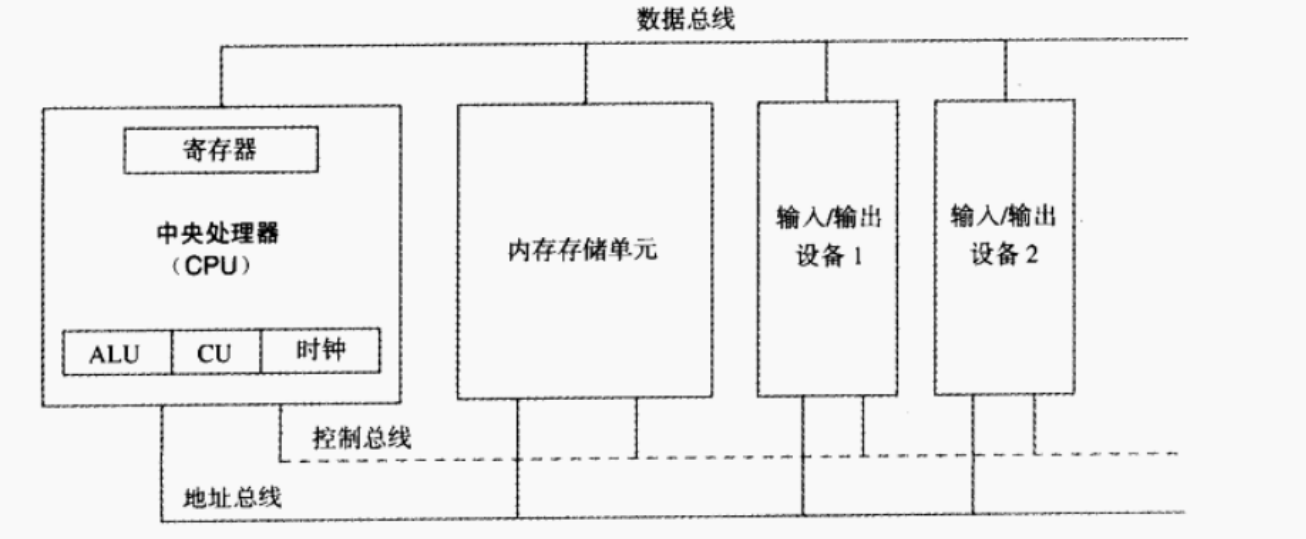
IA-32处理器

基本概念

微机的基本结构



中央处理器（CPU，Central Processor Unit)进行计算和逻辑操作的地方

- 寄存器Register：数据存储，数量有限
- 控制单元CU：控制机器指令的执行步骤
- 算数逻辑单元ALU：算术运算、逻辑运算
- 时钟clock：同步CPU的内部操作。每个时钟周期CPU完成一步操作。时钟频率=1/时钟周期，时钟频率（多为GHz量级）反映了CPU速度的快慢。

内存存储单元：存放指令和数据的地方，核心频率133MHz~200MHz

总线(bus)是一组用于在计算机各部分之间传送数据的并行线。计算机的系统总线一般由三组独立的总线构成：数据总线、控制总线和地址总线。数据总线在CPU和内存之间传送指令和数据；控制总线使用二进制信号同步连接到系统总线上的所有设备的动作；如果当前被执行的指令要在CPU和内存之间传送数据，那么地址总线上保持着指令和数据的地址。多为MHz量级。

