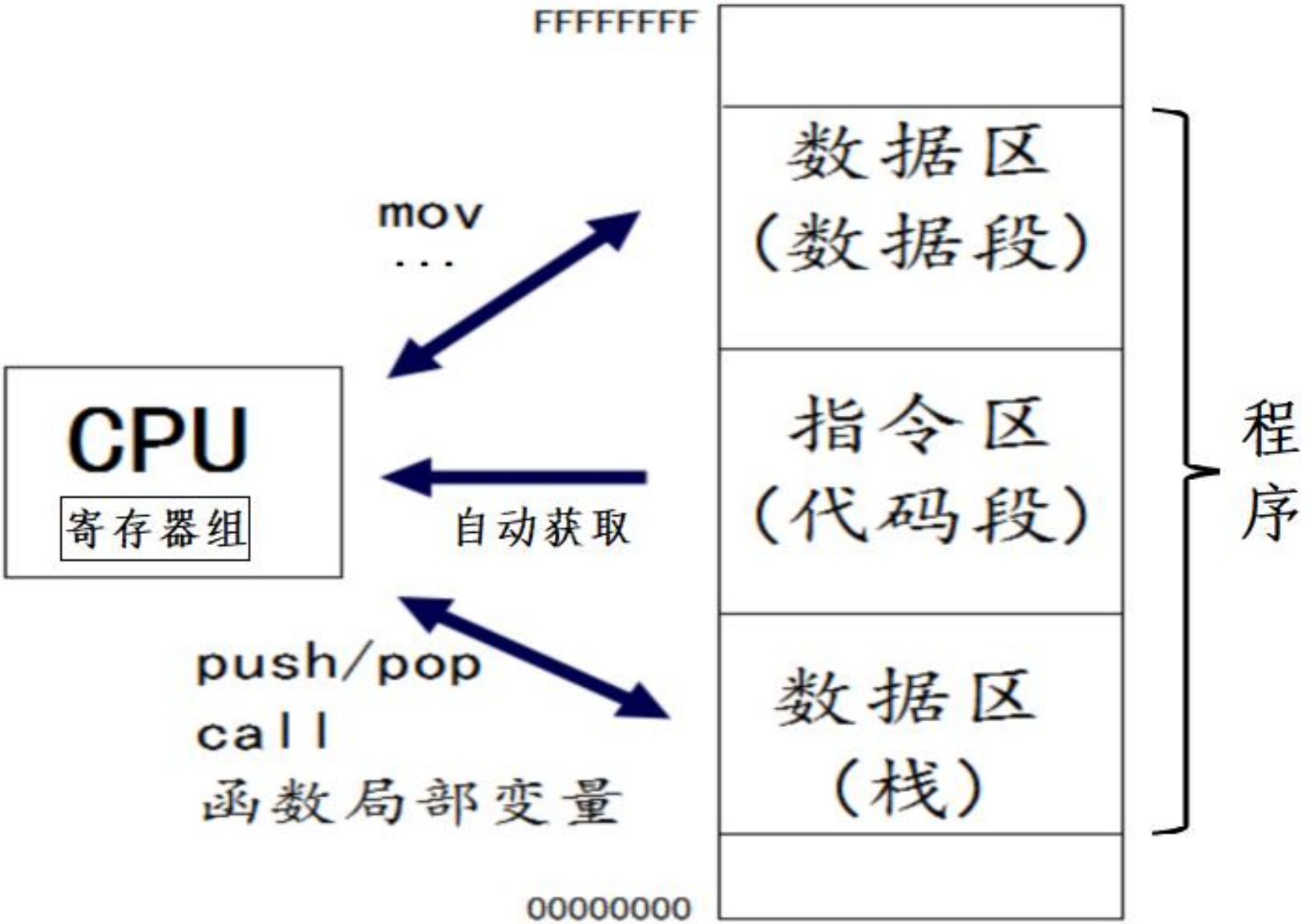
第 2 章
80x86寄存器与程序控制结构

内容

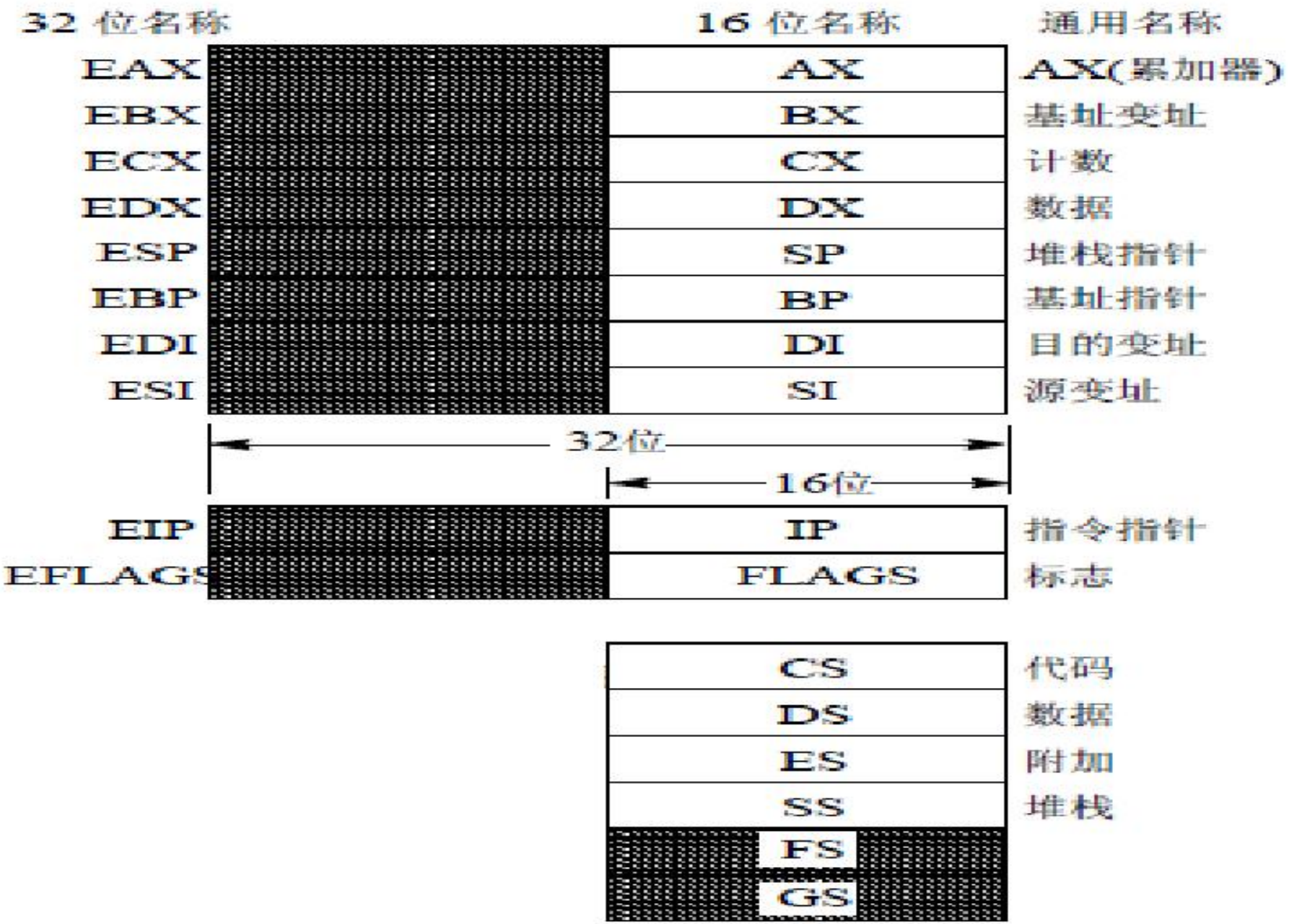
1. 80x86的程序可见寄存器
2. 分支结构与循环结构
3. CPU的三种模式
4. 存储器寻址

