常用机器指令

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) 按位运算指令
(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

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

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、带条件的数据传送指令

CMOVE、CMOVNE、CMOVA ….

除了SAHF、POPF,POPFD外，其他不影响标志位。

5.2.1 一般数据传送指令


```
MOV     OPD, OPS        ; 数据传送
MOVSX   R16/R32, OPS    ; 符号扩展传送
MOVZX   R16/R32, OPS    ; 0（无符号）扩展传送
```

5.2.1 一般数据传送指令

```
int      x = 10;
short  y = 20;
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
```

短的有/无符号数
扩展为
长的有/无符号数

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,z;
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

_x$、_y$是编译时生成的符号常量：_x$ = … _y$ = …

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

PUSH OPS

POP OPD

PUSHA

PUSHAD

POPA

POPAD

5.2.3 堆栈操作指令

1、进栈指令：PUSH OPS

- 字数据入栈
  - ① (ESP) - 2 → ESP
  - ② 字数据 → [ESP]
- 双字数据入栈：
  - ① (ESP) - 4 → ESP
  - ② 双字数据 → [ESP]

ESP —2、—4 取决于是字还是双字入栈

5.2.3 堆栈操作指令

2、出栈指令： POP OPD

字数据出栈

① ([ESP]) → OPD

② (ESP)+2 → ESP

记为：↑(ESP) → OPD

双字数据出栈类似， (ESP)+4->ESP

5.2.3 堆栈操作指令

3、8个16位寄存器入栈　　PUSHA

4、8个16位寄存器出栈　　POPA

5、8个32位寄存器入栈　　PUSHAD

6、8个32位寄存器出栈　　POPAD

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(eflags)→↓(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↑(esp)→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$(eflags)_{15-0} \rightarrow \downarrow (esp)$。

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$\uparrow (esp) \rightarrow eflags_{15-0}$。

5.2.5 地址传送指令

华中科技大学

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。

LEA　ESI, NUM

LEA EDI, [ESI+4]　　(ESI)+4 → EDI

如何实现　（EAX）+ (EBX) * 8 → 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 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
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

算术运算指令

5.3 算术运算指令

华中科技大学

1、加法指令

INC、ADD、ADC

一般对标志位都有影响。
但INC和DEC对CF无影响！

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

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 → OPD

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

```
  1234 F00F
  1234 80F0
-------------
  2469 70FF
```

5.3.2 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

(4) 带借位减指令

SBB OPD,OPS

(OPD) – (OPS) – CF → OPD

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

```
  2469 70FF
  1234 F00F
-------------
  1234 70F0
```

5.3.3 乘法指令

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

OPS与OPD类型相同

IMUL OPS
MUL OPS

字节乘法	AX
字乘法	DX, AX
双字乘法	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) → **OPD**

■ 3个操作数的有符号乘指令

语句格式：**IMUL OPD, OPS，n**
功　　能：**(OPS) * n → OPD**

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

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) → AX

字乘法：　　(AX)*(OPS) → DX , AX

双字乘法：　（EAX) *(OPS) →EDX, EAX

5.3.3 乘法指令

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

OPS与OPD类型相同

IMUL OPS
MUL OPS

字节乘法	AX
字乘法	DX, AX
双字乘法	EDX,EAX

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) → AL(商), AH(余)

字 除 法：(DX, AX) / (OPS) → AX (商), DX (余)

双字除法：(EDX, EAX) / (OPS) → EAX (商), EDX

(2) 无符号除法

DIV OPS

字节除法：(AX) / (OPS) → AL(商),AH(余)

字 除 法：(DX, AX) / (OPS) → AX (商), DX (余)

双字除法：(EDX, EAX) / (OPS) → EAX (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) → AL

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

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

68 %

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

按位运算指令

5.4 按位运算指令

华中科技大学

求反

NOT OPD ; ~ (OPD) →OPD

逻辑乘

AND OPD, OPS ; (OPD) & (OPS) →OPD

测试指令

TEST OPD, OPS ; (OPD) & (OPS)

逻辑加

OR OPD, OPS ; (OPD) | (OPS) →OPD

按位加

XOR OPD, OPS ; (OPD) ^ (OPS) →OPD

5.4 按位运算指令

求反　　按位取反　~

NOT　OPD

逻辑乘　　按位与　&

AND　OPD, OPS

测试指令

TEST　OPD, OPS

逻辑加　　按位或　|

OR　OPD, OPS

按位加　　按位异或　^

XOR　OPD, OPS

5.4 按位运算指令

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 按位运算指令

華中科技大學

```
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 = ffffff98
after & :  y = 1
 x 是奇数
after | : x=103  y= 65537
after exchange : x=65537  y= 103
```