指令执行周期

单条机器指令的执行包括一系列操作

- 取指令：指令指针IP
- 解码：控制单元CU确定执行什么操作
- 取操作数：从内存读操作数

- 执行：算数逻辑单元ALU
- 存储输出操作数：向内存写入

IA-32处理器体系结构

IA-32（Intel Architecture 32-bit）英特尔32位体系结构，1985年 80386 CPU首先使用，32位内存地址，32位数据操作数。

操作模式

保护模式

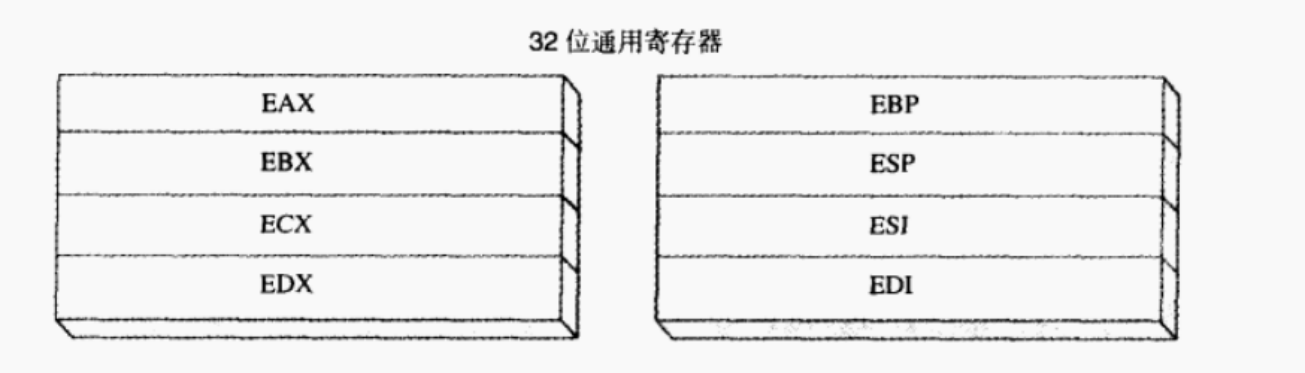
所有指令和特性都是可用的，程序被赋予独立的内存区域（段），处理器阻止程序访问已分配段之外的其他内存。
32位，IA-32程序设计环境
多任务操作系统
程序有独立的4GB内存储存空间

