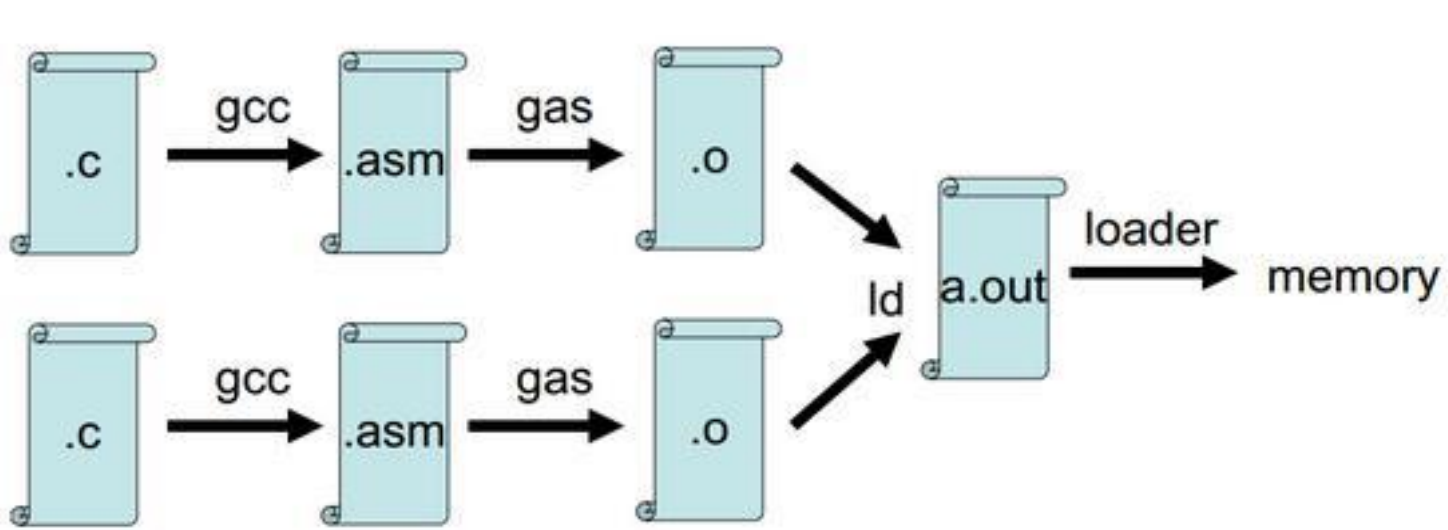
2.2 保护模式之段机制

西安邮电大学

从一个简单“Hello World” 程序说起

```
#include<stdio. h>
int  main（void）
 {
   printf（" Hello world！ "）；
   return  0；
}
```

程序的编译、链接和装载



1. gcc -S hello.c -o hello.s // 编译
2. gcc -c hello.s -o hell.o // 汇编
3. gcc hello.c -o a.out // 链接
4. ./a.out // 装载并执行

