

# 计算机系统基础



华中科技大学

## 常用机器指令

许向阳

xuxy@hust.edu.cn



# 第5章 常用机器指令



华中科技大学

## 一、学习内容

常用的机器指令的使用格式、功能、应用

## 二、学习重点

要求掌握各指令的语法规则、功能，  
最常用指令对标志寄存器的影响。

## 三、学习方法

归类、对比、总结异同点





# 5.1 通用机器指令概述

- (1) 数据传送指令
- (2) 算术运算指令
- (3) 逻辑运算指令
- (4) 移位指令
- (5) 位操作和字节操作指令
- (6) 标志位控制指令
- (7) I/O 指令
- (8) 控制转移指令
- (9) 串操作指令
- (10) 杂项指令



## 5.2 数据传送指令



华中科技大学

# 数据传送指令





## 5.2 数据传送指令

### 1、一般数据传送指令

MOV、MOVSX、MOVZX、XCHG、XLAT

### 2、堆栈操作指令

PUSH、POP、PUSHA、PUSHAD、POPA、POPAD

### 3、标志寄存器传送指令

PUSHF、POPF、PUSHFD、POPFD、LAHF、SAHF

### 4、地址传送指令

**LEA**、LDS、LES、LSS

### 5、带条件的数据传送指令

CMOVE、CMOVNE、CMOVA ....

除了SAHF、  
POPF,POPF外,  
其他不影响标志  
位。





华中科技大学

## 5.2.1 一般数据传送指令

<b>MOV</b>	<b>OPD, OPS</b>	； 数据传送
<b>MOVSX</b>	<b>R16/R32, OPS</b>	； 符号扩展传送
<b>MOVZX</b>	<b>R16/R32, OPS</b>	； 0（无符号）扩展传送
<b>XCHG</b>	<b>OPD, OPS</b>	； 一般数据交换
<b>XLAT</b>		； 查表转换





华中科技大学

## 5.2.1 一般数据传送指令

### 1、MOV指令

语句格式: MOV OPD, OPS

功 能: (OPS)  $\rightarrow$  OPD

```
int x = 10;  
mov dword ptr [ebp-8], 0Ah
```





## 5.2.1 一般数据传送指令

### 2、符号扩展传送指令

语句格式: MOVSWX OPD, OPS

功 能: 将源操作数的**符号向前扩展**成与目的操作数相同的数据类型后,再送入目的地址对应的单元中。

说 明:

- **OPS 不能为立即数;**
- OPD 必须是 16/32位的寄存器;
- 源操作数的位数必小于目的操作数的位数。







## 5.2.1 一般数据传送指令

### 3、无符号扩展传送指令

语句格式: MOVZX OPD, OPS

功 能: 将源操作数的**高位补0**, 扩成与目的操作数相同的数据类型后, 再送入目的地址对应的单元中。

说 明:

- OPS 不能为立即数;
- OPD 必须是 16/32位的寄存器;
- 源操作数的位数必小于目的操作数的位数。





## 5.2.1 一般数据传送指令

### MOVSX , MOVZX示例

例1: MOV BL, 0E3H

MOVSX EBX, BL

(EBX) = ?

0FFFFFFFE3H

若将最后一条指令换成

MOVZX EBX, BL

(EBX) = ?

000000E3H

例2: BYTE0 DB 0A8H

MOV BL, BYTE0

MOVSX ECX, BL ; 寄存器无对应限制

MOVSX EBX, BYTE0





## 5.2.1 一般数据传送指令

int x = 10;      短的 有/无符号数 扩展为  
short y = 20;      长的 有/无符号数

x = y;

**movsx**      eax, word ptr [ebp-14h]

mov      dword ptr [ebp-8], eax

unsigned short z = 20;

mov      eax, 14h

mov      word ptr [ebp-20h], ax

x = z;

**movzx**      eax, word ptr [ebp-20h]

mov      dword ptr [ebp-8], eax





## 5.2.1 一般数据传送指令

### 4、一般数据交换指令

语句格式: XCHG    OPD, OPS

功      能: (OPD)  $\rightarrow$  OPS    (OPS)  $\rightarrow$  OPD

将源、目的地址指明的单元中的内容互换。

例3: XCHG AH, AL

执行前: (AX) = 1234H    执行后: (AX) = 3412H

**Question:** 指令 XCHG BUF1, BUF2 是否正确?





## 5.2.1 一般数据传送指令

### 5、查表转换指令

语句格式: XLAT

功    能:  $[(\text{EBX} + \text{AL})] \rightarrow \text{AL}$

将 (EBX)为首址, (AL)为位移量的字节存储单元中的数据传送给AL。



## 5.2.1 一般数据传送指令

设有一个16进制数码 (0~9, A~F) 在(AL)中, 现请将该数码转换为对应的ASCII。

一般的算法: 判断(AL)是否小于等于9,  
若是: 则将(AL)+30H  $\rightarrow$  AL;  
否则: 将(AL)+37H  $\rightarrow$  AL;

```
MYTAB DB '0123456789ABCDEF'  
MOV    EBX, OFFSET MYTAB  
XLAT
```

XLAT 可用来对文本数据进行编码和译码, 从而实现简单的加密和解密。





## 5.2.2 带条件的数据传送指令

语句格式: **cmov\*\*\*** r32, r32/m32

功 能: 在条件“\*\*\*”成立时,  
传送数据, 即  $(r32/m32) \rightarrow r32$ 。  
cmov 是Conditional MOVe的缩写。

要 求:

- ① r32 表示一个32位的寄存器;
- ② m32位表示一个内存地址;  
m32对应直接、间接、变址、基址加变址寻址;  
m32对应的单元的数据类型是双字, 即32位。

18条带条件传送指令





## 5.2.2 带条件的数据传送指令

- 使用单个标志位判断转移条件是否成立

`cmove/cmovz`、`cmovc`、`cmovs`、`cmovo`、`cmovp`

条件:  $ZF=1$        $CF=1$        $SF=1$        $OF=1$        $PF=1$

`cmovne/cmovnz`、`cmovnc`、`cmovns`、`cmovno`、`cmovnp`

条件:  $ZF=0$        $CF=0$        $SF=0$        $OF=0$        $PF=0$

- 使用多个标志位组合判断转移条件是否成立

`cmova`、`cmovb`、`cmovg`、`cmovl`

`cmovae`、`cmovbe`、`cmovge`、`cmovle`







## 5.2.2 带条件的数据传送指令

- 设有无符号双字类型变量x、y、z，  
将x和y中间的大者存放到z中

x	dd	10
y	dd	20
z	dd	0

## 5.2.2 带条件的数据传送指令

- 设有无符号双字类型变量x、y、z，  
将x和y中间的大者存放到z中

```
mov    eax, x
cmp    eax, y    ; 比较指令，
                ; 根据 (eax)-(y) 设置标志位
```

```
    jae    L1
    mov    eax, y
L1:  mov    z,    eax
```

```
cmovb  eax, y    ; CF=1 且 ZF=0时，传送
mov    z,    eax
```



