从键盘按下一个6，到显示出来，计算机发生了什么？

计算机领域有一个经典的问题：**从你在浏览器中输入URL并按下回车，到网页渲染出来，这中间发生了什么？**

通过这个问题，可以考察候选人对计算机网络的理解程度，因此出现在数不清的面试场合。

毋庸置疑，这是一个好问题，我也看到不下100篇文章在探讨这个问题的答案。

而今天，我想跟大家探讨的是另外一个问题：**从你在键盘上按下一个“6”，到屏幕上显示出来，计算机发生了什么？**

这个问题无论从空间尺度还是时间尺度比起开始那个问题都更小得多。

空间尺度上，这个问题探讨的范围只限于一台计算机上，没有跨越网络。

时间尺度上，第一个问题的时间尺度在秒级别，而这个问题的时间尺度在**毫秒**级别。

尺度虽然小了但背后的技术知识并不少。

我相信，等你看完这篇文章，搞清楚这个问题的答案，你将对计算机组成原理、操作系统、CPU这些东西有完全不一样的理解。

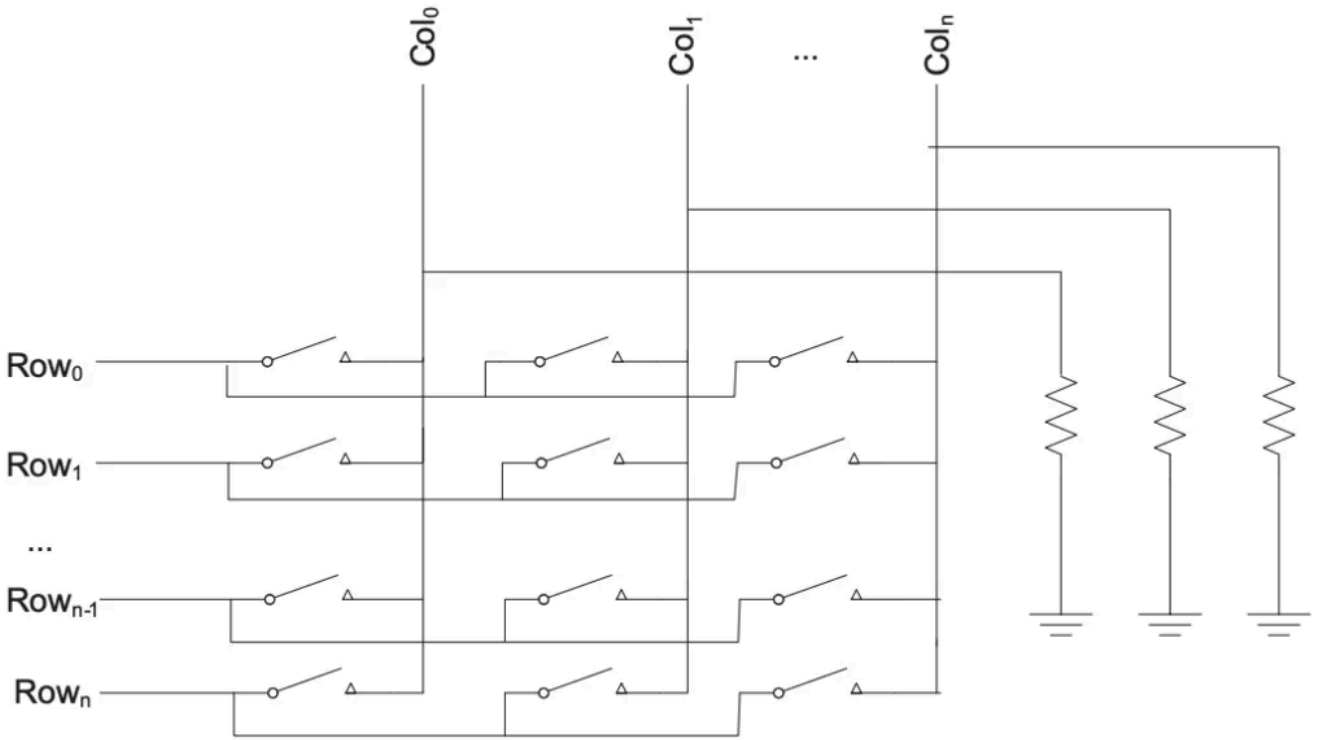
准备好，咱们出发！

0x01: 按下按键，键盘做了什么

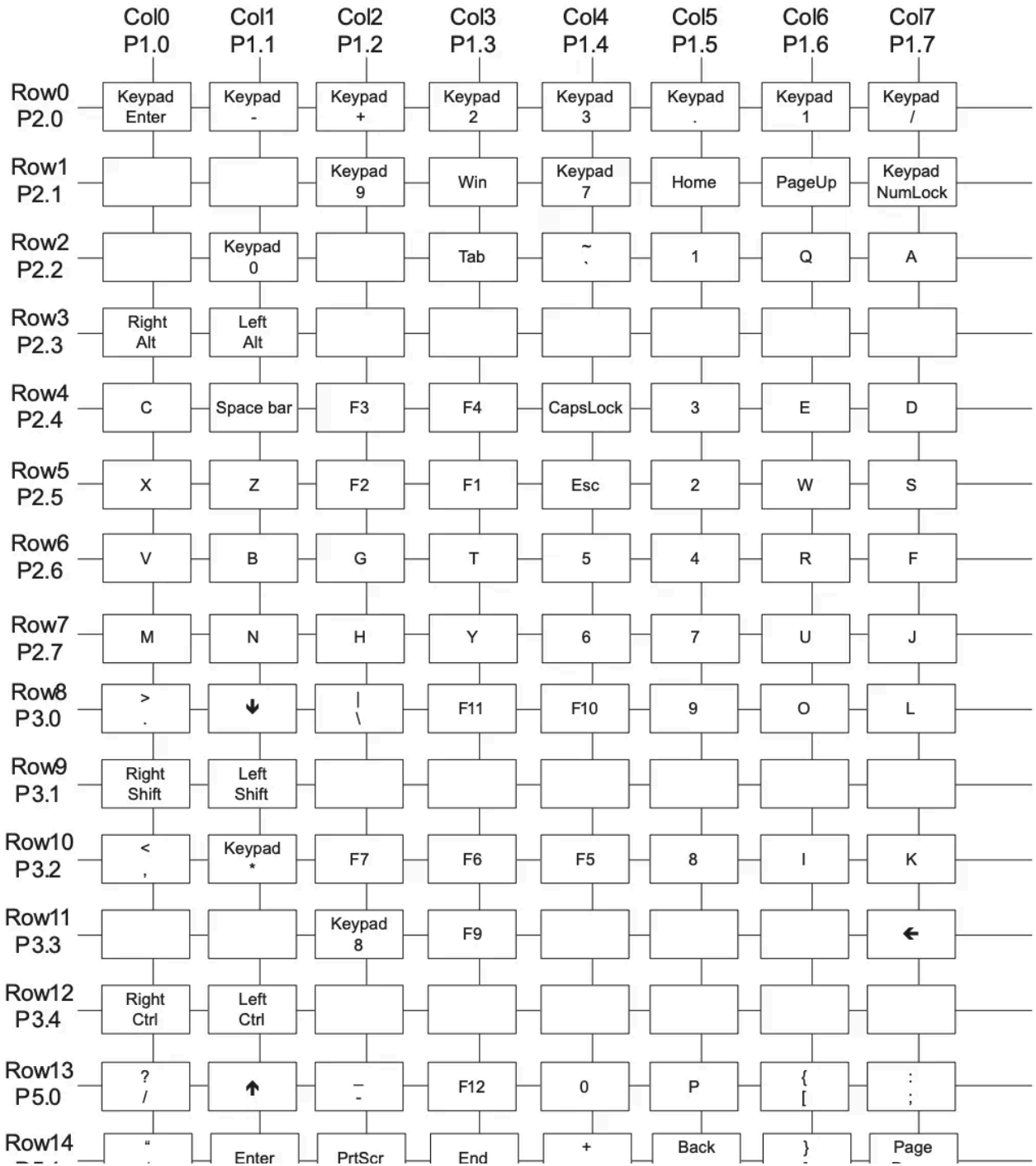
早期的计算机，大部分都是PS2的接口，就是这玩意：

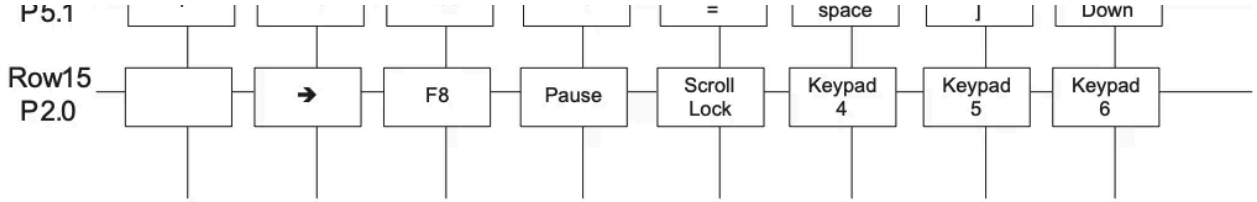
但这种接口插起来不方便，也不通用，近些年USB接口键盘越来越多了，所以咱们就以USB键盘为研究对象。

当你按下键盘按键的瞬间，这个按键位置下的电路“开关”将会被接通，而这样的开关每一个按键下面都有，它们共同组成了一个矩阵：



全局矩阵就是这个样子的：


Col0 P1.0
Col1 P1.1
Col2 P1.2
Col3 P1.3
Col4 P1.4
Col5 P1.5
Col6 P1.6
Col7 P1.7
Row0 P2.0
Keypad Enter
Keypad -