objdump -d a.out //反汇编

从一个简单“Hello World” 程序说起

程序通过编译器GCC将其编译成汇编程序，经过汇编器gas将其汇编成目标代码，经过连接器ld形成可执行文件a，最后通过装载器装入到内存。

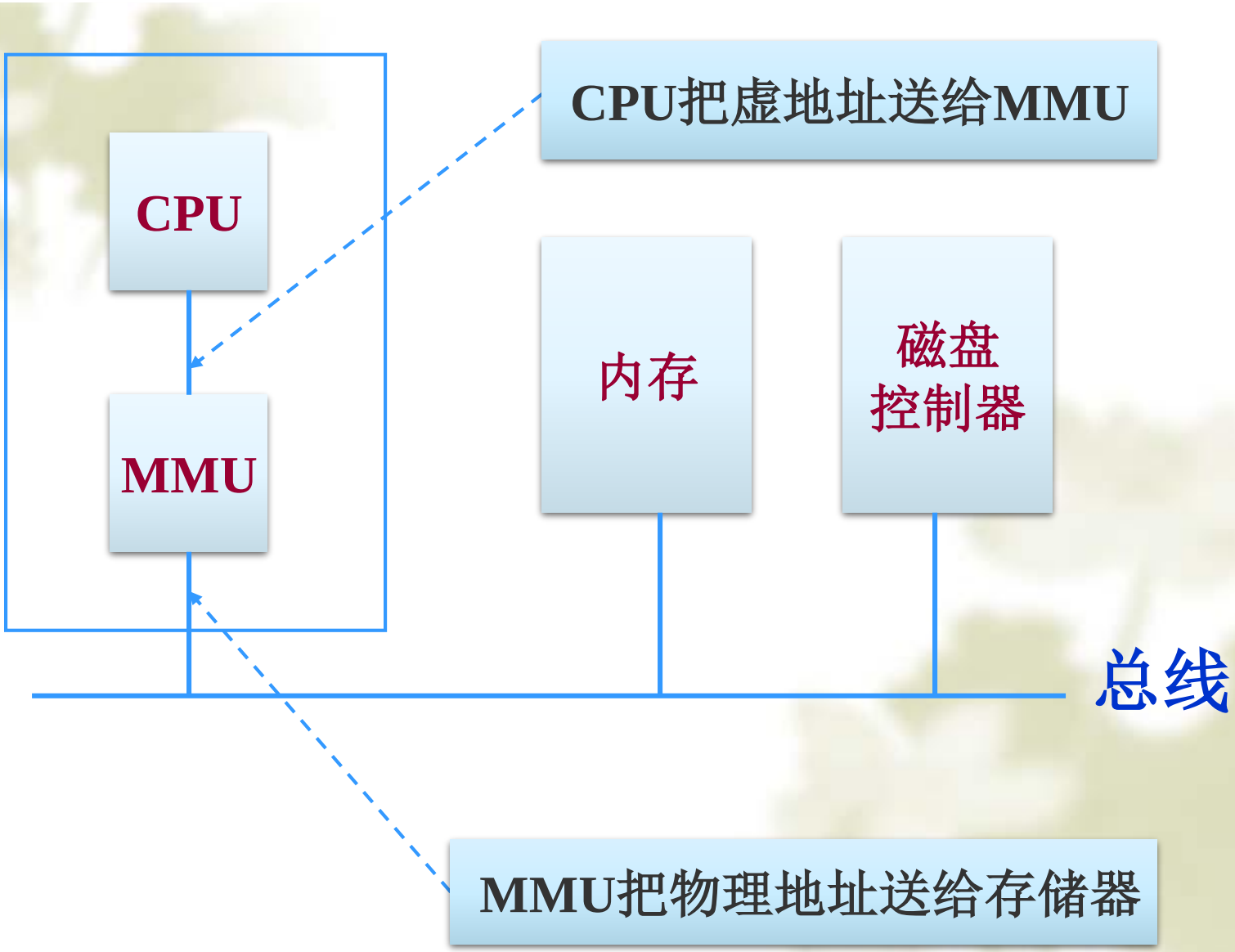
那么，问题来了，链接以后形成的地址是虚地址还是实地址，装入程序把可执行代码装入到虚拟内存还是物理内存？CPU访问的是虚地址还是物理地址？

