第6章 分支程序设计

第6章 顺序和分支程序设计

一、学习重点

转移指令、分支程序的设计

二、学习难点

(1) 无条件转移指令的灵活运用

(2) 条件转移指令的正确选择

(3) 分支出口的安排与汇合

(6) 综合应用前几章的内容，编写和调试程序

第6章 分支程序设计

6.1 转移控制指令

```
if  (cond_expr) {
      then_statements
}
else {
      else_statements
}
```

```
c=cond_expr;
if (!c)
  goto else_p
  then_statements
  goto if_end
else_p:
   else_statements
if_end:
```

6.1 转移控制指令

```
int   x, y;
if   (x ==y ) {
       Statements 1
       .........
}
else {
       Statements 2
       ......
}
```

```
    MOV   EAX, X
    CMP   EAX, Y
    JNE    L1
    Statements 1
       ......
    JMP    L2
L1:
    Statements 2
       ......
L2:
```

Q: c 程序中分支语句的执行流程是什么？与机器指令有何对应关系？

6.1.1 转移指令概述

- 转移指令
  - 条件转移
    - 简单条件转移（10条）
    - 无符号数条件转移(4条)
    - 有符号数条件转移(4条)
  - 无条件转移
    - 直接转移
    - 间接转移

JMP **

6.1.2 简单条件转移指令

根据单个标志位 CF、ZF、SF、OF、PF的值确定是否转移。

语句格式：　[标号:]　操作符　标号

如果转移条件满足，则(EIP)+ 位移量 → EIP。

如条件不成立，则什么也不做。

6.1.2 简单条件转移指令

如果转移条件满足，则(EIP)+ 位移量 → EIP。

如条件不成立，则什么也不做。

取指令，指令译码后，EIP 就加了指令的长度！

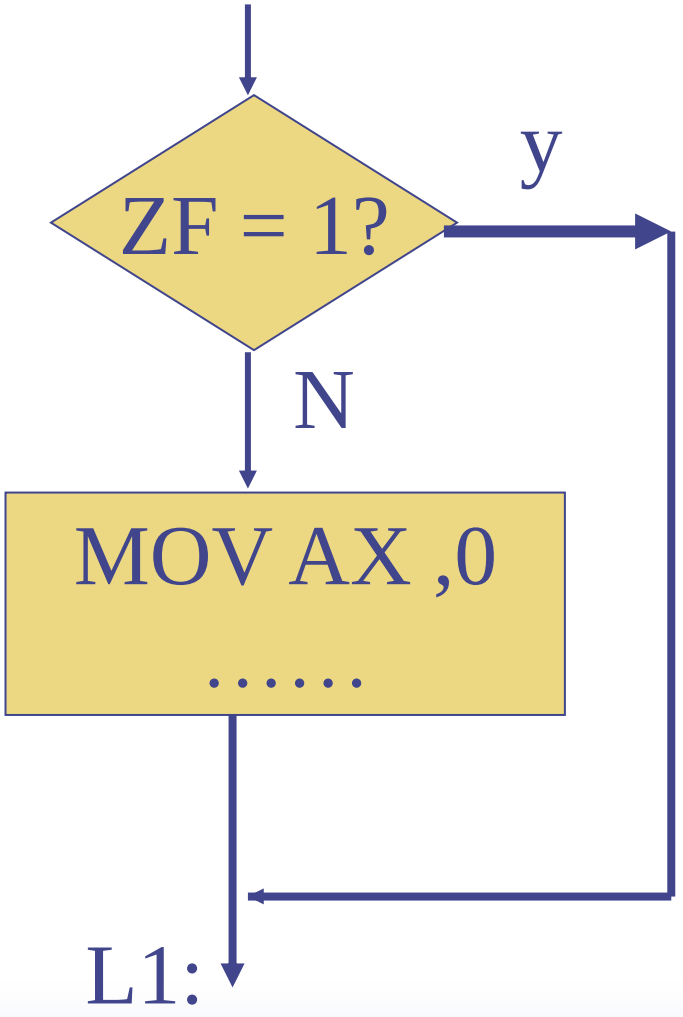
```
  mov  eax, x
00968270   A1 11 90 9D 00       mov   eax,dword ptr [x (09D9011h)]
  cmp  eax, y
00968275  3B 05 15 90 9D 00    cmp  eax,dword ptr [y (09D9015h)]
  jnz  l1
0096827B  75 07                    jne   l1 (0968284h)
  mov  ecx,1
0096827D   B9 01 00 00 00       mov    ecx,1
  jmp  l2
00968282   EB 05                 jmp    l1+5h (0968289h)
l1:mov  ecx,0
00968284  B9 00 00 00 00       mov         ecx,0
l2:
00968289   ……
```

