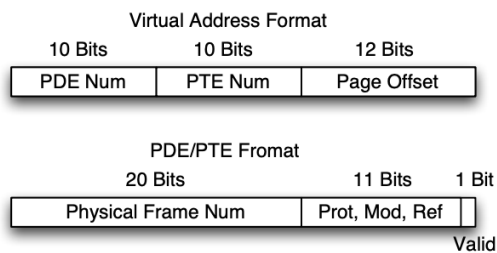


# Intel x86 Memory Architecture

- 2-Level Page Table
- 4KB Page Size
- 32 bit addresses
- PDE/PTE of 32 bits



1

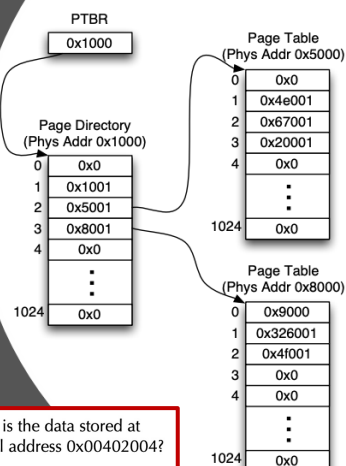
## Translation

Describe the result of accessing the following virtual addresses:

0x0  
0x00803024  
0x00c00136

$2^{22} == 0x400000$ ,  
 $2^{12} == 0x1000$

What is the data stored at virtual address 0x00402004?



2

# 《操作系统》课程

## 第三章 内存管理

授课教师：孙海龙

82339063, [sunhl@buaa.edu.cn](mailto:sunhl@buaa.edu.cn)

2024年春季，北航计算机/软件学院

3

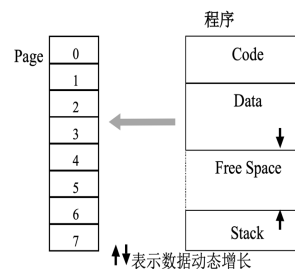
## 内容提要

- 存储管理基础
- 页式内存管理
- 段式内存管理
  - 基本原理
  - 地址变换
  - 段共享
  - 与页式管理优缺点对比
  - 段页式管理
- 虚拟内存管理
- 内存管理实例

4

## 回顾：页式管理

- 从单一逻辑地址空间向不连续物理内存空间映射
- 问题1：难以满足程序运行时对内存的动态需求
  - 编译器：随着编译过程的进行，读取的源程序、解析的符号表、常量表和语法树都在动态增加；子程序调用所需的栈空间不断变化。
  - 动态链接：动态链接在程序运行时才把主程序和要用到的目标程序（程序段）链接起来。
- 问题2：不便于数据共享与保护
  - “数据共享”是程序处理逻辑层面的需求，而“页”是内存分配层面的基本单位；
  - 通过“页共享/保护”实现“数据共享/保护”的难度较大。



## 段式存储管理的主要动机

方便编程：

- 通常一个作业是由多个程序段和数据段组成的，用户一般按逻辑关系对作业分段，并能根据名字来访问程序段和数据段。

信息共享：

- 共享是以信息的逻辑单位为基础的。页是存储信息的物理单位，段是信息的逻辑单位。
- 页式管理中地址空间是一维的，主程序、子程序都顺序排列，共享公用子程序比较困难，一个共享过程可能需要几十个页面。

## 段式存储管理的主要动机

### 信息保护：

- 页式管理中，一个页面中可能装有 2 个不同的子程序的指令代码，不能通过页面共享实现一个逻辑上完整的子程序或数据块的共享。
- 段式管理中，可以使用信息的逻辑单位进行保护。

