，则当一个键被保持 Press 状态时，键盘系统会每 0.25 秒产生一个针对此键的 Repeat code。

有时候，你会同时按下多个键，对于键盘系统而言，针对这些键的 Press key 动作总有现有之分。但你有可能对这些按键按下之后会保持一段时间才放开。这时候，键盘总是对你最后发生 Press key 动作的键产生 Repeat code。而对你之前按下而没有松开的键，在它的当前“击键过程”内不会再产生 Repeat code，即使在它之后被按下的键都已经完全松开。

比如，你现在按下了"A"键，产生了一个 Press key 的动作，键盘系统会为之产生一个 Make code；随后，你保持按着"A"键不放开，键盘系统将会按照一定的频率产生针对"A"键的 Repeat code。这个时候，你由按下了"B"键，键盘随即停止产生"A"键的 Repeat code，然后产生一个"B"键的 Make code。从此以后，在"A"键的当前“击键过程”中，"A"的 Repeat code 再也不会产生，即使"B"在"A"被 Release 之前松开也是这样。而之后如果“B”被保持按着的话，则键盘系统会按照一定的频率产生"B"的 Repeat code，直到"B"被松开，或又有一个键被 Press 为止。但无论"A"或"B"在任何时候被松开，都必然会产生一个 Break code。

结论是：在键盘被打开的情况下，只要一个键发生了 Press key 的动作，就一定会产生一个 Make code；只要一个键(除了"Pause/Break"键)发生了 Rlease key 的动作，就一定会产生一个 Break code；无论它们在什么时候发生。而对于 Repeat Code，则有两个条件，一是，到当前的时刻为止，最后被按下的键；二是，这个最后被按下的键，在被松开之前被按的时间超过键盘所设置的 Repeat code“产生延迟”。

在键盘系统中，由两根线，Data line 和 Clock line，用来控制对 Scan code 的检索和传递。如果 Data line 和 Clock line 都处于高电平状态，则每次 8048 每次检索到一个 Scan code，就会立即将其发送给 8042 芯片。如果 Data line 为高电平，而 Clock line 为低电平，则每次 8048 每次检索到一个 Scan code，不会立即将其发送给 8042 芯片，而是先将其存放在键盘的内部缓冲中，等 Clock line 变成高电平后，再将缓冲中的 Scan code 发送给 8042。如果 Data line 为低电平，则 8048 停止对 Scan code 的检索，转而等待接收来自于 8042 的命令，这种情况下，如果 Clock line 为高电平，8048 则会将接到的命令的回复数据发送给 8042，否则，则无法回复这些命令。所以在 8042 需要向 8048 发送命令时，必须保证 Clock line 为高电平状态。

如果 8042 芯片收到一个来自于 8048 芯片的 Scan code 或者命令回复字节，经过处理后（可能存在的解码操作），会将其放入 8042 的 Output buffer 中，8042 芯片会首先将状态寄存器（Status Register）的 OBF(Output Buffer Full)标志设置，随后将 Output port 的 IBF(bit-4)设为 1，表示将产生一个 IRQ1，然后将 Clock line 置为低电平，以禁止 8042 进一步接收 8048 的数据；然后发送一个 IRQ1 给 Intel 8059A 可编程中断控制器，由它将中断提交给 CPU，CPU 收到此 IRQ 后，将调用此 IRQ 对应的 ISR(中断服务程序,这就是我们键盘 Driver 的一个重要部分)。随后此 ISR 可以从 8042 的数据端口 60H 中将 Output buffer 数据读取出来，并进行进一步的处理。当 Output buffer 中的数据被读取出来之后，8042 会将状态寄存器的 OBF 标志清 0，然后将 Clock line 置为高电平，以允许进一步接收 8048 发送来的数据。

6 Scan Code Set

迄今为止，IBM PC 键盘共有 3 套 Scan Code Set，最早的 IBM PC/XT 使用 Scan Code Set 1，在随后的系统中，默认的都是 Scan Code Set 2，后来出现了 Scan Code Set 3，但并非所有的键盘都支持。为了保证正确性，我们应该以 Scan Code Set 2 为开发对象。

