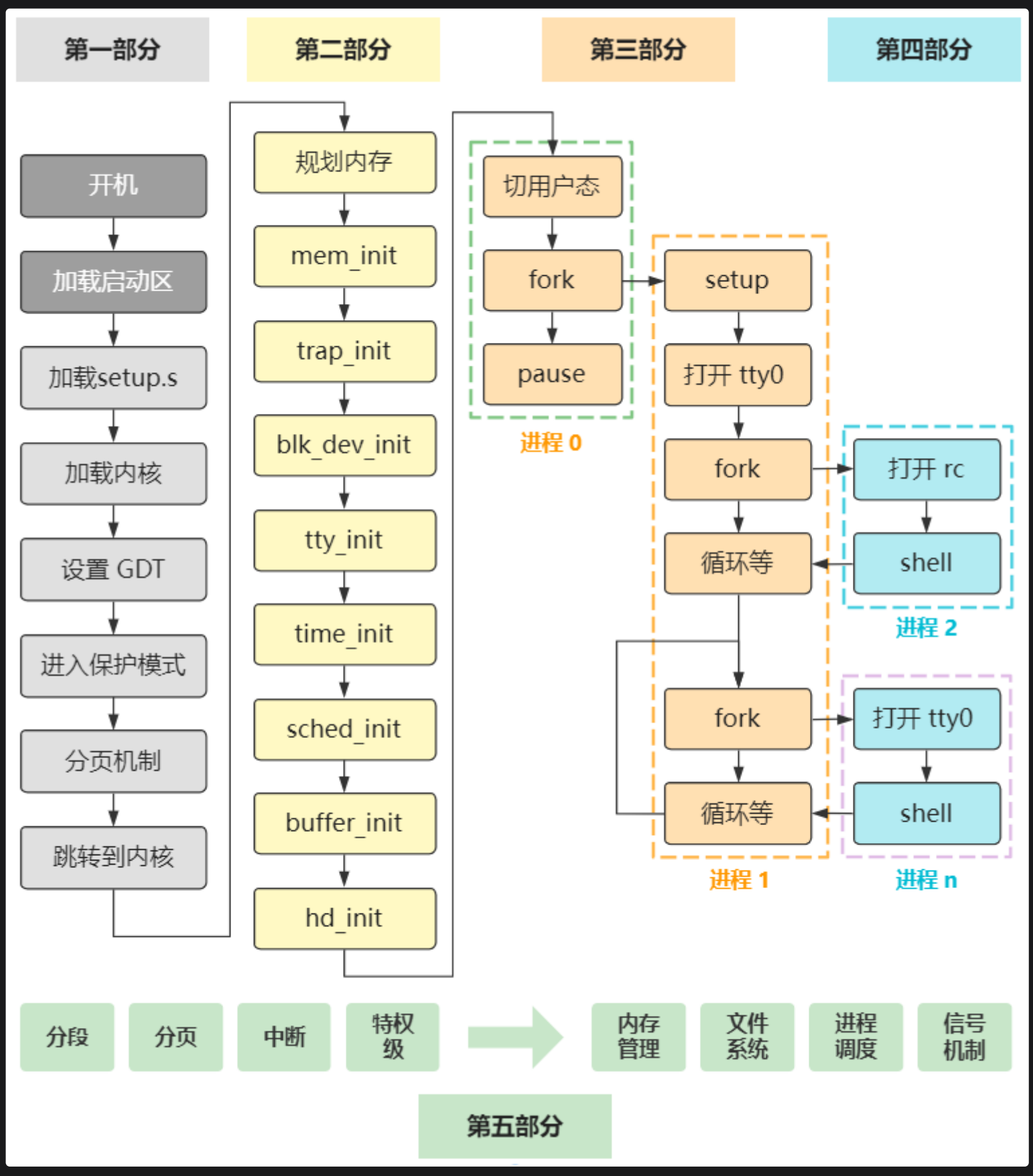
第五回

新读者看这里，老读者直接跳过。

本系列会以一个读小说的心态，从开机启动后的代码执行顺序，带着大家阅读和赏析 Linux 0.11 全部核心代码，了解操作系统的技术细节和设计思想。



你会跟着我一起，看着一个操作系统从啥都没有开始，一步一步最终实现它复杂又精巧的设计，读完这个系列后希望你能发出感叹，原来操作系统源码就是这破玩意。

以下是**已发布文章**的列表，详细了解本系列可以先从开篇词看起。

开篇词

第一回 | 最开始的两行代码

第二回 | 自己给自己挪个地儿

第三回 | 做好最最基础的准备工作