### 动态增长：

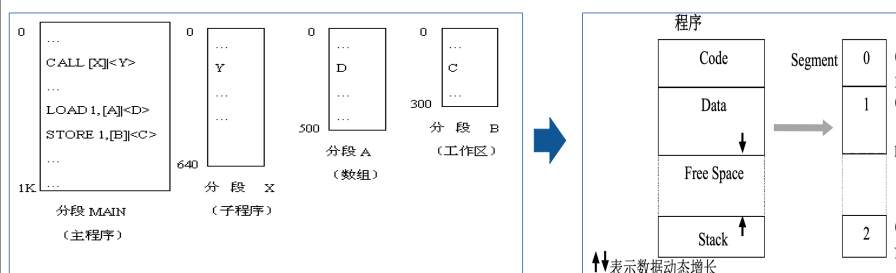
- 实际应用中，某些段（数据段）会不断增长，前面的存储管理方法均难以实现。

### 动态链接：

- 动态链接在程序运行时才把主程序和要用到的目标程序（程序段）链接起来。

## 基本思想：分段地址空间

- 一个段可定义为一组逻辑信息，每个作业的地址空间是由一些分段构成的（由用户根据逻辑信息的相对完整来划分），每段都有自己的名字（通常是段号），且都是一段连续的地址空间，首地址为0。



## 地址变换

### 段表

- 段表记录了段与内存位置的对应关系。
- 段表保存在内存中。
- 段表的基址及长度由段表寄存器给出。

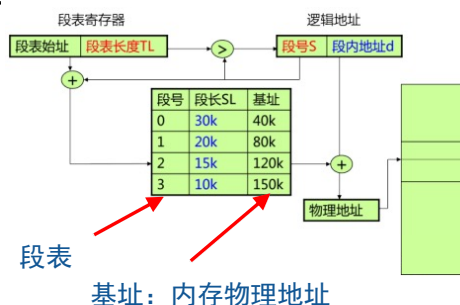
段表始址	段表长度
------	------

- 访问一个字节的数据/指令需访问内存两次（段表一次，内存一次）。
- 逻辑地址由段和段内地址组成。

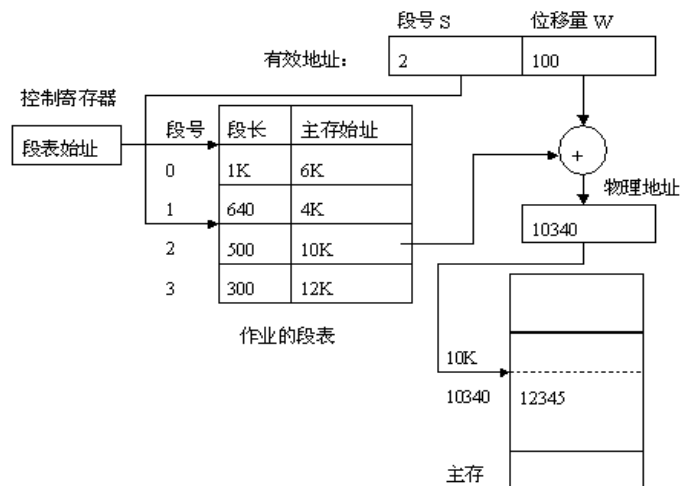
段号	段内地址
----	------

## 地址变换过程

1. 将逻辑地址中的段号  $S$  与段表长度  $TL$  进行比较。
  - 若  $S > TL$ ，表示访问越界，产生越界中断。
  - 若未越界，则根据段表的始址和该段的段号，计算出该段对应段表项的位置，从中读出该段在内存的始址。
2. 再检查段内地址  $d$ ，是否超过该段的段长  $SL$ 。
  - 若超过，即  $d > SL$ ，同样发出越界中断信号。
  - 若未越界，则将该段的基址与段内地址  $d$  相加，即可得到要访问的内存物理地址。



## 地址变换过程举例



Beihang University  
School of Computer Science & Engineering  
北京航空航天大学计算机学院



北京航空航天大学  
第三章 内存管理

2023/3

11

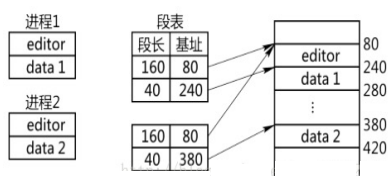
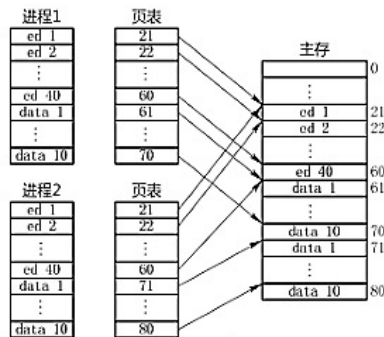
11

