

# 第三章

## 80x86寻址方式与子程序

## 主要内容：

- 指令系统的一般概念
- 80x86寻址方式
- 80x86指令系统
- 编程示例

## 3.1 概述

**指令：** 控制计算机完成指定操作的命令

**机器指令：** 指令的二进制代码形式。

例如： **CD21H**

**汇编指令：** 助记符形式的指令。

例如： **INT 21H**

**指令系统：** CPU所有指令及其使用规则的集合

# 指令按功能分为六大类（92种）

- 数据传送类
- 算术运算类
- 逻辑运算和移位
- 串操作
- 控制转移类
- 处理器控制

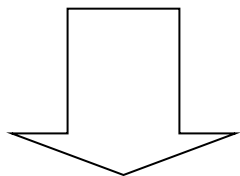
# 一、指令的基本构成

目的

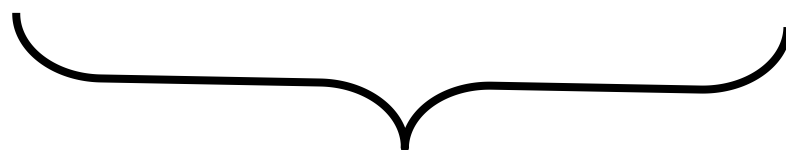
源

操作码

[操作数], [操作数]



说明要执行的  
是什么操作



操作对象，可以  
有0个、1个或2个

指令举例：

MOV EAX, EBX

ADD EAX, [ESI+6]

INC EAX

HLT

## 二、80x86的寻址方式

### 寻址方式

- 寻找操作数的方法
- 寻找转移指令地址
- CALL指令的转向地址

### 与数据有关的寻址方式

### 与转移地址有关的寻址方式

# 寻址方式

立即寻址

直接寻址

寄存器寻址

寄存器间接寻址

寄存器相对寻址

基址-变址寻址

基址-变址相对寻址



# 一、与数据有关的寻址方式

## 【1】立即寻址

- 操作数(为一常数)直接由指令给出 (此操作数称为立即数)
- 立即寻址只能用于源操作数，且源操作数长度和目的操作数长度相等。

例：

MOV      AX,    1C8FH

MOV      BYTE PTR[2A00H], 8FH

错误例：

✗ *MOV      2A00H,    AX           ; 错误!*

## 【2】 寄存器寻址

- 操作数放在某个寄存器中
- 源操作数与目的操作数字长要相同
- 寄存器寻址与段地址无关
- 例：

MOV      AX,    BX

MOV      ECX,  EDX

MOV      CL,    AL

错误例：

✗ MOV      AX,    BL                    ; 字长不同

✗ MOV      ES:  AX,   DX               ; 寄存器与段无关

## 【3】直接寻址

- 指令中直接给出操作数的32位偏移地址

**偏移地址**也称为**有效地址**(EA, Effective Address)

- 默认的段寄存器为DS，但也可以显式地指定其他段寄存器——称为段跨越前缀
- 偏移地址也可用符号地址来表示，如ADDR、VAR  
例：

```
MOV  AX, [2000H]
```

```
MOV  EDX, [2A00H]
```

```
MOV  SI, TABLE_PTR
```

## 【4】 寄存器间接寻址

- 操作数的偏移地址(有效地址EA)放在寄存器中, 操作数在存储器中
- 32位寻址时, 可用寄存器EAX、EBX、ECX、EDX、ESP、EDP、ESI、EDI等8个通用寄存器。ESP、EBP等寻址寄存器的默认段为SS段, 其他寄存器的默认段为DS寄存器。

例: MOV AX, [EBX]

MOV CL, [EDI]; CS指定其他段

错误例 :

× MOV AX, [DX]

× MOV CL, [AX]

## 【5】 寄存器相对寻址

- EA=基址或变址寄存器的内容加指令中指定的位移量
- 32位寻址时，可用寄存器：EAX、EBX、ECX、EDX、ESP、EDP、ESI、EDI等8个通用寄存器。ESP、EBP等寻址寄存器的默认段为SS段，其他寄存器的默认段为DS寄存器。

