公众号【IOT物联网小镇】

> 作 者：道哥，10+年的嵌入式开发老兵。
>
> 公众号：【IOT物联网小镇】，专注于：C/C++、Linux操作系统、应用程序设计、物联网、单片机和嵌入式开发等领域。 公众号回复【书籍】，获取 Linux、嵌入式领域经典书籍。
>
> 转 载：欢迎转载文章，转载需注明出处。

0xFFFF:0x0000
0xF000:0xE05B
0x0000:0x7C00

在第一篇文章中，我们就提到，现代操作系统是从最古老的 8086 系统一步一步发展而来的。

处理器厂商为了向后兼容，很多底层相关的原理都是一样的(如果不兼容，就会丢弃市场份额)。

特别是从系统上电之后，一直到操作系统中第一个进程(Linux 下就是 init 进程)运行起来，这其中经历了 BIOS、引导程序、操作系统这三元大将的接力跑。

今天，我们从几个特殊的地址的角度，来从宏观节点上看一下系统的启动过程。

0xFFFF:0x0000

这个地址，是处理器上电之后的第一个重要的物理地址。

从地址的书写形式上，就可以看出这是 8086 系统中实模式下的段寻址方式：段地址 * 16 + 偏移量。

> 段地址：0xFFFF
>
> 偏移地址：0x0000
>
> 计算得到物理地址：0xFFFF0

当处理器的 reset 引脚被触发后，处理器首先进行硬件初始化，也就是把处理器内部的每个寄存器都设置为一个初始的默认状态：

> 把段寄存器 cs 设置为 0xFFFF，指令寄存器 ip 设置为 0x0000;
>
> 把其它的所有寄存器设置为 0x0000;

当所有的初始化完成之后，CPU 就开始执行第一条指令。

之前说过，CPU 是很傻、很单纯的，它只知道去 cs:ip 所指向的地址处，取出一条指令，执行完之后，再取出下一条指令继续执行。。。

> 每一条指令的第一个字节都是操作码，CPU 根据操作码，能够知道当前指令的字节长度，并把 ip 寄存器指向下一条指令。

既然硬件初始化时，已经把 cs 初始化为 0xFFFF，把 ip 初始化为 0x0000，经过段寻址的公式计算之后，就得到了物理地址：0xFFFF0，也就是说，CPU 执行的第一条指令位于物理地址 0xFFFF0 这个地方。