Scan Code Set 1 和 Scan Code Set 2 是不相同的，但你可以让 8042 芯片帮你将从 8048 芯片得到的属于 Scan Code Set 2 的 Scan Code 转换为 Scan Code Set 1 中对应的 Scan Code，这样，你的驱动程序只需要都以一种方式—Scan Code set 1 处理就行了。设置 Scan Code 转换的方法为将 8042 Command Byte 的 bit-6 清 0。如果你设置了 Scan Code 转换，则 8042 在得到一个来自于 8048 的 Scan Code 之后，会首先将其转换为 Scan Code set 1 中对应的 Scan Code，然后再将其放入 8042 的 Output buffer 中。这样 Keyboard Driver 从中读出的时候，就已经是属于 Scan Code Set 1 的 Scan Code 了。

需要注意的是，Scan Code 和 ASCII 码完全不相同，所以 Keyboard Driver 的一个重要任务是将 Scan Code 和 ASCII 之间建立一种映射关系，将从 8042 读到的 Scan Code 转换为 ASCII 码。这就提供了一个特性：你可以通过建立不同的映射表，将键映射成你所喜欢的方式。比如你完全可以将键“A”映射为字母“B”的 ASCII 码。事实上，你也可以其映射到扩展 ASCII 码上，以实现其它语言的输入，比如简体中文—假如你的键盘驱动中实现了 GB2312 的映射，则你敲击两次键盘则可以输入一个汉字。

6.1 Scan code set 1

在 Scan Code Set 1 中，对于绝大多数键而言，其 Make Code，Break Code，以及 Repeat Code 都是单字节的。其规则为：如果 Make Code 为 nn，则其 Repeat code 与 Make Code 相同也是 nn，而其 Break Code 则是将 nn 与 80h 进行按位 OR 运算，也就是将 Make Code 的最高位 bit-7 设置为 1。比如：键"A"的 Make Code 位 1Eh，其 Repeat Code 也为 1Eh，而其 Break Code 则为 1Eh|80h=9Eh。

还有一些键的 Scan Code 是双字节的。其规则为：它们的第一个字节都是 E0h，对于第 2 个字节，其规则与单字节 Scan Code 的规则一样。

对 PrtSc/SysRq 键而言，其 make code = E02AE037,repeat code = E037, break code = E0B7E0AA。

Pause/Break 键没有 Repeat Code，也没有 Break Code，只有 Make Code。其 Make Code 很长，为 E11D45E19DC5。

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,C7
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

6.2 Scan code set 2

在 Scan Code Set 2 中，对于绝大多数键而言，其 Make Code，Repeat Code 都是单字节的，而其 Break Code 为双字节的。其规则为：如果 Make Code 为 nn，则其 Repeat code 与 Make Code 相同也是 nn，而其 Break Code 的第一个字节为 F0，而第二个字节与 Make Code 相同。比如：键"A"的 Make Code 位 1Ch，其 Repeat Code 也为 1Ch，而其 Break Code 则为 F01Ch。

还有一些键的 Make Code，Repeat Code 是双字节的，其 Break Code 则是 3 字节的。其规则为：它们的第一个字节都是 E0h，对于后两个字节，其规则与单字节 Scan Code 的规则一样。

对 PrtSc/SysRq 键而言，其 Make code = E012E07C, repeat code = E07C, break code = E0F07CE0F012

Pause/Break 键没有 Repeat Code，也没有 Break Code，只有 Make Code。其 Make Code 很长，为 E11477E1F014F077。

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

6.3 Scan code set 3

Scan Code Set 3 的 Scan Code 只有一种规则，与 Scan Code Set 2 的单字节 Make Code 的规则一样。这非常有利于 Keyboard driver 的实现。可惜的就是只有一部分键盘支持它。