系统结构-程序员视角


FFFFFFFF
数据区
(数据段)
mov
...
CPU
寄存器组
指令区
(代码段)
自动获取
程序
push/pop
call
函数局部变量
数据区
(栈)
00000000

一、80x86的程序可见寄存器

- 寄存器：
  - 可见寄存器
  - 不可见寄存器
- 程序可见寄存器：在汇编语言中用到的寄存器。
  - 通用寄存器
  - 段寄存器
  - 专用寄存器



80x86的程序可见寄存器组

通用寄存器

- 数据寄存器

  32位：EAX，　EBX，　CX，　DX

  16位：AX，　BX，　CX，　DX

  8位：AH,AL，　BH,BL，　CH,CL,DH,DL

- 地址指针寄存器

  32位：ESP,EBP　16位：SP，BP

- 变址寄存器

  32位：ESI,EDI　16位：SI，DI

数据寄存器的习惯用法(1/2)

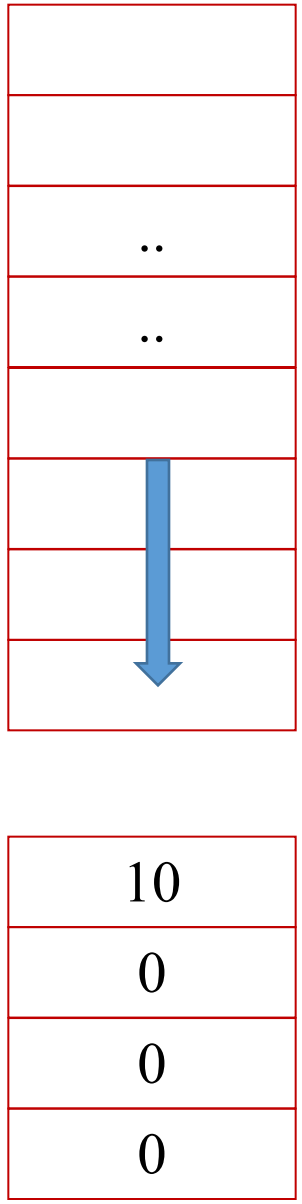
- EAX：累加器
  - 多用于存放中间运算结果
  - 所有I/O指令必须都通过AX与接口传送信息
- EBX：基址寄存器
  - 在间接寻址中用于存放基地址

数据寄存器习惯用法(2/2)

- ECX：计数寄存器
  - 用于在循环或串操作指令中存放循环次数或重复次数;
- EDX：数据寄存器
  - 在32位乘除法运算时，存放高32位数
- DX:
  - 在间接寻址的I/O指令中存放I/O端口地址

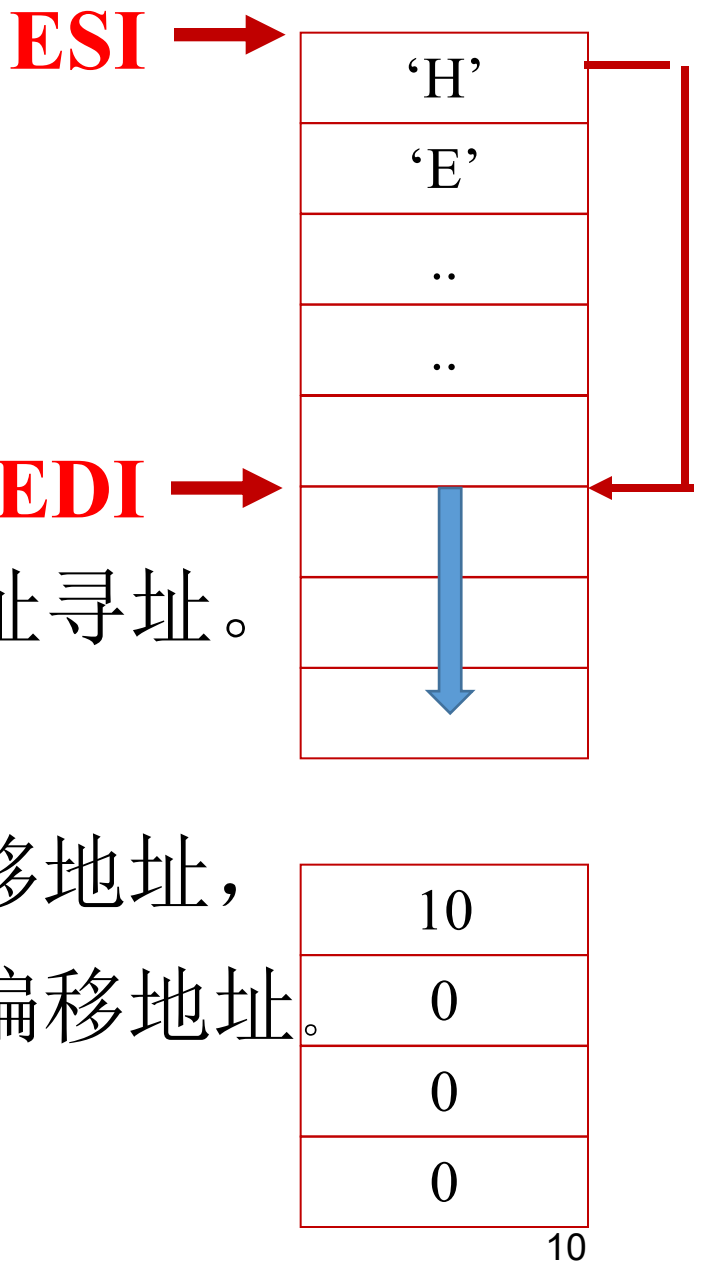
地址指针寄存器

- ESP：堆栈指针寄存器
  - 其内容为栈顶的偏移地址
- EBP：基址指针寄存器
  - 常用于在访问内存时存放内存单元的偏移地址



变址寄存器

- ESI：源变址寄存器
- EDI：目标变址寄存器
  - 用于指令的间接寻址或变址寻址。
  - 在串操作指令中
    - 用ESI存放源操作数的偏移地址，
    - 用EDI存放目标操作数的偏移地址。



段寄存器

存放逻辑段的段基地址信息

- CS：代码段寄存器
- DS：数据段寄存器
- ES：附加段寄存器

  数据段和附加段用来存放操作数
- SS：堆栈段寄存器
  - 存放返回地址，保存寄存器内容 ，传递参数
- FS、GS：从80386起增加两个附加的数据段寄存器。