实地址模式

16位，8086程序设计环境
储存空间1MB，20条地址线

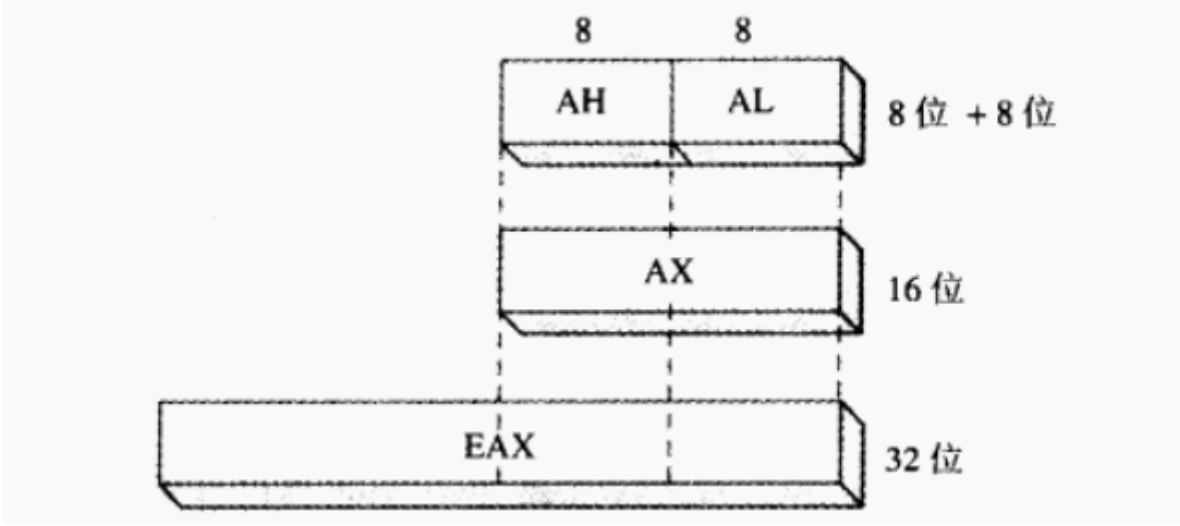
寄存器

CPU内部的高速储存单元，比内存的访问速度快很多，优化循环结构执行速度，把循环计数变量放到寄存器中。



通用寄存器

主要用于算术运算和数据的传送。如下图所示，每个寄存器可作为一个32位值或两个16位值来寻址使用。



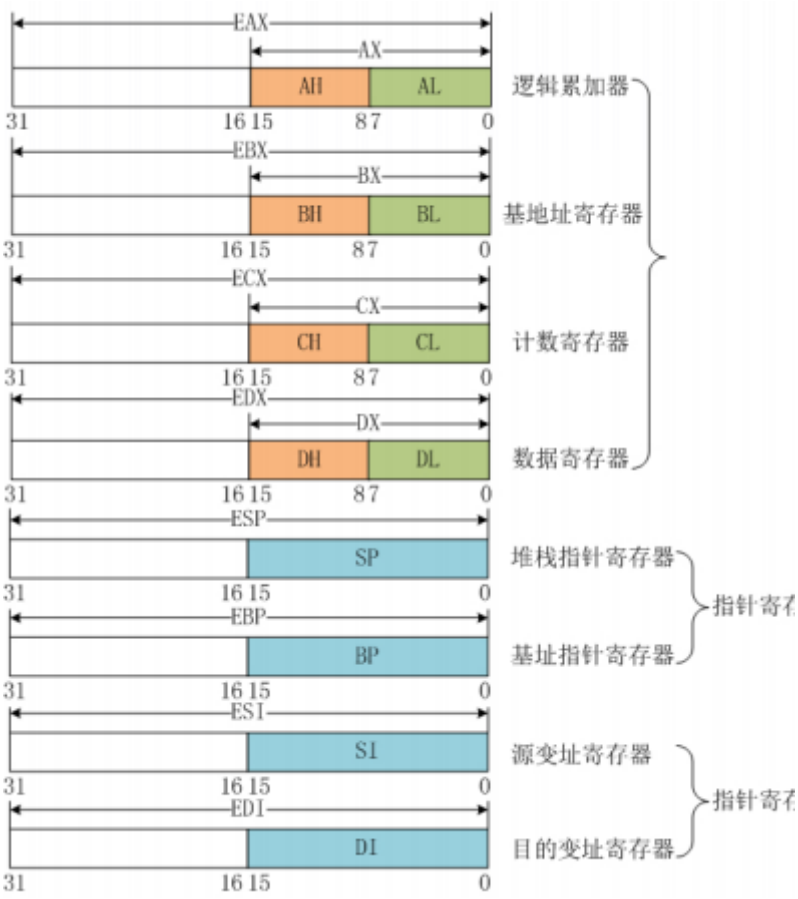
某些16位的寄存器能按8位值寻址使用。例如，32位的EAX寄存器的低16位称AX，AX寄存器的高8位称AH，低8位称AL。EAX,EBX,ECX,EDX寄存器都存在这种交迭的关系。

其余通用寄存器只有低16位有特别的名字，这里列出的16位寄存器通常在编写实地址模式程序时使用。

32 位	16 位
ESI	SI
EDI	DI
EBP	BP
ESP	SP

某些通用寄存器有特殊用法：

- EAX在乘法和除法指令中被自动使用，通常称为扩展累加寄存器。
- 在某些指令中，CPU自动使用ECX作为循环计数器。
- ESP寻址堆栈 (一种系统内存结构)上的数据，极少用于普通的算术运算和数据传送，通常称之为扩展堆栈指针寄存器。
- ESI和EDI由高速内存数据传送指令使用，通常称为扩展源指针和扩展目的指针寄存器。