## 5.2.2 带条件的数据传送指令

```
int    x, y, x;  
z = x > y ? x : y;  
mov     ecx, DWORD PTR _y$[ebp]  
add     esp, 16      ; 00000010H  
cmp     DWORD PTR _x$[ebp], ecx  
cmovg   ecx, DWORD PTR _x$[ebp]
```

注：下面使用 `z` 时直接使用了 `ecx`

Release 版，不同程序编译结果不同

红色语句与`z`的赋值无关，是下一条C语句翻译结果的部分。  
将其穿插到前面，可提高流水线的处理性能。



## 5.2.3 堆栈操作指令



华中科技大学

**PUSH OPS**

**POP OPD**

**PUSHA**

**PUSHAD**

**POPA**

**POPAD**





## 5.2.3 堆栈操作指令

### 1、进栈指令： PUSH

语句格式： PUSH    OPS

功能       ： 将立即数、寄存器、段寄存器、存储器中的一个**字/双字**数据压入堆栈中。

例： PUSH    AX  
      PUSH    EAX  
      PUSH    X  
      PUSH    DWORD PTR [EBX]



## 5.2.3 堆栈操作指令

### 1、进栈指令：PUSH OPS

#### ➤ 字数据入栈

①  $(ESP) - 2 \rightarrow ESP$

② 字数据  $\rightarrow [ESP]$

#### ➤ 双字数据入栈：

①  $(ESP) - 4 \rightarrow ESP$

② 双字数据  $\rightarrow [ESP]$

记为：(OPS)  $\rightarrow \downarrow$  (ESP)

ESP -2、-4 取决于是字还是双字入栈





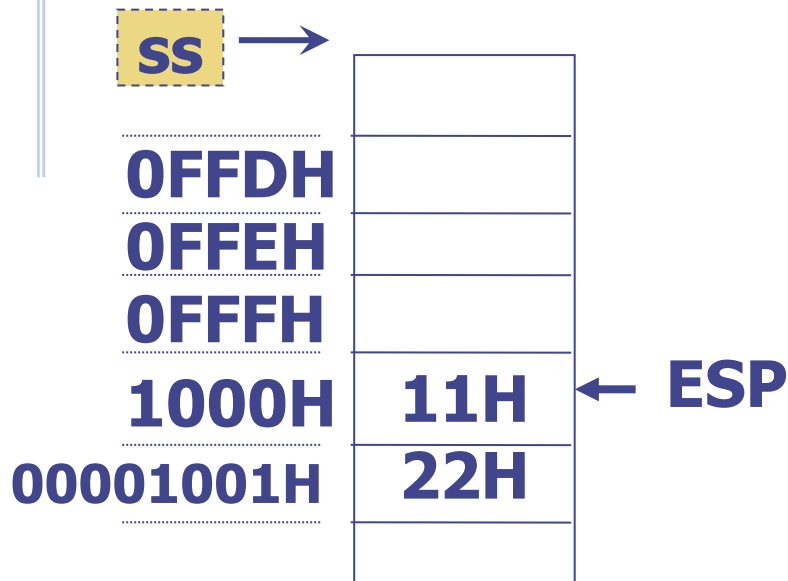
## 5.2.3 堆栈操作指令

### 堆栈示意图画法中应注意的问题

体会指针的含义

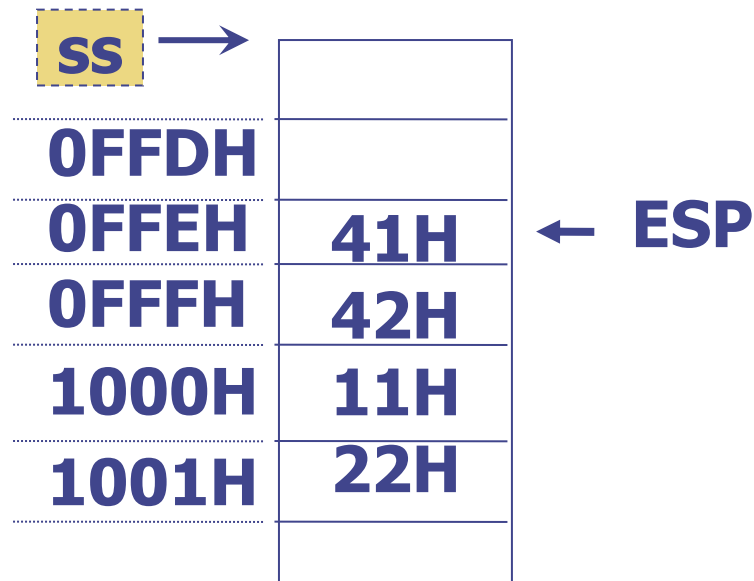
### PUSH AX

执行前:  $(AX)=4241H$   $(ESP)=00001000H$



指令执行堆栈状态

ESP 00001000H



执行“PUSH AX”后的堆栈状态

ESP 00000FFEH





## 5.2.3 堆栈操作指令

### 2、出栈指令： POP

语句格式： POP    OPD

功    能： 将栈顶元素弹出送至某一寄存器、  
段寄存器（CS除外）、存储器中。

字数据出栈

①  $([ESP]) \rightarrow OPD$

②  $(ESP)+2 \rightarrow ESP$

记为：  $\uparrow (ESP) \rightarrow OPD$

双字数据出栈类似，  $(ESP)+4 \rightarrow ESP$







## 5.2.3 堆栈操作指令

### 3、8个16位寄存器入栈

格式: **PUSHA**

功能: 将8个16位寄存器按AX, CX, DX, BX, SP, BP, SI, DI顺序入堆栈。

说明: 入栈的SP的值是执行PUSHA之前的SP值。

### 4、8个16位寄存器出栈

格式: **POPA**





## 5.2.3 堆栈操作指令

### 5、8个32位寄存器入栈

格式: **PUSHAD**

功能: 将8个16位寄存器按EAX, ECX, EDX, EBX, ESP, EBP, ESI, EDI顺序入堆栈。

说明: 入栈的ESP的值是执行PUSHAD之前的ESP值。

### 6、8个32位寄存器出栈

格式: **POPAD**





## 5.2.4 标志寄存器传送指令

### 1、将标志位传送到 AH中

格式: LAHF

功能: 将标志寄存器的低8位送入ah中,  
即  $(\text{eflags})_{7 \sim 0} \rightarrow \text{ah}$ 。

说明: 该指令的执行对标志位无影响。





华中科技大学

## 5.2.4 标志寄存器传送指令

### 2、将 (ah) 传送到标志寄存器

格式: **SAHF**

功能: 将 (ah) 送入标志寄存器的低8位中,  
高位保持不变, 即  $(ah) \rightarrow \text{eflags}_{7 \sim 0}$ 。