6.1.2 简单条件转移指令

JZ / JE	ZF=1时，转移
JNZ / JNE	ZF=0时，转移
JS	SF=1时，转移
JNS	SF=0时，转移
JO	OF=1时，转移
JNO	OF=0时，转移
JC	CF=1时，转移
JNC	CF=0时，转移
JP / JPE	PF=1时，转移
JNP / JPO	PF=0时，转移

6.1.2 简单条件转移指令

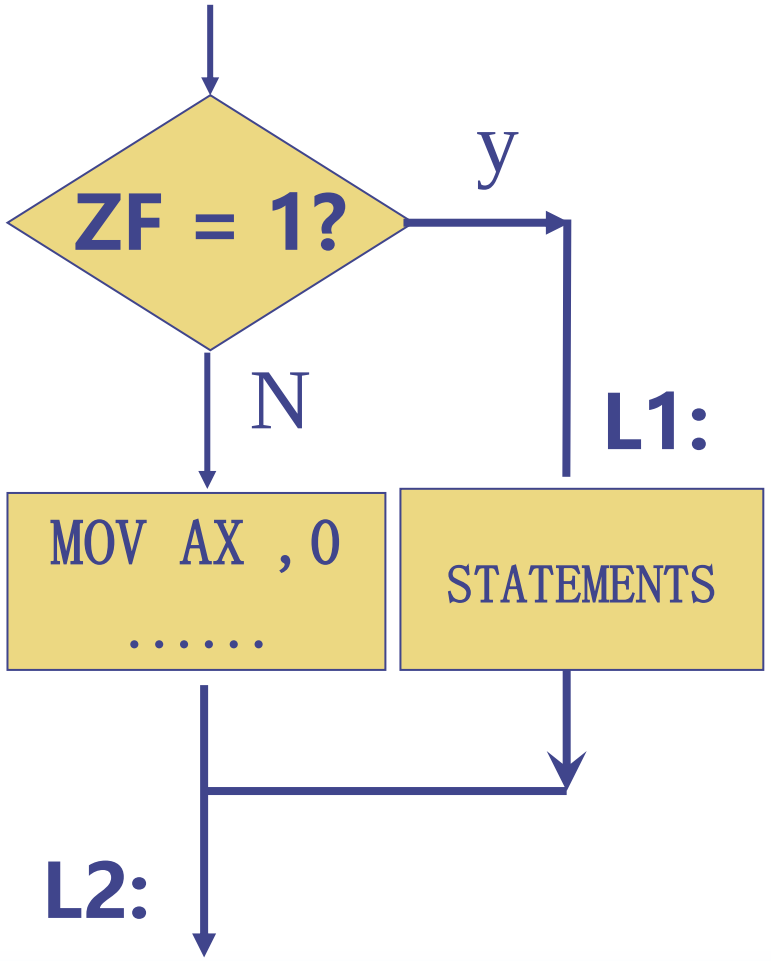
```
      JZ    L1
      MOV   AX , 0
      ……
L1:
```



指令与流程图的对应关系

6.1.2 简单条件转移指令


```
    JZ   L1
    MOV  AX , 0
         ……
    JMP  L2
L1: STATEMENTS
L2:
```



指令与流程图的对应关系

6.1.3 无符号条件转移指令

JA / JNBE　　标号　　(CF=0 且 ZF=0，转移)

JAE / JNB　　标号　　(CF=0 或 ZF=1，转移)

JB / JNAE　　标号　　(CF=1 且 ZF=0，转移)

JBE / JNA　　标号　　(CF=1 或 ZF=1，转移)

6.1.3 无符号条件转移指令

无符号数条件转移指令的理解

```
      CMP    AX，BX
      JA     L1
        ……          CF=0 且 ZF=0，转移
L1:
```