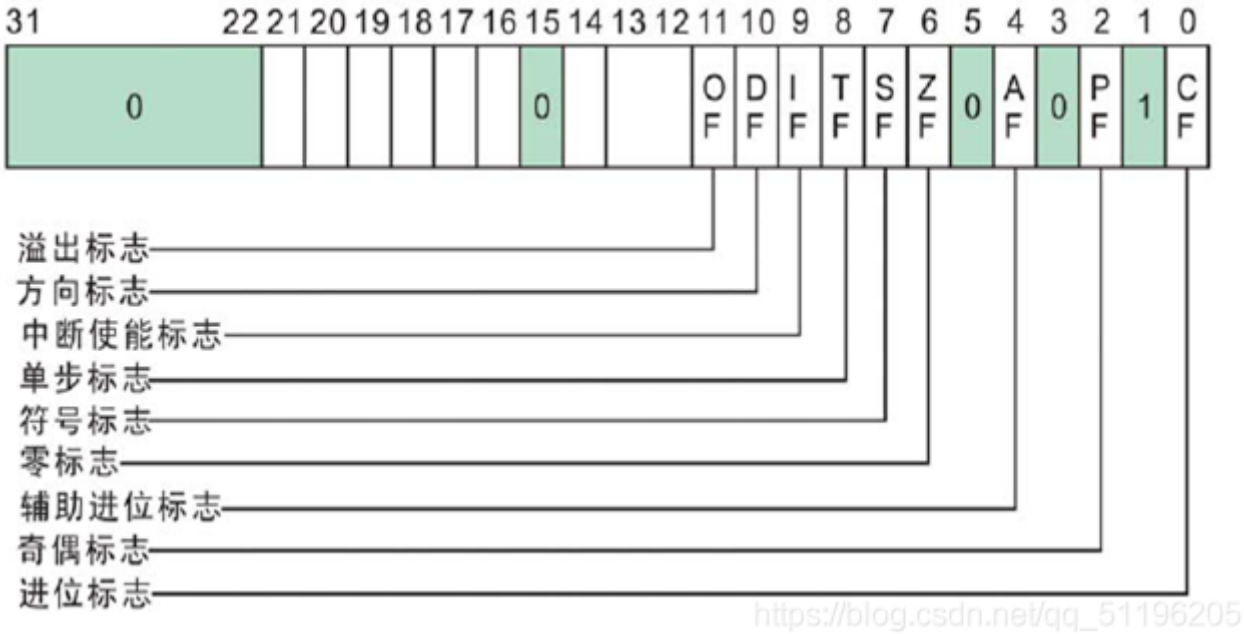


段寄存器

实地址模式下，段寄存器用于存放段的基址，段是一块预分配的内存区域。保护模式下，段寄存器存放段描述符表的指针(索引)。有些段存放程序的指令(代码)有些则存放变量(数据)，另外还有其他的段(名为堆栈段)存放着函数的局部变量和函数参数。

- CS：Code Segement，代码段寄存器
- SS：Stack Segment，栈段寄存器
- DS：Data Segment，数据段寄存器
- ES：Extra(Data) Segment，数据段寄存器
- FS：Data Segment，数据段寄存器
- GS：Data Segment，数据段寄存器

EFLAGS寄存器



由**控制CPU的操作**或**反映CPU某些运算的结果**的**独立二进制位**构成。有些机器指令可以测试和修改单个处理器标志。

当某标志等于**1**时就说其被**置位**；等于**0**时就说其**清除**(或复位)。

控制标志

控制标志控制CPU的操作。例如，某些控制标志可使CPU在每条指令执行后、检测到算术运算溢出后、进人虚拟8086模式或保护模式后中断。
程序可以通过设置EELAGS的单个位来控制CPU的操作。例如,设置方向标志位和中断标志位。

- 方向标志(DF)
  设置DF标志使得串指令自动递减（从高地址向低地址方向处理字符串），清除该标志则使得串指令自动递增
  STD以及CLD指令分别用于设置以及清除DF标志。

状态标志

状态标志反映了CPU执行的算术和逻辑运算的结果，包括溢出标志、符号标志、零标志、辅助进位标志、奇偶标志和进位标志。下列标志的名字后面列出了其对应的简写:

- **进位标志**(CF)：在无符号算术运算的结果太大而目的操作数无法容纳时置位。
- **溢出标志**(OF)：在有符号算术运算的结果是**较大**的正数或**较小**的负数，并且**目的操作数无法容纳**时置位。
- **符号标志**(SF):在算术或逻辑运算的**结果为负**时置位
- **零**标志(ZF)：在算术或逻辑运算的**结果为零**时置位。
- **辅助进位**标志(AC)：在算术运算导致**8位操作数的位3到位4产生进位**时置位.
- **奇偶**标志(PF)：结果的**最低有效字节**为**1的位**的数目为**偶数**时置位，否则PF复位。通常PF标志位用于在数据有可能被改变或丢失的情况下进行错误检查

系统标志

TF：将该位设置为1以允许单步调试，清零则禁用该模式

指令指针寄存器

指令指针寄存器EIP可以存放下一条机器指令的内存地址，跳转指令可以修改指令寄存器，使程序分支转移到新的地址执行。

IA-32内存管理

IA-32保护模式的内存管理比实地址模式要复杂，多任务、多用、段模式、页模式（页模式也是基于段模式的，通常称为段页式）。

平坦模式

当处理器运行于保护模式下时，每个程序可以寻址4GB的内存，地址范围是从十六进制数的0-FFFFFFFF。平坦内存模式非常易于使用，因为只需要使用一个32位整数就可以存放任何指令和变量的地址、处理器在后台进行地址的计算和转换，所有这一切对应用程序员都是透明的。