## 信息共享（分段与分页比较）

例：一个系统可同时接纳40个用户，都执行一个文本编辑程序。程序有160KB代码和40KB数据区，如果不共享，则共需8MB内存。可采用共享代码，在内存中只需保留一份代码副本，则所需内存仅为 1760 KB

(40 × 40 = 1600) 页共享: 40个页表项

段共享: 1个段表项



要求: 代码是可重入的



Beihang University  
School of Computer Science & Engineering  
北京航空航天大学计算机学院



北京航空航天大学  
第三章 内存管理

2023/3

12

12


## 补充知识：可重入代码

- 可重入代码 (Reentrant Code) 又称为“纯代码” (Pure Code)，是指在多次并发调用时能够安全运行的代码。**要求：**不能使用全局/静态变量；代码不能修改代码本身；不能调用其他的不可重入代码。

```
int v = 1;


int f()
{
    v += 2;
    return v;
}

int g()
{
    return f() + 2;
}
```



```
int f(int i)
{
    return i + 2;
}

int g(int i)
{
    return f(i) + 2;
}
```



[https://en.wikipedia.org/wiki/Reentrancy\\_\(computing\)](https://en.wikipedia.org/wiki/Reentrancy_(computing))



Beihang University  
School of Computer Science & Engineering  
北京航空航天大学计算机学院



北京航空航天大学  
COLLEGE OF SOFTWARE  
软件学院

第三章 内存管理

2023/3

13

13

## 分段管理的优缺点

- 优点：
  - 分段系统易于实现段的共享，对段的保护也十分简单。
  - 更好地支持动态的内存需求
- 缺点：
  - 处理机要为地址变换花费时间；要为段表提供附加的存储空间。
  - 为满足分段的动态增长和减少外碎片，要采用内存紧凑的技术手段。
  - 在辅存中管理不定长度的分段比较困难（交换）。
  - 分段的最大尺寸受到主存可用空间的限制。



Beihang University  
School of Computer Science & Engineering  
北京航空航天大学计算机学院



北京航空航天大学  
COLLEGE OF SOFTWARE  
软件学院

第三章 内存管理

2023/3

14

14

## 分页与分段的比较

- 分页的作业的地址空间是单一的线性地址空间（注：页式管理所用的一维的虚拟地址也称为线性地址），分段作业的地址空间是二维的。

- “页”是信息的“物理”单位，大小固定。“段”是信息的逻辑单位，即它是一组有意义的信息，其长度不定。

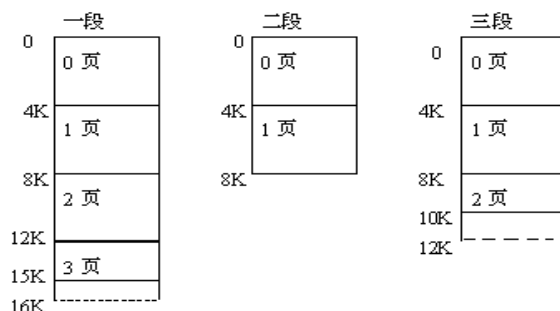
- 分页对用户是透明的，是系统对于主存的管理。分段是用户可见的（分段可以在用户编程时确定，也可以在编译程序对源程序编译时根据信息的性质来划分）。

	页式存储管理	段式存储管理
目的	实现非连续分配，解决碎片问题	更好地满足用户需要
信息单位	页（物理单位）	段（逻辑单位）
大小	固定（由系统定）	不定（由用户程序定）
内存分配单位	页	段
作业地址空间	一维	二维
优点	有效解决了碎片问题（没有外碎片，每个内碎片不超过页大小）；有效提高内存的利用率；程序不必连续存放。	更好地实现数据共享与保护；段长可动态增长；便于动态链接