将(AX),(BX)中的数据当成无符号数，
若 (AX)>(BX)，执行 (AX)-(BX)，
则 CF一定会为0，ZF=0

例1：(AX)= 1234H,　　(BX)=0234H

例2：(AX)= 0A234H,　　(BX)=0234H

例3：(AX)= 0A234H,　　(BX)=09234H

6.1.3 无符号条件转移指令

```
int flag=0;
unsigned int  ux = -1; // ux=0xffffffff;
unsigned int  uy = 3;
if (ux > uy)
   flag = 1;
```

```
mov   dword ptr [flag],0
mov   dword ptr [ux],0FFFFFFFFh
mov   dword ptr [uy],3
mov   eax,dword ptr [ux]
cmp   eax,dword ptr [uy]
00A517E3   jbe   main+4Ch (0A517ECh)//机器码 76 07
00A517E5   mov   dword ptr [flag],1
00A517EC   ……
```

6.1.4 有符号条件转移指令

JG / JNLE　　标号

当 SF=OF 且 ZF=0时，转移

JGE / JNL　　标号

当 SF=OF 或者 ZF=1时，转移

JL / JNGE　　标号

当 SF≠OF 且 ZF=0时，转移

JLE / JNG　　标号

当 SF≠OF 或者 ZF=1时，转移

6.1.4 有符号条件转移指令


```
      CMP    AX，BX
      JG     L1
        ……
L1:
```

有符号数条件转移指令的理解

将(AX),(BX)中的数据当成有符号数，
若 (AX)>(BX)，执行 (AX) - (BX)，
则 SF、OF会相等，ZF=0。

例1: (AX)= 1234H, (BX)=0234H

SF=0、OF=0, ZF=0, CF=0

不论使用 JA 还是 JG ，转移的条件均成立

6.1.4 有符号条件转移指令

例2：(AX)= 0A234H, (BX) =0234H

执行(AX) - (BX)后：

SF = 1, ZF=0, CF=0, OF = 0

对于 JA ，条件成立 （CF=0 ， ZF=0）

对于 JG ，条件不成立 （因为SF≠OF）

例3：(AX)= 0A234H, (BX) =09234H

SF = 0 ， ZF=0, CF=0, OF=0

对于JA、JG，条件均成立

6.1.4 有符号条件转移指令

```
int flag=0;
int  ux = -1; // ux=0xffffffff;
int  uy = 3;
if (ux > uy)
   flag = 1;
```

```
mov   dword ptr [flag],0
mov   dword ptr [ux],0FFFFFFFFh
mov   dword ptr [uy],3
mov   eax,dword ptr [ux]
cmp   eax,dword ptr [uy]
00A517E3  jle   main+4Ch (0A517ECh)//机器码 7E 07
00A517E5  mov   dword ptr [flag],1
00A517EC  ……
```

6.1.5 无条件转移指令

格式	名称	功能
JMP 标号	直接	(EIP)+位移量 → EIP
JMP OPD	间接	(OPD) → EIP

6.1.5 无条件转移指令

间接转移方式中，除了立即数寻址方式外，其它方式均可以使用。

```
BUF     DD    L1     ; L1为标号

(1)   JMP    L1
(2)   JMP    BUF
(3)   LEA    EBX , BUF
      JMP    DWORD PTR [EBX]
(4)   MOV    EBX , BUF
      JMP    EBX
```

功能等价的转移指令

6.1.5 无条件转移指令

```
 jmp 11
00F68270 EB 14                jmp   11 (0F68286h)
    mov   eax, x
00F68272 A1 11 90 FD 00       mov   eax,dword ptr [x (0FD9011h)]
     ......
11:mov   ecx,0
00F68286 B9 00 00 00 00       mov   ecx,0
```

6.1.5 无条件转移指令

```
 jmp 11
00578270 E9 40 01 00 00          jmp          10+14h (05783B5h)
    db 300 dup(0)
00578275 00 00                   add          byte ptr [eax],al
00578277 00 00                   add          byte ptr [eax],al

11:mov  ecx,0
005783B5 B9 00 00 00 00          mov          ecx,0
```