常用于存取表格或一维数组中的元素——把表格的起始地址作为位移量，利用修改基址或变址寄存器的内容来取得表格中的值

例: **MOV AX, [EBX+8]**  
**MOV CX, TABLE[ESI]**  
**MOV AX, [EBP+1000H] ; 默认段寄存器**  
**为SS**

## 【6】 基址-变址寻址

- 由一个基址寄存器的内容和一个变址寄存器的内容相加而形成操作数的偏移地址，称为基址-变址寻址。
  - 同一组内的寄存器不能同时出现。

• 例:

**MOV AX, [EBX] [ESI]**

**MOV AX, [EBX+ESI]**

**MOV EAX, [EBP] [EDI]**

**MOV EDX, [EBX] [ESI]**



## 【7】 相对的基址-变址寻址

- 在基址-变址寻址的基础上再加上一个相对位移量
- 注意事项同基址-变址寻址
- 这种寻址方式通常用于二维数组的寻址。

例：

**MOV AX, BASE [EBX] [ESI]**

**MOV AX, [BASE +EBX] [ESI]**

**MOV AX, [BASE + EBX+ESI]**

使用相对的基址-变址寻址方式可以很方便地访问二维数组。

位移量

数组首地址

(偏移地址)

基址寄存器

数组元素行址

(行位移地址)

变址寄存器

数组元素列址

(行内元素下标)

二维数组例： 内存图示（按行存储）

$$A = \begin{bmatrix} 1 & 8 & 3 \\ 2 & 5 & 2 \\ 4 & 0 & 9 \end{bmatrix}$$

寻址格式:

$\text{DATA}[\text{reg1}][\text{reg2}] = [\text{DATA} + \text{reg1} + \text{reg2}]$

语义—物理地址:

$[\text{DATA}] + [\text{reg1}] + [\text{reg2}]$

➤ DATA

➤ [reg]

➤ [reg1][reg2]

➤ DATA[reg]

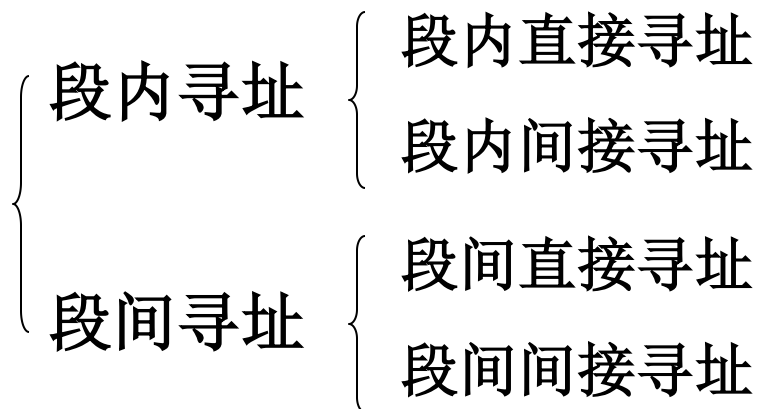
➤ DATA[reg1][reg2]

## 寄存器间接、寄存器相对、基址变址、相对基址变址四种寻址方式的比较：

寻址方式	指令操作数形式
寄存器间接	只有一个寄存器
寄存器相对	一个寄存器加上位移量
基址—变址	两个不同类别的寄存器
相对基址-变址	两个不同类别的寄存器加上位移量

## 二、与转移地址有关的寻址方式

用来确定 转移指令 及 CALL指令 的转向地址。



段内：转移指令与转向的目标指令在**同一段代码**中, **(CS)不变**。

段间：转移指令与转向的目标指令在**两个代码段**中, **(CS)变化**。

以 转移（条件转移/无条件转移）指令为例

# 表示转移距离（称为位移量）的操作符：

- **SHORT** 短转移，位移量用1个字节表示，  
在-128 ~ 127字节之间。

**JMP SHORT NEXT**

- **NEAR** 近转移，在同一段内转移，位移量用1个字表示  
在-32768~32767字节范围内。

**JMP NEAR PTR AGAIN**

- **FAR** 远转移，表示转移距离超过±32K字节，  
或是在不同段之间转移。

**JMP FAR PTR WAIT**

**PTR----**指针操作符

# 段内直接寻址

转向的有效地址EA =

(EIP)<sub>新</sub> ←

(EIP)<sub>当前</sub>

+

位移量( 8bit / 16bit )

例：段内直接寻址方式的示例

1060:000D EB04

**JMP SHORT NEXT**

EIP当前值→ 1060:000F ...