5.4 按位运算指令

华中科技大学

Q：下面函数实现两个变量中的值互换，有无问题？

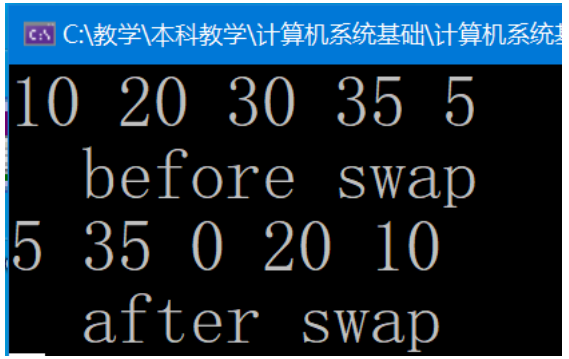
```
void xor_swap(int *x, int *y)
{   *y = *x ^ *y;
    *x = *x ^ *y;
    *y = *x ^ *y;
}
```

```
int x = 10;
int y = 20;
printf("before swap :%d %d\n", x, y);
xor_swap(&x, &y);
printf("after swap :%d %d\n",x,y);
```

```
before swap :10 20
after swap :20 10
```

5.4 按位运算指令


```
#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;
}
```



5.4 按位运算指令


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

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

按位运算与逻辑运算

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   //Debug version
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
```

按位运算与逻辑运算

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

按位运算与逻辑运算

```
    z = x && y;
003A1036   cmp         dword ptr [esp+8],0
003A103B   je          003A104B
003A103D   cmp         dword ptr [esp+4],0
003A1042   je          003A104B
003A1044   mov         eax,1
003A1049   jmp         003A104D
003A104B   xor         eax,eax
```

Release 版本下的程序与 Debug 版本下的程序不同；
设置不同的编译开关，产生的程序也不同。

按位运算与逻辑运算

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
```

按位运算与逻辑运算

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;
}
```

按位运算与逻辑运算

思考题：编写C程序，实现计算 -x

用按位运算方法实现

-x 等于 ~x+1

```
NEG  x      NOT  x
            INC  x
```

用乘法指令实现？

```
MOV   EAX，x
IMUL  EAX, -1
MOV   x,  EAX
```

用减法指令实现？

```
MOV   EAX，0
SUB   EAX, x
MOV   x,  EAX
```

5.5 移位指令

(1)算术左移 SAL　Shift Arithmetic Left

(2)逻辑左移 SHL　SHift Logical Left

(3)逻辑右移 SHR　SHift Logical Right

(4)算术右移 SAR　Shift Arithmetic Right

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

REVIEW

华中科技大学

数据传送指令

1、一般数据传送指令
MOV、MOVSX、MOVZX

2、堆栈操作指令
PUSH、POP

3、地址传送指令
LEA

REVIEW

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

REVIEW

华中科技大学

逻辑运算指令

NOT、AND、TEST、OR、XOR

移位指令

SAL、 SHL

SAR、 SHR

优化

```
int   x, y;
y = x * 9;
imul     eax, dword ptr [x], 9
mov      dword ptr [y], eax
```

DEBUG 版

```
mov      ecx, dword ptr [x]
lea      eax, [ecx+ecx*8]
mov      dword ptr [y], ecx
```

RELEASE 版

优化


```
int   x, y;
y = x * 17;
imul      eax, dword ptr [x], 11h
mov       dword ptr [y], eax
```

DEBUG 版

```
mov       ecx, dword ptr [x]
mov       eax, ecx
shl       eax, 4
add       eax, ecx
mov       dword ptr [y], eax
```

RELEASE 版

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
```

练习1：用C语言的异或（^）操作实现整数的求反（not）。

```
int not_op(int x) {
    return (0xFFFFFFFF ^ mask);
}
```

练习2：用C语言实现求一个整数的绝对值，只能使用位运算、算术运算等指令，不使用转移之类的语句。

```
int abs_op(int x){
    int y,mask;
    mask = x >> 31; //算术右移,得到 FFFFFFFF / 00000000
    y = (x ^ mask) + ~mask + 1 ;      //method 1:
    y = (x ^ mask) + (1-mask) - 1;    //method 2
    y = ((x & 0x80000000) >> 31) + (x^mask); //method 3
    return y;
}
//注意运算的优先级
//单纯的 x >> 31 是算术右移
//（x & 0x80000000）>> 31 是逻辑右移
```

练习3：用C语言用编程，不使用转移之类的语句，判断两个有符号数相加是否出现溢出。

```
//返回: 0 = 不溢出，非0 =溢出
int addOV(int x, int y)  {
   int  z = x + y;
   int  sx = (unsigned int)x >> 31;
   int  sy = (unsigned int)y >> 31;
   int  sz = (unsigned int)z >> 31;
   //Overflow:  sx = sy  &&  sx != sz
    return ( !(sx ^ sy) & (sx ^ sz) );
}
```

第5章 常用机器指令

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]	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	-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)	

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	

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**

指令系统的扩展知识

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)

复杂指令集计算机CISC

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使计算机的研制周期变长，
  - 难以保证设计的正确性，难以调试和维护，
  - 因指令操作复杂而增加机器周期，从而降低了系统性能。

- 对CISC进行测试，发现一个事实：
  - 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%。
  - 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%。
- 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 。
- 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 MOV M to R	22%
2	conditional branch Jcc	20%
3	compare CMP	16%
4	store MOV R to M	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。

Microcomputer without interlocked pipeline stages