程序的地址空间

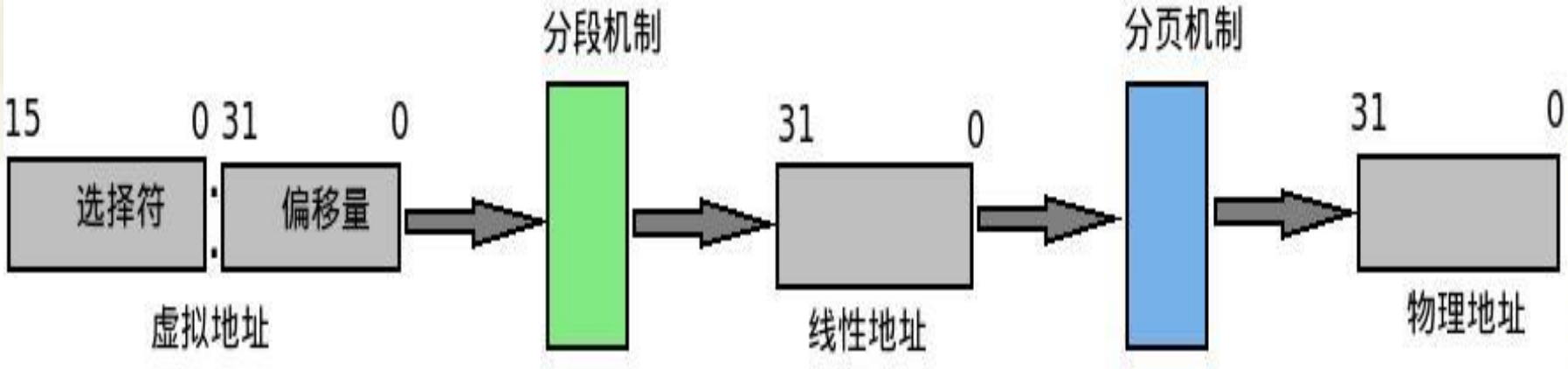
```
00000000004006cd <main>:
  4006cd:       55                      push   %rbp
  4006ce:       48 89 e5                mov    %rsp,%rbp
  4006d1:       48 81 ec a0 00 00 00    sub    $0xa0,%rsp
  4006d8:       89 bd 6c ff ff ff       mov    %edi,-0x94(%rbp)
  4006de:       48 89 b5 60 ff ff ff    mov    %rsi,-0xa0(%rbp)
  4006e5:       83 bd 6c ff ff ff 03    cmpl   $0x3,-0x94(%rbp)
  4006ec:       74 19                   je     400707 <main+0x3a>
  4006ee:       bf 80 08 40 00          mov    $0x400880,%edi
  4006f3:       b8 00 00 00 00          mov    $0x0,%eax
  4006f8:       e8 63 fe ff ff          callq  400560 <printf@plt>
  4006fd:       bf 01 00 00 00          mov    $0x1,%edi
  400702:       e8 b9 fe ff ff          callq  4005c0 <exit@plt>
  400707:       48 8b 85 60 ff ff ff    mov    -0xa0(%rbp),%rax
  40070e:       48 83 c0 08             add    $0x8,%rax
  400712:       48 8b 00                mov    (%rax),%rax
```