Keypad +
Keypad 2
Keypad 3
Keypad .
Keypad 1
Keypad /
Row1 P2.1
Keypad 9
Win
Keypad 7
Home
PageUp
Keypad NumLock
Row2 P2.2
Keypad 0
Tab
~ `
1
Q
A
Row3 P2.3
Right Alt
Left Alt
Row4 P2.4
C
Space bar
F3
F4
CapsLock
3
E
D
Row5 P2.5
X
Z
F2
F1
Esc
2
W
S
Row6 P2.6
V
B
G
T
5
4
R
F
Row7 P2.7
M
N
H
Y
6
7
U
J
Row8 P3.0
> .
| \
F11
F10
9
O
L
Row9 P3.1
Right Shift
Left Shift
Row10 P3.2
< ,
Keypad *
F7
F6
F5
8
I
K
Row11 P3.3
Keypad 8
F9
Row12 P3.4
Right Ctrl
Left Ctrl
Row13 P5.0
? /
_ -
F12
0
P
{ [
: ;
Row14
Enter
PrtScr
End
+
Back
}
Page



如果你拆开键盘看过，你会发现在键盘的内部有类似下面这样的一个芯片，它负责周期性的扫描电路，检测哪些位置的按键被按下。

当它检测到按键按下事件，将拿到对应键位的键盘扫描码（注意按下和弹起对应不同的扫描码），然后通过USB接口的通信协议，封装一个按键消息传递出去。在这个消息中，包含了你按下/弹起键位的扫描码，如果有多个按键，消息中就会有多个扫描码。

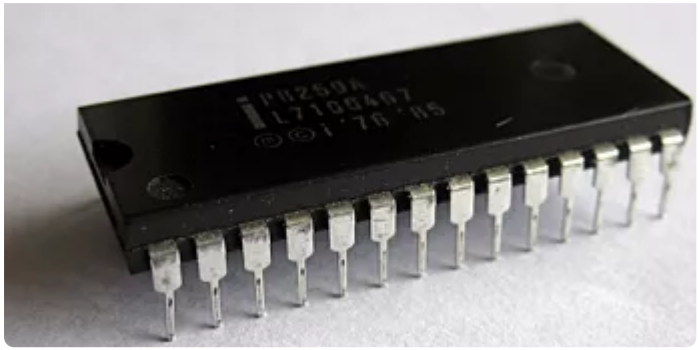
键盘USB连接头连接到了计算机主板上的USB接口，USB接口背后是主板上的USB总线系统，于是这个按键消息顺着键盘的连线，穿过USB接口来到了USB总线上。