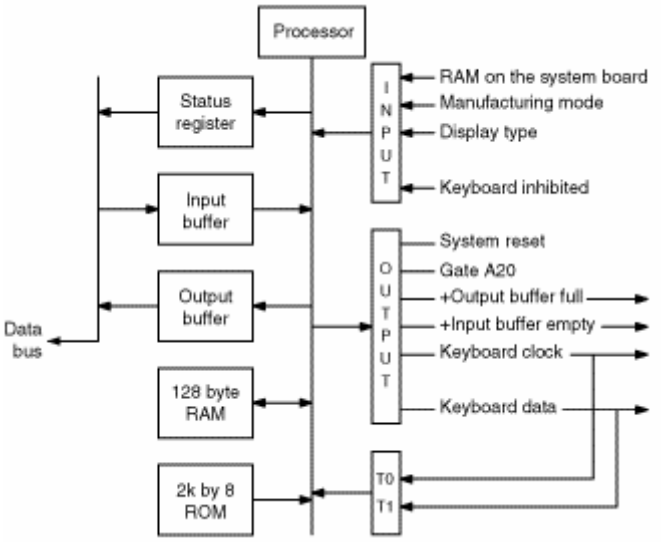
KEY	MAKE	BREAK	-----	KEY	MAKE	BREAK	-----	KEY	MAKE	BREAK
A	1C	F0,1C		9	46	F0,46		[	54	F0,54
B	32	F0,32		`	0E	F0,0E		INSERT	67	F0,67
C	21	F0,21		-	4E	F0,4E		HOME	6E	F0,6E
D	23	F0,23		=	55	F0,55		PG UP	6F	F0,6F
E	24	F0,24		\	5C	F0,5C		DELETE	64	F0,64

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				

7 8042 Controller

4.2.4 节所描述的过程就是如何把操作用户的外部输入转化为系统软件（也就是操作系统）可以获取的数据的过程。除了上述的外部输入的转化过程之外，操作系统还可以通过向 8042 芯片发送命令来分别控制 8042 芯片和 8048 芯片。

下图表现的就是 8042 芯片的内部结构，图中 Processor 不是指计算机的 CPU，而是 8042 芯片自身的处理器。



8042 芯片除了被用来控制键盘，作为键盘和 CPU 之间的桥梁之外，还可以被用来控制 A20 Gate，以决定 CPU 是否可以访问以 MB 为单位的偶数内存；以及向系统发送 Reset 信号，让主机重新启动。另外，8042 芯片还可以支持 PS/2 类型的鼠标，但这一点本部分不会进行讨论。

8042 有 4 个寄存器：

- 1 个 8-bit 长的 Input buffer；Write-Only；
- 1 个 8-bit 长的 Output buffer； Read-Only；
- 1 个 8-bit 长的 Status Register；Read-Only；
- 1 个 8-bit 长的 Control Register；Read/Write。

其中 Control Register 在上图中没有表现，又被称作 Command Byte。

另外，8042 有 2 个端口：60h 和 64h。

Port	R/W	Function
0x60	Read	Read Output Buffer
0x60	Write	Write Input Buffer(8042 Data&8048 Command)
0x64	Read	Read Status Register
0x64	Write	Write Input Buffer(8042 Command)

上面提到的 4 个寄存器中，前 3 个寄存器都可以通过 60h 和 64h 直接访问，但第 4 个寄存器只能够通过向 64h 端口发送命令，然后通过 60h 端口存取。

Status Register 中存放的是一些与缓冲状态，数据状态，及键盘状态有关的状态信息，是一个 8-bit 长的积存器。它对于 OS 来说是只读的，并且只能够通过 64h 端口读取，任何时候，只要读取 64h 端口，都会读到 Status Register 的内容。

bit	Meaning
0	output register (60h) has data for system
1	input register (60h/64h) has data for 8042
2	system flag (set to 0 after power on reset)
3	data in input register is command (1) or data (0)
4	1=keyboard enabled, 0=keyboard disabled (via switch)
5	1=transmit timeout (data transmit not complete)
6	1=receive timeout (data transmit not complete)
7	1=even parity rec'd, 0=odd parity rec'd (should be odd)