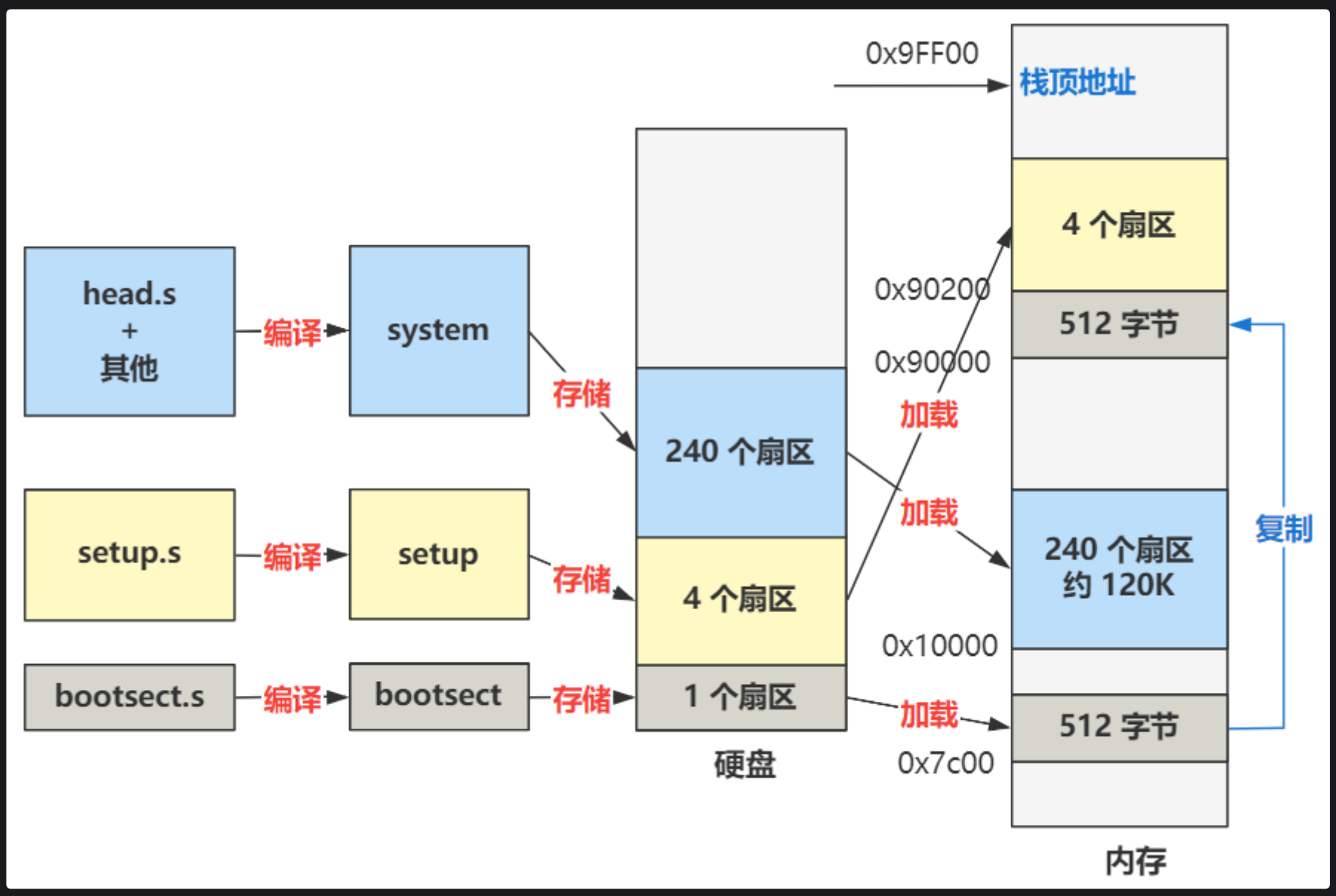
第四回 | 把自己在硬盘里的其他部分也放到内存来

本系列的 GitHub 地址如下（文末阅读原文可直接跳转）

https://github.com/sunym1993/flash-linux0.11-talk

------- **正文开始** -------

书接上回，上回书咱们说到，操作系统已经完成了各种从硬盘到内存的加载，以及内存到内存的复制。



至此，整个 **bootsect.s** 的使命就完成了，也是我们品读完的第一个操作系统源码文件。之后便跳转到了 **0x90200** 这个位置开始执行，这个位置处的代码就是位于 **setup.s** 的开头，我们接着来看。