专用寄存器

- EIP（32位） IP（16位）、

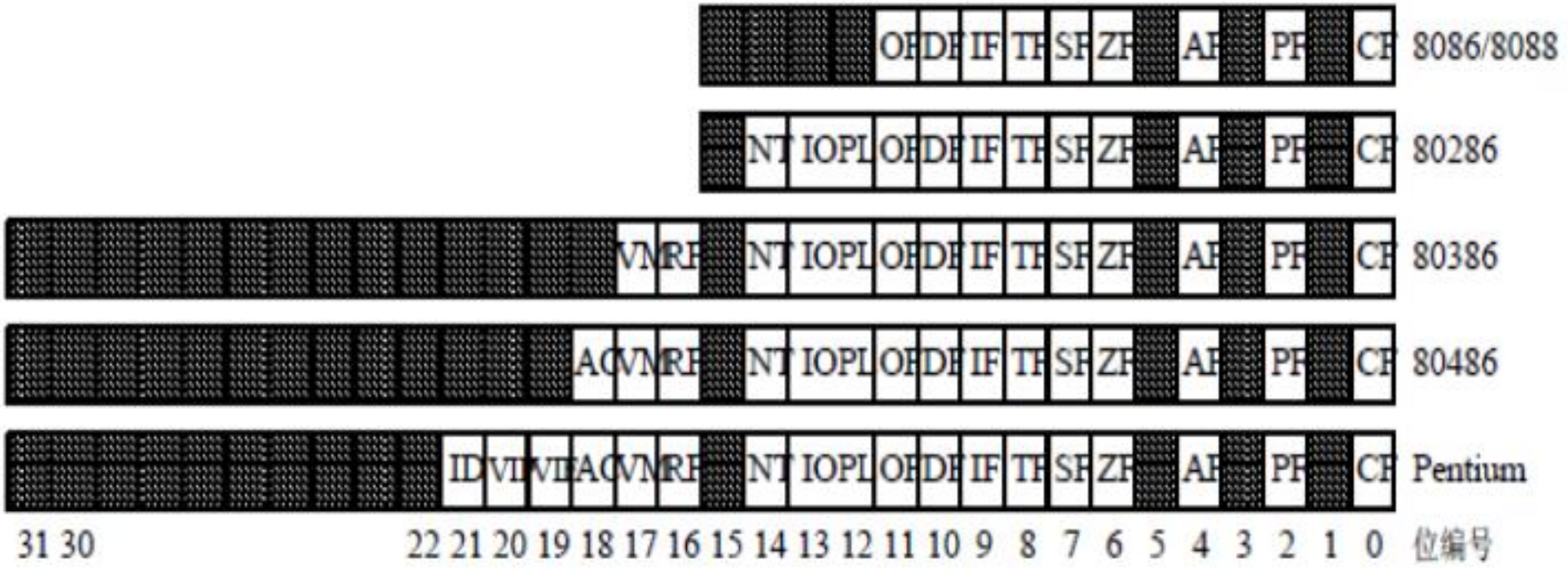
  指令指针寄存器，其内容为下一条要执行的指令的偏移地址，其中IP为EIP的低16位。

- FLAGS （16位）、EFLAGS（32位）

  标志寄存器，其中FLAGS为EFLAGS的低16位

  6个状态标志位(CF，SF，AF，PF，OF，ZF)：存放运算结果的特征

  3个控制标志位(IF，TF，DF)：控制某些特殊操作



80x86的标志寄存器

二、分支与循环结构的汇编实现

- 比较指令：　　cmp x, y
- 跳转指令：　　jmp, jXXX（ja, jb, jz）

cmp x, y 语义：

执行操作 x-y (x与y的值不变),根据操作结果改变EFLAG相应的位。

ja loc: 若x与y是无符号数(程序员定义)且x>y,则程序跳转到地址loc处执行

jz/je loc: 若x与y是无符号数(程序员定义)且x==y, 则程序跳转到地址loc处执行

jb loc: 若x与y是无符号数(程序员定义)且x<y,则程序跳转到地址loc处执行

jg loc: 若x与y是有符号数(程序员定义)且 x>y,则程序跳转到地址loc处执行

jz/je loc: 若x与y是有符号数(程序员定义)且x==y,则程序跳转到地址loc处执行

jl loc: 若x与y是有符号数(程序员定义)且 x<y,则程序跳转到地址loc处执行

简单条件分支结构汇编实现

简单条件：两整数大小

```
if(x1<y1) {
    Block_1 ;
} else if(x2>y2){
    Block_2;
} else if(x3<y3){
    Block_3;
}else {
    Block_4;
}
```

```
        cmp  x1,y1
        jnb  Loc2
        Block_1
        jmp  Final
Loc2:  cmp x2, y2
        jna Loc3
         Block_2
        jmp Final
Loc3:  cmp x3,y3
        jnb Loc4
         Block_3
        jmp Final
Loc4:  Block_4
        jmp Final;(可略)
Final:
```

指令简介

- 数据移送指令：mov dest，src
- 函数调用指令：Call f
- **输入输出函数**

ReadInt PROC uses ebx ecx edx esi

输入：无

返回值：

- CF=0， 输入存在EAX
- CF=1， 输入无效，EAX=0

WriteInt proc

输入：显示的整数存在EAX

返回值：　　无

;

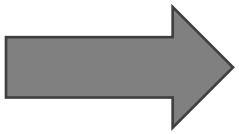
例1

求整数 a与b最大值,并在屏幕中输出最大值。

步骤1： 算法设计

```
if a > b  then
    max=a
else
    max=b
```

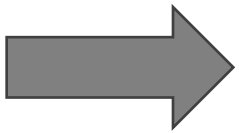
算法汇编实现--逐步转换



```
        cmp  a, b
        jna  maxb
        max=a
        jmp  final
maxb:   max=b
final :
```

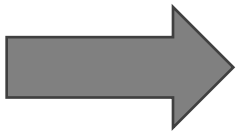
算法汇编实现

```
        mov  eax, a
        cmp  eax, b
        jna   maxb
        mov max,a
        jmp  final
maxb: mov  max, b
final :
```



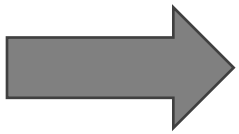
```
        mov  eax, a
        cmp  eax, b
        jna    maxb
        mov   eax, a
        mov  max,eax
        jmp  final
maxb: mov   eax, b
        mov  max,eax
final :
```

算法汇编实现



```
        mov  eax, a
        cmp  eax, b
        jna  maxb
        mov  max,eax
        jmp  final
maxb: mov eax, b
        mov  max, eax
final :
```

算法汇编实现



```
        mov  eax, a
        cmp  eax, b
        jna     maxb
        jmp final
maxb: mov eax,b
final: ;eax是最大值
```

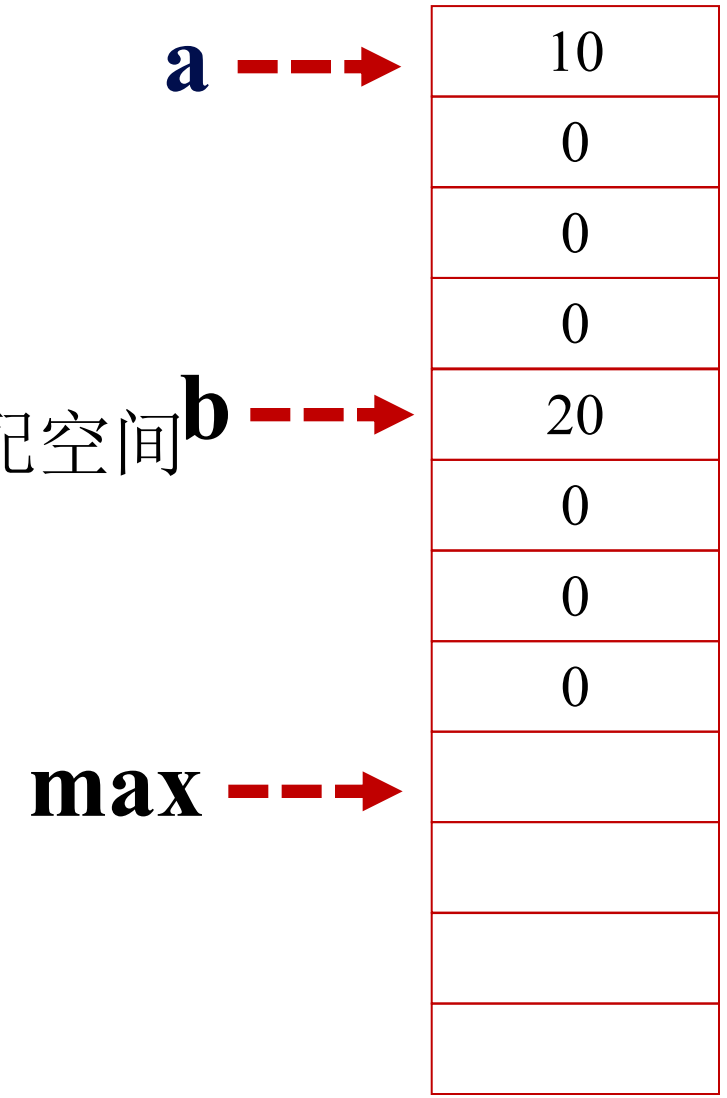
步骤2 编写程序

步骤2.1 包含头文件

```
Include Irvine32.inc
```

步骤2.2 定义数据段---为数据分配空间

```
.data
    a    dd   10h
    b    dd   20h
    max  dd   ?
```