## 5.2.4 标志寄存器传送指令

### 3、32位标志寄存器进栈指令

格式: **PUSHFD**

功能: 将标志寄存器的内容压入堆栈,  
记为  $(\text{eflags}) \rightarrow \downarrow (\text{esp})$ 。

### 4、32位标志寄存器出栈指令

格式: **POPFD**

功能: 将栈顶内容弹出送入标志寄存器中,  
记为  $\uparrow (\text{esp}) \rightarrow \text{eflags}$ 。





## 5.2.4 标志寄存器传送指令

### 5、16位标志寄存器进栈指令

格式: **PUSHF**

功能: 将标志寄存器的内容压入堆栈,  
记为  $(\text{eflags})_{15-0} \rightarrow \downarrow (\text{esp})$ 。

### 6、16位标志寄存器出栈指令

格式: **POPF**

功能: 将栈顶内容弹出送入标志寄存器中,  
记为  $\uparrow (\text{esp}) \rightarrow \text{eflags}_{15-0}$ 。





## 5.2.5 地址传送指令

### 1、传送偏移地址指令

语句格式: LEA R32, M32

功 能: 将M32 对应的地址送入R32中。

说明:

- **R32** 是一个32位的通用寄存器;
- **M32** 所提供的 是一个32位的存储器地址;  
直接寻址、寄存器间接寻址、变址寻址、基址加变址寻址



## 5.2.5 地址传送指令

### 1、传送偏移地址指令

**MOV ESI, OFFSET NUM**

**LEA ESI, NUM**      与上一行语句等效;

**LEA EDI, [ESI+4]**       $(ESI)+4 \rightarrow EDI$

**MOV EDI, [ESI+4]**

**DS: ([ESI]+4)  $\rightarrow$  EDI**

如何实现       $(EAX) + (EBX)*8 \rightarrow ECX$  ?

**LEA ECX, [EAX+EBX\*8]**







## 5.2.5 地址传送指令

```
int x=20;
005D192F  mov          dword ptr [ebp-0Ch], 14h
          int* p=&x;
005D1936  lea          eax, [ebp-0Ch]
005D1939  mov          dword ptr [ebp-18h], eax
```

寄存器

EAX = 0019FA98 EBX = 00290000 ECX = 005DC029  
EDX = 00000001 ESI = 005D1023 EDI = 0019FAA4  
EIP = 005D1939 ESP = 0019F9BC EBP = 0019FAA4  
EFL = 00000246

68 %

执行 `lea eax, [ebp-0Ch]` 后,  $(eax) = (ebp-0Ch)$

监视 1

搜索(Ctrl+E)



搜索深度: 3

名称	值	类型
▸ &x,x	0x0019fa98 {0x00000014}	int *
▾ x	20	int





## 5.2.5 地址传送指令

### 2、传送偏移地址及数据段首址指令

在32位扁平内存管理模式下，不需要使用这些指令。

语句格式: LDS opd, ops

功 能:  $(ops) \rightarrow opd$ ,  $(ops+2) \rightarrow ds$

说 明:

- ① opd一定要是一个16/32位的通用寄存器。
- ② ops所提供的一定要是是一个存储器地址，  
且类型为dd/df。



## 5.2 数据传送指令

记住

一般传送	MOV	OPD, OPS
有符号数传送	MOVSX	R16/R32, OPS/非立即数
无符号数传送	MOVZX	R16/R32, OPS/非立即数
一般数据交换	XCHG	OPD, OPS
查表转换	XLAT	
传送偏移地址	LEA	R32 , M32
进栈	PUSH	OPS
出栈	POP	OPD
32位通用寄存进栈、出栈		PUSHAD、POPAD
标志寄存器进栈、出栈		PUSHFD、POPFD



## 5.3 算术运算指令



华中科技大学

# 算术运算指令





## 5.3 算术运算指令

一般对标志位都有影响

### 1、加法指令

INC、ADD、ADC

### 2、减法指令

DEC、NEG、SUB、SBB、CMP

### 3、乘法指令

IMUL、MUL

### 4、除法指令

IDIV、DIV

### 5、符号扩展指令

CBW、CWD、CWDE、CDQ





## 5.3.1 加法指令

### (1) 加1指令

**INC OPD ; (OPD) +1 → OPD**

### (2) 加指令

**ADD OPD,OPS ; (OPD)+(OPS) → OPD**

例: **MOV AX, 0FFFDH**

**ADD AX, -7FFFH**

执行上述语句后, (AX)= ?

**OF= ?    CF=?    ZF= ?    SF=?**

**(AX)=7FFEH    OF=1    CF=1    ZF=0    SF=0**





## 5.3.1 加法指令

### (3) 带进位加指令

语句格式: `ADC OPD,OPS`

功能:  $(OPD) + (OPS) + CF \rightarrow OPD$

例: 计算 `1234 F00FH + 1234 80F0H`  
(只允许使用16位寄存器)

`1234 F00F`

`1234 80F0`

---

`2469 70FF`





## 5.3.1 加法指令

例：计算 1234 F00FH + 1234 80F0H  
(只允许使用16位寄存器)

```
dn1 dw 0f00fh, 1234h
dn2 dw 80f0h, 1234h
dsum dw 0,0
```

```
mov ax, dn1
add ax, dn2
mov dsum, ax
mov ax, dn1+2
adc ax, dn2+2
mov dsum+2,ax
```





## 5.3.2 减法指令

**DEC OPD**

**NEG OPD**

**SUB OPD, OPS**

**SBB OPD, OPS**

**CMP OPD, OPS**

DEC对OF,SF,ZF,PF,AF  
有影响;  
其它指令对  
**CF**,OF,SF,ZF,PF,AF有影  
响;



## 5.3.2 减法指令

### (1) 减1指令

**DEC OPD ; (OPD) - 1 → OPD**

### (2) 求补指令

**NEG OPD ; (OPD)求反加1 → OPD**

执行如下程序段后，(AX)=?