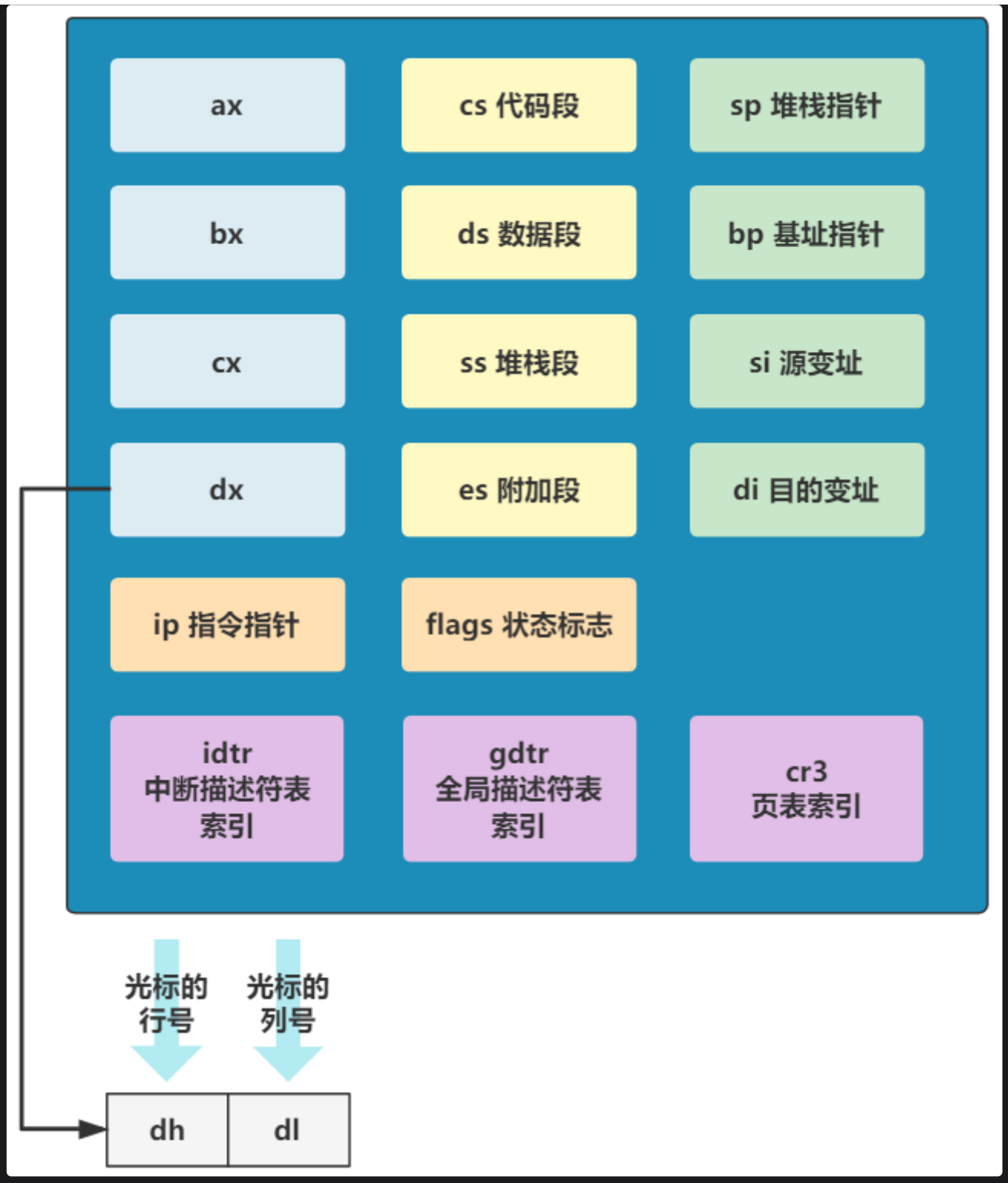
```
start:  mov ax,#0x9000  ; this is done in bootsect already, but...  mov ds,ax  mov ah,#0x03  ; read cursor pos  xor bh,bh  int 0x10  ; save it in known place, con_init fetches  mov [0],dx  ; it from 0x90000.
```