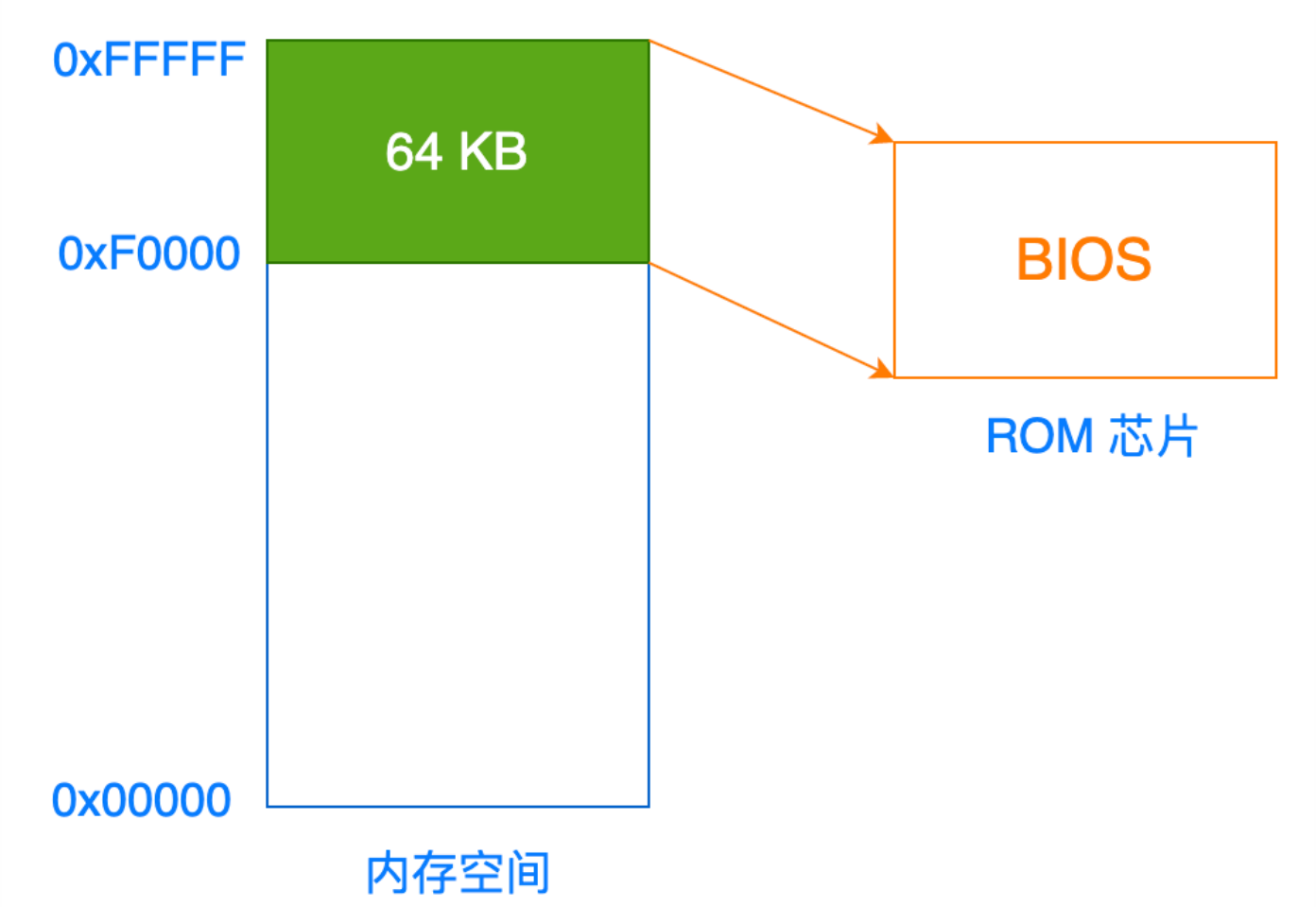
那么，这个物理地址中，存放着什么指令呢？

首先来复习一下地址范围的相关知识：

> 8086 处理器有 20 根地址线，寻址范围是：0x00000 ~ 0xFFFFF，最大就是 1 MB。
>
> 但是 8086 的处理器是 16 位的，寄存器最大表示的范围是 0xFFFF，也就是 64 KB。
>
> 采用【段基址:偏移量】来表示一个段时，这个段的最大偏移范围就是 64 KB。