而USB总线上，连接了USB控制器芯片，是它在与USB设备进行“通话”。

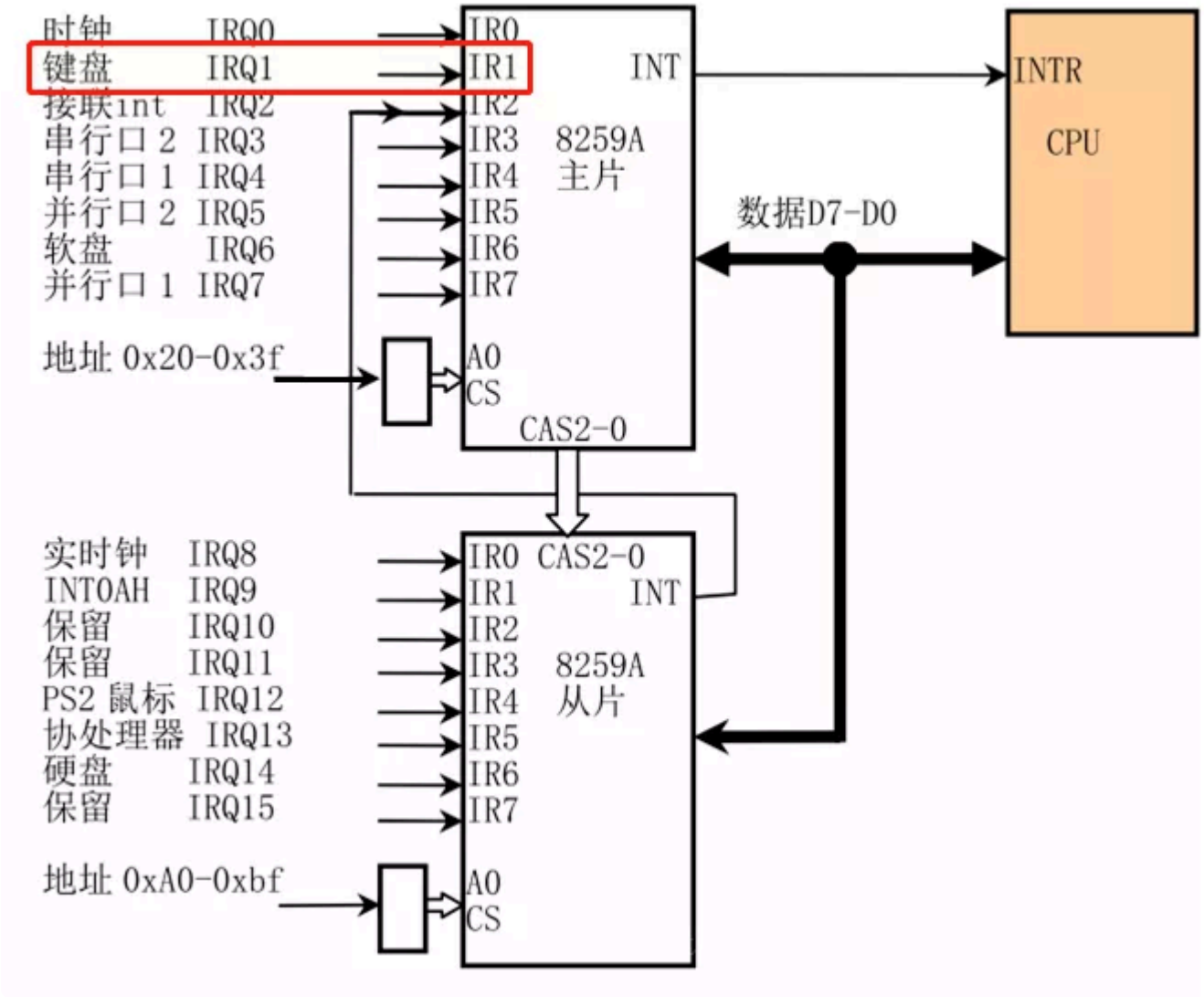
0x02: 高级可编程中断控制器APIC

USB控制器拿到了按键消息后，并不能直接提交给CPU，还要通过另外一个管事儿的投递这个消息，这个管事儿的就是**中断控制器**。

提到中断控制器，你可能在很多地方看到过一个叫8259A的芯片：



然后会告诉你键盘通过IRQ1的中断输入源连接进去：



但现在请忘记它，这玩意已经是上个世纪作古的产物，我保证你拆开你的计算机，一定找不到它。

究其原因，还是因为CPU多核技术的兴起，8259A这个东西早已满足不了时代的需要，换了另外一个更高级的中断控制器，**APIC**。

没错，它的名字就是这么简单直接：**高级可编程中断控制器**。