15

## 段页式存储管理：分段 + 分页

- 基本思想：用分段方法来分配和管理虚拟存储器，而用分页方法来分配和管理实存储器。



16



## 实现原理

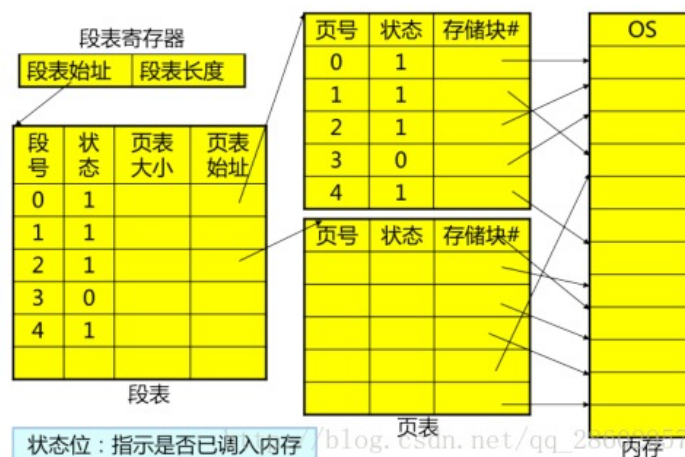
- 段页式存储管理：分段和分页原理的结合，即先将用户程序分成若干个段（段式），并为每一个段赋一个段名，再把每个段分成若干个页（页式）。
- 地址结构由段号、段内页号、及页内位移三部分所组成。

段号 (S)	段内页号 (P)	页内地址 (W)
0	0	00000000
0	1	00000001
0	2	00000010
0	3	00000011
0	4	00000100
0	5	00000101
0	6	00000110
0	7	00000111
0	8	00001000
0	9	00001001
0	10	00001010
0	11	00001011
0	12	00001100
0	13	00001101
0	14	00001110
0	15	00001111
1	0	00010000
1	1	00010001
1	2	00010010
1	3	00010011
1	4	00010100
1	5	00010101
1	6	00010110
1	7	00010111
1	8	00011000
1	9	00011001
1	10	00011010
1	11	00011011
1	12	00011100
1	13	00011101
1	14	00011110
1	15	00011111
2	0	00100000
2	1	00100001
2	2	00100010
2	3	00100011
2	4	00100100
2	5	00100101
2	6	00100110
2	7	00100111
2	8	00101000
2	9	00101001
2	10	00101010
2	11	00101011
2	12	00101100
2	13	00101101
2	14	00101110
2	15	00101111
3	0	00110000
3	1	00110001
3	2	00110010
3	3	00110011
3	4	00110100
3	5	00110101
3	6	00110110
3	7	00110111
3	8	00111000
3	9	00111001
3	10	00111010
3	11	00111011
3	12	00111100
3	13	00111101
3	14	00111110
3	15	00111111
4	0	01000000
4	1	01000001
4	2	01000010
4	3	01000011
4	4	01000100
4	5	01000101
4	6	01000110
4	7	01000111
4	8	01001000
4	9	01001001
4	10	01001010
4	11	01001011
4	12	01001100
4	13	01001101
4	14	01001110
4	15	01001111
5	0	01010000
5	1	01010001
5	2	01010010
5	3	01010011
5	4	01010100
5	5	01010101
5	6	01010110
5	7	01010111
5	8	01011000
5	9	01011001
5	10	01011010
5	11	01011011
5	12	01011100
5	13	01011101
5	14	01011110
5	15	01011111
6	0	01100000
6	1	01100001
6	2	01100010
6	3	01100011
6	4	01100100
6	5	01100101
6	6	01100110
6	7	01100111
6	8	01101000
6	9	01101001
6	10	01101010
6	11	01101011
6	12	01101100
6	13	01101101
6	14	01101110
6	15	01101111
7	0	01110000
7	1	01110001
7		

- 系统中设段表和页表，均存放于内存中。读一次指令或数据须访问内存三次。为提高速度可增设高速缓存。
- 每个进程一张段表，每个段一张页表。
- 段表含段号、页表始址和页表长度；页表包含页号和页框号。