我们再回到系统的启动流程。

在上电之后，硬件会把一个 ROM 芯片，映射到内存地址空间的最高地址空间，也即是 1 MB 的位置，如图：



ROM 芯片中存放的就是 BIOS 代码，称作：基本输入输出系统(Basic Input/Output System)。

此时，cs:ip 计算得到的物理地址为 0xFFFF0，正好落在映射到 ROM 的这块内存空间。

因此，从这个地址中获取到指令，其实就是从 ROM 中读取的。

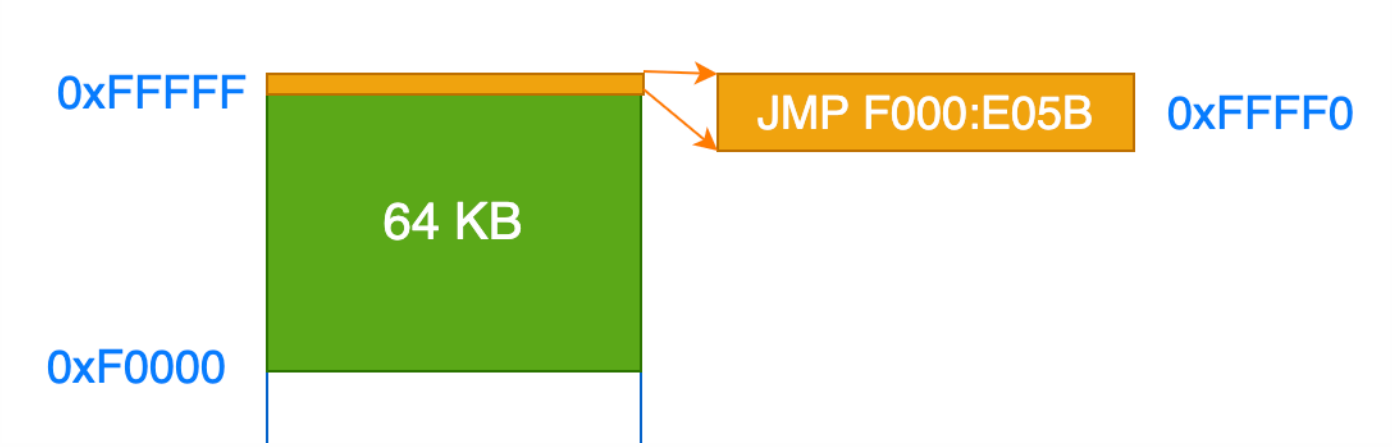
> 所谓的映射：就是访问某个地址空间中的内容时，就会自动定位到被映射的目标物理设备中进行访问，这是由硬件来保证的。

CPU 在执行指令的时候，ip 寄存器是递增的，也就是说会从低地址到高地址，依次执行每一条指令。