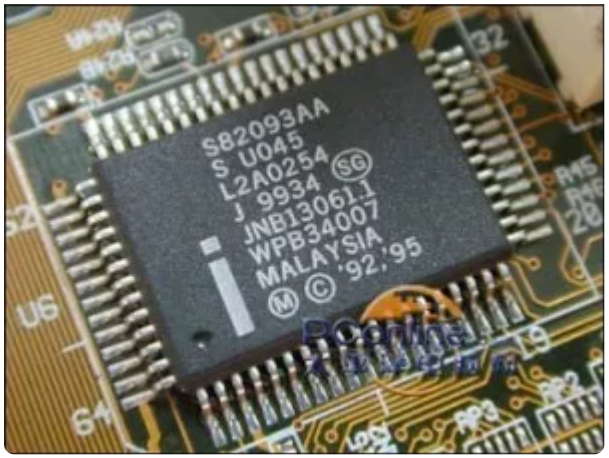
这个更高级的管事儿的到底哪里高级呢？

首先，它不是一块芯片，而是分了两部分：Local APIC和I/O APIC。

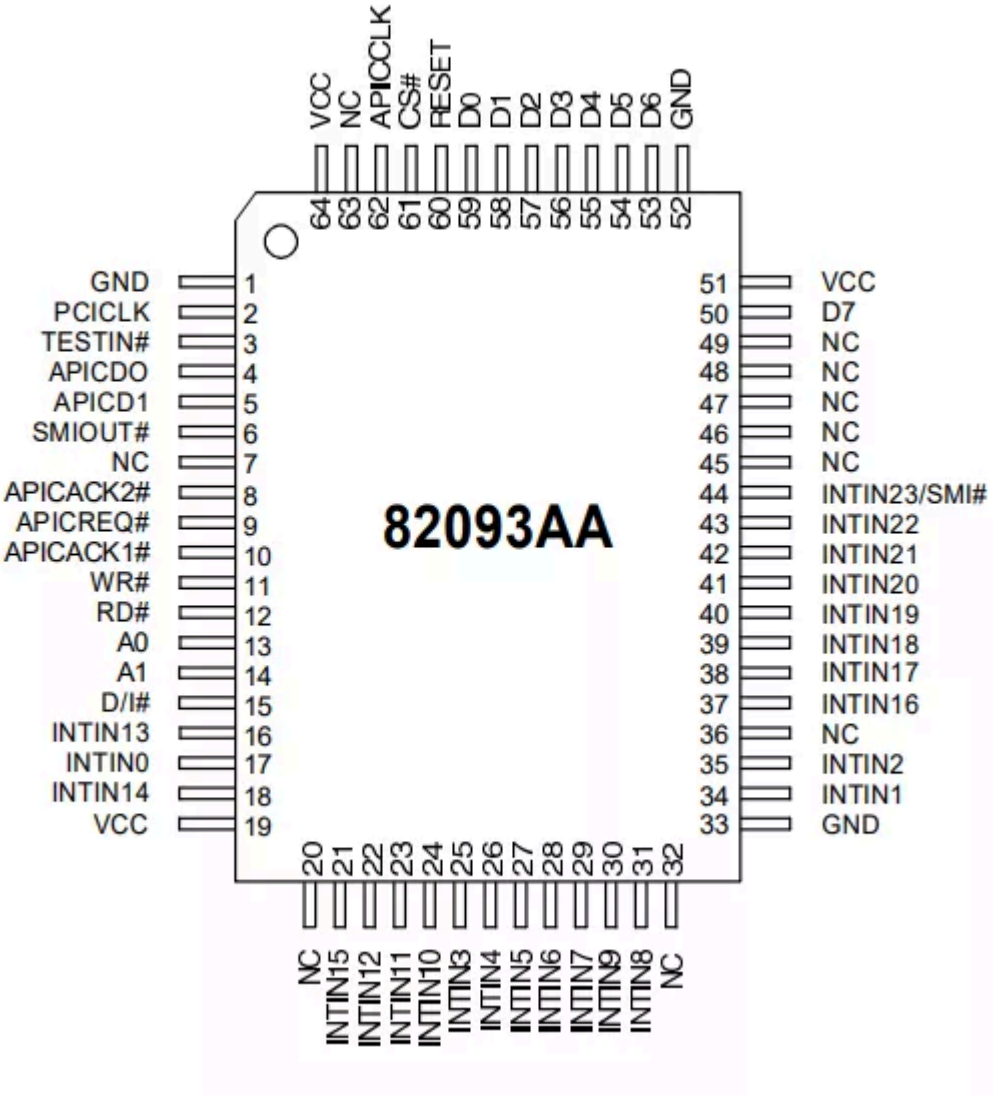
Local APIC像是外包团队一样，入驻到了CPU的每个核心，负责中断每个核。

I/O APIC则独立在CPU外面，接收所有I/O设备的中断源。

来看一个早期的IOAPIC芯片：82093AA



就是它代替了传统的8259A的PIC来总管主板上这些外设的中断信号，这家伙的管脚图长这样：



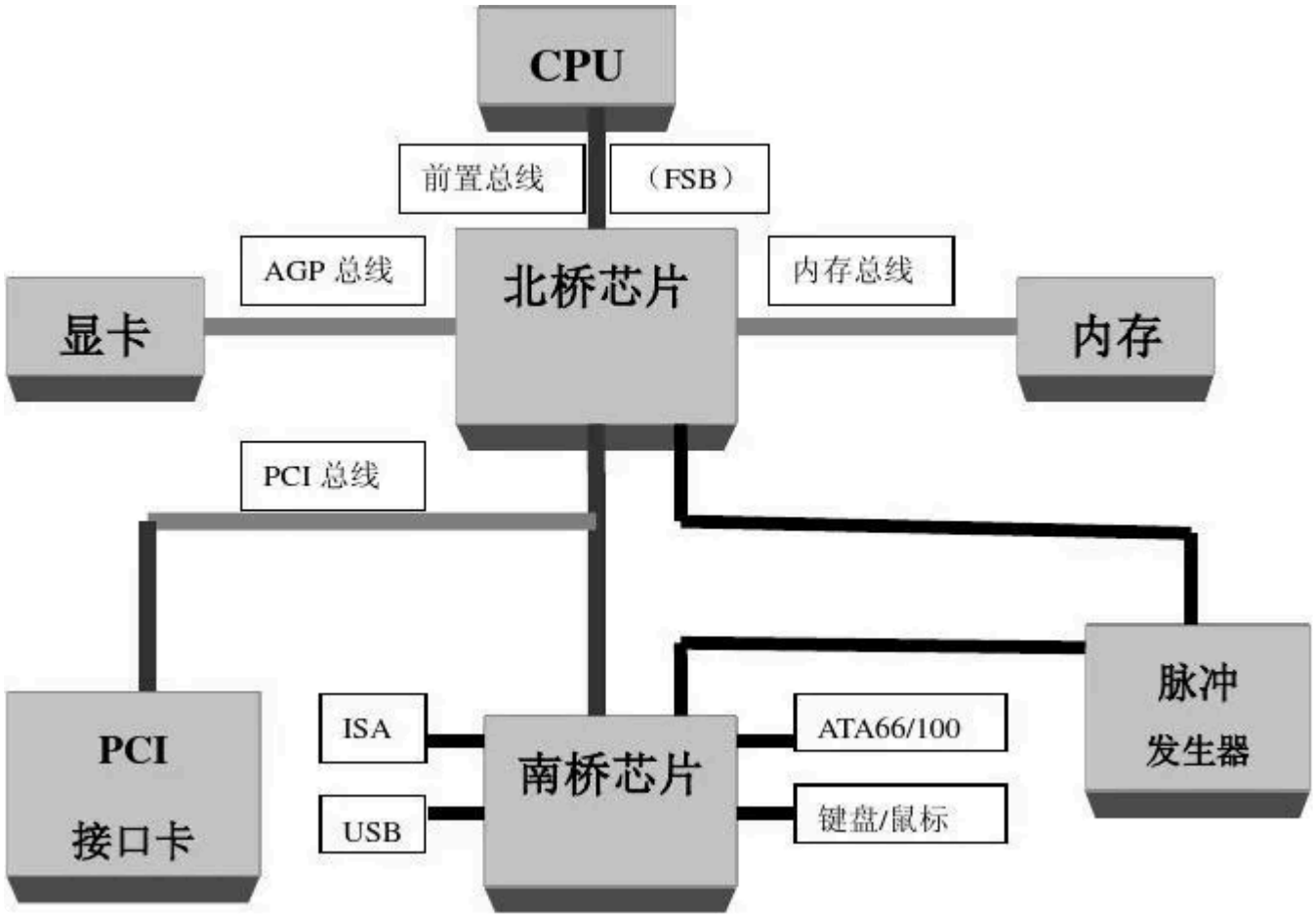
你可以数一下，负责中断源的输入引脚有INTIN0-INTIN23，总共24个，比传统的两块8259A的芯片级联起来的数量还要多。