步骤2.3 编写指令

```
 .code
 main PROC
        mov  eax, a
        cmp  eax, b
        jna     maxb
        jmp  final
maxb: mov   eax, b
final :  call  writeint
        exit
 main ENDP
 END main
```

步骤3 生成可执行程序

make32 ex01

步骤4 运行可执行程序

ex01

比较汇编源代码与反汇编代码 (发现了什么？)

```
main   PROC
       mov  eax, a
       cmp  eax, b
       jna  maxb
       jmp  final
maxb : mov  eax, b
final: call writeint
       exit
main ENDP
```

```
00401005 JMP 24-4.00401010
0040100A INT3
0040100B INT3
0040100C INT3
0040100D INT3
0040100E INT3
0040100F INT3
00401010 MOV EAX,DWORD PTR DS:[404000]
00401015 CMP EAX,DWORD PTR DS:[404004]
0040101B JBE SHORT 24-4.0040101F
0040101D JMP SHORT 24-4.00401024
0040101F MOV EAX,DWORD PTR DS:[404004]
00401024 CALL 24-4.0040189A
00401029 PUSH 0
0040102B CALL <JMP.&KERNEL32.ExitProcess>
00401030 INT3
00401031 INT3
```


00401005 $ E9 06000000 JMP 24-4.00401010
0040100A CC INT3
0040100B CC INT3
0040100C CC INT3
0040100D CC INT3
0040100E CC INT3
0040100F CC INT3
00401010 > A1 00404000 MOV EAX,DWORD PTR DS:[404000]
00401015 . 3B05 04404000 CMP EAX,DWORD PTR DS:[404004]
0040101B . 76 02 JBE SHORT 24-4.0040101F
0040101D . EB 05 JMP SHORT 24-4.00401024
0040101F > A1 04404000 MOV EAX,DWORD PTR DS:[404004]
00401024 > E8 71080000 CALL 24-4.0040189A
00401029 . 6A 00 PUSH 0
0040102B . E8 58090000 CALL <JMP.&KERNEL32.ExitProcess>
00401030 CC INT3
00401031 CC INT3
00401032 CC INT3
00401033 CC INT3
机器指令
汇编指令
a
b
变量a
变量b
Address Hex dump
00404000 10 00 00 00 20 00 00 00
00404008 00 00 00 00 00 00 00 00
00404010 00 20 20 20 20 20 20 20
小端编码

例2

从键盘输入两个整数，求这两个整数 的最大值,并在屏幕中输出最大值

步骤2 编写程序

步骤2.1 包含头文件

```
Include Irvine32.inc
```

步骤2.2 定义数据段---为数据分配空间

```
.data
    a dd ？     ;注意这里改变了！！！
    b dd ？
```

步骤2.3 编写指令

```
.code
main PROC
        mov  eax, a
        cmp  eax,  b
        ja   output
        mov eax, b
output:
        call  writeint
        exit
main ENDP
END main
```

在此处插入

```
call    readint
mov  a, eax
call    readint
mov   b,eax
```

步骤3 生成可执行程序

步骤4 运行可执行程序

例3. 生成10个伪随机整数,并将所生成的数保存在内存。

需解决的问题：

- 如何生成伪随机数： RandomRange
- 如何在内存中保存10个整数：数组
- 如何表示数组元素
- 如何表示循环结构

用到的知识：

1. 生成伪随机整数

```
mov   eax,1000
call randomrange; eax为0..999之间伪随机整数
```

2. 定义数组

```
arr dword 10 dup(?);申请4*10个字节空间
```

3. 数组第i个元素首地址

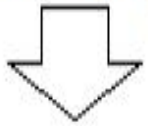
```
arr[4*i]
```

汇编以下程序，用调试器观察数据段数据

```
include irvine32.inc
.data
    a dword 1, 2, 3, 4, 5, 6, 7
.code
start:
        mov eax, a
        call writeint
        exit
end start
```

汇编成可执行文件，用ollydbg调试器打开该文件，可得如下：

mov eax,a



```
00401000  A1 00404000  mov   eax, dword ptr [00404000]
00401005  E8 70080000  call  WriteInt
0040100A  6A 00        push  0
0040100C  E8 57090000  call  ExitProcess

00404000  01 00 00 00 02 00 00 00 03 00 00 00 04 00 00 00
00404010  05 00 00 00 06 00 00 00 07 00 00 00 00 00 00 00
```

循环结构的汇编实现

```
for(i=0;i<10;i++) {
    BLOCK;
}
```

语义：

```
    i=0;
again: if (i<10){
    BLOCK;
    i=i+1;
    goto again
}
```

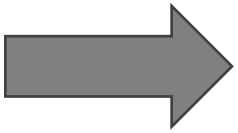
```
        mov   i,0
again:  cmp   i,10
        jnb   final
        BLOCk
        add i,1
        jmp again
final:
```

```
include irvine32.inc
.data
   arr dd 10 dup(?)
.code
main proc
      mov  ebx,0
again:cmp  ebx,10
      jnb  final
      mov  eax,1000
      call randomrange
      mov  arr[4*ebx],eax
      add  ebx,1
      jmp  again
final: exit
main  endp
end   main
```

例4. 在内存中存有10个整数，求这10整数最大值,并在屏幕中输出最大值

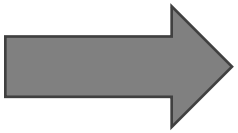
算法：

```
max=arr[0];
For(i=0;i<10;i++){
    if max<arr[i] then
        max=arr[i];
}
```

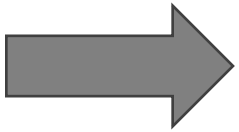


```
        max=arr[0];
        i=0;
again:  cmp i,10
        jnb  final
        if max<arr[i] then
            max=arr[i];
        add i,1
        jmp again
final:
```

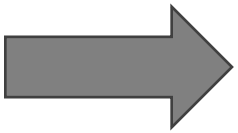
```
max=arr[0];
For(i=0;i<10;i++){
    if max<arr[i] then
        max=arr[i];
}
```



```
        max=arr[0];
        i=0;
again:  cmp i,10
        jnb  final
        cmp max, arr[i]
        jnl  next
        mov max, arr[i]
next:
        add i,1
        jmp again
final:
```



```
        mov max, arr[0];
        mov  i, 0;
again:  cmp i,10
        jnb  final
        cmp max, arr[i]
        jnl  next
        mov max, arr[i]
next:    add i,1
        jmp again
final:
```



```
        mov eax,  arr[0]     ;  eax=max
        mov  ebx, 0          ;      ebx=i
again: cmp ebx,10
        jnb  final
        cmp eax, arr[4*ebx]
        jnl  next
        mov eax, arr[4*ebx]
next:    add ebx,1
        jmp again
final:
```