这是编译链接后的64位的地址空间，最左边是地址，中间是指令码 ，右边是AT&T格式的汇编指令。

保护模式下的寻址



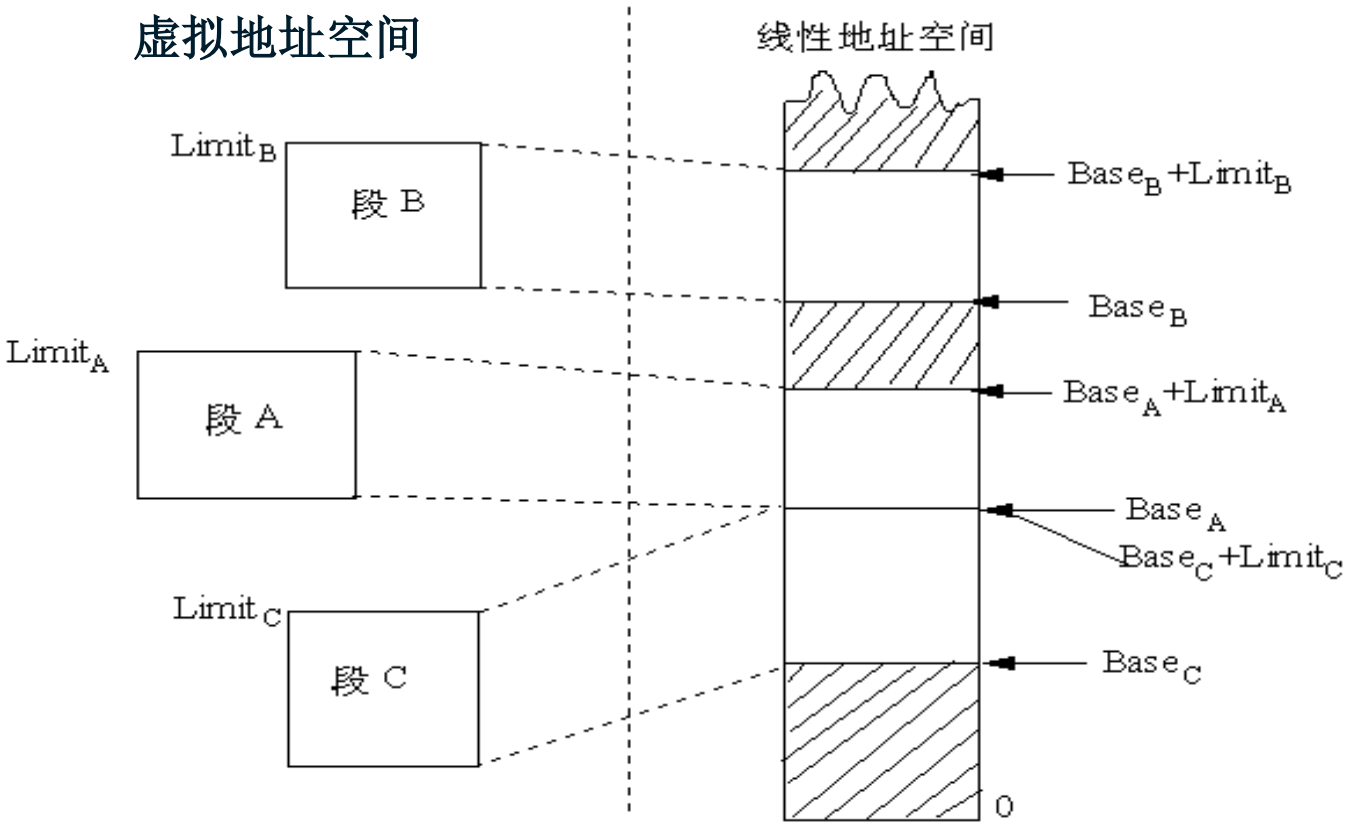
MMU的地址转换



MMU把虚拟地址转化为物理地址

MMU的转换分两个阶段，分段机制和分页机制，分段把虚拟地址转换为线性地址，分页把线性地址转换为物理地址。

虚拟—线性地址的转换



虚拟地址空间中偏移量从0到limit范围内的一个段，映射到线性地址空间中就是从Base到Base+Limit。

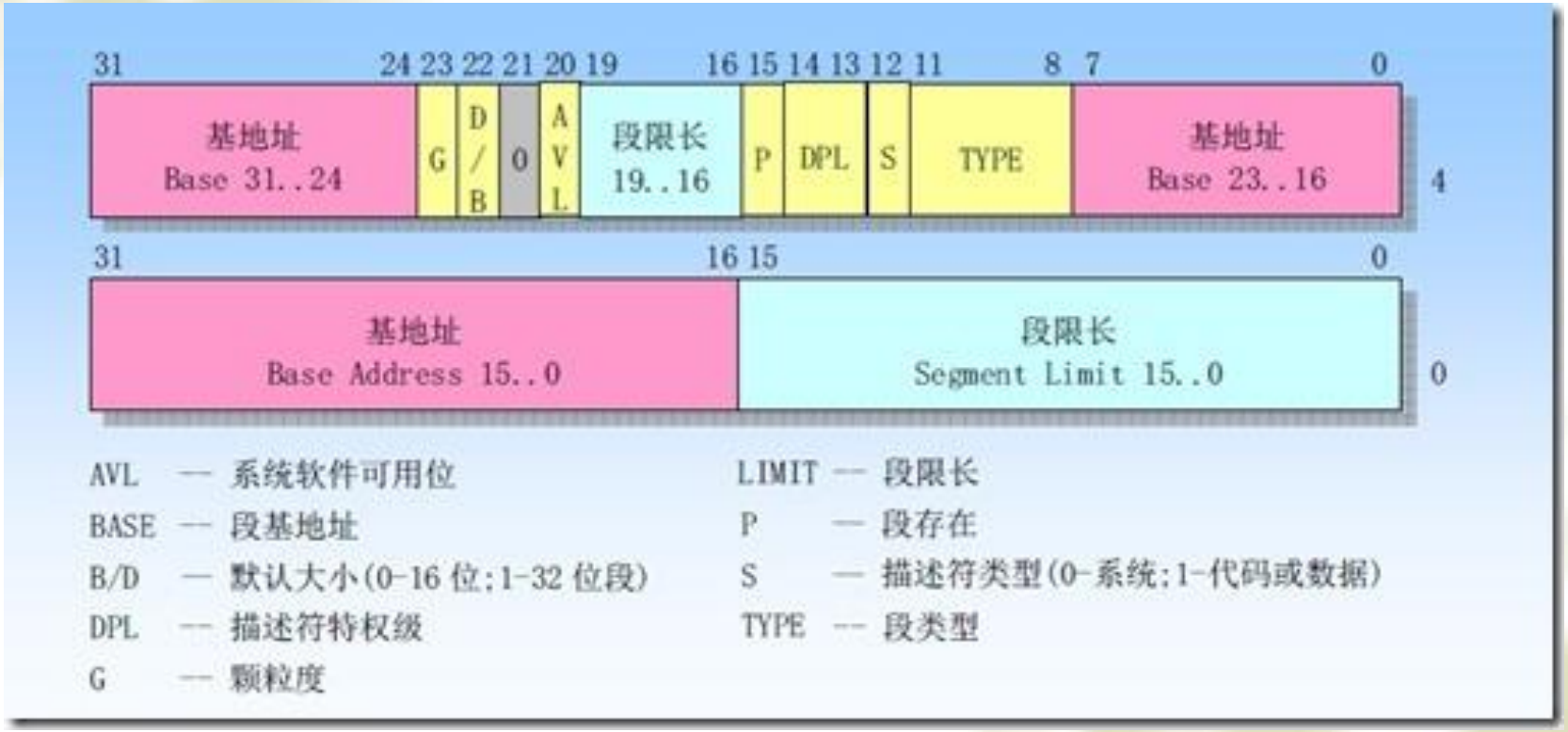
段描述符表—段表

★如图所示的段描述符表（或叫段表）来描述转换关系。段号描述的是虚拟地址空间段的编号，基地址是线性地址空间段的起始地址。

★段描述符表中的每一个表项叫做段描述符

索引（段号）	基地址	界限	属性
0	$Base_b$	$Limit_b$	$Attribute_b$
1	$Base_a$	$Limit_a$	$Attribute_a$
2	$Base_c$	$Limit_c$	$Attribute_c$

段描述符—描述段的结构

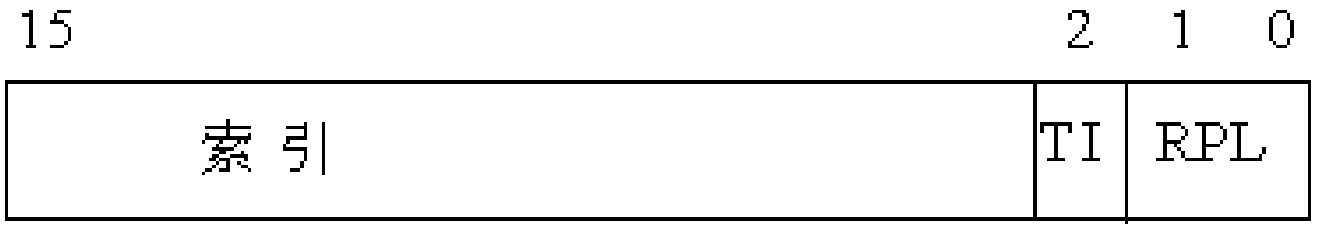


从图可以看出，一个段描述符指出了段的32位基地址和20位段界限(即段长)，1.5个字节用于描述段的属性。

保护模式下段寄存器中存放什么

★存放索引或叫段号，因此，这里的段寄存器也叫选择符，即从描述符表中选择某个段。

★选择符（段寄存器）的结构：



★RPL表示请求者的特权级（Requestor Privilege Level）

★TI（Table Index）

保护模式下的特权级

- ★保护模式提供了四个特权级，用0~3四个数字表示
- ★很多操作系统（包括Linux,Windwos）只使用了其中的最低和最高两个，即0表示最高特权级，对应内核态；3表示最低特权级，对应用户态。
- ★保护模式规定，高特权级可以访问低特权级，而低特权级不能随便访问高特权级。

保护模式下的其他描述符表简介

- ★全局描述符表GDT（Gloabal Descriptor Table）
- ★中断描述符表IDT（Interrupt Descriptor Table）
- ★局部描述符表LDT（Local Descriptor Table）
- ★为了加快对这些表的访问，Intel设计了专门的寄存器，以存放这些表的基地址及表的长度界限 。这些寄存器只供操作系统使用。

