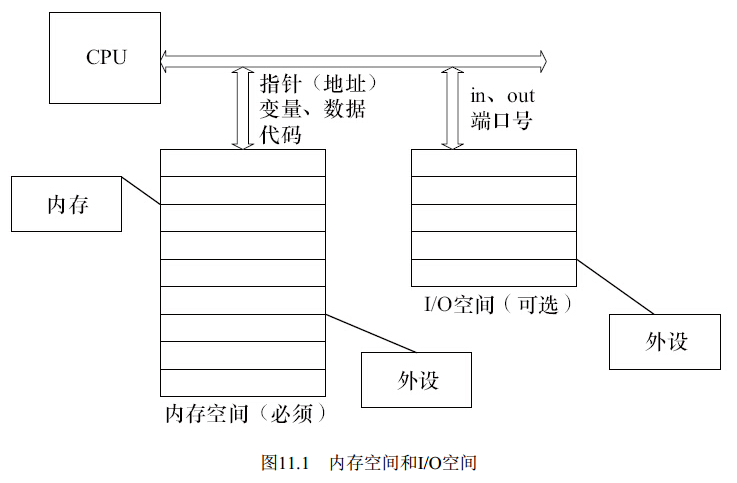
# 概述

## 内存空间与IO空间

x86存在着IO空间的概念，需要通过特定指令in、out来进行访问。而很多其他处理器如arm不存在IO空间，所有的地址统一编址。



## I/O端口与I/O内存

设备通常会提供一组寄存器来控制设备、 读写设备和获取设备状态， 即控制寄存器、 数据寄存器和状态寄存器。 这些寄存器可能位于I/O空间中， 也可能位于内存空间中。 当**位于I/O空间时， 通常被称为I/O端口**； **当位于内存空间时， 对应的内存空间被称为I/O内存**。

## MMU

高性能处理器一般会提供一个内存管理单元（MMU） ， 该单元辅助操作系统进行内存管理， 提供虚拟地址和物理地址的映射、 内存访问权限保护和Cache缓存控制等硬件支持。 操作系统内核借助MMU可以让用户感觉到程序好像可以使用非常大的内存空间， 从而使得编程人员在写程序时不用考虑计算机中物理内存的实际容量。

为了理解基本的MMU操作原理， 需先明晰几个概念。

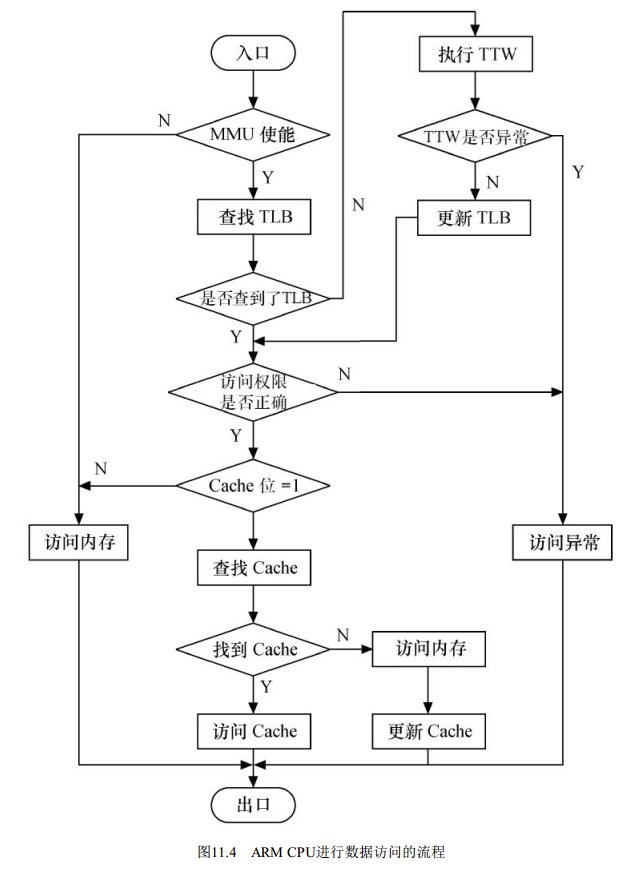
1） TLB（Translation Lookaside Buffer） ： 即转换旁路缓存， TLB是MMU的核心部件， 它缓存少量的虚拟地址与

物理地址的转换关系， 是转换表的Cache， 因此也经常被称为“快表”。

2） TTW（Translation Table walk） ： 即转换表漫游， 当TLB中没有缓冲对应的地址转换关系时， 需要通过对内

存中转换表（大多数处理器的转换表为多级页表， 如图11.2所示） 的访问来获得虚拟地址和物理地址的对应关

系。 TTW成功后， 结果应写入TLB中。



# 方法一：Hard code

适用于没有MMU的情况。

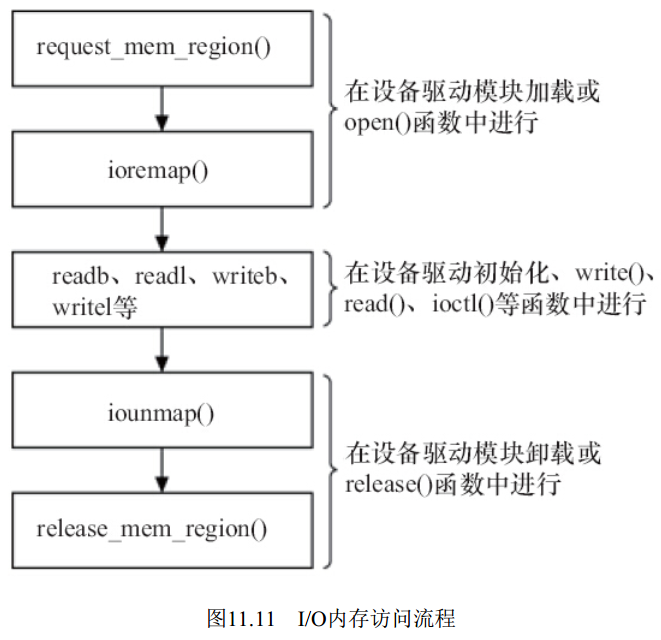
unsigned char \*p = (unsigned char \*)0xF000FF00;

\*p=11;

以上程序的意义在x86架构是在绝对地址0xF0000+0xFF00（186处理器使用16位段地址和16位偏移地址） 中写入11。

而在ARM、 PowerPC等未采用段地址的处理器中， p指向的内存空间就是0xF000FF00， 而\*p=11就是在该地址写入11。

# 方法二：ioremap



注：readb/readl等函数也可以直接写hard code 读写ioremap返回的地址。

# 方法三：regmap

<http://blog.csdn.net/hellowxwworld/article/details/10737569>

<http://blog.csdn.net/scarecrow_byr/article/details/48625085>

<http://blog.csdn.net/kris_fei/article/details/72842119>

regmap是基于ioremap的。进行了一些封装。可以参考上面的连接了解为啥需要regmap以及他的作用。底层的读写仍然使用的是readb、readw等函数。