新知识：循环指令

➢**循环指令　loop**

```
    mov ecx, count;
again:
      .......
      ......
    loop again
```

Loop语义(操作语义）：

1. ecx ←- ecx - 1
2. Ecx ==0 ?
   - 2.1 是， 循环结束
   - 2.2 否，跳转至 标号again处

使用循环指令实现代码

```
main PROC
        mov  eax,  arr[0] ;eax存放最大值
        mov  ebx,  0;  ebx存放数组元素下标
        mov   ecx, 10
again: cmp  eax,  arr[4* ebx]
         jnl   next
         mov  eax, arr[4* ebx]
next:   add  esi, 1
        loop again
         call  writeint
        exit
main ENDP
```

3.编程练习

问题1(猴子吃桃)：猴子第一天摘下若干个桃子，当即吃了一半，还不过瘾，又多吃了一个。第二天早上又将第一天剩下的桃子吃掉一半，有多吃了一个。以后每天早上都吃了前一天剩下的一半零一个。到第 10 天早上想再吃时，发现只剩下一个桃子了。编写程序求猴子第一天摘了多少个桃子。

问题2:从键盘输入两个正整数 a 和 b，求其最大公约数

1.猴子吃桃C算法

```
#include <stdio.h>
int main( ) {
   int day, x1, x2;      /*定义3 个变量day、x1、x2 */
   day=9;
   x2=1;
   while(day>0)   {
      x1=(x2+1)*2; /*前一天的桃子数是当天桃子数加1后的2倍*/
      x2=x1;
      day--;   /*因为从后向前推所以天数递减*/
   }
   printf("the total is %d\n",x1);   /* 输出桃子的总数*/
   return 0;
}
```

2.求最大公约数 C算法

```
#include <stdio.h>
int main(){
    int a,b,c
    printf("请输入两个数:\n");
    scanf("%d%d",&a,&b);
    c=a%b;
    while(c!=0)  {
        a=b;
        b=c;
        c=a%b;
    }
    printf("最大公约数是:\n%d\n",b);
  return 0;
}
```

思考题

1.从键盘输入10个整数，并存放在内存中，求这10整数 最大值,并在屏幕中输出最大值

2. 在内存存放有两组整数中，分别求两组整数最大值,并在屏幕中输出各自最大值

3.内存中有10个分布在0至100内的正整数，试编程统计大于或等于60的数的个数num1,小于60的数的个数num2。

4.内存中有10个分布在0至100内的正整数，试编程求统计小于60的数的个数num1,大于或等于60且小于80的数的个数num2, 大于或等于80且小于100的数的个数num3。