如果你拆开你的电脑主板，我保证你依然看不到这个叫IOAPIC的芯片。因为这个家伙现在已经被集成到了**南桥**之中了。

啥？南桥是啥？接下来需要补充一点计算机主板的知识了。

0x03: 计算机主板结构

在传统计算机主板上，分为了**CPU+北桥+南桥**的经典架构：



北桥和南桥是主板上除CPU外最重要的2个芯片，所谓南北，是因为在画图位置上，上北下南，因而得名。

北桥联通着CPU，负责连接内存、显卡等高速设备。

南桥联通着北桥，负责连接网卡、硬盘、键盘、鼠标这些低速设备。

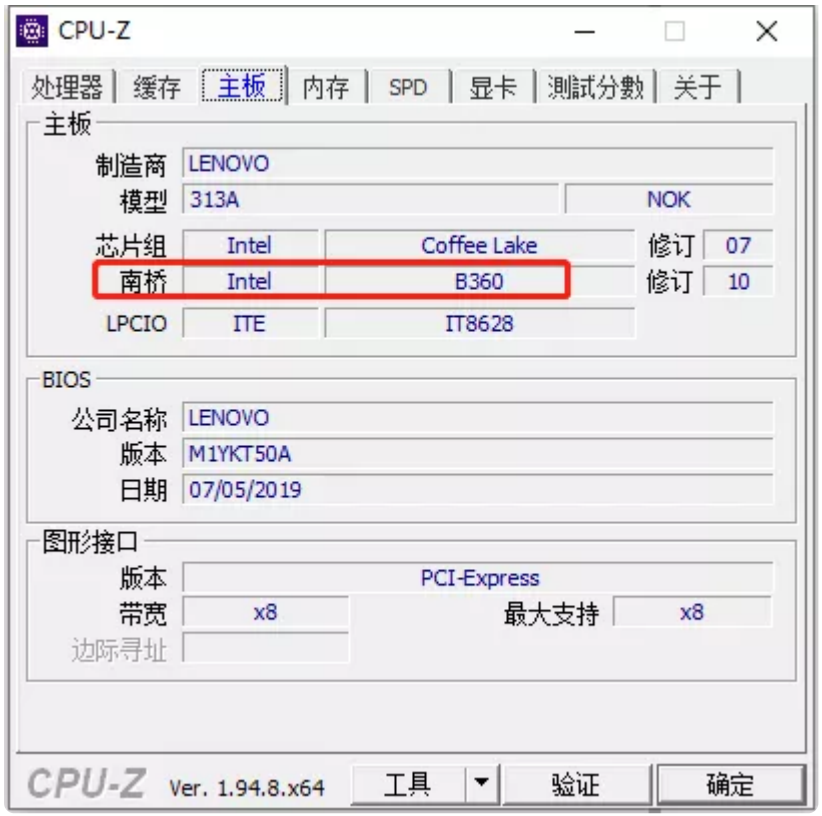
你可以这样理解：CPU是整个主板上的大明星，主板上其他所有设备都要围绕它来转，这明星有两个经纪人，一个负责对接速度快的，一个负责对接速度慢的。

从Intel的酷睿处理器开始（2008年），将北桥芯片的功能集成到了CPU之中，从此主板上就只剩一个南桥了，于是也没有南北之分了，甚至改头换面，换了个名字：**PCH**。