又有个 **int** 指令。

前面的文章好好看过的话，一下就能猜出它要干嘛。还记不记得之前有个 **int 0x13** 表示触发 BIOS 提供的**读磁盘**中断程序？这个 **int 0x10** 也是一样的，它也是触发 BIOS 提供的**显示服务**中断处理程序，而 **ah** 寄存器被赋值为 **0x03** 表示显示服务里具体的**读取光标位置功能**。

具体 BIOS 提供了哪些中断服务，如何去调用和获取返回值，请大家自行寻找资料，这里只说结果。

这个 int 0x10 中断程序执行完毕并返回时，**dx** 寄存器里的值表示**光标的位置**，具体说来其高八位 dh 存储了**行号**，低八位 dl 存储了列号。



这里说明一下：计算机在加电自检后会自动初始化到文字模式，在这种模式下，一屏幕可以显示 25 行，每行 80 个字符，也就是 80 列。

那下一步 **mov [0],dx** 就是把这个光标位置存储在 [0] 这个内存地址处。注意，前面我们说过，这个内存地址仅仅是偏移地址，还需要加上 ds 这个寄存器里存储的段基址，最终的内存地址是在 **0x90000** 处，这里存放着光标的位置，以便之后在初始化控制台的时候用到。

所以从这里也可以看出，这和我们平时调用一个方法没什么区别，只不过这里的**寄存器**的用法相当于**入参和返回值**，这里的 **0x10 中断号**相当于**方法名**。

这里又应了之前说的一句话，操作系统内核的最开始也处处都是 BIOS 的调包侠，有现成的就用呗。

再接下来的几行代码，都是和刚刚一样的逻辑，调用一个 BIOS 中断获取点什么信息，然后存储在内存中某个位置，我们迅速浏览一下就好咯。

```
比如获取内存信息。; Get memory size (extended mem, kB)  mov ah,#0x88  int 0x15  mov [2],ax 获取显卡显示模式。; Get video-card data:  mov ah,#0x0f  int 0x10  mov [4],bx  ; bh = display page  mov [6],ax  ; al = video mode, ah = window width 检查显示方式并取参数; check for EGA/VGA and some config parameters  mov ah,#0x12  mov bl,#0x10  int 0x10  mov [8],ax  mov [10],bx  mov [12],cx 获取第一块硬盘的信息。; Get hd0 data  mov ax,#0x0000  mov ds,ax  lds si,[40x41]  mov ax,#INITSEG  mov es,ax  mov di,#0x0080  mov cx,#0x10  rep  movsb 获取第二块硬盘的信息。; Get hd1 data  mov ax,#0x0000  mov ds,ax  lds si,[40x46]  mov ax,#INITSEG  mov es,ax  mov di,#0x0090  mov cx,#0x10  rep  movsb
```

以上原理都是一样的。

我们就没必要细琢磨了，对操作系统的理解作用不大，只需要知道最终存储在内存中的信息是什么，在什么位置，就好了，之后会用到他们的。

内存地址	长度(字节)	名称
0x90000	2	光标位置
0x90002	2	扩展内存数
0x90004	2	显示页面
0x90006	1	显示模式
0x90007	1	字符列数
0x90008	2	未知
0x9000A	1	显示内存
0x9000B	1	显示状态
0x9000C	2	显卡特性参数
0x9000E	1	屏幕行数
0x9000F	1	屏幕列数
0x90080	16	硬盘 1 参数表
0x90090	16	硬盘 2 参数表
0x901FC	2	根设备号

由于之后很快就会用 c 语言进行编程，虽然汇编和 c 语言也可以用变量的形式进行传递数据，但这需要编译器在链接时做一些额外的工作，所以这么多数据更方便的还是**双方共同约定一个内存地址**，我往这里存，你从这里取，就完事了。这恐怕是最最原始和直观的变量传递的方式了。