此页之后为课后阅读内容，请同学们课后完成

三、 80x86 CPU的三种工作模式

1.实模式

与8086兼容的工作模式，只有低20位地址线起作用，仅能寻址第一个1MB的内存空间。MS DOS运行在实模式下。

2.保护模式

32位80x86 CPU的主要工作模式，提供对程序和数据进行安全检查的保护机制。Windows 9x/NT/2000运行在保护模式下。

3.虚拟8086模式

在Windows 9x下，若打开一个MS DOS窗口，运行一个DOS应用程序，那么该程序就运行在虚拟8086模式下。

四、实模式下存储器寻址

1. 内存分段

- 80x86采用分段内存管理机制，主要包括下列几种类型的段：

  代码段：用来存放程序的指令序列。

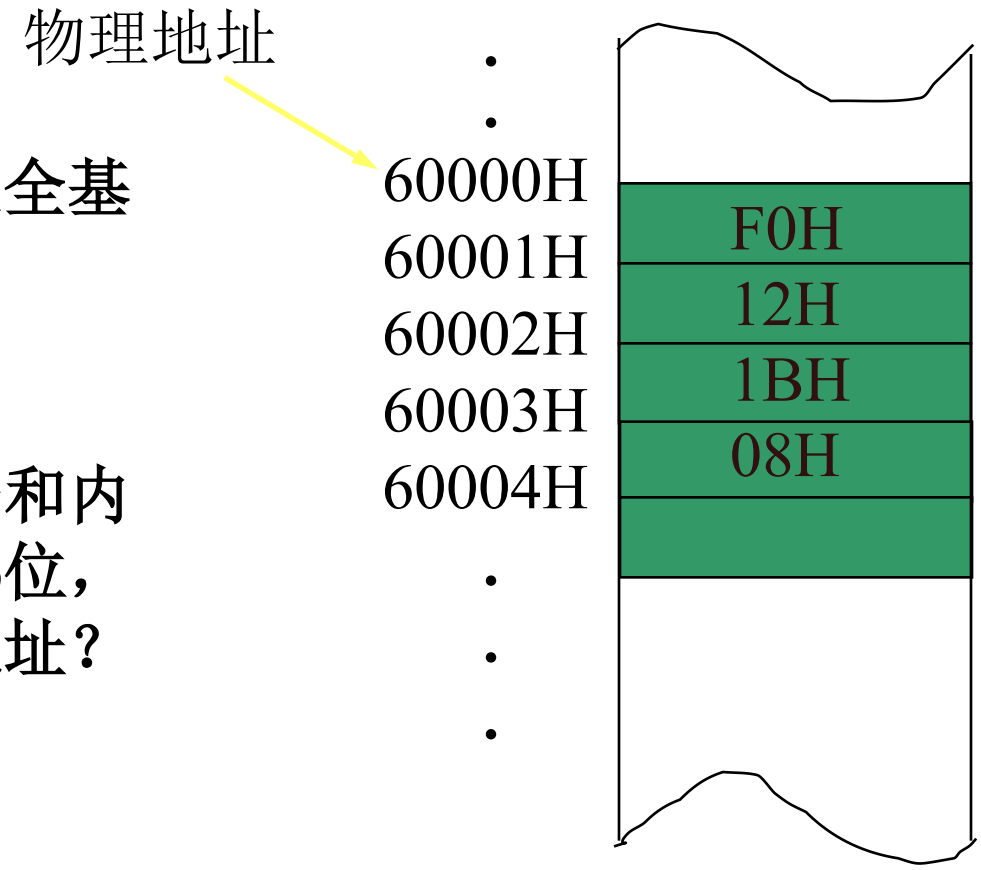
  数据段：用来存放程序的数据。

  堆栈段：作为堆栈使用的内存区域，用来存放过程返回地址、过程参数等。

- 一个程序可以拥有多个代码段、多个数据段甚至多个堆栈段。

物理地址

- **8086/88**： 20根地址线，可寻址2^{20}(1MB)个存储单元
- CPU送到AB上的20位的地址称为**物理地址**



存储器的操作完全基于物理地址。

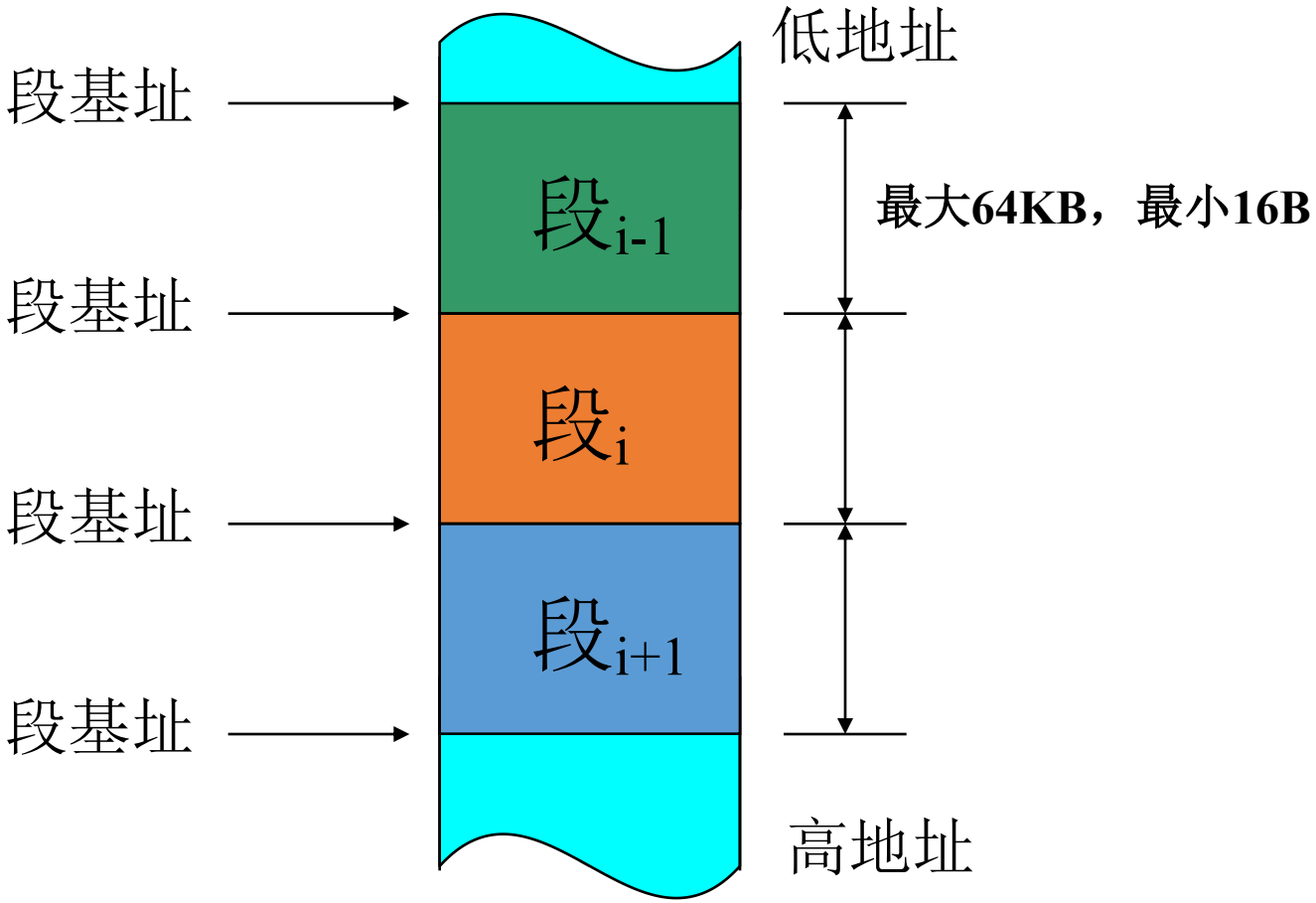
- 问题：

8088的内部总线和内部寄存器均为16位，如何生成20位地址？

- 解决：

存储器分段

存储器分段