7.1 Status Register

Input buffer 被用来向 8042 芯片发送命令与数据，以控制 8042 芯片和 8048 芯片。Input buffer 可以通过 60h 端口和 64h 端口写入，其中通过 64h 端口写入的是用来控制 8042 芯片的命令，而通过 60h 写入的数据有两种：一种是通过 64h 端口发送的被用来控制 8042 芯片的命令所需要的进一步数据；另一种则是直接发给键盘用来控制 8048 芯片的命令。

Output buffer 被存放可以通过 60h 端口读取的数据。这些数据分为 2 大类：一类是那些通过 64h 端口发送的，被用来控制 8042 芯片的命令的返回结果；另一类则是 8048 芯片所发过来的数据。后者的数据又分为 Scan code，和对那些通过 60h 端口发送给 8048 的命令的回复结果。

8042 自身有少量的 RAM，以及一些 ROM，这些内存与我们常说的内存（称为系统内存）没有任何关系，它们和系统内存不使用相同的地址空间。这些 RAM 和 ROM 是 8042 芯片处理器自身运算的需要。

除了上述元素之外，8042 还有 3 个内部端口：Input port, Output port 和 Test Port。Output port 有 System Reset 和 A20 Gate 两个与键盘无关的重要的控制位，其它的位都是向 8048 芯片输出，让 8048 芯片参考的控制位。程序员最好不要动这些除了 System Reset 和 A20 Gate 之外的控制位。IBM AT 和 IBM PS/2 的这 3 个端口的格式有少许不同，主要因为在 IBM PS/2 上，8048 同时支持 PS/2 鼠标。

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

7.2 Input Port

Pin	Name	PS/2 Function	AT Function
0	P20	System Reset 1: Normal 0: Reset computer	System Reset 1: Normal 0: Reset computer
1	P21	A20 Gate 1: Enable 0: Disable	A20 Gate 1: Enable 0: Disable
2	P22	Mouse Data: 1: Pull Data low 0: High-Z	Undefined
3	P23	Mouse Clock: 1: Pull Clock low 0: High-Z	Undefined
4	P24	Keyboard IBF interrupt: 1: Assert IRQ 1 0: De-assert IRQ 1	Output Buffer Full
5	P25	Mouse IBF interrupt: 1: Assert IRQ 12 0: De-assert IRQ 12	Input Buffer Empty
6	P26	Keyboard Clock 1: Pull Clock low 0: High-Z	Keyboard Clock 1: Pull Clock low 0: High-Z
7	P27	Keyboard Data: 1: Pull Data low 0: High-Z	Keyboard Data: 1: Pull Data low 0: High-Z

7.3 Output Port

Pin	Name	Function
0	T0	Keyboard Clock
1	T1	Mouse Clock
2	--	Undefined
3	--	Undefined
4	--	Undefined
5	--	Undefined
6	--	Undefined
7	--	Undefined

7.4 Test Port

Command Byte 不能直接通过 60h 和 64 端口读取，若要访问它，必须首先通过 64h 端口向 8042 发布相应命令（20h/read,60h/write），然后再通过 60h 存取。

bit	Meaning
0	1=enable output register full interrupt
1	should be 0
2	1=set status register system, 0=clear
3	1=override keyboard inhibit, 0=allow inhibit
4	disable keyboard I/O by driving clock line low
5	disable auxiliary device, drives clock line low
6	IBM scancode translation 0=AT, 1=PC/XT
7	reserved, should be 0