把这些信息存储好之后，操作系统又要做什么呢？我们继续往下看。

```
    cli         ; no interrupts allowed ;
```

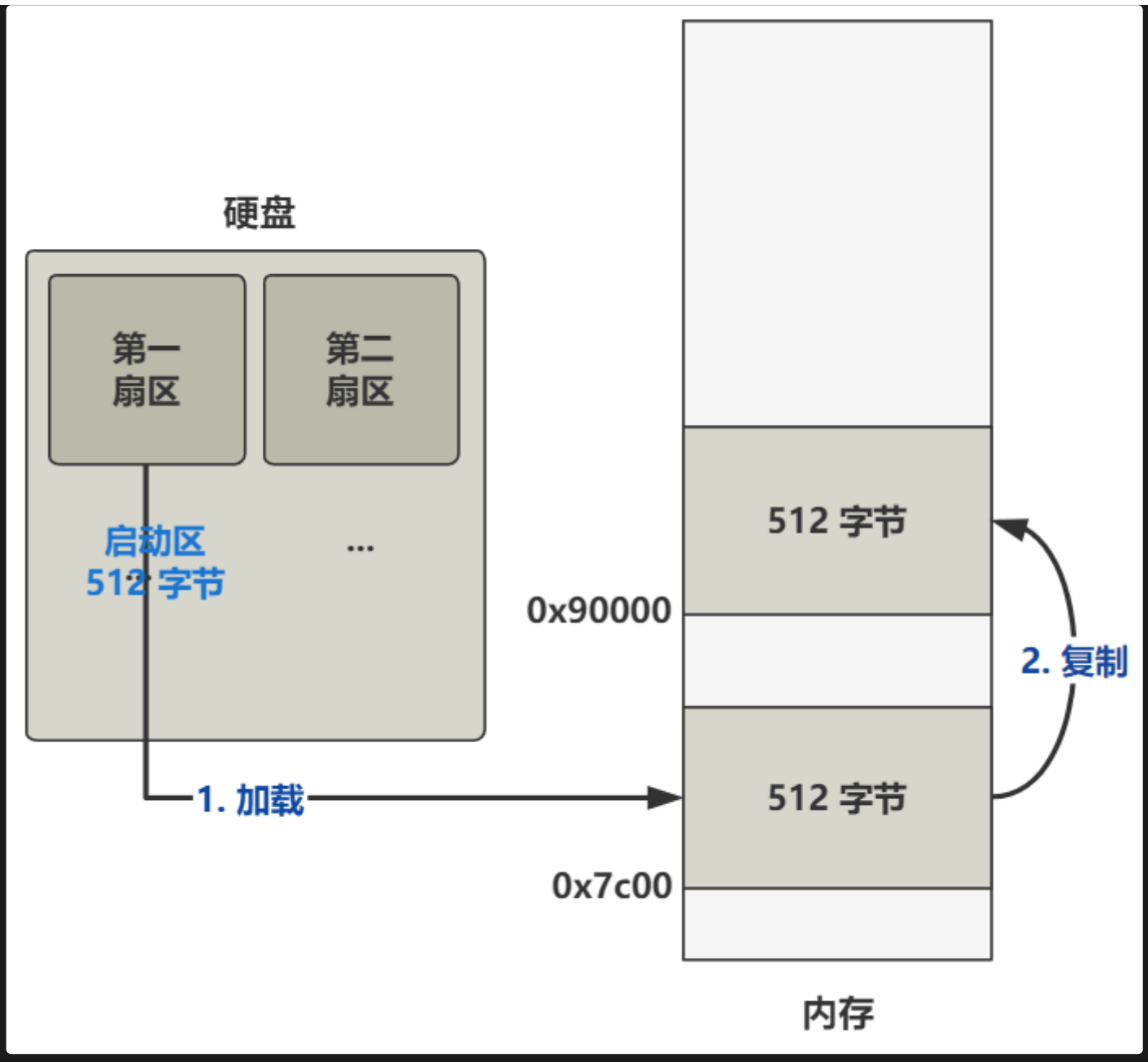
就一行 cli，表示**关闭中断**的意思。

因为后面我们要把原本是 BIOS 写好的中断向量表给覆盖掉，也就是给破坏掉了，写上我们自己的中断向量表，所以这个时候是不允许中断进来的。

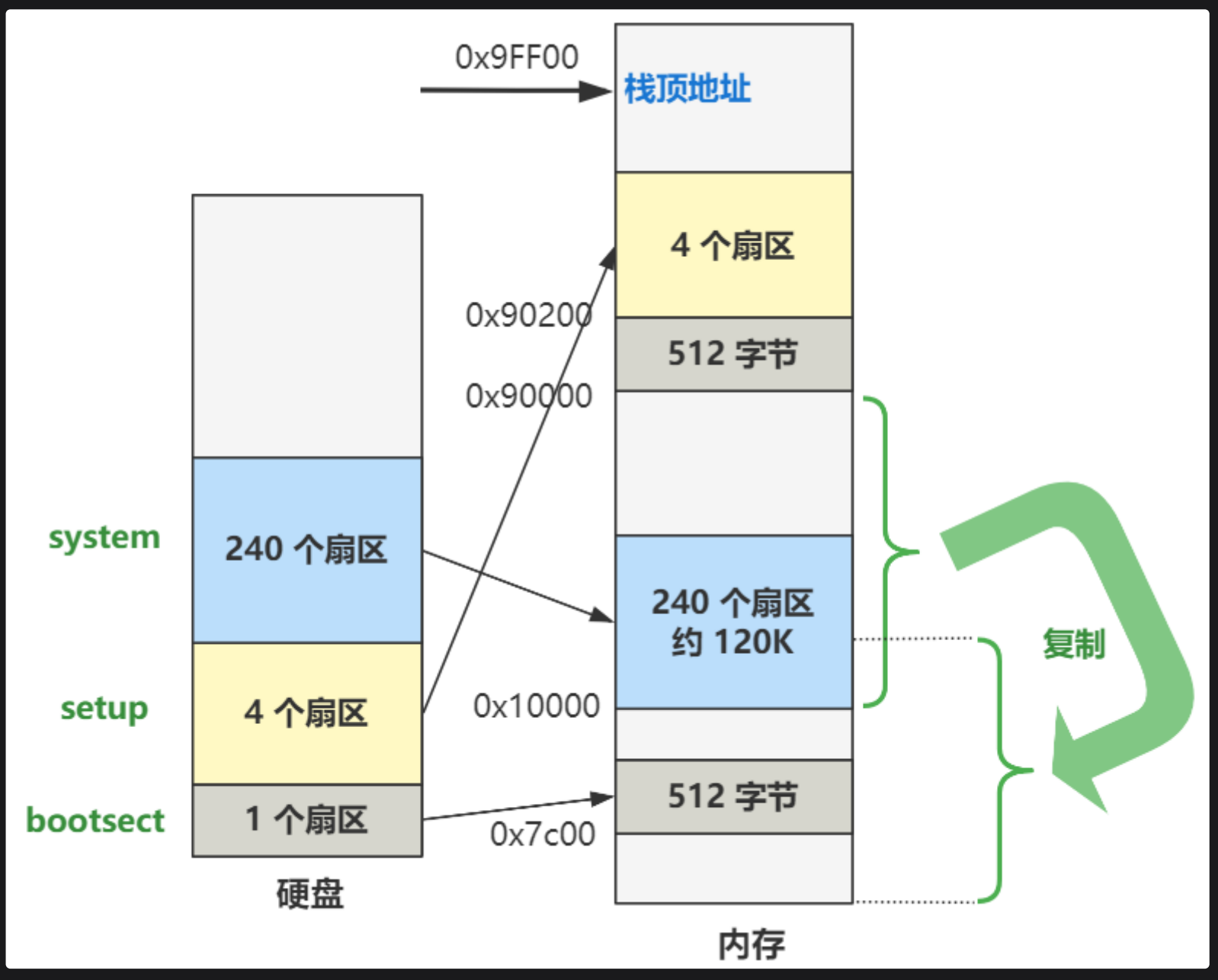
继续看。

```
    ; first we move the system to it's rightful place    mov ax,#0x0000    cld         ; 'direction'=0, movs moves forwarddo_move:    mov es,ax       ; destination segment    add ax,#0x1000    cmp ax,#0x9000    jz  end_move    mov ds,ax       ; source segment    sub di,di    sub si,si    mov cx,#0x8000    rep movsw    jmp do_move; then we load the segment descriptorsend_move:    ...
```

