

第3课 虚拟内存的原理与实现

《跟着瓦利哥学写OS》

联系方式

- 自己动手写操作系统QQ群：82616767。

- 申请考试邮箱：sangwf@gmail.com

- 所有课程的代码都在Github：

<https://github.com/sangwf/walleclass>

思考题回顾

- 在实验中，我们的是将第2扇区先加载到
0x100000，然后再复制到0x0000000的，为什么
不直接加载到0x0000000？

讲解更正1：

```
40 ;这里加 dword是为了生成一个32位偏移地址的指令
41 ;这条指令执行的时候，已经是在保护模式了
42 ;是利用了CPU预取的机制，在执行 mov cr0, eax时，
43 ;就将其读入了指令寄存器
44 ;0x0008表示GDT中第一个描述符
45 ;0x00000000是跳转的32位偏移地址，有效的
46 jmp dword 0x0008: 0x00000000
```


讲解更正2：

- 第1节课的思考题，我讲到使用 `jmp`
`BOOTSEG: LOOP1`，而不能使用 `jmp`
`LOOP1`。这里讲解错误，实际是可以的。

认识机器码

```
1  nop
2  nop
3  nop
4  LOOP1:
5      jmp  LOOP1
6  nop
7  jmp  0x1234:  LOOP1
```


认识机器码

```
sangwf:cls3 sangwf$ nasm -f bin machinecode.s -o machinecode
sangwf:cls3 sangwf$ hexdump machinecode
00000000 90 90 90 eb fe 90 ea 03 00 34 12
0000000b
```


重新认识jmp指令

- 迄今为止，我们已经学习了jmp指令的两种用法

:

- `jmp LOOP1`

- `jmp BOOTSEG: LOOP1`

jmp LOOP1

- LOOP1这个标记符，会转化为相对于jmp LOOP1后面一条指令 ($IP + 2$) 的偏移。而非相对于文件的起始地址。
- 如LOOP1: jmp LOOP1翻译为机器指令后，成为0xeb 0xfe，其中0xeb为相对跳转的机器码。0xfe = 0b11111110 = $od(-2)$ 。
- 这是因为jmp LOOP1指令的长度为2。

jmp BOOTSEG: LOOP1

- 段间跳转，设置CS = BOOTSEG。LOOP1翻译为相对于文件头部的偏移地址。
- 如例子中的代码翻译为0xea 0x03 0x00 0x34 0x12。0xea表示段间跳转的机器码，0x0003表示LOOP1相对文件起始的偏移是3。

分段的问题

- 段占用的物理内存连续，且前后无法扩展，就要整体移位。
- 内存不足时，需要将整个段置换到磁盘（回顾段描述符的P标记位），来回置换代价大。

段描述符通用格式

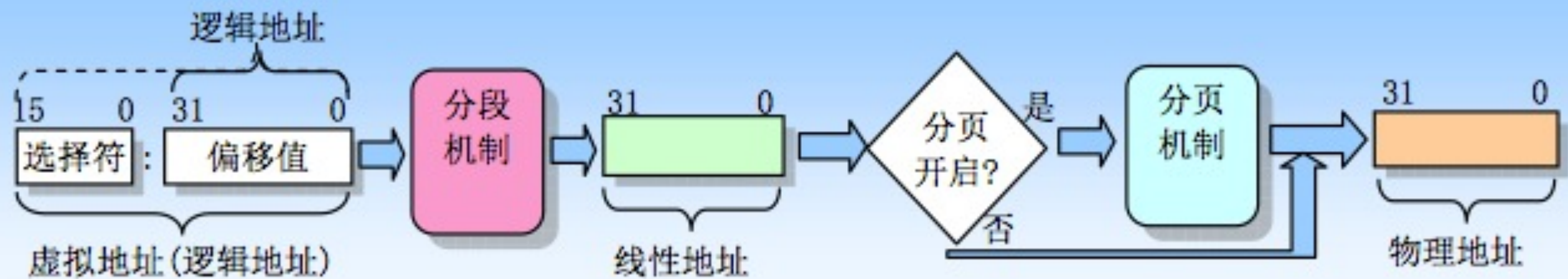


- | | | | |
|------|---------------------------|-------|--------------------------|
| AVL | -- 系统软件可用位 | LIMIT | -- 段限长 |
| BASE | -- 段基地址 | P | -- 段存在 |
| B/D | -- 默认大小 (0-16 位; 1-32 位段) | S | -- 描述符类型 (0-系统; 1-代码或数据) |
| DPL | -- 描述符特权级 | TYPE | -- 段类型 |
| G | -- 颗粒度 | | |