Commnand Byte

8 Command

8.1 Command 介绍

通过 8042 芯片，可以：

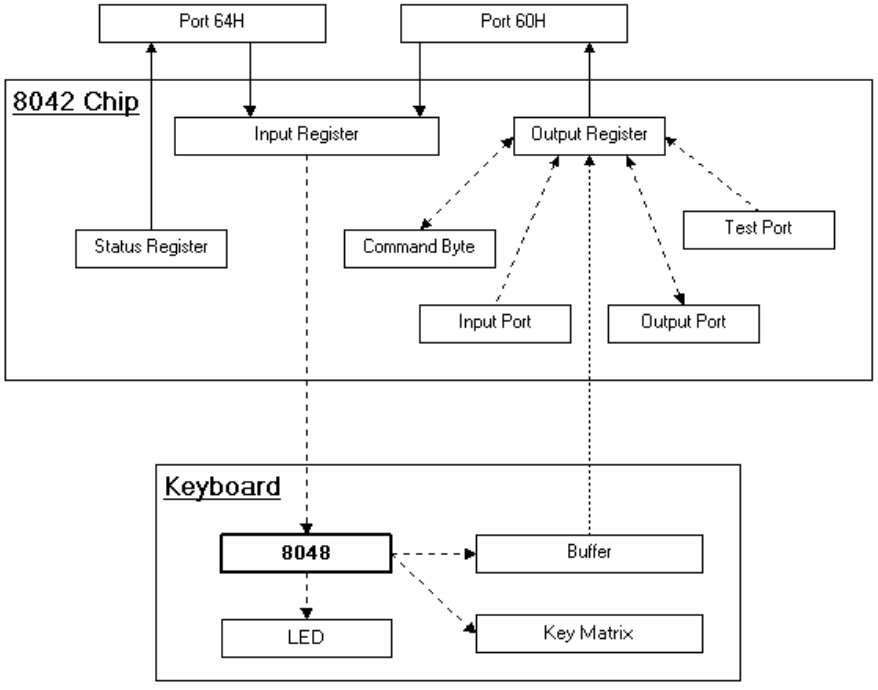
向 8042 芯片发布命令（通过 64h），并通过 60h 读取命令的返回结果（如果有的话），或通过 60h 端口写入命令所需的数据（如果需要的话）。

读取 Status Register 的内容（通过 64h）；

向 8048 发布命令（通过 60h）；

读取来自于 Keyboard 的数据（通过 60h）。这些数据包括 Scan Code（由按键和释放键引起的），对 8048 发送的命令的确认字节（ACK）及回复数据。

再次强调一遍，Command(命令)分为发送给 8042 芯片的命令和发送给 8048 的命令。它们是不相同的，并且使用的端口也是不相同的（分别为 64h 和 60h）。



- **64h 端口（读操作）**

对 64h 端口进行读操作，会读取 Status Register 的内容。

inb %0x64

执行这个指令之后，AL 寄存器中存放的就是 Status Register 的内容。

- **64h 端口（写操作）**

向 64h 端口写入的字节，被认为是对 8042 芯片发布的命令（Command）：

写入的字节将会被存放在 Input Register 中；

同时会引起 Status Register 的 Bit-3 自动被设置为 1，表示现在放在 Input Register 中的数据是一个 Command，而不是一个 Data；

在向 64h 端口写某些命令之前**必须**确保键盘是被禁止的，因为这些被写入的命令的返回结果将会放到 Output Register 中，而键盘如果不被禁止，则也会将数据放入到 Output Register 中，会引起相互之间的数据覆盖；

在向 64h 端口写数据之前必须确保 Input Register 是空的（通过判断 Status Register 的 Bit-1 是否为 0）。

```
void wait_input_empty(void)
{
   char __b;

   do{
     __b = inb(0x64);
   }while(!(__b&0x02));
}

void disable_keyboard(void)
{
   wait_input_empty();
   outb(0x64, 0xAD);
}
```

- **60h 端口（读操作）**

对 60h 端口进行读操作，将会读取 Output Register 的内容。Output Register 的内容可能是：

来自于 8048 的数据。这些数据包括 Scan Code，对 8048 发送的命令的确认字节（ACK)及回复数据。

通过 64h 端口对 8042 发布的命令的返回结果。

在向 60h 端口读取数据之前必须确保 Output Register 中有数据（通过判断 Status Register 的 Bit-0 是否为 1）。

```
void wait_output_full(void)
{
   char __b;

   do{
     __b = inb(0x64);
   }while(__b&0x01);
}
```

```
unsigned char read_output(void)
{
  wait_output_full();
  return inb(0x60);
}
```