17

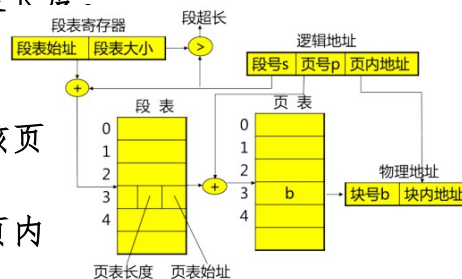
## 利用段表和页表实现地址映射



18

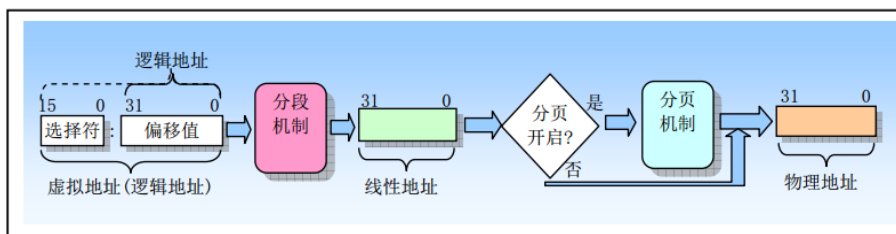
## 段页式存储管理的地址变换

- 从 PCB 中取出段表始址及段表长度，装入段表寄存器
- 将段号与段表长度进行比较，若段号大于或等于段表长度，产生越界中断。
- 利用段表始址与段号得到该段表项在段表中的位置。
- 将段表项中的段表地址和段表长度进行比较，若页号大于或等于页表长度，产生越界中断。
- 利用页表始址与页号得到该页表项在页表中的位置。
- 取出该页的物理块号，与页内地址拼接得到物理地址。