6.1.5 无条件转移指令

6.1.5 无条件转移指令

例：根据不同的输入，执行不同的程序片段。

构造指令地址列表

输入1，执行程序段 LP1 ：
输入2，执行程序段 LP2 ：
输入3，执行程序段 LP3 ：
……

```
JMP   LP1
……
JMP   LP2
…..
JMP   LP3
```

如果分支很多，每个分支均使用JMP 标号，程序难看，臃肿！

6.1.5 无条件转移指令

例：根据不同的输入，执行不同的程序片段。

构造指令地址列表

```
FUNCTAB   DD   LP1, LP2, LP3
JMP   FUNCTAB[EBX*4]
```

(EBX)=0, 跳转到 LP1处
(EBX)=1, 跳转到 LP2处

订正：书P123 ，例6.4程序中
应将 ptable dd l1,l2,l3 移到 main endp 之前

6.1.5 无条件转移指令


```
void arraysubtract_colsfirst( )  {……}
void arraysubtract_rowsfirst( )  {……}
void arraysubtract_onedim ( )  {……}

int main()
{
    int   i;
    void (*funcp[3])() = { arraysubtract_colsfirst ,
                           arraysubtract_rowsfirst,
                           arraysubtract_onedim };
    funcp[i]();   // i=0,1,2 会执行不同的函数
    …….
}
```