- **60h 端口（写操作）**

向 60h 端口写入的字节，有两种可能：

1．如果之前通过 64h 端口向 8042 芯片发布的命令需要进一步的数据，则此时写入的字节就被认为是数据；

2．否则，此字节被认为是发送给 8048 的命令。

在向 60h 端口写数据之前，必须确保 Input Register 是空的（通过判断 Status Register 的 Bit-1 是否为 0）。

8.2 发给 8042 的命令

- **20h**

准备读取 8042 芯片的 Command Byte；其行为是将当前 8042 Command Byte 的内容放置于 Output Register 中，下一个从 60H 端口的读操作将会将其读取出来。

```
unsigned char read_command_byte(void)
{
  wait_input_empty();
  outb(0x64,0x20);
  wait_output_full();
  return inb(0x60);
}
```

- **60h**

准备写入 8042 芯片的 Command Byte；下一个通过 60h 写入的字节将会被放入 Command Byte。

```
void write_command_byte(unsigned char command_byte)
{
  wait_input_empty();
  outb(0x64,0x60);
  wait_input_empty();
  outb(0x60,command_byte);
}
```

- **A4h**

测试一下键盘密码是否被设置；测试结果放置在 Output Register，然后可以通过 60h 读取出来。测试结果可以有两种值：FAh=密码被设置；F1h=没有密码。

```
bool is_set_password(void)
{
  wait_input_empty();
  outb(0x64,0xA4);
  wait_output_full();
  return inb(0x60)==0xFA?true:false;
}
```

- A5h

设置键盘密码。其结果被按照顺序通过 60h 端口一个一个被放置在 Input Register 中。密码的最后是一个空字节（内容为 0）。

```
void set_password(unsigned char* password)
{
   char* p = password;

   if(p == NULL)
      return;
   wait_input_empty();
   outb(0x64,0xA5);
   do{
      wait_input_empty();
      outb(0x60, *p);
   }while(*p++ != 0);
}
```

- A6h

让密码生效。在发布这个命令之前，必须首先使用 A5h 命令设置密码。

```
void enable_password(void)
{
   if(!is_set_password())
      return;
   wait_input_empty();
   outb(0x64,0xA6);
}
```

- AAh

自检。诊断结果放置在 Output Register 中，可以通过 60h 读取。55h=OK。

```
bool is_test_ok(void)
{
   wait_input_empty();
   outb(0x64,0xAA);

   wait_output_full();
   return inb(0x60)==0x55?true:false;
}
```

- ADh

禁止键盘接口。Command Byte 的 bit-4 被设置。当此命令被发布后，Keyboard 将被禁止发送数据到 Output Register。

```
void disable_keyboard(void)
{
   wait_input_empty();
   outb(0x64,0xAD);
}
```

- **AEh**

打开键盘接口。Command Byte 的 bit-4 被清除。当此命令被发布后，Keyboard 将被允许发送数据到 Output Register。

```
void enable_keyboard(void)
{
   wait_input_empty();
   outb(0x64,0xAE);
}
```

- **C0h**

准备读取 Input Port。Input Port 的内容被放置于 Output Register 中，随后可以通过 60h 端口读取。

```
unsigned char read_input_port(void)
{
   wait_input_empty();
   outb(0x64,0xC0);

   wait_output_full();

   return inb(0x60);
}
```

- **D0h**

准备读取 Outport 端口。结果被放在 Output Register 中，随后通过 60h 端口读取出来。

```
unsigned char read_output_port(void)
{
   wait_input_empty();
   outb(0x64,0xD0);

   wait_output_full();

   return inb(0x60);
}
```

- **D1h**

准备写 Output 端口。随后通过 60h 端口写入的字节，会被放置在 Output Port 中。

```
void write_output_port(unsigned char __c)
{
   wait_input_empty();
   outb(0x64,0xD1);

   wait_input_empty();
   outb(0x60,__c);
}
```

- **D2h**

准备写数据到 Output Register 中。随后通过 60h 写入到 Input Register 的字节会被放入到 Output Register 中，此功能被用来模拟来自于 Keyboard 发送的数据。如果中断被允许，则会触发一个中断。

```
void put_data_to_output_register(unsigned char __data)
{
  wait_input_empty();
  outb(0x64,0xD2);

  wait_input_empty();
  outb(0x60,__c);
}
```