**MOV AX, 20H**

**NEG AX**

**(AX)= 0FFE0 H**



## 5.3.2 减法指令

### (3) 减指令

**SUB OPD,OPS ;(OPD)-(OPS) → OPD**

例: A DW 50

B DW 100

MOV AX, 200

SUB AX, A ; (AX)=? 150

**Q:** 写出完成  $(B) - (A) \rightarrow B$  的指令

~~SUB B, A~~

MOV BX, A  
SUB B, BX



## 5.3.2 减法指令

### (4) 带借位减指令

**SBB OPD,OPS**

**$(OPD) - (OPS) - CF \rightarrow OPD$**

例：计算 2469 70FF - 1234 F00FH  
(只允许使用16位寄存器)

2469 70FF

1234 F00F

---

1234 70F0



## 5.3.2 减法指令

### (5) 比较指令

**CMP OPD, OPS ; (OPD) – (OPS)**

比较指令的作用是什么？



## 5.3.3 乘法指令

**IMUL R16/R32, OPS**

**IMUL R16/R32, OPS, n**

**IMUL OPS**

**MUL OPS**

字节乘法

字乘法

双字乘法

**OPS与OPD  
类型相同**

AX

DX, AX

EDX, EAX





## 5.3.3 乘法指令

### (1) 有符号乘法

#### ■ 双操作数的有符号乘指令

语句格式: IMUL OPD, OPS

功 能:  $(OPD) * (OPS) \rightarrow OPD$

说 明: **OPD** 为 16/32位寄存器

**OPS** 为同类型的寄存器、存储器  
操作数或立即数。

例: IMUL AX, BX

IMUL EAX, DWORD PTR[ESI]

IMUL AX, 3





## 5.3.3 乘法指令

### (1) 有符号乘法

#### ■ 3个操作数的有符号乘指令

语句格式: **IMUL OPD, OPS, n**

功 能:  **$(OPS) * n \rightarrow OPD$**

说 明: **OPD** 为 16/32位寄存器

**OPS**为同类型的寄存器、存储器  
操作数或立即数。

例: **IMUL AX, BX, -10**

**IMUL EAX, DWORD PTR[ESI], 5**

**IMUL BX, AX, 3**







## 5.3.3 乘法指令

### (1) 有符号乘法

#### ■单操作数的有符号乘法

语句格式: **IMUL OPS**

字节乘法: **(AL)\*(OPS) → AX**

字乘法: **(AX)\*(OPS) → DX, AX**

双字乘法: **(EAX)\*(OPS) → EDX, EAX**

说明: **OPS**不能是立即数

如果乘积的高位不是低位的符号扩展,  
而是包含有效位, 则**CF=1, OF=1**.





## 5.3.3 乘法指令

### (2) 无符号乘法

语句格式: **MUL OPS**

功 能:

字节乘法:  $(AL) * (OPS) \rightarrow AX$

字乘法:  $(AX) * (OPS) \rightarrow DX, AX$

双字乘法:  $(EAX) * (OPS) \rightarrow EDX, EAX$

说 明: **OPS**不能是立即数





## 5.3.3 乘法指令

### 无符号乘法与有符号乘法的比较

**.code**

**begin:**

**mov al,10H**

**mov bl,-2 ; (bl)=FE**

**imul bl**

**(ax)=0FFE0H,结果高字节无有效位, 有NC, NV**

**mov al,10H**

**mul bl**

**(ax)=0FE0H, 结果高字节有有效位, 有CY, OV**

**mov al,-10h**

**mov bl,2**

**imul bl**

**(ax) = 0FFE0H**

**mov al,-10h**

**mov bl,2**

**mul bl**

**(ax) = 01E0H**

**end begin**





## 5.3.3 乘法指令

**OPS与OPD  
类型相同**

**IMUL R16/R32, OPS**

**IMUL R16/R32, OPS, n**

**IMUL OPS**

**MUL OPS**

字节乘法

AX

字乘法

DX, AX

双字乘法

EDX, EAX

**Question:** 为什么字乘法的结果在 DX, AX中, 而不是在EAX中?





## 5.3.3 乘法指令

```
unsigned short us1,us2;  
unsigned int   ui;  
short  s1,s2 ;  
int    i;
```

无符号乘法 向  
有符号乘法 的 转化

```
u1 = us1 * us2;
```

00651959	movzx	eax, word ptr [ebp-0Ch]
0065195D	movzx	ecx, word ptr [ebp-18h]
00651961	imul	eax, ecx
00651964	mov	dword ptr [ebp-24h], eax
i=s1*s2;		
00651967	movsx	eax, word ptr [ebp-30h]
0065196B	movsx	ecx, word ptr [ebp-3Ch]
0065196F	imul	eax, ecx
00651972	mov	dword ptr [ebp-48h], eax





## 5.3.4 除法指令

### (1) 有符号除法

**IDIV OPS**

**字节除法:**  $(AX)/(OPS) \rightarrow AL(\text{商}), AH(\text{余})$

**字除法:**  $(DX,AX)/(OPS) \rightarrow AX(\text{商}), DX(\text{余})$

**双字除法:**  $(EDX,EAX)/(OPS) \rightarrow EAX(\text{商}), EDX$

### (2) 无符号除法

**DIV OPS**

**字节除法:**  $(AX)/(OPS) \rightarrow AL(\text{商}), AH(\text{余})$

**字除法:**  $(DX,AX)/(OPS) \rightarrow AX(\text{商}), DX(\text{余})$

**双字除法:**  $(EDX,EAX)/(OPS) \rightarrow EAX(\text{商}), EDX$





## 5.3.4 除法指令

### 使用除法指令应注意的问题

- (1) 除数为0
- (2) 除法溢出

$(AX)=1234, (BL)=1, (AX)/(BL) \rightarrow AL$



华中科技大学

## 5.3.5 符号扩展指令

### (1) 将字节转换成字

**CBW**

将AL中的符号扩展至AH中。

### (2) 将字转换成双字

**CWD**

将AX中的符号扩展至DX中。







## 5.3.5 符号扩展指令

(3) 将AX中的有符号数扩展为32位送EAX  
CWDE

(4) 将EAX中的有符号数扩展为64位数  
送 EDX, EAX  
CDQ



## 5.3.5 符号扩展指令

unsigned short us1,us2;  
unsigned int ui;  
short s1,s2 ;  
int i;

us1 = ui / us2;

```
00EF1975 movzx ecx,word ptr [ebp-18h]
00EF1979 mov eax,dword ptr [ebp-24h]
00EF197C xor edx,edx
00EF197E div eax,ecx
00EF1980 mov word ptr [ebp-0Ch],ax
```

s1 = i / s2;

```
00EF1984 movsx ecx,word ptr [ebp-3Ch]
00EF1988 mov eax,dword ptr [ebp-48h]
00EF198B cdq
00EF198C idiv eax,ecx
00EF198E mov word ptr [ebp-30h],ax
```