段寄存器 (CS，DS，SS，ES，FS和 GS)指向段描述符表，操作系统使用段描述符表定位程序使用的段的位置。一个典型的保护模式程序有个3段：代码段、数据段和堆栈段，使用CS，DS和SS三个段寄存器：

- CS包含描述符表中的代码段描述符
- DS包含描述符表中的数据段描述符
- SS包含描述符表中的堆栈段描述符

平坦分段模式

在平坦分段模式下，所有段都被映射到计算机的**32位物理地址空间**中。一个程序至少需要两个段：**代码段和数据段**。每个段都由一个**段描述符**定义，段描述符通常是一个存放在**全局描述符表**(GDT，Global Descriptor Table) 中的一个64位的值。图给出了一个基地址域指向内存中第一个可用地址(00000000)的段描述符，**段界限域**可用于表示系统中**物理内存的数量**，在当前的图中，段界限是0040。访问类型域包含了规定段如何使用的数据位。

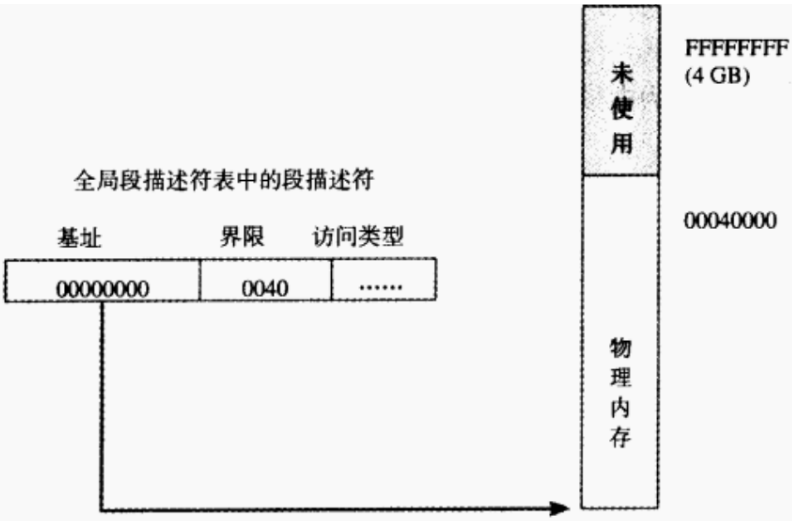


图 2.12　平坦分段模式

假设一台计算机有 256 MB 的内存，某个段描述符表示所有可用的物理内存，那么段界限域将包含十六进制值 10000，因为其值隐含地要乘以十六进制值 1000，最终得到十六进制值 10000000（256 MB）。

段管理

GDT（Global Descriptor Table）全局描述符表

- 整个系统只有一个GDT（64bit）
- Intel提供了一个寄存器GDTR用来存放GDT的入口地址

LDT（Local Descriptor Table）局部描述符表

- IA-32为LDT的入口地址也提供了一个寄存器LDTR
- 因为在任何时刻只能有一个任务在运行，所以LDTR也只需要有一个

段寄存器

index（13bits），段描述符在表中的索引

TI（1bit），0是GDT，1是LDT

RPL（2bits），Request Privilege Level，权限

- Ring 0, Kernel Mode
- Ring 3, User Mode

多段模式

在多段模式（multi-Segment model）下，每个任务或程序都有自己的段描述符表，称为局部描述符表（LDT，Local Descriptor Table）。每个描述符都可以指向一个与其他所有进程使用的段都不同的段，并且每个段都位于独立的地址空间中。图 2.13 中，LDT 的每个表项（段描述符）都指向内存中的一个不同的段，每个段描述符都指定了段的大小，例如从 3000 开始的段的大小是十六进制值 2000，其计算过程为（十六进制数值）0002 * 1000，而从 8000 开始的段大小为十六进制数值 A000。