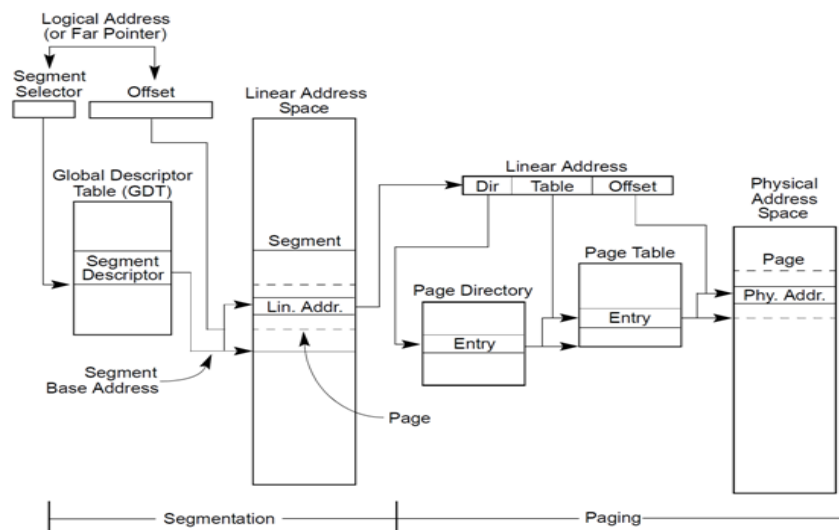


## 实例：Intel X86的段页式地址映射

- X86的地址映射机制分为两个部分：
  - 段映射机制，将逻辑地址映射到线性地址；
  - 页映射机制，将线性地址映射到物理地址。

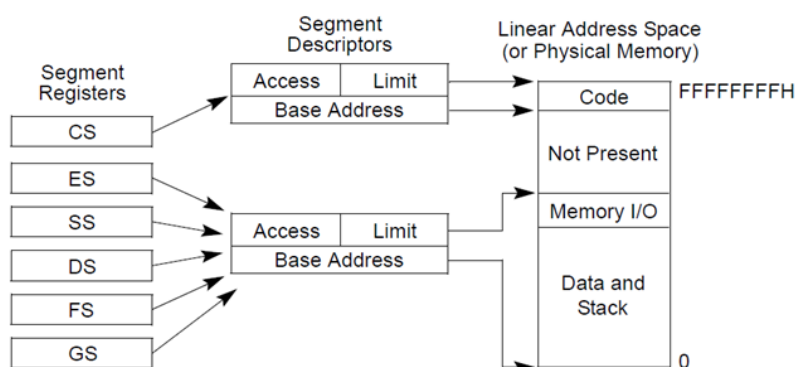


## X86的段页式地址映射



21

## 第一阶段：段式地址映射



22

## 段式地址映射过程

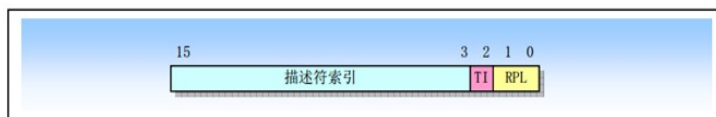
1. 根据指令的性质来确定应该使用哪一个段寄存器（Segment Selector），例如转移指令中的地址在代码段，而取数据指令中的地址在数据段；
2. 根据段寄存器的内容，找到相应的“地址段描述结构”（Segment Descriptor），段描述结构都放在一个表（Descriptor Table）中（GDT或LDT等），而表的起始地址保存在GDTR、LDTR等寄存器中。
3. 从地址段描述结构中找到基地址（Base Address）；
4. 将指令发出的地址作为位移，与段描述结构中规定的段长度相比，看看是否越界；
5. 根据指令的性质和段描述符中的访问权限来确定是否越权；
6. 将指令中发出的地址作为位移，与基地址相加而得出线性地址（Linear Address）。



23

## Segment Selector

- 80386之后的处理器共有6个段选择子，
  - CS寄存器：程序指令段起始地址；
  - DS寄存器：程序数据段起始地址；
  - SS寄存器：栈起始地址；
  - ES, FS, GS寄存器：额外段寄存器。



段选择符结构

TI（加载指示）：值为0处理器从GDT中加载；1则从LDT中加载。

RPL（请求优先级）：00最高，11最低。

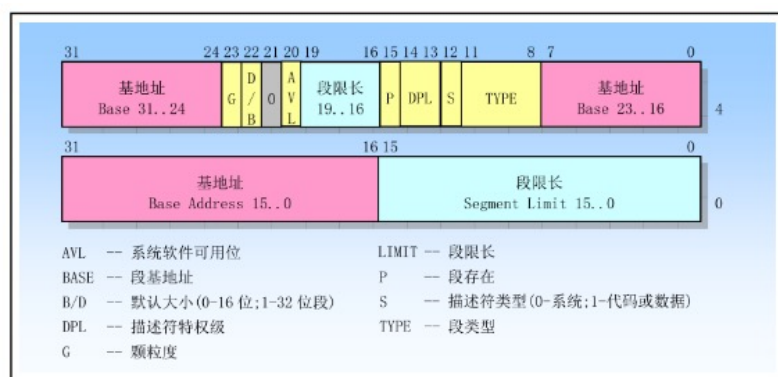


24

## GDT及LDT

- GDT (Global Descriptor Table): 全局描述符表，是全局性的，为所有的任务服务，不管是内核程序还是用户程序，都是把段描述符放在GDT中。
- LDT (Local Descriptor Table): 局部描述符表，为了有效实施任务间的隔离，把专属于某个任务的段描述符放到LDT中。

## Segment Descriptor



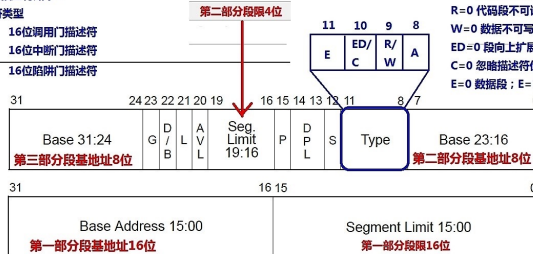
段描述符通用格式

# Segment Descriptor

Type类型域4个bits的特定组合表示的门描述符如下：

bit11	bit10	bit9	bit8	门描述符类型
0	1	0	0	16位调用门描述符
0	1	1	0	16位中断门描述符
0	1	1	1	16位陷阱门描述符
1100				32位调用门描述符
1110				32位中断门描述符
1111				32位陷阱门描述符