## 5.3.5 符号扩展指令

```
mov eax, 123
00708270 B8 7B 00 00 00      mov     eax, 7Bh
mov ecx, 10
00708275 B9 0A 00 00 00      mov     ecx, 0Ah
mov edx, 0
0070827A BA 00 00 00 00    mov     edx, 0
div ecx
0070827F F7 F1            div     eax, ecx
```

寄存器

```
007 EAX = 0000000C EBX = 0050A000 ECX = 0000000A
007 EDX = 00000003 ESI = 00BD8BF8 EDI = 00BD8C58
007 EIP = 00708281 ESP = 006FFA88 EBP = 006FFACC
007 EFL = 00000202
```

68 %

正确理解 `div eax, ecx`; 本质是  $(edx, eax) / (ecx)$



## 5.4 逻辑运算指令



华中科技大学

# 逻辑运算指令





## 5.4 逻辑运算指令

### 求反

NOT OPD ; (OPD)求反 $\rightarrow$ OPD

### 逻辑乘

AND OPD, OPS ; (OPD) $\wedge$ (OPS)  $\rightarrow$ OPD

### 测试指令

TEST OPD, OPS ; (OPD) $\wedge$ (OPS)

### 逻辑加

OR OPD, OPS ; (OPD) $\vee$ (OPS)  $\rightarrow$ OPD

### 按位加

XOR OPD, OPS ; (OPD)异或(OPS)  $\rightarrow$ OPD





## 5.4 逻辑运算指令

例1：指出各指令执行后，(AX)=?

**MOV AX, 1234H**

**NOT AX ; (AX) = ? 0EDCBH**

**AND AX, 0FH ; (AX)= ? 000BH**

**OR AX, 1255H ; (AX)= ? 125FH**

**XOR AX, AX ; (AX) = ? 0000H**



## 5.4 逻辑运算指令

**测试指令** TEST OPD, OPS


**功 能：** 根据  $(OPD) \wedge (OPS)$  设置标志位，  
(OPD)、(OPS)不变。

CF=0, OF=0。 ZF、SF、PF依结果而定。  
AND、OR、XOR亦是如此。

例：判断(AX)的最高位是否为0，若为0，转L

TEST AX, 8000H

JZ L


$$\begin{array}{r} \wedge \quad 1000 \quad 0000 \quad 0000 \quad 0000B \\ \hline \quad ? \, 000 \quad 0000 \quad 0000 \quad 0000B \end{array}$$





## 5.4 逻辑运算指令

求反

NOT OPD

逻辑乘

AND OPD, OPS

测试指令

TEST OPD, OPS

逻辑加

OR OPD, OPS

按位加

XOR OPD, OPS

按位取反 ~

按位与 &

按位或 |

按位异或 ^







华中科技大学

## 5.4 逻辑运算指令

### C 语句 与机器指令的对应

int x = 100, y;

y = ~x;

0065194C 8B 45 F8

mov eax, dword ptr [ebp-8]

0065194F F7 D0

not eax

00651951 89 45 EC

mov dword ptr [ebp-14h], eax

y = x & 1;

00651954 8B 45 F8

mov eax, dword ptr [ebp-8]

00651957 83 E0 01

and eax, 1

0065195A 89 45 EC

mov dword ptr [ebp-14h], eax

y |= 0x10000;

0065197B 8B 45 EC

mov eax, dword ptr [ebp-14h]

0065197E 0D 00 00 01 00

or eax, 10000h

00651983 89 45 EC

mov dword ptr [ebp-14h], eax

x = x ^ y;

0065199B 8B 45 F8

mov eax, dword ptr [ebp-8]

0065199E 33 45 EC

xor eax, dword ptr [ebp-14h]

006519A1 89 45 F8

mov dword ptr [ebp-8], eax





## 5.4 逻辑运算指令

```
void bit_op() {  
    int x = 103;  
    int y;  
    y = ~x;  
    printf("after ~ : x= %x y = %x \n",x,y);  
    y = x & 1;  
    printf("after & : y = %x \n", y);  
    if (y == 0) printf("x 是偶数 \n");  
    else printf("x 是奇数 \n");  
    y |= 0x10000;  
    printf("after | : x=%d y= %d \n",x, y);  
    x = x ^ y;  
    y = x ^ y;  
    x = x ^ y;  
    printf("after exchange : x=%d y= %d \n", x, y);  
}
```

C:\教学\本科教学\计算机系统基础\计算机系统基础\_程序\C05\_常用指令\Debug\C05\_常用指令.exe

```
after ~ : x= 00000067 y = ffffffff98  
after & : y = 1  
x 是奇数  
after | : x=103 y= 65537  
after exchange : x=65537 y= 103
```

## 5.4 逻辑运算指令

挑战：下面函数实现两个变量中的值互换，有无问题？

```
void xor_swap(int* x, int* y)
{
    *y = *x ^ *y;
    *x = *x ^ *y;
    *y = *x ^ *y;
}
```



## 5.4 逻辑运算指令

```
#define LEN 5
int main()
{
    int a[LEN] = { 10, 20, 30, 35, 5 };
    int head, tail;
    for (head = 0; head < LEN; head++)
        printf("%d ", a[head]);
    printf("\n before swap\n");
    for (head = 0, tail=LEN-1; head <= tail; head++, tail--)
        xor_swap(a + head, &a[tail]);
    for (head = 0; head < LEN; head++)
        printf("%d ", a[head]);
    printf("\n after swap\n");
    return 0;
}
```

C:\教学\本科教学\计算机系统基础\计算机系统基

```
10 20 30 35 5
  before swap
5 35 0 20 10
  after swap
```