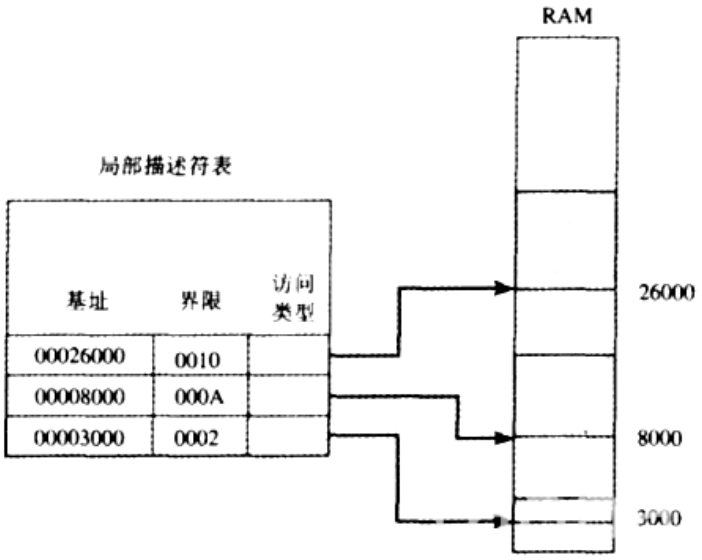


图 2.13　多段模式

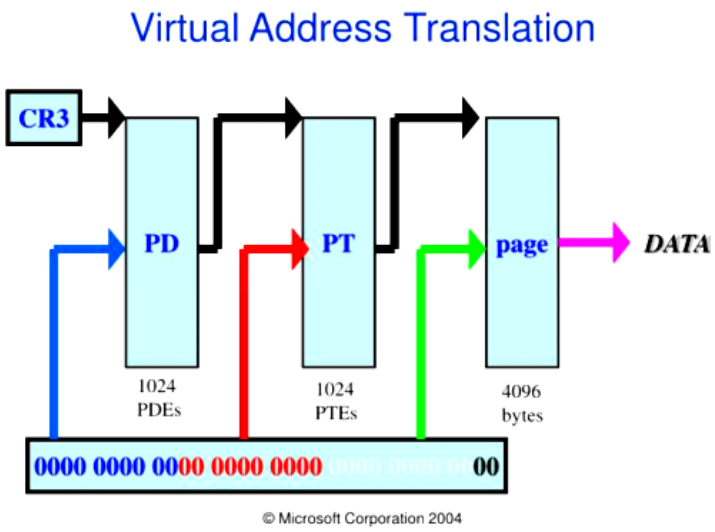
分页

IA-32 处理器支持一种称为分页（paging）的特性，允许一个段被分割成称为页（page）的 4096 字节的内存块。分页机制允许同时运行的程序使用的总内存远大于计算机的物理内存。操作系统映射的所有页的集合称为虚拟内存（virtual memory）。操作系统通常包含一个名为虚拟内存管理器的实用程序。

分页机制解决了一个一直困扰着软硬件设计者的难题：程序在运行前必须装入内存，但内存是非常昂贵的，用户总是想要在内存中装入大量程序并随意进行切换。另一方面，磁盘存储是廉价而海量的，不过访问磁盘要比访问主存储器慢得多。分页机制（通过使用后备磁盘存储）会使人产生内存几乎是无限大的错觉。然而，一个程序越依赖于分页机制，其运行也就可能越慢。

当任务运行时，如果程序的一部分当前未被使用，那么这部分可以保留在磁盘上。任务的一部分可能已经被换页（交换）到磁盘上了，任务的其他部分，如当前活跃的执行代码用到的页，可以保留在内存中。当处理器开始执行已经被换页交换出主存的代码时，将产生一个页错误（page fault），这将导致包含有所需代码及数据的页被重新载入内存。要想观察分页机制导致的页交换，读者可以找一台内存很少的计算机，并同时运行多个大型程序，读者应该能注意到从一个程序切换到另一个程序时会有明显的延迟，因为操作系统必须将每个程序交换出的部分从磁盘传输到主存。当安装了更多内存时，计算机会运行得更快，因为大型应用程序和文件可完全存放在内存中，这就减少了换页的数量。

分页机制



页目录表（PDT）的每一项元素称为页目录表项（PDE）
每个页目录表项指向一个**页表（PTT）**
每个页表的大小为**4KB**，即一个页表可以存储**1024个**页表项（PTE）

页表（PTT）的每一个元素称为页表项（PTE）
页表项（PTE）所指向的才是真正的物理页