8.3 发给 8048 的命令

- **EDh**

设置 LED。Keyboard 收到此命令后，一个 LED 设置会话开始。Keyboard 首先回复一个 ACK（FAh），然后等待从 60h 端口写入的 LED 设置字节，如果等到一个，则再次回复一个 ACK，然后根据此字节设置 LED。然后接着等待。。。直到等到一个非 LED 设置字节(高位被设置)，此时 LED 设置会话结束。

- **EEh**

诊断 Echo。此命令纯粹为了检测 Keyboard 是否正常，如果正常，当 Keyboard 收到此命令后，将会回复一个 EEh 字节。

- **F0h**

选择 Scan code set。Keyboard 系统共可能有 3 个 Scan code set。当 Keyboard 收到此命令后，将回复一个 ACK，然后等待一个来自于 60h 端口的 Scan code set 代码。系统必须在此命令之后发送给 Keyboard 一个 Scan code set 代码。当 Keyboard 收到此代码后，将再次回复一个 ACK，然后将 Scan code set 设置为收到的 Scan code set 代码所要求的。

- **F2**

读取 Keyboard ID。由于 8042 芯片后不仅仅能够接 Keyboard。此命令是为了读取 8042 后所接的设备 ID。设备 ID 为 2 个字节，Keyboard ID 为 83ABh。当键盘收到此命令后，会首先回复一个 ACK，然后，将 2 字节的 Keyboard ID 一个一个回复回去。

- **F3h**

设置 Typematic Rate/Delay。当 Keyboard 收到此命令后，将回复一个 ACK。然后等待来自于 60h 的设置字节。一旦收到，将回复一个 ACK，然后将 Keyboard Rate/Delay 设置为相应的值。

- **F4h**

清理键盘的 Output Buffer。一旦 Keyboard 收到此命令，将会将 Output buffer 清空，然后回复一个 ACK。然后继续接受 Keyboard 的击键。

- **F5h**

设置默认状态(w/Disable)。一旦 Keyboard 收到此命令，将会将 Keyboard 完全初始化成默认状态。之前所有对它的设置都将失效—Output buffer 被清空，Typematic Rate/Delay 被设置成默认值。然后回复一个 ACK，接着等待下一个命令。需要注意的是，这个命令被执行后，键盘的击键接受是禁止的。如果想让键盘接受击键输入，必须 Enable Keyboard。

- **F6h**

设置默认状态。和 F5 命令唯一不同的是，当此命令被执行之后，键盘的击键接收是允许的。

- **FEh**

Resend。如果 Keyboard 收到此命令，则必须将刚才发送到 8042 Output Register 中的数据重新发送一遍。当系统检测到一个来自于 Keyboard 的错误之后，可以使用自命令让 Keyboard 重新发送刚才发送的字节。

- **FFh**

Reset Keyboard。如果 Keyboard 收到此命令，则首先回复一个 ACK，然后启动自身的 Reset 程序，并进行自身基本正确性检测（BAT-Basic Assurance Test）。等这一切结束之后，将返回给系统一个单字节的结束码（AAh=Success, FCh=Failed），并将键盘的 Scan code set 设置为 2。

8.4 8048 到 8042 的数据

- **00h/FFh**

当击键或释放键时检测到错误时，则在 Output Bufer 后放入此字节，如果 Output Buffer 已满，则会将 Output Buffer 的最后一个字节替代为此字节。使用 Scan code set 1 时使用 00h，Scan code 2 和 Scan Code 3 使用 FFh。

- **AAh**

BAT 完成代码。如果键盘检测成功，则会将此字节发送到 8042 Output Register 中。

- **EEh**

Echo 响应。Keyboard 使用 EEh 响应从 60h 发来的 Echo 请求。

- **F0h**

在 Scan code set 2 和 Scan code set 3 中，被用作 Break Code 的前缀。

- **FAh**

ACK。当 Keyboard 任何时候收到一个来自于 60h 端口的合法命令或合法数据之后，都回复一个 FAh。

- **FCh**

BAT 失败代码。如果键盘检测失败，则会将此字节发送到 8042 Output Register 中。

- **FEh**

Resend。当 Keyboard 任何时候收到一个来自于 60h 端口的非法命令或非法数据之后，或者数据的奇偶交验错误，都回复一个 FEh，要求系统重新发送相关命令或数据。