有关这些表的详细内容请参看有关保护模式的参考书。

Linux中的段

线性地址=段的起始地址+偏移量

Linux在启动的过程中设置了段寄存器的值和全局描述符表**GDT**的内容，内核代码中可以这样定义段：

```
#define __KERNEL_CS      0x10    /*内核代码段，index=2,TI=0,RPL=0*/
#define __KERNEL_DS      0x18    /*内核数据段，index=3,TI=0,RPL=0*/
#define __USER_CS        0x23    /*用户代码段，index=4,TI=0,RPL=3*/
#define __USER_DS        0x2B    /*用户数据段，index=5,TI=0,RPL=3*/
```

段	Base	G	Limit	S	Type	DPL	D/B	P
用户代码段	0x00000000	1	0xfffff	1	10	3	1	1
用户数据段	0x00000000	1	0xfffff	1	2	3	1	1
内核代码段	0x00000000	1	0xfffff	1	10	0	1	1
内核数据段	0x00000000	1	0xfffff	1	2	0	1	1

保护模式寻址实例

```
.set PROT_MODE_CSEG, 0x8          # kernel code segment selector
.set PROT_MODE_DSEG, 0x10         # kernel data segment selector
.set CR0_PE_ON,      0x1          # protected mode enable flag
lgdt    gdtdesc

  movl    %cr0, %eax
  orl     $CR0_PE_ON, %eax
  movl    %eax, %cr0

  # Jump to next instruction, but in 32-bit code segment.
  # Switches processor into 32-bit mode.
  #  实模式跳转到保护模式
  ljmp    $PROT_MODE_CSEG, $protcseg

protcseg:
  movw    0x10, %ax
  movw    %ax,  %dx
  movl    0xf0000000, %ebx
  movl    0x20(%ebx), %eax

gdt:                                    #gdt表的内容
  SEG_NULL                                          # null seg
  SEG(STA_X|STA_R, 0x0, 0xffffffff)       # code seg
  SEG(STA_W, 0x0, 0xffffffff)           # data seg

gdtdesc:
  .word   0x17                            # sizeof(gdt) - 1
  .long   gdt
```