看到后面那个 **rep movsw** 熟不熟悉，一开始我们把操作系统代码从 **0x7c00** 移动到 **0x90000** 的时候就是用的这个指令，来图回忆一下。



同前面的原理一样，也是做了个内存复制操作，最终的结果是，把内存地址 **0x10000** 处开始往后一直到 **0x90000** 的内容，统统复制到内存的最开始的 **0** 位置，大概就是这么个效果。



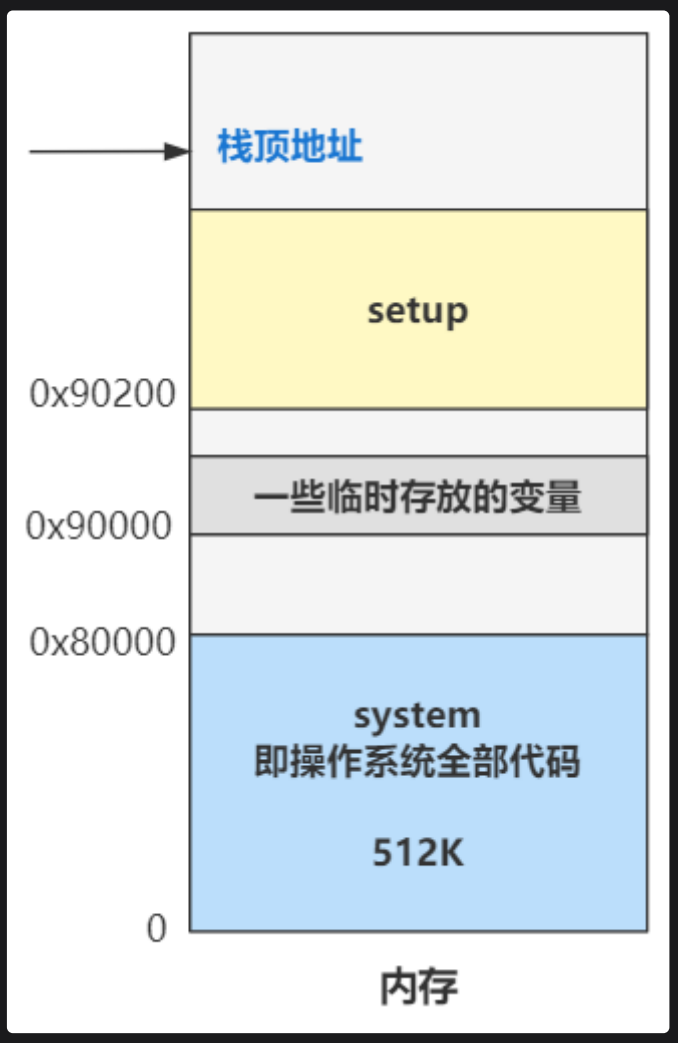
由于之前的各种加载和复制，导致内存看起来很乱，是时候进行一波取舍和整理了，我们重新梳理一下此时的内存布局。

栈顶地址仍然是 **0x9FF00** 没有改变。

0x90000 开始往上的位置，原来是 **bootsect** 和 **setup** 程序的代码，现 bootsect 的一部分代码在已经被操作系统为了记录内存、硬盘、显卡等一些**临时存放的**数据给覆盖了一部分。

内存最开始的 **0** 到 **0x80000** 这 512K 被 **system** 模块给占用了，之前讲过，这个 system 模块就是除了 bootsect 和 setup 之外的全部程序链接在一起的结果，可以理解为**操作系统的全部。**

那么现在的内存布局就是这个样子。



好了，记住上面的图就好了，这回是不是又重新清晰起来了？之前的什么 0x7c00，已经是过去式了，**赶紧忘掉它**，向前看！

接下来，就要进行有点技术含量的工作了，那就是**模式的转换**，需要从现在的 16 位的**实模式**转变为之后 32 位的**保护模式**，这是一项大工程！也是我认为的这趟操作系统源码旅程中，第一个颇为精彩的地方，大家做好准备！

后面的世界越来越精彩，欲知后事如何，且听下回分解。