## 5.4 逻辑运算指令

### 逻辑 && 与 按位 & 的差别

int z;

z = x && y;

x 非 0 且 y 非 0, 则 z = 1;

x 为 0 或 y 为 0, 则 z = 0;

```
001719F5  cmp     dword ptr [ebp-8],0
001719F9  je      00171A0D
001719FB  cmp     dword ptr [ebp-14h],0
001719FF  je      00171A0D
00171A01  mov     dword ptr [ebp+FFFFFF18h],1
00171A0B  jmp     00171A17
00171A0D  mov     dword ptr [ebp+FFFFFF18h],0
00171A17  mov     eax,dword ptr [ebp+FFFFFF18h]
00171A1D  mov     dword ptr [ebp-20h],eax
```





## 5.4 逻辑运算指令

### 逻辑非 ! 与 按位反 ~ 的差别

```
int x, z;
```

```
z = ~x;
```

```
00D11AC0 mov     eax,dword ptr [x]
```

```
00D11AC3 not     eax
```

```
00D11AC5 mov     dword ptr [z],eax
```

```
z = !x;
```

```
00D11AC8 cmp     dword ptr [x],0
```

```
00D11ACC jne     __$EncStackInitStart+10Eh (0D11ADAh)
```

```
00D11ACE mov     dword ptr [ebp-0E8h],1
```

```
00D11AD8 jmp     __$EncStackInitStart+118h (0D11AE4h)
```

```
00D11ADA mov     dword ptr [ebp-0E8h],0
```

```
00D11AE4 mov     eax,dword ptr [ebp-0E8h]
```

```
00D11AEA mov     dword ptr [z],eax
```





## 5.4 逻辑运算指令

思考题：编写C程序，判断两个整数是否相等，  
相等为1，不相等为 0

```
int isEqual(int x, int y) {  
    return !(x ^ y);  
  
    // return x==y;  
}
```



## 5.5 移位指令

- (1) 算术左移 **SAL**      Shift **A**rithmetic **L**eft
- (2) 逻辑左移 **SHL**      **S**hift Logical **L**eft
- (3) 逻辑右移 **SHR**      **S**hift Logical **R**ight
- (4) 算术右移 **SAR**      Shift **A**rithmetic **R**ight
- (5) 循环左移 **ROL**      Rotate Left
- (6) 循环右移 **ROR**      Rotate Right
- (7) 带进位的循环左移 **RCL**  
Rotate left through Carry
- (8) 带进位的循环右移 **RCR**





## 5.5 移位指令

语句格式:

**操作符** **OPD, n 或 CL**

功能: 将(OPD)中的所有位按**操作符**规定的方式移动, 结果存在**OPD**对应的单元中。

说明:

**OPD**可以是寄存器, 也可以是地址表达式。



## 5.5 移位指令

- (1) 算术左移 SAL OPD, n
- (2) 逻辑左移 SHL OPD, n

(OPD)向左移动n位, 低位补0

CF



SAL AX, 3 等价于

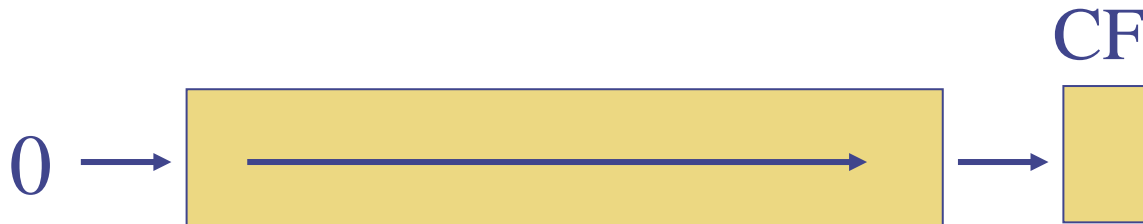
SAL AX, 1

SAL AX, 1

SAL AX, 1 ; CF为执行最后一次移位时送入的值

## 5.5 移位指令

(3) **逻辑右移** SHR OPD, n  
(OPD) 向右移动n位, 高位补0



MOV AH, 5

SHR AH, 1

; (AH)= ?                    2

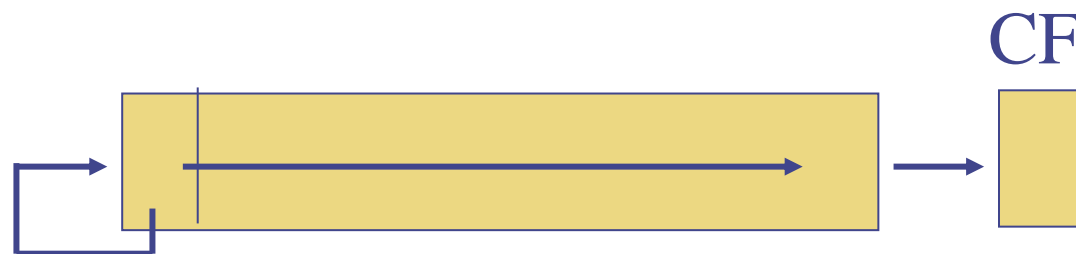
; (CF)= ?                    1

## 5.5 移位指令



华中科技大学

**(4)算术右移 SAR OPD, n**  
(OPD)向右移动n位, 最高位不变。



```
MOV    AH, 0F5H
SHR     AH, 1
; (AH)= ?    7AH
; (CF)= ?    1
```

```
MOV     AH, 0F5H
SAR     AH, 1
; (AH)= ?    0FAH
; (CF)= ?    1
```

比较两种指令, 其结果说明什么?



## 5.5 移位指令

### (5) 循环左移 ROL OPD, n

将(OPD)的最高位与最低位连接起来，组成一个环。将环中的所有位一起向左移动n位，CF的内容为最后移入位的值。



```
MOV DL, 0EAH
```

```
ROL DL, 4
```

(DL) = ?

CF = ?