函数指针、函数指针数组、函数入口地址表

6.2 简单分支程序设计

(1) 选择合适的转移指令；

(2) 为每个分支安排出口；

(3) 将分支中的公共部分尽量放到分支前或分支后的公共程序段中；

(4) 流程图、程序对应

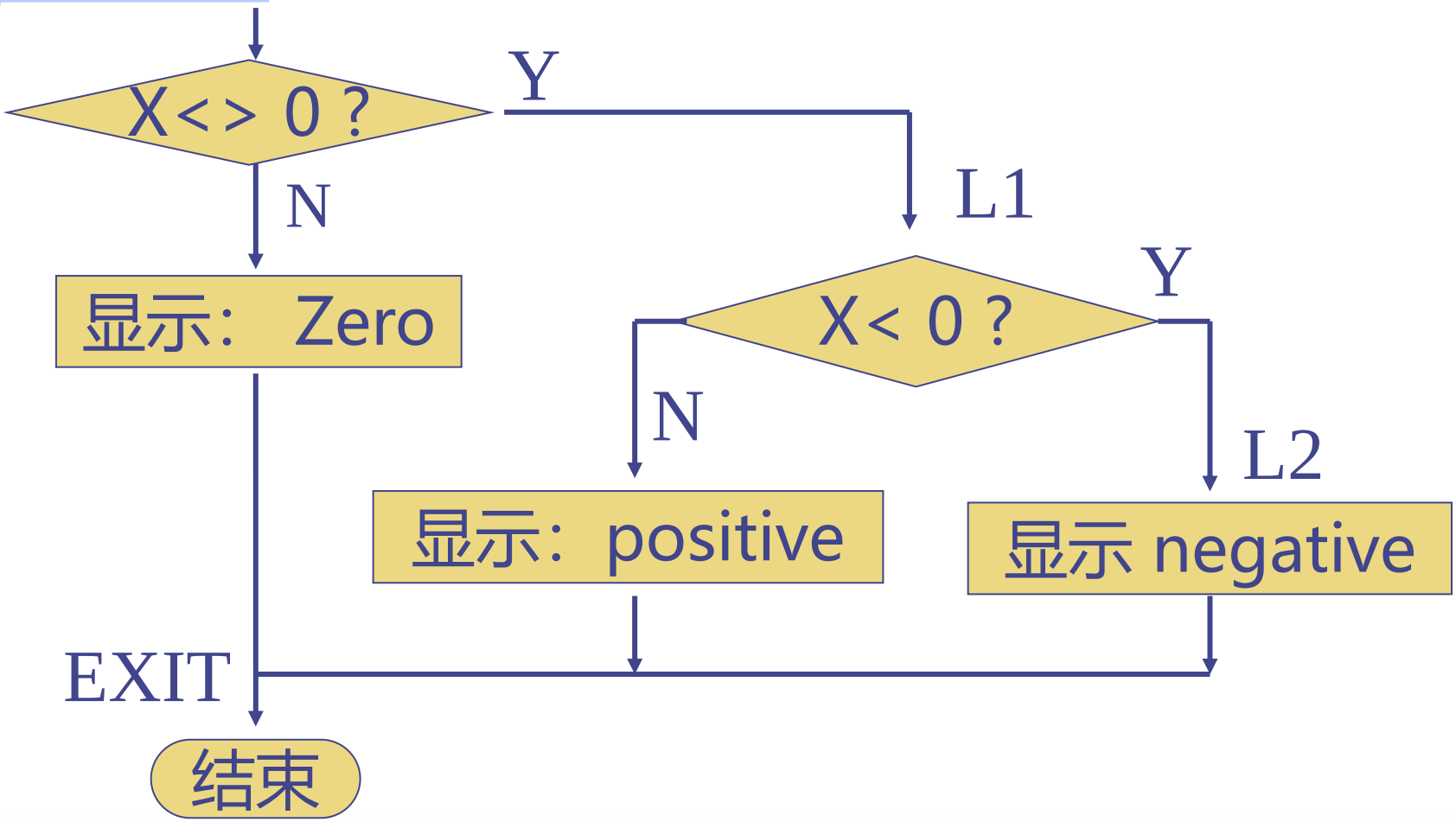
(5) 调试时，逐分支检查

6.2 简单分支程序设计

例：判断 x 中的内容,
为正,显示 positive > 0 ；
为负,显示 < 0;
为0, 显示 =0

实验：使用不同转移指令后的结果比较合分析

6.2 简单分支程序设计



6.2 简单分支程序设计

例：实现|(AX)|+(BX)→ Y，试简化程序。

```
        OR    AX, AX
        JS    L1
        ADD   AX, BX
        MOV   Y,  AX
        JMP   EXIT
L1:     NEG   AX
        ADD   AX, BX
        MOV   Y,  AX
        JMP   EXIT
EXIT:
```

```
        OR    AX, AX
        JNS   L1
        NEG   AX
L1:     ADD   AX, BX
        MOV   Y,  AX
EXIT:
```

公共部分的简化

6.2 简单分支程序设计

TIPS：

- 不要随意摆放各个分支、不要随意跳转；
- 最好是将两个分支紧靠在一起，使得整个条件语句像一个小模块，有一个起始点和一个终止点；
- 从起始点到终止点之间的语句全部都是该条件语句的组成部分，而不含有其他语句。

6.3.1 多分支向无分支的转化

例：统计一个字符串中各个字母出现的次数。

例：当x==1时，显示‘Hello，One’；
当x==2时，显示‘Two’；
当x==3时，显示‘Welcome,Three’,……,
即x为不同的值，显示不同的串。

6.3 多分支程序设计

例：当x==1时，显示‘Hello，One’；
当x==2时，显示‘Two’；
当x==3时，显示‘Welcome,Three’,……,

用汇编语言编写程序：
多分支向无分支的转化

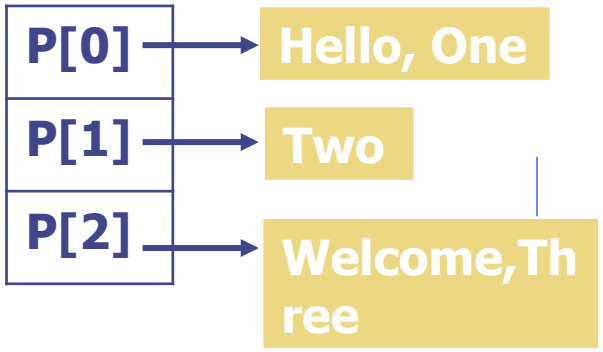
```
lpFmt   db   "%s",0ah, 0dh, 0
 x       dd     3
 MSG1  db   'Hello,One',0
 MSG2  db  'Two',0
 MSG3  db  'Welcome,Three',0
 VTABLE  dd  MSG1, MSG2, MSG3

  MOV    EBX,  X
  MOV    EAX,  VTABLE[EBX*4 -4]
  invoke  printf, offset lpFmt, EAX
```