1060:0011 ...

1060:0013 0207

**NEXT: ADD AL,[BX]**

**EA 000F+04=0013**

**CS 不变**

# 段内间接寻址

转向的有效地址EA是一个寄存器或存储单元的内容。

**JMP EBX**

**JMP TABLE[EBX]**

**JMP TABLE[EBX]**

**JMP [EBX][ESI]**

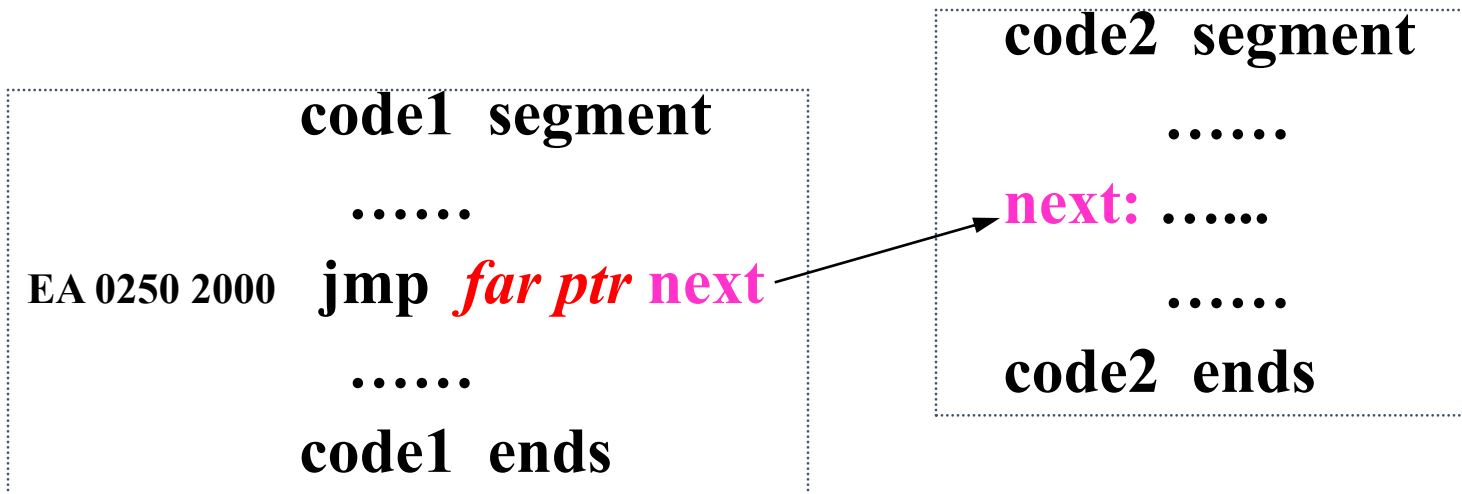


# 段间直接寻址(16位)

用指令中提供的转向段地址和偏移地址取代CS和IP

$$\text{物理地址} = 16d \times (\text{CS}) + (\text{IP})$$

例:



**far ptr:** 段间转移操作符

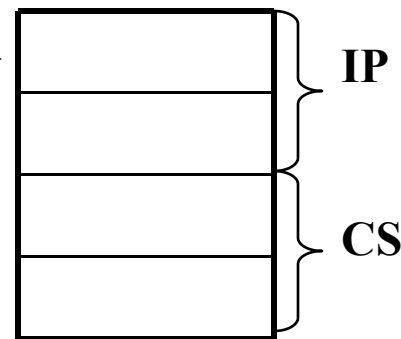
# 段间间接寻址(16位)