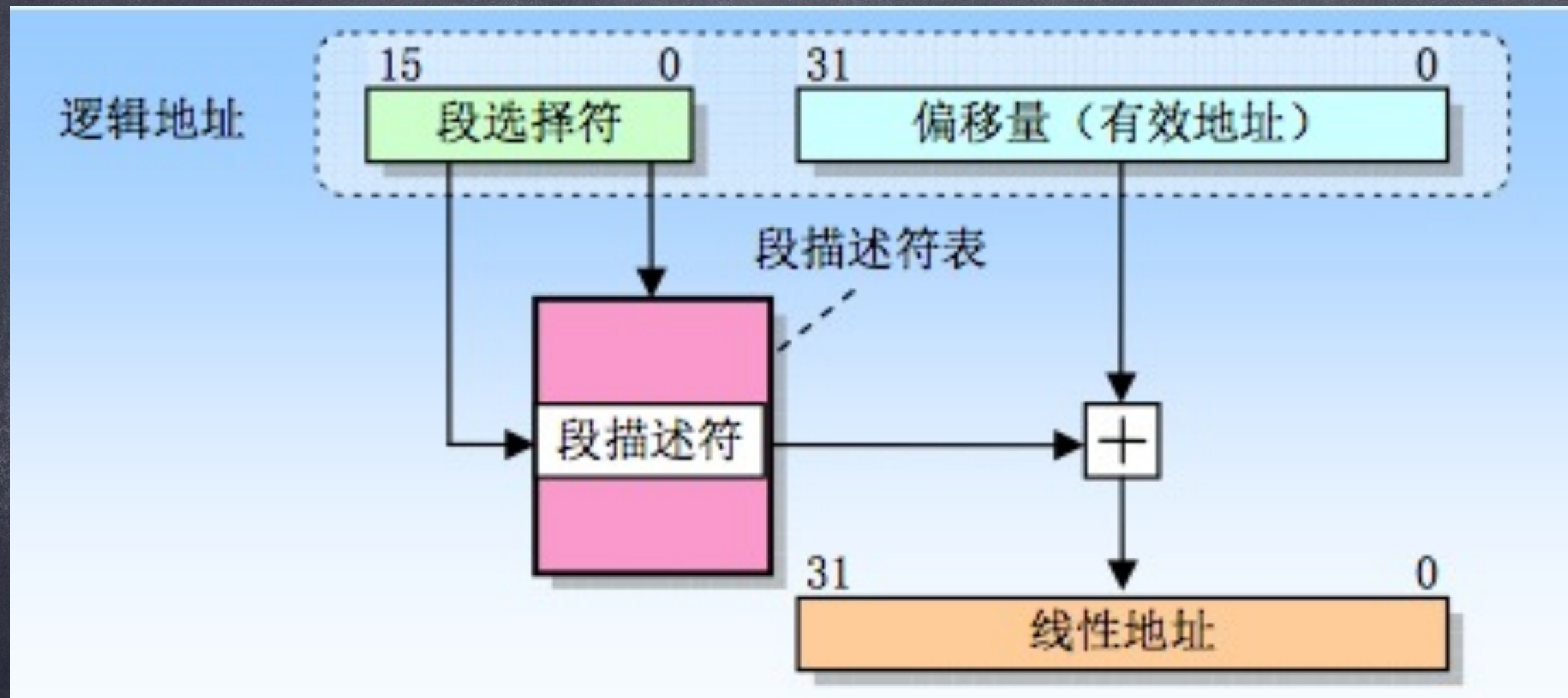
分页的解决之道

- 在保护模式下，可以开启分页，每页4K，可以按需分配。
- 好处：
 - 页面之间独立，不要求连续。
 - 实际是将内存当成了磁盘的缓存，按需加载。
(传说中的虚拟内存)