6.3 多分支程序设计

例：当x==1时，显示‘Hello，One’；
当x==2时，显示‘Two’；
当x==3时，显示‘Welcome,Three’,……,

用 C 语言编写程序：
多分支向无分支的转化

```
void myprint()
{    int  x;
     char msg1[] = "Hello,One";
     char msg2[] = "Two";
     char msg3[] = "Welcome, Three";
     char* p[3] = { msg1,msg2,msg3 };
     printf("please input 0,1,2 \n");
     scanf("%d", &x);
     printf("%s\n", p[x]);
}
```



➢ 编译优化：向无分支转换

```
#include <iostream>
using namespace std;
int absdiff(int x, int y)
{
    int result;
    if (x < y)
        result = y - x;
    else
        result = x - y;
    return result;
}
```

```
int main()
{
    int r=absdiff(10, 25);
    cout << r << endl;
    return 0;
}
```

```
mov    eax, 15
ret
```

6.3 多分支程序设计

➢ 编译优化：向无分支转换

```
int absdiff(int x, int y)
; _x$ = ecx
; _y$ = edx
push   esi
mov    esi, ecx
mov    eax, edx
sub    esi, edx
sub    eax, ecx
cmp    ecx, edx
cmovge eax, esi
pop    esi
ret    0
```

```
if (x < y)  result = y - x;
   else  result = x - y;

int main()
{   int r=absdiff(10, 25);
    cout << r << endl;
    r = absdiff(15, 50);
    cout << r << endl;
    return 0;
}
```

先做　x - y -> esi

　　y - x -> eax

再比较　x、y

最后 cmovge eax, esi

结果在 eax中

6.3 多分支程序设计

- 编译优化：向无分支转换

Q：向无分支转换有什么好处？

Q：什么情况下能够转换？什么时候不能转换？

```
; _p$ = ecx
  test  ecx, ecx
  je    SHORT $LN3@getv
  mov   eax, DWORD PTR [ecx]
  ret   0
$LN3@getv:
   xor   eax, eax
   ret   0
```

```
int getv(int* p)
{
    return p ? *p : 0;
}
```

Q：能优化为 取 *p，0 再条件传送吗？

6.3.2 switch语句的编译

从生成的汇编代码，体会 C 中switch语句的写法

- break 对应什么指令？
- 少写break，导致的后果是什么？
- 各个分支摆放顺序对结果有无影响？
- default 分支能不能写在最开头？

从编译结果中体会多分支向无分支的转化技巧

6.3 多分支程序设计


```
#include <stdio.h>
int main(int argc, char* argv[])
{     int  x = 3, y = -1,  z;
      char c;
      c = getch();
      switch (c) {
              case '+':
              case 'a':      // 用 字符'a'来表示 '+'
                      z= x+y;
                      break;
              case '-':
              case 's':     // 用 字符's'来表示 '-'
                      z= x-y;
                      break;
              default: z=0;
      }
      printf(" %d %c %d = %d \n", x,c,y,z);
      return 0;
}
```

输入代表运算的 字符

+、a ： x+y

-、s ： x-y

其他 ： 0

6.3 多分支程序设计

DEBUG 版编译结果

```
        case '+':
        case 'a' :  z= x+y;
00C2188B 8B 45 F8             mov     eax,dword ptr [x]
00C2188E 03 45 EC             add     eax,dword ptr [y]
00C21891 89 45 E0             mov     dword ptr [z],eax
                break;
00C21894 EB 12                jmp     $LN5+12h (0C218A8h)
        case '-':
        case 's' : z= x-y;
00C21896 8B 45 F8             mov     eax,dword ptr [x]
00C21899 2B 45 EC             sub     eax,dword ptr [y]
00C2189C 89 45 E0             mov     dword ptr [z],eax
                break;
00C2189F EB 07                jmp     $LN5+12h (0C218A8h)
        default:              z=0;
00C218A1 C7 45 E0 00 00 00 00 mov     dword ptr [z],0
        }
00C218A8 ……
```