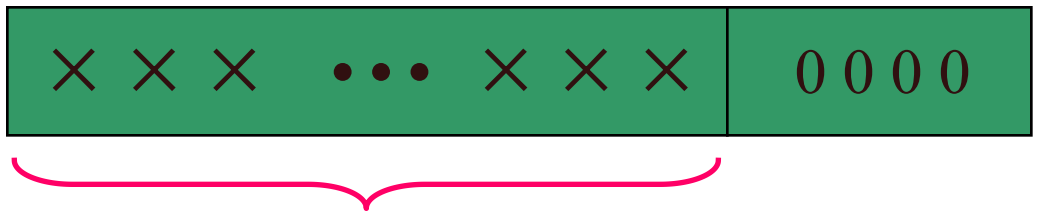


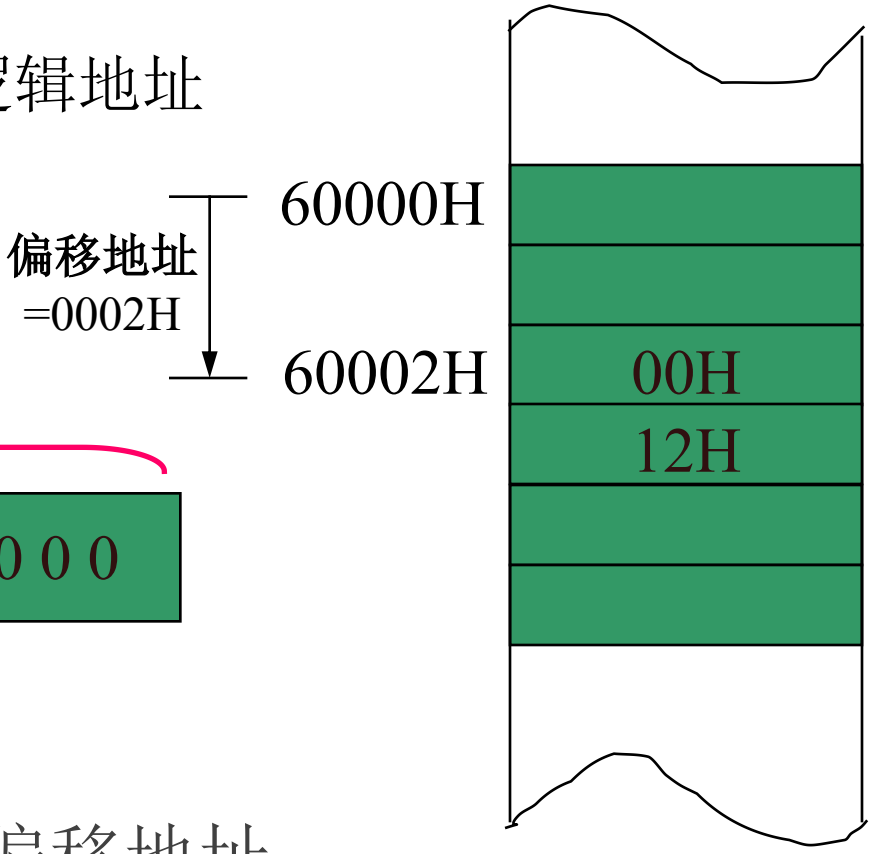
逻辑地址

- 段基地址和段内偏移组成了逻辑地址

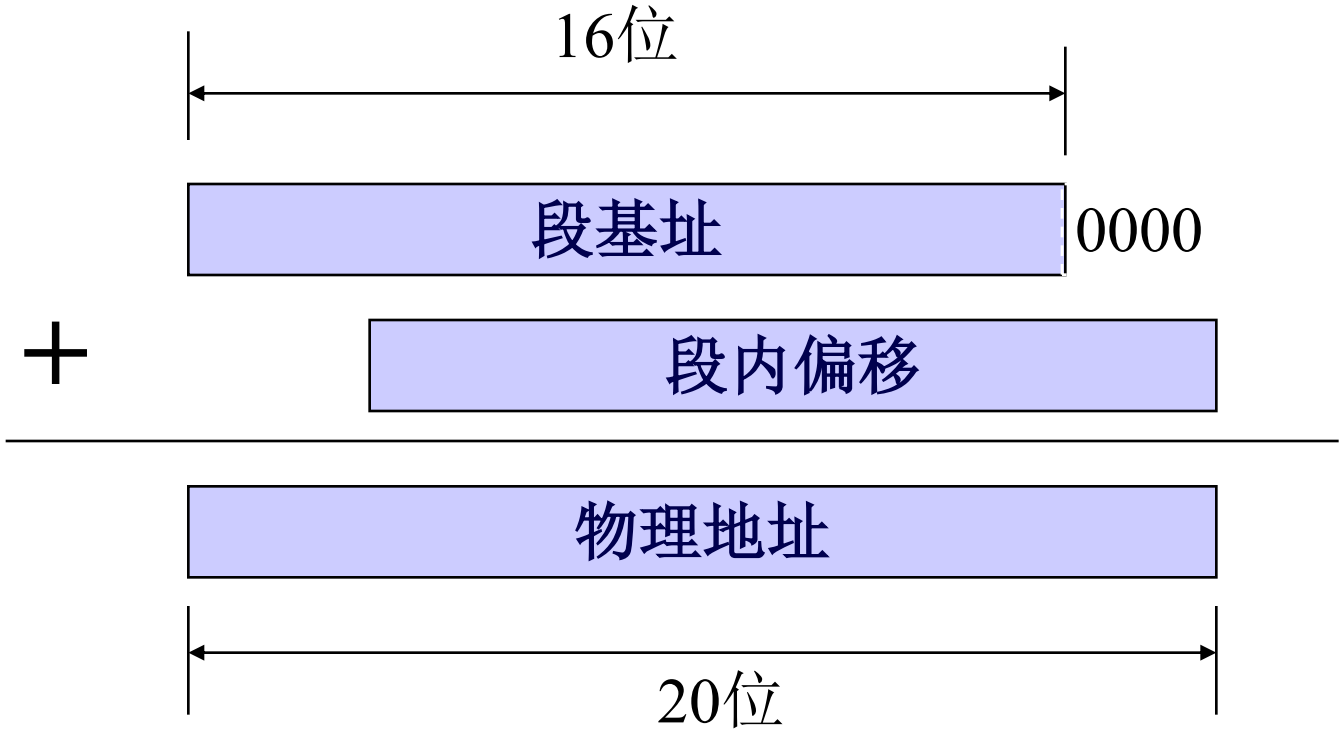
段地址　偏移地址(偏移量)

格式为：段地址:偏移地址





物理地址=段基地址×16+偏移地址



- BIU(总线接口单元）中的地址加法器用来实现逻辑地址到物理地址的变换
- 8086～80286的程序允许访问4个存储段，4个段寄存器中的内容指示了每个段的基地址。其他的80X86程序可允许访问6个段。

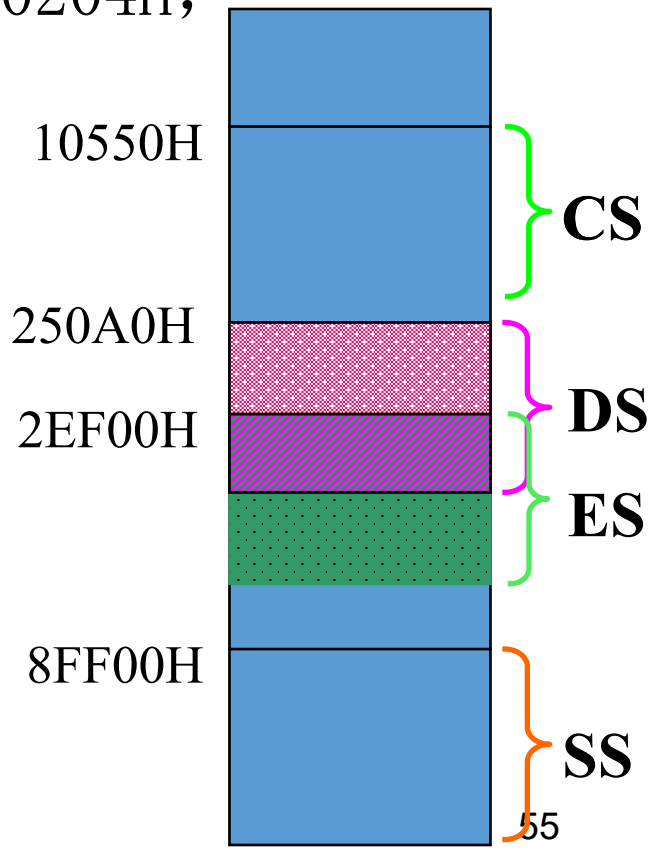
[例]:

- 已知CS=1055H，DS=250AH，ES=2EF0H，SS=8FF0H，
  DS段有一操作数，其偏移地址=0204H，
  1)画出各段在内存中的分布
  2)指出各段首地址
  3)该操作数的物理地址=?

解：　各段分布及段首址见右图所示。

操作数的物理地址为：

250AH×10H+0204H = 252A4H



堆栈及堆栈段的使用

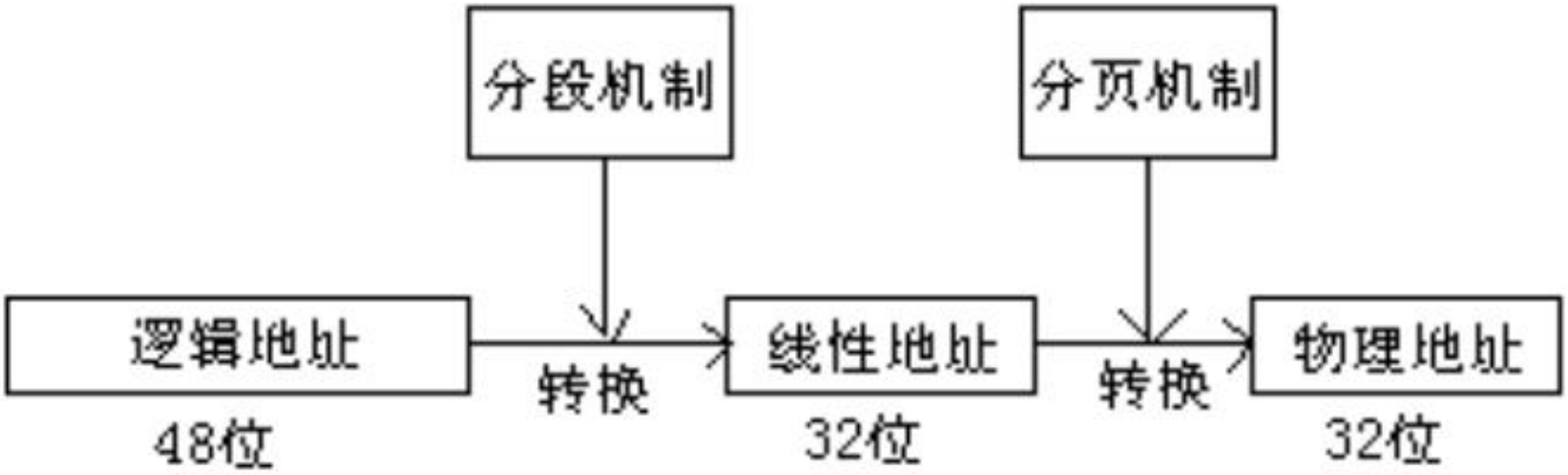
- 内存中一个按FILO方式操作的特殊区域
- 每次压栈和退栈均以DWORD为单位
- SS存放堆栈段地址信息，ESP存放段内偏移，SS:ESP构成了堆栈指针
- 堆栈用于存放返回地址、过程参数或需要保护的数据
- 常用于响应中断或子程序调用