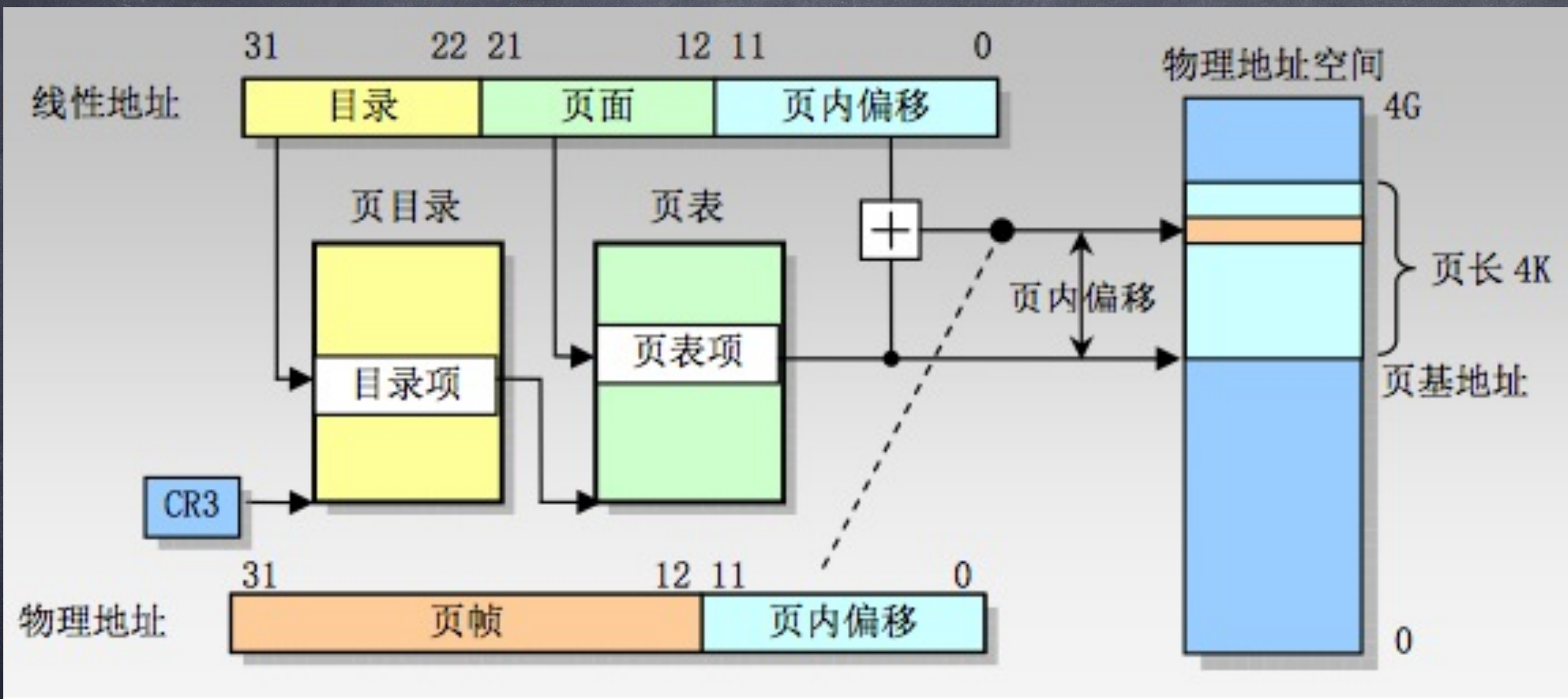
从虚拟地址到物理地址



段地址的转换



分页的地址转换



页目录表/页表的格式



P：表示页是否在物理内存中，是虚拟内存工作的最关键的标记。

D：页面是否被修改。当页面要被置换到磁盘时，该位决定了是否要回写。

A：页面是否被访问。没被访问过的页面，优先被换出到磁盘。

半吊子的虚拟内存

- 本节课所讲的部分，只包括开启分页机制。一个真正支持虚拟内存的系统，需要在内存不足时，将某些页面置换到磁盘。也访问的地址不存在时，分配物理页面，并从磁盘加载内容。
- 因为牵涉到缺页中断与保护模式下的磁盘读取，这个放到以后再讲。