0AEH 0

## 5.5 移位指令

### (6) 循环右移 ROR OPD, n

将(OPD)的最高位与最低位连接起来, 组成一个环。将环中的所有位一起向右移动n位, CF的内容为最后移入位的值。



```
MOV DL, 0EAH
```

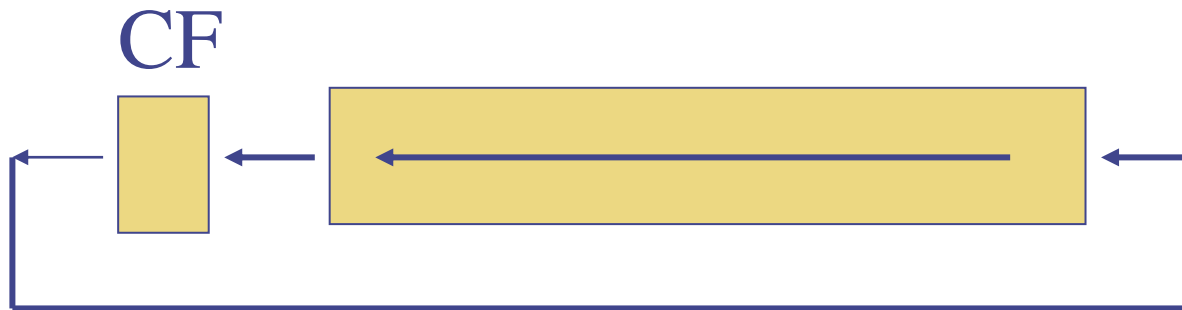
```
ROR DL, 4
```

```
(DL) = ?    CF = ?    0AEH    1
```

## 5.5 移位指令

### (7) 带进位的循环左移 RCL OPD, n

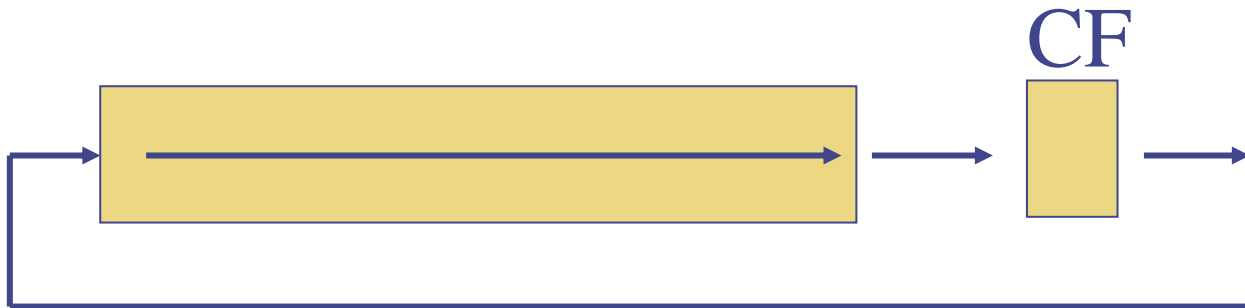
将 (OPD) 的最高位、CF、(OPD) 最低位连接起来，组成一个环。将环中的所有位一起向左移动n位，CF的内容为最后移入位的值。



## 5.5 移位指令

### (8) 带进位的循环右移 RCR OPD, n

- (OPD)、CF连接组成一个环；
- 将环中的所有位一起向右移动n位；
- CF的内容为最后移入位的值。







## 5.5 移位指令

SAL、	SHL
SAR、	SHR
ROL、	ROR
RCL、	RCR

移动方向？

CF的摆放位置？

移动规则？



## 5.5 移位指令

### C 语句 与机器指令的对应

```
int x = 100;
00A11C05  mov     dword ptr [ebp-8],64h
        x = x << 1;
00A11C0C  mov     eax,dword ptr [ebp-8]
00A11C0F  shl     eax,1
00A11C11  mov     dword ptr [ebp-8],eax
        x = x >> 1;
00A11C14  mov     eax,dword ptr [ebp-8]
00A11C17  sar     eax,1
00A11C19  mov     dword ptr [ebp-8],eax
        unsigned int y = 100;
00A11C1C  mov     dword ptr [ebp-14h],64h
        y = y >> 1;
00A11C23  mov     eax,dword ptr [ebp-14h]
00A11C26  shr     eax,1
00A11C28  mov     dword ptr [ebp-14h],eax
```





## 5.5 移位指令

```
int x = 100;
unsigned int y = 100;
x = x >> 1;
mov     eax,dword ptr [x]
sar     eax,1
mov     dword ptr [x],eax
y = y >> 1;
mov     eax,dword ptr [y]
shr     eax,1
mov     dword ptr [y],eax
x = (unsigned int)x >> 1;
mov     eax,dword ptr [x]
shr     eax,1
mov     dword ptr [x],eax
```

### C 语句 与机器指令 的对应

比较:

有符号数、无符号数

右移 对应的指令



跳过

**5.6 位操作和字节操作指令**

**5.7 标志位控制指令和杂项指令**

**5.8 I/O 指令**



华中科技大学

# REVIEW

## 数据传送指令

### 1、一般数据传送指令

**MOV、MOVSX、MOVZX、XCHG、XLAT**

### 2、堆栈操作指令

**PUSH、POP、PUSHA、PUSHAD、POPA、POPAD**

### 3、标志寄存器传送指令

**PUSHF、POPF、PUSHFD、POPFD、LAHF、SAHF**

### 4、地址传送指令

**LEA**

### 5、输入、输出指令

**IN、OUT**



## 算术运算指令

### 1、加法指令

**INC、ADD、ADC**

### 2、减法指令

**DEC、NEG、SUB、SBB、CMP**

### 3、乘法指令

**IMUL (三种形式)、MUL**

### 4、除法指令

**IDIV、DIV**

### 5、符号扩展指令

**CBW、CWD**



华中科技大学

# REVIEW

## 逻辑运算指令

**NOT、AND、TEST、OR、XOR**

## 移位指令

<b>SAL、</b>	<b>SHL</b>
<b>SAR、</b>	<b>SHR</b>
<b>ROL、</b>	<b>ROR</b>
<b>RCL、</b>	<b>RCR</b>



# 第5章 常用机器指令



华中科技大学

试用不同指令将 (AX)置0。

```
MOV  AX, 0
SUB  AX, AX
AND  AX, 0
XOR  AX, AX
SHL  AX, 16
```

试用不同的指令，将AX的高、低字节内容互换。

```
XCHG AH, AL
ROL  AX, 8
ROR  AX, 8
```





# 第5章 常用机器指令



华中科技大学

作业 5.1 将 (x)和(x+2) 中的字数据交换位置。

`x dw 10, 20`

- (1) 只使用 `mov` 指令
- (2) 只使用 `xchg`指令
- (3) 混合使用 `mov`和`xchg`指令
- (4) 用`push`、`pop`指令
- (5) 用循环左移指令
- (6) 用循环右移指令