用存储器中的两个相继字的内容取代CS和IP以达到段间转移的目的。

(存储单元的地址可用除立即数和寄存器以外的任何一种数据寻址方式得到)

$$\text{物理地址} = 16 \times (\text{DS}) + \text{EA}$$

存储器



根据数据寻址方式计算出EA值。

例: **JMP *DWORD PTR* [INTERS+BX]**

$$\text{EA} = (\text{BX}) + \text{INTERS}$$

***DWORD PTR***: 双字操作符。

# 寻址方式应用示例

试编程实现如下功能：

键盘输入1， 屏幕输出“Insert Sort”

键盘输入2， 屏幕输出“Bubble Sort”

键盘输入3， 屏幕输出“Quick Sort”

键盘输入4， 屏幕输出“Heap Sort”

# 子程序(函数)

- 1.用于模块化、是重要的封装机制
- 2.函数定义方式与执行逻辑
- 3.参数传递方法:
  - 内存变量(数据段)方式
  - 寄存器方式
  - 栈方式

# 函数定义方式

fName proc

指令序列

ret

fName endp

push locF

eip <-----fName地址

ret指令的语义:

pop eip

## 函数调用方式

call fName

指令f; <==地址 设为locF

函数调用的语义如右

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

# 算法设计

## 输入整数算法

```
int a[10];  
for (int i=0; i<10; i++){  
    scanf("%d",&a[i]);  
}
```

## 求数组最大值算法

```
int max;  
max=a[0];  
for (int i=0; i<10; i++){  
    if(max<a[i]) max=a[i];  
}  
printf("%d",max);
```

# 算法实现1-全局变量传参

```
include irvine32.inc
```

```
.data
```

```
    a dword 10 dup(?)
```

```
.code
```

```
intputInts proc
```

```
    push ebx
```

```
    mov ebx, 0
```

```
again:    cmp ebx, 10
```

```
    jge final
```

```
    call readint
```

```
    mov a[4*ebx], eax
```

```
    inc ebx
```

```
    jmp again
```

```
final:    pop ebx
```

```
    ret
```

```
intputInts endp
```



max proc

push ebx

mov ebx,0

**mov eax,a[0]**

again: cmp ebx,10

jge final

**cmp eax,a[4\*ebx]**

jge next

**mov eax,a[4\*ebx]**

next: inc ebx

jmp again

final: pop ebx

ret

max endp

main proc

call intputInts

call max

call writeint

exit

main endp

end main

缺点：函数与数据耦合  
不具有通用性

# 算法实现2-寄存器传参

```
include irvine32.inc
```

```
.data
```

```
    a dword 10 dup(?)
```

```
.code
```

```
intputInts proc
```

```
    push ebx
```

```
    mov ebx, 0
```

```
again:    cmp ebx, ecx
```

```
    jge final
```

```
    call readint
```

```
    mov [edx+4*ebx], eax
```

```
    inc ebx
```

```
    jmp again
```

```
final:    pop ebx
```

```
    ret
```

```
intputInts endp
```

```
max  proc
    push ebx
    mov ebx,0
    mov eax,[edx]
again: cmp ebx,ecx
    jge final
    cmp eax,[edx+4*ebx]
    jge next
    mov eax,[edx+4*ebx]
next:  inc ebx
    jmp again
final: pop ebx
    ret
max  endp
```

```
main proc
    mov edx, offset a
    mov ecx,10
    call intputInts
    mov edx, offset a
    mov ecx,10
    call max
    call writeint
    exit
main endp
end main
优点： 具有通用性
```

# 算法实现3-栈传参

```
include irvine32.inc
```

```
.data
```

```
    a dword 10 dup(?)
```

```
.code
```

```
intputInts proc
```

```
    push ebp
```

```
    mov  ebp, esp
```

```
    push ebx
```

```
    push ecx
```

```
    push edx
```

```
    mov  ecx, [ebp+12]
```

```
    mov  edx, [ebp+8]
```

```
    mov  ebx, 0
```

```
again:    cmp ebx, ecx  
          jge final  
          call readint  
          mov [edx+4*ebx], eax  
          inc ebx  
          jmp again
```

```
final:    pop edx  
          pop ecx  
          pop ebx  
          leave  
          ret 8
```

```
intputInts endp
```