但是此时第一条指令的地址就是 0xFFFF0，已经快接近 1 MB 地址空间的顶端了，只有 16 个字节的地址空间。

如果执行到顶端，溢出之后，就会回绕到最低地址 0x00000。

因此，在这个第一条指令的位置处，是一条跳转指令：



跳转目标是 0xF000:0xE05B，计算得到物理地址 0xFE05B，可以看到同样是落在映射到 ROM 的地址空间中(好像是废话：此时只能执行 BIOS 中的代码)。

0xF000:0xE05B

这个地址处的代码，才是 BIOS 真正开始执行的地方。BIOS 所做的事情包括：

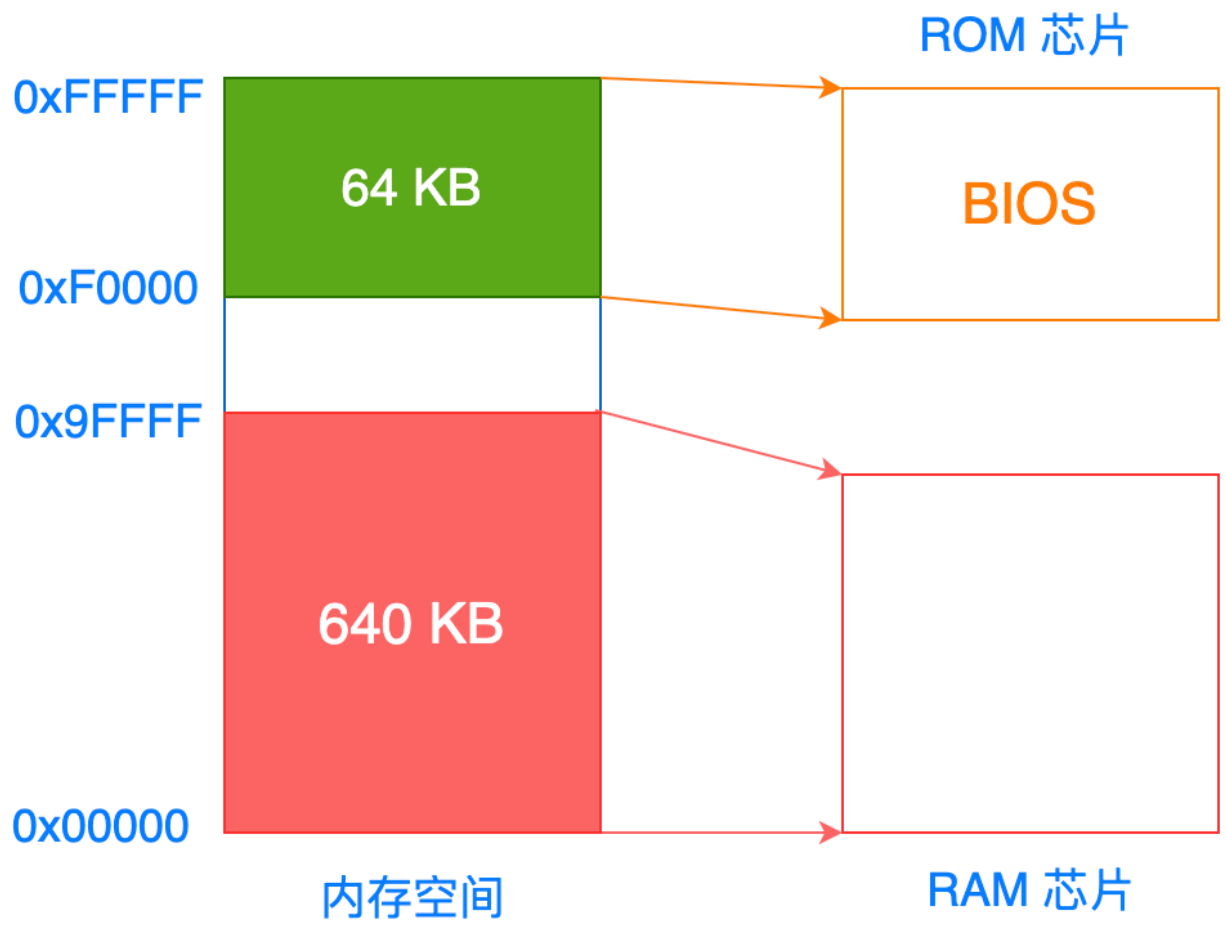
> 侦测硬件设备：系统中有哪些硬件设备，工作状态是什么；
>
> 对硬件设备进行初始化：比如最初始的中断向量表；
>
> 侦测操作系统启动设备：选择好一个系统盘之后，把系统盘中主引导扇区中的引导程序读取到内存中；