所有类型门描述符的bit12位，即5位都是0，属于系统描述符  
调用门描述符存储在全局描述符表（GDT）中；中断门、陷阱门描述符存储在中断描述符表（IDT）中



A=0 段未被访问；A=1 段已被访问

R=0 代码段不可读；R=1 代码段可读

W=0 数据不可写入；W=1 数据可写入

ED=0 段向上扩展（数据段）；ED=1 段向下扩展（堆栈段）

C=0 忽略描述符优先级；C=1 遵循描述符优先级

E=0 数据段；E=1 代码段

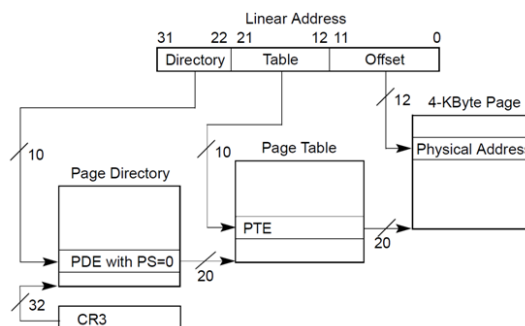
L — 64-bit code segment (IA-32e mode only)  
AVL — Available for use by system software  
BASE — Segment base address 用于计算最终线性地址（关闭分页功能）的32位段基地址，由3部分组成  
D/B — D=0 16位指令模式，使用16位偏移地址和16位寄存器；D=1 32位指令模式，使用32位偏移地址和32位寄存器  
DPL — Descriptor privilege level 描述符优先级 / 描述符特权级，00=ring0, 01=ring1, 10=ring2, 11=ring3  
G — Granularity 粒度位，G=0时，段限为20位，即00000~FFFF，即1B~1MB；G=1时，段限乘以4KB，即4KB~4MB，即00000FFF~FFFFFFFF，段限为32位  
LIMIT — Segment Limit 20位段限，由20位组成  
P — Segment present  
S — Descriptor type (0 = system; 1 = code or data) 段描述符类型，S=1为代码和数据段描述符，S=0为系统段描述符  
TYPE — Segment type

Segment Descriptor 64位段描述符，分成低32位和高32位

27

## 第二阶段：页式地址映射

1. 从CR3寄存器中获取页面目录表（Page Directory）的基地址；
2. 以线性地址的Directory位段为下标，在目录（Page Directory）中取得相应页表（Page Table）的基地址；
3. 以线性地址中的Table位段为下标，在所得到的页表中获得相应的页描述项；
4. 将页描述项中给出的页基地址与线性地址中的offset位段相加得到物理地址。