max proc

**push ebp**

**mov ebp,esp**

**push ebx**

**push ecx**

**push edx**

**mov ecx,[ebp+12]**

**mov edx,[ebp+8]**

**mov ebx, 0**

**mov eax, [edx]**

```
again:    cmp  ebx,ecx
          jge  final
          cmp  eax,[edx+4*ebx]
          jge  next
          mov  eax,[edx+4*ebx]
next:     inc  ebx
          jmp  again
final:    pop  edx
          pop  ecx
          pop  ebx
          leave
          ret  8
max      endp
```

main proc

**push 10**

**push offset a**

call intputInts

**push 10**

**push offset a**

call max

call writeint

exit

main endp

end main

优点：具有通用性



3.在内存中存有10个整数，求这10整数 最大值，及最大值在数组中的位置(下标)

4.在内存中存有10个整数，求试用选择排序方法对这10个整数排序、在屏幕输出排好序的10个整数

### 3.算法

```
max=arr[0]
```

```
j=0;
```

```
for(int i=0;i<n;i++){
```

```
    if( arr[j]<arr[i]) {
```

```
        max=arr[i];
```

```
        j=i;
```

```
    }
```

```
}
```

```
18
19 max proc
20 ;输入: edx存放数组首地址
21 ;      ecx存放当前数组最后元素的下标
22 ;输出: eax存放最大值
23 ;      esi存放最大值在数组中的下标
24         mov ebx,0
25         mov eax,[edx]
26         mov esi,0
27 again:
28         cmp ebx,ecx
29         ja  final
30         cmp eax,[edx+ebx*4]
31         ja  next
32         mov eax,[edx+ebx*4]
33         mov esi,ebx
34 next:    add ebx,1
35         jmp again
36 final:
37         ret
38 max endp
39
```

```
1  Include Irvine32.inc
2
3  .data
4      arr dd 32,44,55,1,10,2,5
5  .code
6  main PROC
7      mov  edx,offset arr
8      mov  ecx,7
9      dec  ecx
10     call  max
11     call  writeint;显示最大值
12     mov  eax,esi
13     call  writeint;;显示最大值下标
14     exit
15  main ENDP
```

## 选择排序

```
1  Include Irvine32.inc
2  .data
3      arr dd 32,44,55,1,10,2,5
4  .code
5  main PROC
6      mov     edx,offset arr
7      mov     ecx,7
8      call    sort
9      mov     ecx,7
10     call    output
11     exit
12 main ENDP
13
14 sort proc
15     sub     ecx,1
16     L1:     call max;eax:最大值
17     mov     ebx,[edx+4*ecx]
18     mov     [edx+4*ecx],eax
19     mov     [edx+4*esi],ebx
20     loop    L1
21     ret
22 sort endp
23
```




```

max proc
;输入: edx存放数组首地址
;      ecx存放当前数组最后元素的下标
;输出: eax存放最大值
;      esi存放最大值在数组中的下标
        mov ebx,0
        mov eax,[edx]
        mov esi,0
again:
        cmp ebx,ecx
        ja  final
        cmp eax,[edx+ebx*4]
        ja  next
        mov eax,[edx+ebx*4]
        mov esi,ebx
next:    add ebx,1
        jmp again
final:
        ret
max endp

```

```
45 output proc
46         mov     ebx, 0
47 L2:
48         mov     eax, [edx+4*ebx]
49         call    writeint
50         add     ebx, 1
51         mov     al, ' '
52         call    writechar
53         loop    L2
54         ret
55 output endp
```

## 主函数改进

```
1  Include Irvine32.inc
2
3  .data
4      arr    dd    32,44,55,1,10,2,5
5       count dd    ($-arr)/4
6  .code
7  main PROC
8      mov     edx,offset arr
9       mov     ecx,count
10      call    sort
11         mov     ecx,count
12         call    output
13         exit
14  main ENDP
15
```



# 作业

练习1：在内存中存有10个整数，求试用选择排序方法对这10个整数排序、在屏幕输出排序过程中间结果

练习2：在内存中存有10个整数，求试用冒泡排序方法对这10个整数排序、在屏幕输出排序结果