这个叫PCH的家伙可不简单，它现在要对接CPU，还要对接PCI总线、ISA总线上的一堆设备。

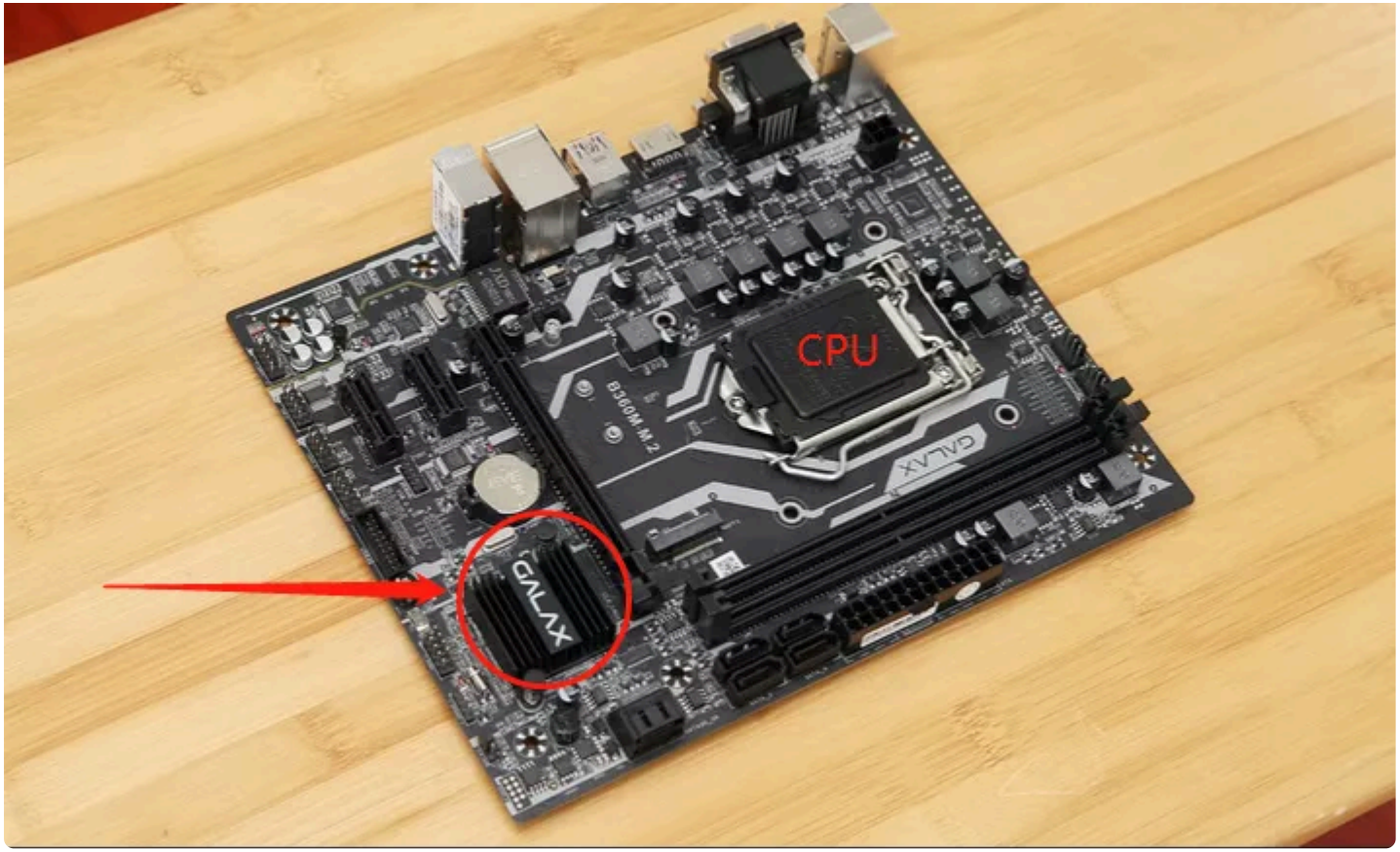
我们的键盘连接到的是USB总线，也是对接到这个PCH芯片。

通过**cpu-z**工具，可以看到自己电脑主板上的PCH芯片型号：



如上图所示，我的这台电脑是B360芯片，你可以在Intel的官网查询到它的详细资料。

那这玩意儿在电脑主板哪个位置呢：



拿掉上面的散热片，这家伙长这样，其貌不扬：

在这个小小的芯片里，就集成有负责跟USB设备进行通信的**USB控制器**，还有前面说的负责中断CPU的**高级可编程中断控制器IOAPIC**，这两个家伙在今天讨论的问题中扮演了关键角色。

USB控制器负责与USB设备通信，它将拿到USB键盘传输过来的那个按键消息包。

0x04: 中断信号的投递

现在USB控制器和APIC已经都集成到了PCH中，内部的结构不得而知，但总体来说，USB控制器拿到按键消息后，然后通过IOAPIC的中断源输入管脚发起通知：老哥，我这有情况，快帮我通知CPU老大。

在IOAPIC的内部，有一个表格PRT，记录了中断分发的配置信息，24个中断源就有24个表项（其实还有一部分保留的）。表格中的每一项叫RTE，每项占据64bit。

来自USB控制器的电信号输入到IOAPIC之后，IOAPIC会根据事先编程配置的信息，通过对应的表项RTE格式化出一条中断消息，然后通过总线系统发出去。

> 在早期，IOAPIC和CPU内部的Local APIC之间有专属的APIC总线来联系，但从奔腾4开始就取消了，使用公共的总线系统来传递中断消息。

消息发出去后，谁来接收呢？

在这个中断消息中，填写有收件人：Local APIC的标识号。

总线系统上的信号通过CPU的针脚传输到了CPU内部，内部所有核的Local APIC都能收到这个中断消息，但只有一个核的Local APIC检测后发现收件人是自己，其他人都会忽略这条消息。

发现收件人是自己的那个Local APIC，开始通知自己所在的这个核有中断请求来了。

CPU的核心一直在不停的执行指令，在每个指令周期的最后，都会去检查一下是不是有中断请求过来，在执行完手头这条指令后，它发现了Local APIC提交的中断请求。

接下来，就是CPU开始来处理这个中断消息的时候了。

0x05: 中断处理

第一个动作，保存执行上下文。

所谓中断，从字面来讲就是中途打断的意思，就好比你正在写着代码，突然有产品来找你增加需求，你被打断了。人倒还好，咱们有记忆能力，跟产品沟通完成后，还能回去接着原来的地方继续写代码。但机器没有记忆思维，在打断去干别的事情之前，必须把原来做的事情保存起来，这样一会儿才能回来继续做剩下的事。

这个保存的过程，就叫执行上下文保存。那保存在哪里呢？

答案就是线程的**栈**。

但是要注意，这里的栈，不是咱们平时看到的那个线程栈，而是另外一个位于内核地址空间的栈。

不管是Windows还是Linux，基本上每个线程在执行的时候都有两个栈，一个用于我们编写的应用程序在用户态模式下执行代码时使用，叫**用户栈**，另一个用于程序因为系统调用、异常、中断等情况进入内核模式下执行的时候使用，叫**内核栈**，相比用户栈，内核栈的空间要小得多。

> 注意：也不是每个线程都有两个栈，有一些操作系统的纯内核线程就只有内核栈，没有用户栈。

发生中断时，CPU将自动将当前执行的上下文保存在内核栈的顶部，所谓上下文，其实就是一堆寄存器的值。注意这个动作不是操作系统软件完成的，而是CPU内部的硬件电路自动完成。

第二个动作，执行中断处理函数

保存完上下文，接着就要去处理中断了。怎么处理，那就是操作系统的工作了。

CPU的每一个核，都有一个中断描述符表IDT，位于内存之中，这个表有256项，每一个表项都记录了一个处理函数的地址。每个核的内部还有一个叫IDTR的寄存器，指向了这个表。

> 要注意，IDT虽然是叫做中断描述符表，但里面的256项内容却不全是用来记录中断处理函数的，还有异常、陷阱（软中断）、任务这些。

表格中的处理函数地址，是操作系统在启动之初就安排好了，这其中就有我们的键盘中断处理函数。

当中断发生时，CPU将根据中断向量号，从IDTR寄存器指向的表格中，取出索引是向量号的那一个表项，跳转到里面记录的函数地址，开始执行代码，这个过程依然是CPU的硬件电路完成的。