开启分页的步骤

- 分页是在分段的基础上进行的，所以是进入保护模式之后。
- 1, 初始化CR3为页目录表的基地址（实际高20位有效）。
- 2, 填充页目录表中的项为页表的基地址（实际是页帧号）。
- 3, 给每个页表的项赋值，即设置实际的物理页起始地址。
- 4, 开启CR0的PG标记位（bit 31）。
- 5, 进入分页模式。

第一个实验

- 在上节课的基础上，实现分页模式下，在屏幕打印一个字符串Hello, world!
- 我们将 $0 \sim 4M$ 的线性地址映射到和物理地址完全一样。
- 将页目录表放在 $1M$ 的物理地址。
- 将第 0 个页表放在 $1M+4K$ 的物理地址。

代码 (boot.s增加一个数据段)

```
81      ;第三个描述符 (0x18), 表示数据段
82      dw 0x07FF ;段限长低16位, 则段限长为:
83              ; (0x07FF + 1) * 4KB = 8MB
84      dw 0x0000 ;地址低16位0, 所以地址为0
85      dw 0x9200 ;0x92 = 1001 0010
86              ;高4位分别表示P = 1段在内存中
87              ;DPL = 0, 最高权限
88              ;S = 1, 非系统段
89              ;且TYPE的bit 3为0, 表示数据段
90              ;低8位表示地址23~16为0
91      dw 0x00C0 ;高8位表示地址31~24为0
92              ;G标志 (bit 7) = 1, 表示颗粒度为4KB
93              ;注意: 保护模式下颗粒度一般都设4KB
94              ;最后4位为0, 表示段限长19~16位是0
```


代码

(printhello_pg.s)

```
6 ;设置页目录表基址
7 ;即给CR3赋值，将页目录表起始地址的高20位
8 ;写入CR3的高20位，其中低12位是保留字段，无用
9 mov eax, 0x00100000 ;我们放到物理地址1M的地方
10 mov CR3, eax
```


代码

(printhello_pg.s)

```
12 ;只给页目录表的第0项赋值
13 ;即真正有效的地址为4M
14 ;赋值为物理地址1M+4K的高20位
15 mov eax, DATA_SEL ;先设置数据段
16 mov ds, eax
17
18 ;0x00101001的高20位表示页帧地址0x00101
19 ;低12位中的bit 0位P = 1, 表示该项有效
20 ;bit 1位R/W = 0, 表示只能读, 不能写
21 mov dword [0x00100000], 0x00101001
```


代码

(printhello_pg.s)