五、保护模式存储器寻址*

在保护模式存储器寻址中，程序员在程序中指定**逻辑地址**，**逻辑地址由选择器和偏移地址组成**。其中选择器存放在段地址中。

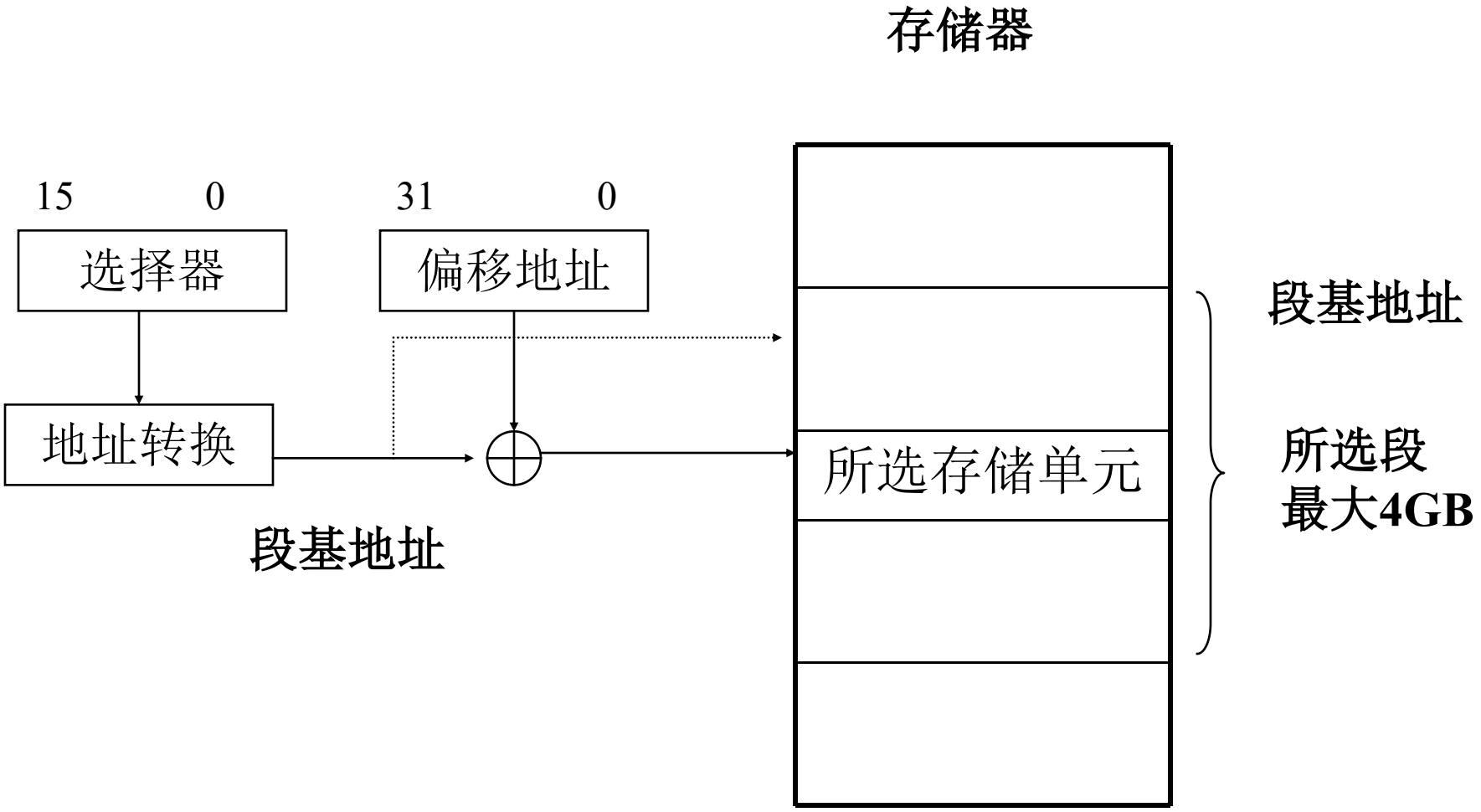
系统根据**选择器**的内容按指定的途径找到所选段对应的**描述符**，根据其给出的基地址和界限值，确定所要找的存储单元所在的段，加上逻辑地址中指定的偏移地址，确定相应的存储单元。

描述符：用来描述段的大小、段在存储器中的位置及其控制和状态信息，由基地址、界限、访问权限和附加字段组成。

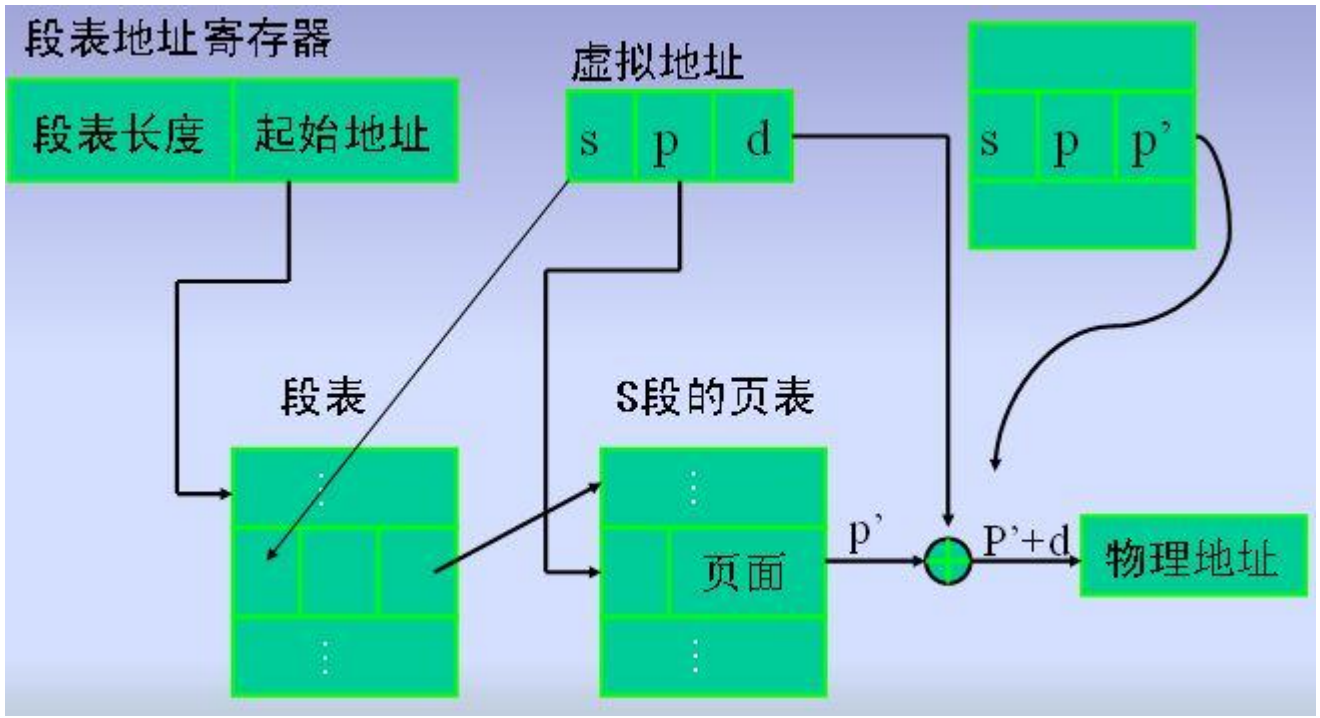


80386 中的地址转换

保护模式存储器寻址示意图




32bit 线性地址
31
22 21
12 11
0
页表目录
页表
页内偏址
(10bit)
(10bit)
(12bit)


段表地址寄存器
段表长度
起始地址
虚拟地址
s
p
d
s
p
p'
段表
S段的页表
页面
p'
P'+d
物理地址