在 BIOS 的最后一个步骤中，它把引导程序读取到内存中 0x0000:0x7C00 地址处，计算得到物理地址就是：0x07C00。

这个地址的内存空间，被硬件映射到 RAM 芯片中。

具体的说就是，硬件把内存空间 0x00000 ~ 0x9FFFF 映射到随机存储器中，一共是 640 KB 的空间。

注意：虽然地址空间有 640 KB 这么大，但是实际的 RAM 大小可能只有可怜的 32 KB，因此实际可用的空间取决于物理芯片。



> 中间空着的那块地址空间，映射到一些外设。

0x0000:0x7C00

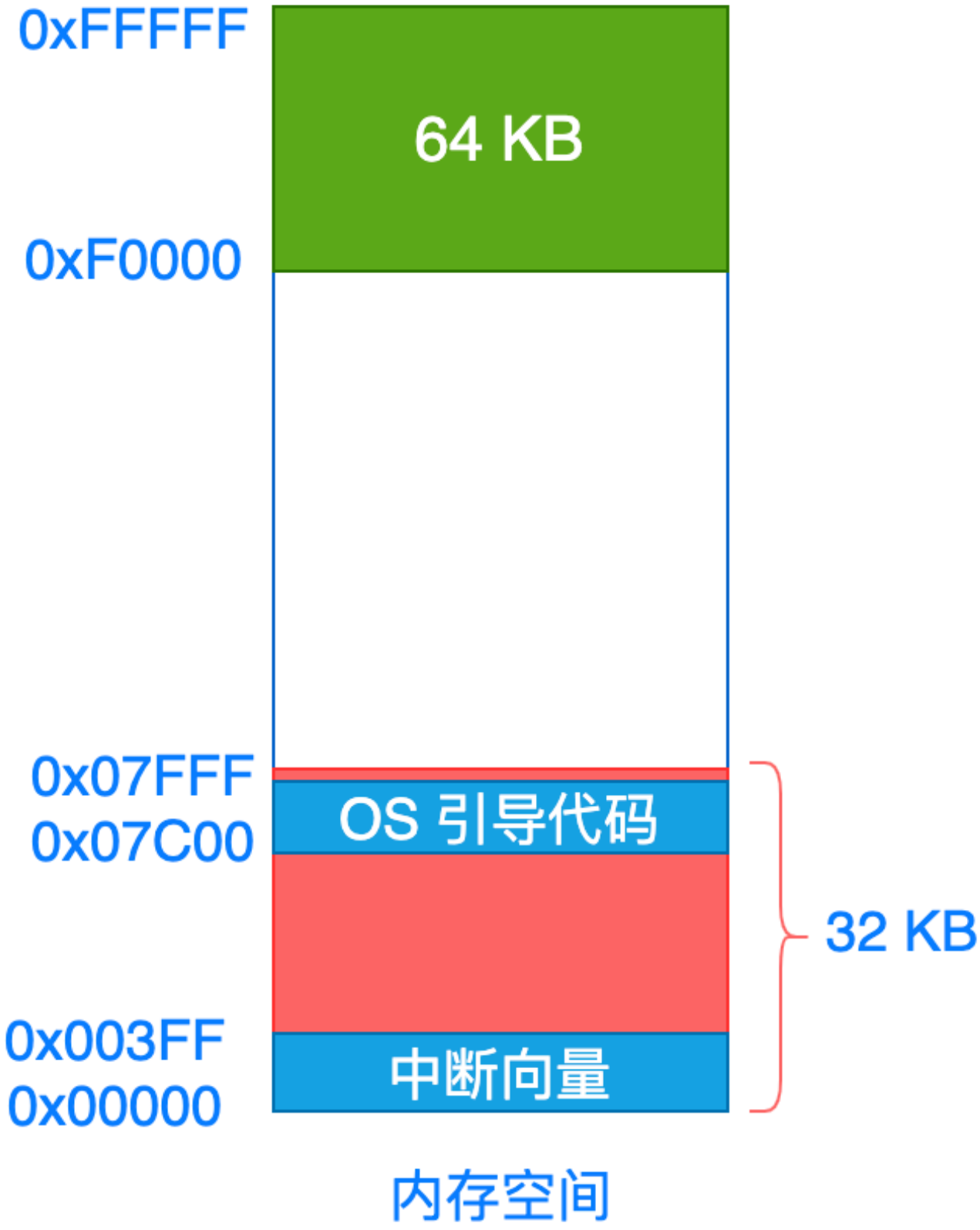
这个地址，就是操作系统的引导代码被读取到内存中的地方。

在内存地址的刚开始位置(0x00000 ~ 0x003FF)，存放着中断向量表。

可以看到：操作系统引导代码并没有从中断向量表之后的 `0x00400` 开始存放，而是被放在了 `0x07C00` 这个地方：

至于为什么要这么放置，有很多的说法，比较靠谱的解释是这样的：

假如实际的 RAM 芯片只有 32 KB(不要用现代的眼光来看，在 N久 之前，RAM 还是非常的珍贵)，那么内存布局就是这样：



> 在此也鄙视一下现在很多的应用软件，动不动就占用那么多的内存，都以为整个电脑只为它一家软件服务的？！

可以看到，引导代码几乎位于 RAM 的顶端了，这样的话，从中断向量开始的 `0x00400`，一直到引导代码的 `0x07C00`，这块地址空间就是连续的一整块，可以被操作系统更方便的操作。

另外，把引导代码放在 RAM 的高地址处，还有一个好处:

当引导代码最终把接力棒交给操作系统之后，引导代码就没有任何用处了。

因此，操作系统就可以直接把引导代码所在的地址空间中内容，全部抹掉，为自己所用！

------ End ------

推荐阅读

【1】C语言指针-从底层原理到花式技巧，用图文和代码帮你讲解透彻
【2】一步步分析-如何用C实现面向对象编程
【3】原来gdb的底层调试原理这么简单
【4】内联汇编很可怕吗？看完这篇文章，终结它！

其他系列专辑：精选文章、C语言、Linux操作系统、应用程序设计、物联网

公众号【IOT物联网小镇】

C/C++、物联网、嵌入式、Lua语言

Linux 操作系统、应用程序开发设计

扫码关注公众号

道哥 个人微信

喜欢请分享，满意点个赞，最后点在看。