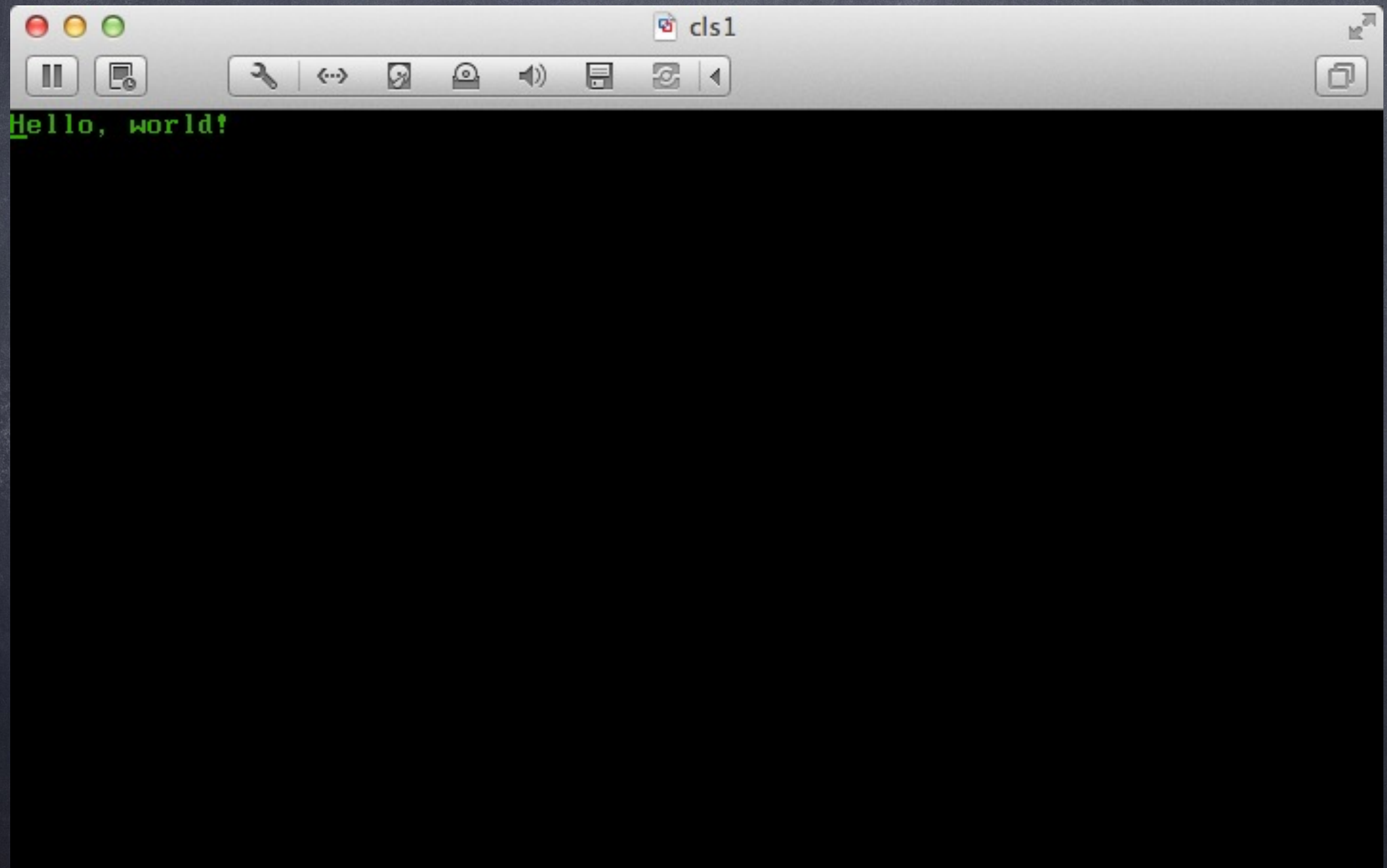
```
23 ;给第0项页表赋值
24 ;页表中的每一项，都和实际物理地址一致
25 ;即页表的第0项的页帧为0x00000
26 ;第1项为0x00001...以此类推
27 ;P设置为1，R/W设置为1
28 mov ecx, 0
29 mov eax, 0x00101000
30 mov ebx, 0x00000003
31
32 set_page:
33 mov [eax + ecx * 4], ebx
34 add ebx, 0x00001000 ;页帧号++
35
36 inc ecx
37 cmp ecx, 1024 ;共有1024项
38 jne set_page
```


代码

(printhello_pg.s)

```
41 ;开启分页模式
42 ;即将CR0寄存器的第31位置1
43 mov eax, CR0
44 or eax, 0x80000000 ;最高位置1
45 mov CR0, eax
```


运行结果



A screenshot of a Java IDE window titled "cls1". The window has a standard macOS-style title bar with red, yellow, and green window control buttons. Below the title bar is a toolbar with icons for running, debugging, and other IDE functions. The main area of the window is black, and the text "Hello, world!" is displayed in green, indicating successful execution of a Java program.

```
Hello, world!
```


思考

- 在实验中，如何实现虚拟地址被访问时，再分配物理内存？

谢谢！