- **83ABh**

当键盘收到一个来自于 60h 的 F2h 命令之后，会依次回复 83h，ABh。83AB 是键盘的 ID。

- **Scan code**

除了上述那些特殊字节以外，剩下的都是 Scan code。

8042 控制命令的下達方式：

```
;
; ========================================================
; 以下為 8042 界面函數之內部呼叫函數
; ========================================================
;
Flush8042       PROC
                ; 清除所有的輸出緩衝器資料
                push ax
FlushOBF:       in   al,64h
                test al,1
                jz   NotOBF
                in   al,60h
                jmp  short FlushOBF
NotOBF:         pop  ax
                retn
Flush8042       ENDP
;
WaitIBF         PROC
                ; 等待 8042 輸入緩衝器有空
WaitIBFLoop:    push ax
                in   al,64h
                test al,2
                pop  ax
                jnz  WaitIBFLoop
                retn
WaitIBF         ENDP
;
WaitOBF         PROC
                ; 等待 8042 輸出緩衝器有資料送來
WaitOBFLoop:    push ax
                in   al,64h
                test al,1
                pop  ax
                jz   WaitOBFLoop
                retn
WaitOBF         ENDP
;
Read8042Data    PROC
                ; 讀取 8042 回應資料
                ; 傳回: AL =  回應資料
                call WaitOBF        ; 等待資料回庄
                in   al,60h
                retn
Read8042Data    ENDP
;
```

```
Write8042Data   PROC
                ; 送出資料給 8042
                ; 參數: AL = 8042 系統命令或資料
                ; 備註: 本函數亦為送出系統命令或資料的函數
                call WaitIBF        ; 等待輸入緩衝區有空
                out  60h,al         ; 送出資料
                call WaitIBF        ; 確認 8042 收到
                retn
Write8042Data   ENDP
;
RealSend8042Cmd PROC
                ; 送出一般命令碼給 8042
                ; 參數: AL = 8042 一般控制命令
                ; 備註: 8042 命令之參數或傳回值由外界處理
                call WaitIBF        ; 等待輸入緩衝區有空
                out  64h,al         ; 送出命令
                call WaitIBF        ; 確認 8042 收到
                retn
RealSend8042Cmd ENDP
;
RealSend8042Sys PROC
                ; 送出系統命令碼或參數給 8042
                ; 參數: AL = 8042 系統控制命令或參數
                ; 傳回: AL = 8042 回應值
                ; 備註: 除回音外,其餘命令或參數應檢查是否傳回 ACK
                call Flush8042
                call Write8042Data
                call Read8042Data
                retn
RealSend8042Sys ENDP
;
; ======================================================
; 以下為 8042 界面函數,使用前必須使用 CLI 將岔斷禁能
; ======================================================
;
Send8042Cmd     PROC
                ; 送出一般命令碼給 8042
                ; 參數: AL = 8042 一般控制命令
                call RealSend8042Cmd
                retn
Send8042Cmd     ENDP
;
Read8042Cmd     PROC
                ; 送出讀取命令給 8042
                ; 參數: AL = 8042 讀取命令
                ; 傳回: AL =  讀取值
```

```
                call Flush8042
                call RealSend8042Cmd
                call Read8042Data
                retn
Read8042Cmd     ENDP
;
Write8042Cmd    PROC
                ; 送出寫入命令給 8042
                ; 參數: AL = 8042 寫入命令
                ;       AH =  寫入資料
                ; 備註: AX 值會被破壞
                call RealSend8042Cmd
                xchg al,ah
                call Write8042Data
                retn
Write8042Cmd    ENDP
;
Echo8042        PROC
                ; 送出 8042 回音命令
                ; 傳回: AL = EEh
                mov  al,0EEh        ; 回音命令
                call RealSend8042Sys
                retn
Echo8042        ENDP
;
```