# 第5章 常用机器指令



华中科技大学

(1) 只使用 mov 指令

MOV AX, X

MOV BX, X+2

MOV X+2, AX

MOV X, BX

(2) 只使用 xchg指令

XCHG AX, X

XCHG AX, X+2

XCHG AX, X



# 第5章 常用机器指令



华中科技大学

## (3) 混合使用 mov和xchg指令

**MOV AX, X**

**XCHG AX, X+2**

**MOV X, AX**

## (4) 用push、pop指令

**PUSH X**

**PUSH X+2**

**POP X**

**POP X+2**



# 第5章 常用机器指令



华中科技大学

(5) 用循环左移指令

**ROL DWORD PTR X, 16**

(6) 用循环右移指令

**ROR DWORD PTR X, 16**



# 第5章 常用机器指令



华中科技大学

练习:

用C语言编程，只使用位运算、算术运算等指令，不使用转移之类的语句，实现求一个数的绝对值

```
int abs_op(int x) {  
    int y, mask;  
    mask = x >> 31; // FFFFFFFF vs 00000000  
    y = (x ^ mask) + ~mask + 1; //method1:  
    y = (x ^ mask) + (1-mask) - 1; //method2  
    y = ((x & 0x80000000) >> 31) + (x ^ mask); // method3  
    return y;  
}
```

注意：运算的优先级



# 第5章 常用机器指令



华中科技大学

优先级	运算符	名称或含义	使用形式	结合方向
1	[]	数组下标	数组名[常量 表达式 <sup>Q</sup> ]	左到右
	()	圆括号	(表达式) /函数名(形参表)	
	.	成员选择 (对象)	对象.成员名	
	->	成员选择 (指针)	对象指针->成员名	
2	-	负号 运算符 <sup>Q</sup>	-表达式	右到左
	~	按位取反运算符	~表达式	
	++	自增运算符	++变量名/变量名++	
	--	自减运算符	--变量名/变量名--	
	*	取值运算符	*指针变量	
	&	取地址运算符	&变量名	
	!	逻辑非运算符	!表达式	
	(类型)	强制类型转换	(数据类型)表达式	
	sizeof	长度运算符	sizeof(表达式)	



# 第5章 常用机器指令



华中科技大学

3	/	除	表达式/表达式	左到右
	*	乘	表达式*表达式	
	%	余数（取模）	整型表达式%整型表达式	
4	+	加	表达式+表达式	左到右
	-	减	表达式-表达式	
5	<<	左移	变量<<表达式	左到右
	>>	右移	变量>>表达式	
6	>	大于	表达式>表达式	左到右
	>=	大于等于	表达式>=表达式	
	<	小于	表达式<表达式	
	<=	小于等于	表达式<=表达式	
7	==	等于	表达式==表达式	左到右
	!=	不等于	表达式!= 表达式	



# 第5章 常用机器指令



华中科技大学

8	&	按位与	表达式&表达式	左到右
9	^	按位异或	表达式^表达式	左到右
10		按位或	表达式 表达式	左到右
11	&&	逻辑与	表达式&&表达式	左到右
12		逻辑或	表达式  表达式	左到右
13	?:	条件运算符	表达式1? 表达式2: 表达式3	右到左





# 第5章 常用机器指令



华中科技大学

练习:

用C语言编程，不使用转移之类的语句，  
判断两个数相加 是否出现溢出。

```
int addOK(int x, int y)
{
    int sum = x+y;
    int x_neg = x>>31;
    int y_neg = y>>31;
    int s_neg = sum>>31;
    /* Overflow when x and y have same sign, but s is different */
    return !((~(x_neg ^ y_neg) & (x_neg ^ s_neg)));
}
```



# 第5章 常用机器指令



华中科技大学

作业

**P102-103      5.2 – 5.5**



# 按指令格式的复杂度来分

---

按指令格式的复杂度来分，有两种类型计算机：

**复杂指令集计算机CISC (Complex Instruction Set Computer)**

**精简指令集计算机RISC (Reduce Instruction Set Computer)**

早期CISC设计风格的主要特点

**(1) 指令系统复杂**

**变长操作码 / 变长指令字 / 指令多 / 寻址方式多 / 指令格式多**

**(2) 指令周期长**

**绝大多数指令需要多个时钟周期才能完成**

**(3) 各种指令都能访问存储器**

**除了专门的存储器读写指令外，运算指令也能访问存储器**

**(4) 采用微程序控制**

**(5) 难以进行编译优化来生成高效目标代码**

例如，VAX-11/780小型机

**16种寻址方式；9种数据格式；303条指令；一条指令包括1~2个字节的操作码和下续N个操作数说明符。一个说明符的长度达1~10个字节。**

# 复杂指令集计算机CISC

## ◆ CISC的缺陷

- 日趋庞大的指令系统不但使计算机的研制周期变长，而且难以保证设计的正确性，难以调试和维护，并且因指令操作复杂而增加机器周期，从而降低了系统性能。

## ◆ 1975年IBM公司开始研究指令系统的合理性问题，John Cocks提出精简指令系统计算机 **RISC** ( Reduce Instruction Set Computer )。

### ◦ 对CISC进行测试，发现一个事实：

- 在程序中各种指令出现的频率悬殊很大，最常使用的是一些简单指令，这些指令占程序的80%，但只占指令系统的20%。而且在微程序控制的计算机中，占指令总数20%的复杂指令占用了控制存储器容量的80%。

### ◦ 1982年美国加州伯克利大学的**RISC I**，斯坦福大学的**MIPS**，IBM公司的**IBM801**相继宣告完成，这些机器被称为**第一代RISC机**。

# Top 10 80x86 Instructions

° Rank	instruction	Integer Average	Percent total executed
1	load <b>MOV M to R</b>		22%
2	conditional branch <b>Jcc</b>		20%
3	compare <b>CMP</b>		16%
4	store <b>MOV R to M</b>		12%
5	add		8%
6	and		6%
7	sub		5%
8	move register-register		4%
9	call		1%
10	return		1%
	Total		96%

° Simple instructions dominate instruction frequency

( 简单指令占主要部分, 使用频率高! )

# RISC设计风格的主要特点

---

## (1) 简化的指令系统

指令少 / 寻址方式少 / 指令格式少 / 指令长度一致

## (2) 以RR方式工作

除Load/Store指令可访存外，其余指令都只访问寄存器

## (3) 指令周期短

以流水线方式工作，因而除Load/Store指令外，其他简单指令都只需一个或一个不到的时钟周期就可完成

## (4) 采用大量通用寄存器，以减少访存次数

## (5) 采用硬连线路控制器，不用或少用微程序控制

## (6) 采用优化的编译系统，力求有效地支持高级语言程序

MIPS是典型的RISC处理器，82年以来新的指令集大多采用RISC体系结构  
x86因为“兼容”的需要，保留了CISC的风格，同时也借鉴了RISC思想