保护模式寻址实例

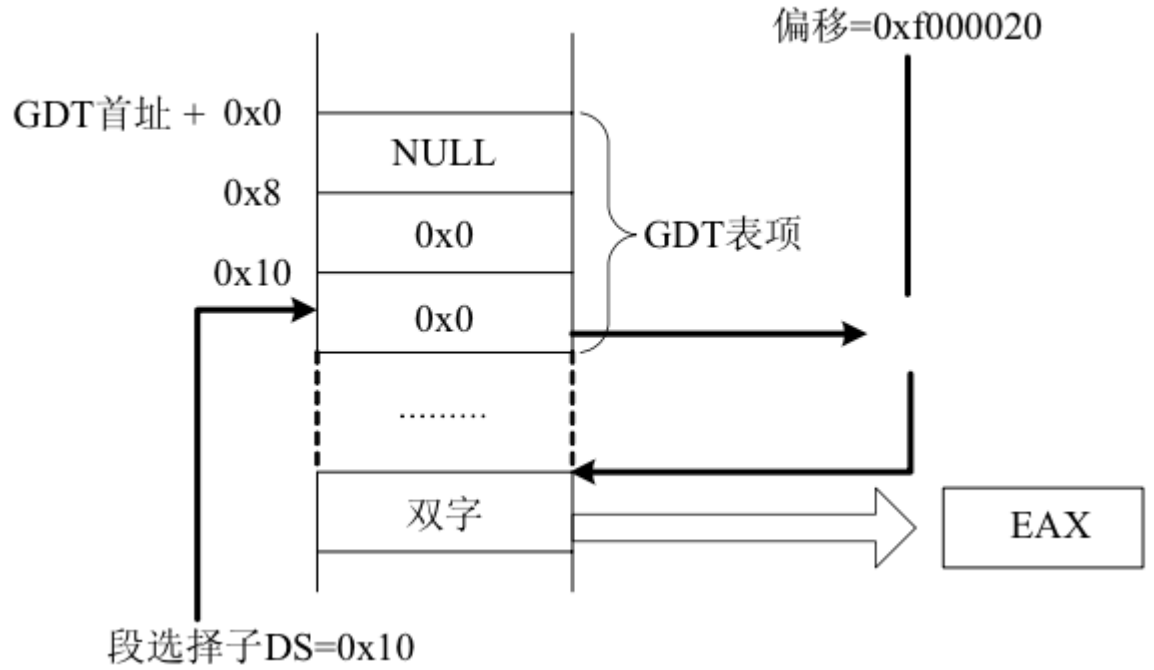
以上这一小段程序展示了系统进入保护模式以及在保护模式中利用寄存器寻址的过程。

首先lgdt指令将GDT表的地址和表长装入GDTR寄存器，在gdtdesc标识的地方存有一个字及一个双字，前者为0x17表示表的长度(字节数)，后者是表的物理地址。

接着再将CR0的保护模式开启位打开，系统便进入了保护模式，开始采用保护模式的寻址模式进行地址的转换。这个时候，内存中有GDT的3个表项。

进入保护模式后系统立即执行了一个长跳转指令，由于是在保护模式中，所以 PROT_MODE_CSEG（前者）被当作段选择子，而protcseg（后者）是偏移地址。段选择子的值是0x8，于是对应的段描述符会是表中的第一项，即是SEG(STA_X|STA_R, 0x0, 0xffffffff)这一项，0x0表示段首地址是0，所以最终得到的物理地址为0 + protcseg，程序便会跳到 protcseg 所标识的位置来执行。

保护模式寻址实例



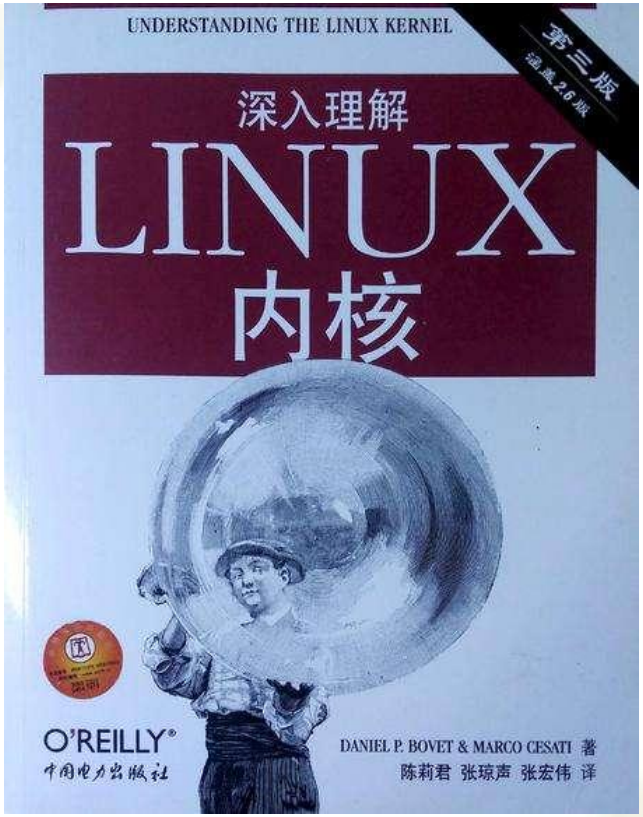
之后执行指令：movl 0x20(%ebx)， %eax， 如图所示可知段基址为0，于是物理地址是0+0x20+0xf000000=0xf000020，内存中这个位置的一个双字会被复制到eax寄存器中。

这里能访问的到的0到4G的地址空间实际上是虚拟地址空间，在开启分页机制后，还要经过页表转换才能得到真实地址，而在开启分页之前系统一般会控制只访问低地址。

动手实战

- Linux内核之旅网站http://www.kerneltravel.net/
- 电子杂志栏目是关于内核研究和学习的资料
- 第二期“《i386体系结构》上”，上半部分让大家认识一下Intel系统中的内存寻址和虚拟内存的来龙去脉。
- 下载代码进行调试

参考资料



深入理解Linux内核第二章

带着思考离开

硬件设计和操作系统设计中，到底进行怎样的取舍和折中？

谢谢大家！

THANK YOU