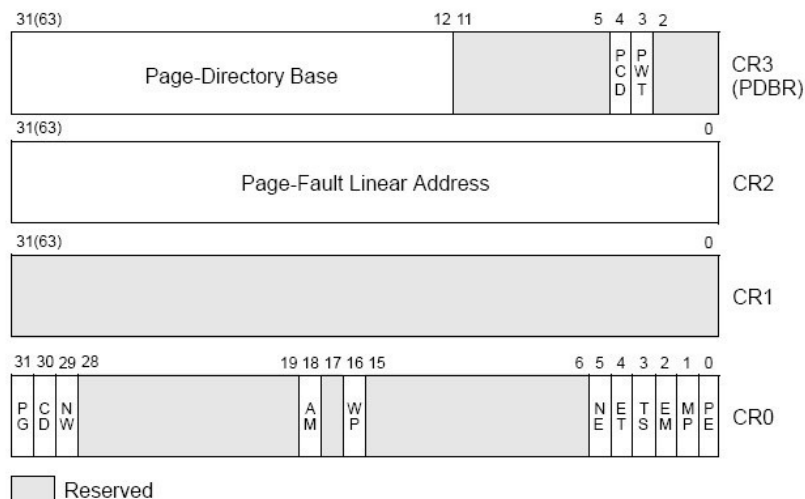


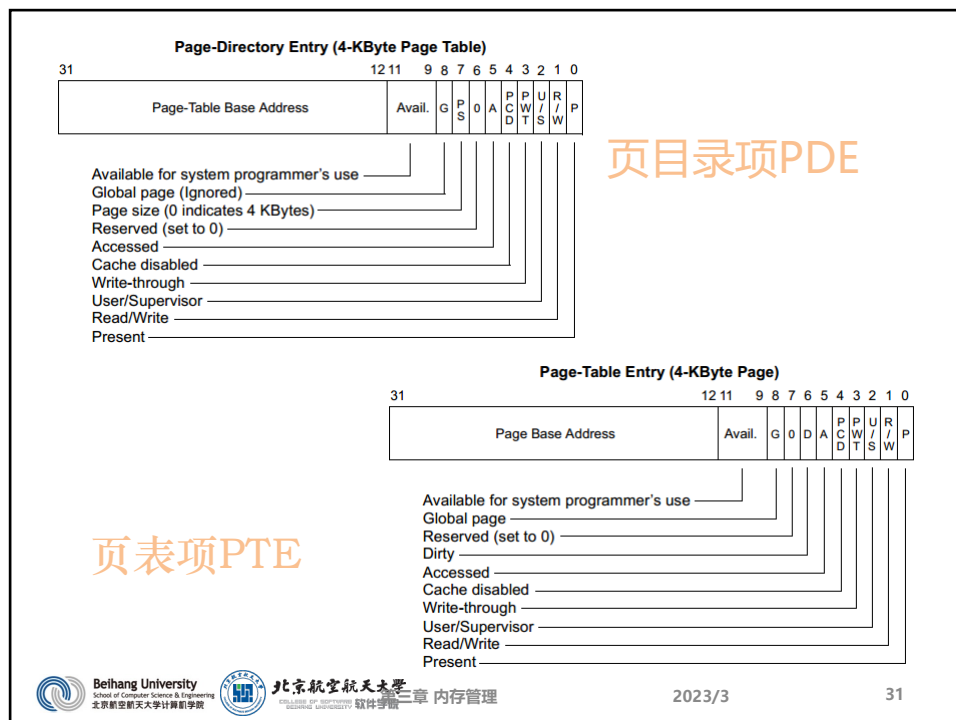
28

## X86的控制寄存器

- 控制寄存器（CR0~CR3）用于控制和确定处理器的操作模式以及当前执行任务的特性：
- CR0含有控制处理器操作模式和状态的系统控制标志；
- CR1保留不用；
- CR2保存最后一次出现页故障的线性地址；
- CR3中含有页目录表物理内存基地址，因此该寄存器也被称为页目录基地址寄存器PDBR（Page-Directory Base address Register），低12位是0。

## X86的控制寄存器





31

- **【P】**：存在位。为1表示页表或者页位于内存中。否则，表示不在内存中，必须先予以创建或者从磁盘调入内存后方可使用。
- **【R/W】**：读写标志。为1表示页面可以被读写，为0表示只读。当处理器运行在0、1、2特权级时，此位不起作用。页目录中的这个位对其所映射的所有页面起作用。
- **【U/S】**：用户/超级用户标志。为1时，允许所有特权级别的程序访问；为0时，仅允许特权级为0、1、2的程序访问。页目录中的这个位对其所映射的所有页面起作用。
- **【PWT】**：Page级的Write-Through标志位。为1时使用Write-Through的Cache类型；为0时使用Write-Back的Cache类型。当CR0.CD=1时（Cache被Disable掉），此标志被忽略。对于我们的实验，此位清零。
- **【PCD】**：Page级的Cache Disable标志位。为1时，物理页面是不能被Cache的；为0时允许Cache。当CR0.CD=1时，此标志被忽略。对于我们的实验，此位清零。
- **【A】**：访问位。该位由处理器固件设置，用来指示此表项所指向的页是否已被访问（读或写），一旦置位，处理器从不清这个标志位。这个位可以被操作系统用来监视页的使用频率。
- **【D】**：脏位。该位由处理器固件设置，用来指示此表项所指向的页是否写过数据。
- **【PS】**：Page Size位。为0时，页的大小是4KB；为1时，页的大小是4MB（for normal 32-bit addressing）或者2MB（if extended physical addressing is enabled）。
- **【G】**：全局位。如果页是全局的，那么它将在高速缓存中一直保存。当CR4.PGE=1时，可以设置此位为1，指示Page是全局Page，在CR3被更新时，TLB内的全局Page不会被刷新。
- **【AVL】**：被处理器忽略，软件可以使用。

Beihang University  
School of Computer Science & Engineering  
北京航空航天大学计算机学院

北京航空航天大学  
BEIHANG UNIVERSITY  
第三章 内存管理

2023/3 32

32



## 小结

- 基本原理
  - 多个独立的逻辑地址空间：一维→二维
- 地址变换：段表
- 段共享：与页共享的区别
- 与页式管理优缺点对比
- 段页式管理
  - 基于分段的地址空间管理
  - 基于分页的内存分配管理

33

## Q&A

微信群/课程中心论坛

[sunhl@buaa.edu.cn](mailto:sunhl@buaa.edu.cn)

34