那这个中断向量号从哪儿来的呢？

答案是在IOAPIC发来的那条消息中，除了收件人Local APIC的标识，还有处理中断所需要的**中断向量号**。

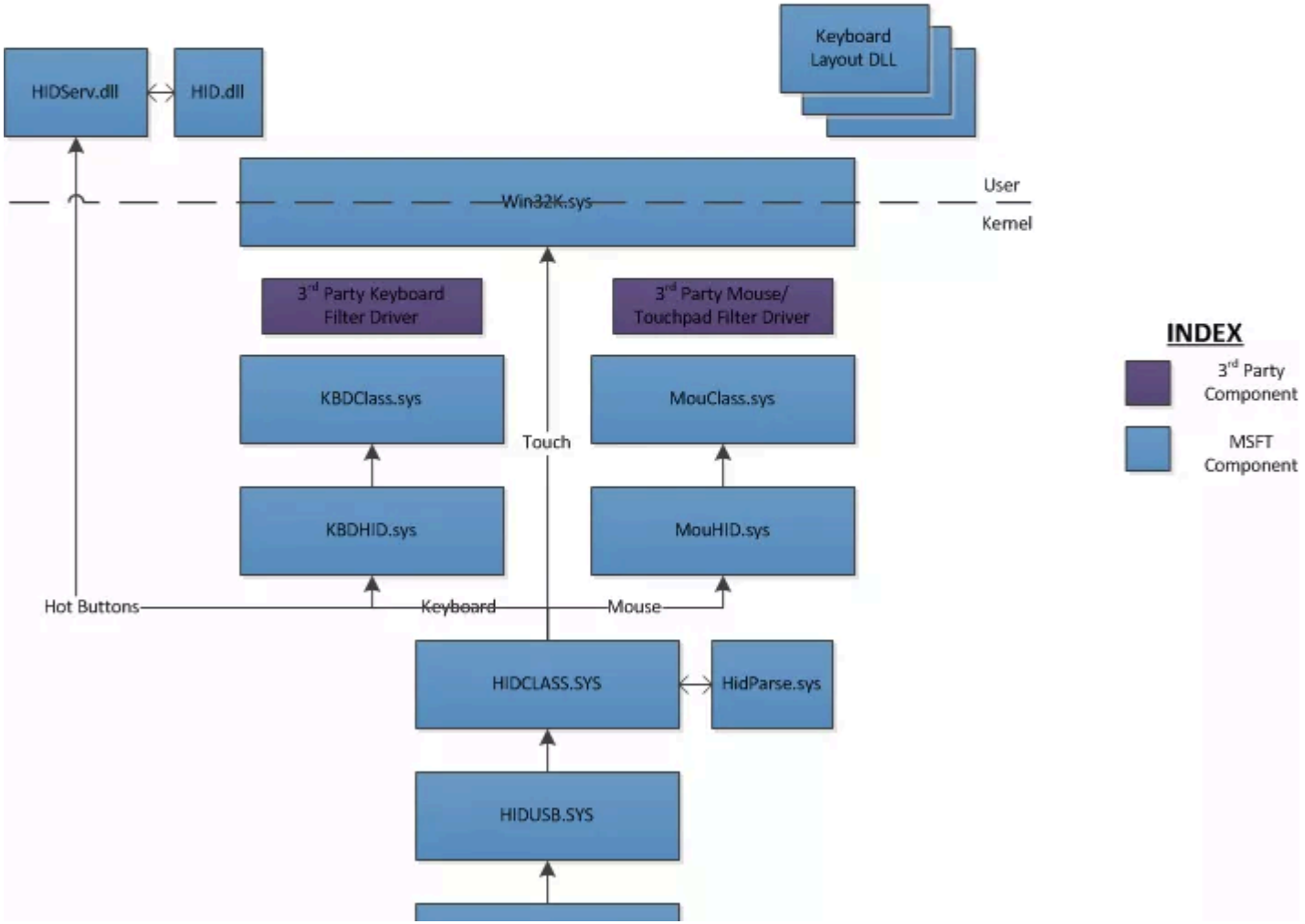
再往前追溯，这个中断向量号其实是配置在前面说的IOAPIC内部的那个叫PRT的表格中的，操作系统启动之初一项重要的工作就是对APIC进行编程（所谓编程其实就是写他们内部的这些配置表，也叫寄存器），设定好每一个中断源对应的中断向量号是多少，这样24个中断源与对应的中断向量号之间的映射关系就被确立起来了。

> 除了给中断源分配向量号，操作系统还有一项工作就是指定哪些核来处理哪些中断。我之前写过一篇趣文故事就是讲的这部分知识：CPU明明8个核，网卡为啥拼命折腾一号核？

接下来就是操作系统（准确来说是操作系统中的设备驱动程序）开始来处理这个中断消息了。

具体的驱动处理部分就不详述了，不同版本的系统处理略有不同，在微软的官网上，可以找到这么一张图，针对USB输入设备（键盘、鼠标）的驱动处理栈结构图：

HID USB Stack Diagram



总体来说，Windows操作系统介入中断处理后，经过一系列驱动程序（USB、HID等）的处理后，进行扫描码的转换，然后把按键的消息最终投递到了一个叫**Win32k.sys**的家伙那里。

0x06: 操作系统介入

让我们把视线从硬件部分转移到操作系统上来。Windows是一个基于视窗的图形化的操作系统，绝大部分程序都是基于**消息**驱动。这一点，做过Windows客户端开发的朋友应该不会陌生。

Windows上有图形窗口的程序形态各异，功能千差万别，但它们都有一个共同之处：**基于消息驱动**。

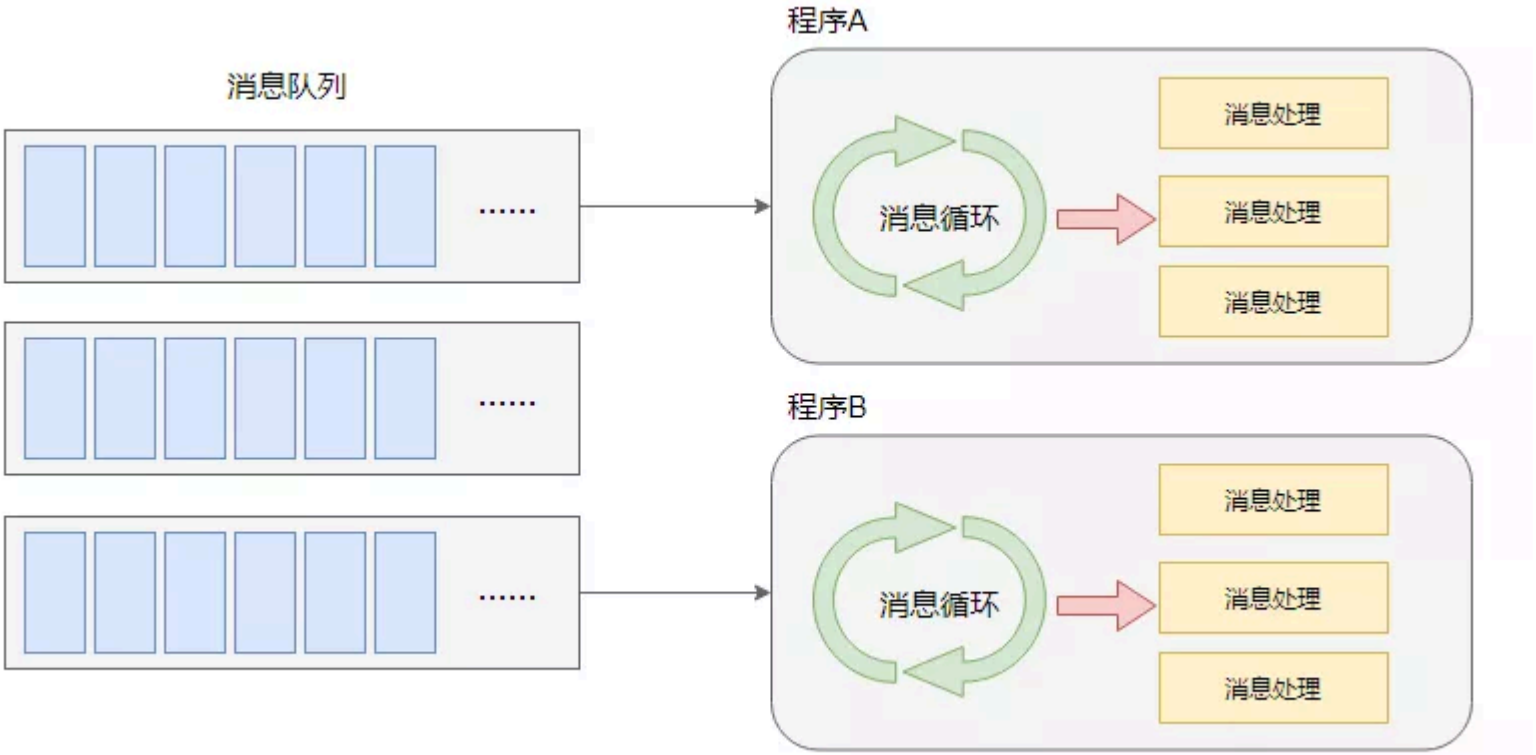
这些消息可能来自于键盘、鼠标、其他进程甚至网络，一个典型的Windows程序，其主线程一定有一个下面的消息循环：

```
while(GetMessage()) {
  TranslateMessage();
  DispatchMessage();
}
```

主线程不断调用**GetMessage()** 获取消息，然后分发处理，如果没有消息，GetMessage将会阻塞。

这个GetMessage()是从哪里获取消息呢？

答案是**消息队列**。



每一个具有图形可视化窗口的程序都有一个消息队列，维护在内核空间，GetMessage()就是从这里源源不断的取出消息来处理。你的每一次键盘按键，每一次鼠标点击，每一次鼠标移动，都会产生消息被投放到这个队列中，等待取出处理。

那么问题又来了，你在键盘按下后产生的消息，是被谁投递到了这里呢？还有，每一个窗口程序都有消息队列，那我按下的键盘消息，到底该被投递给谁呢？

答案正是在前面说的那个叫**Win32k.sys**的家伙之中！这是Windows内核实现图形用户界面一个重要的模块，里面有一个内核线程在专门负责干这事——不断从键盘驱动获取按键事件，然后封装成消息，再结合当前

桌面激活的窗口，定位到对应的消息队列，把这个消息给投递过去。

于是，应用程序的消息循环中，GetMessage()函数将会拿到一个代表键盘按键被按下的**WM_KEYDOWN**消息。

再回过头去看下那个消息循环，拿到消息后会有一个“转换动作”：TranslateMessage()。这个函数将对按键消息进行一次翻译，翻译成一个**WM_CHAR**消息，表示有字符输入消息来了，这个消息的一个字段会标识输入的是6这个字符。

最终，应用程序终于收到了一个参数是6的**WM_CHAR**消息，知道用户按了一个6，接下来就是在显示器上把它给显示出来了。

总结

文章有点长，现在来总结梳理下，按下键盘上的6以后，计算机到底发生了什么。

1. 按下按键的瞬间，按键所在位置的开关被接通，随后被键盘内部芯片检测到，得到按键的扫描码。
2. 键盘控制器芯片发送一个按键消息，通过USB连接口传输到计算机主板上的USB控制器。
3. USB控制器被集成到了主板上的PCH之中（以前的南桥芯片），一同被集成的还有负责管理所有外设中断源的中断控制器：IOAPIC。
4. USB以中断请求形式通知IOAPIC。
5. IOAPIC根据对应中断源的配置，生成一条中断消息，通过系统总线发送出去。
6. 这条消息之中有收件人ID，所有核的Local APIC拿到以后比对收件人是否是自己，不是自己则丢弃。
7. Local APIC收到中断消息后，向所在的CPU核心发起中断
8. CPU执行完手头的指令后就会转而处理中断，先进行上下文保存，然后调取IDT表中对应中断向量号的处理函数执行。
9. 中断处理函数是USB驱动程序，它将读取键盘按键消息的扫描码，并转换成程序处理所需的编码。
10. 操作系统内核线程从USB驱动程序拿到输入消息，并分发到对应程序的消息队列。
11. 应用程序从自己的消息队列中获取到键盘被按下的消息。