6.3 多分支程序设计

華中科技大學

DEBUG 版编译结果

```
        switch (c) {
00C21855 0F BE 45 CB          movsx   eax,byte ptr [c]
00C21859 89 85 00 FF FF FF    mov     dword ptr [ebp-100h],eax
00C2185F 8B 8D 00 FF FF FF    mov     ecx,dword ptr [ebp-100h]
00C21865 83 E9 2B             sub     ecx,2Bh
00C21868 89 8D 00 FF FF FF    mov     dword ptr [ebp-100h],ecx
00C2186E 83 BD 00 FF FF FF 48 cmp     dword ptr [ebp-100h],48h
00C21875 77 2A                ja      $LN5+0Bh (0C218A1h)
00C21877 8B 95 00 FF FF FF    mov     edx,dword ptr [ebp-100h]
00C2187D 0F B6 82 E8 18 C2 00 movzx   eax,byte ptr [edx+0C218E8h]
00C21884 FF 24 85 DC 18 C2 00 jmp     dword ptr [eax*4+0C218DCh]
        case '+':
        case 'a':……
```

内存 1

地址: 0x00C218DC　　列: 自动

```
0x00C218DC  8b 18 c2 00 96 18 c2 00 a1 18 c2 00
0x00C218E8  00 02 01 02 02 02 02 02 02 02 02 02
0x00C218F4  02 02 02 02 02 02 02 02 02 02 02 02
```

+ : 2Bh
- : 2Dh
a : 64h
s : 73h
73h-2Bh =48h

6.3 多分支程序设计

RELEASE 版编译结果

```
; 10   :      switch (c) {
      movsx   ecx, al
      lea     eax, DWORD PTR [ecx-43]   ;  43->2Bh
      cmp     eax, 72                   ; 00000048H
      ja      SHORT $LN8@main
      movzx   eax, BYTE PTR $LN10@main[eax]
      jmp     DWORD PTR $LN11@main[eax*4]
$LN4@main:
; 11   : case '+' : ; 12  :  case 'a' : ; 13   : z = x + y;
      mov     eax, 2                    // int  x = 3, y = -1 x+y=2
; 14   :      break;
      jmp     SHORT $LN2@main
$LN6@main:
; 15   : case '-' : ; 16   : case 's' : ; 17   : z = x - y;
      mov     eax, 4                    // int  x = 3, y = -1 x-y=4
; 18   :      break;
      jmp     SHORT $LN2@main
$LN8@main:  ; 19   : default: ; 20   : z = 0;
      xor     eax, eax
$LN2@main:
```

6.4 条件控制流伪指令

条件控制流伪指令

```
.IF   条件表达式
      语句序列
.ENDIF

.IF   条件表达式
      语句序列
.ELSE
      语句序列
.ENDIF
```

条件表达式：

(1)关系运算

==

!=

>

>=

<

<=

(2) 逻辑运算

&&

||

6.4 条件控制流伪指令

条件控制流伪指令

```
.IF    条件表达式
       语句序列
.ELSEIF    条件表达式
           语句序列
……
.ELSE
       语句序列
.ENDIF
```

6.4 条件控制流伪指令

例： 判断字节单元 x 中的内容

为正, 显示 positive > 0

为负, 显示 < 0

为0, 显示 zero

6.4 条件控制流伪指令


```
.data
x       db   -5
bufp  db  'positive > 0`,0
bufn  db  ' < 0 `,0
zero  db  'zero `,0
Fmt   db "%s",0ah,0dh, 0

.code ……
```

```
.if  x==0
    lea ebx, zero
.elseif  x>0
    lea ebx, bufp
.else
    lea ebx, bufn
.endif
invoke  printf,
        offset Fmt, ebx
```

```
invoke  printf, offset Fmt, offset bufp
```

注意：显示一个串，就要给出串的首地址，串以0结束.

6.4 条件控制流伪指令

实践：结果不正确，为什么？

分析原因：观察目标程序，发现其用的是无符号数比较转移指令。即masm 将X当作无符号数来翻译的。

若要将其当作有符号数，定义形式是：

```
X    SBYTE    -5
```

同样，有SWORD, SDWORD……

6.4 条件控制流伪指令


```
start:
    mov r, 0
    .if x>=0 && y>=0
          mov  r, 1
    .endif
```

组合条件的编译

```
start:
00592030   mov        dword ptr [r (0595019h)],0
0059203A   cmp        dword ptr [x (0595011h)],0
00592041   jl         _start+26h (0592056h)
00592043   cmp        dword ptr [y (0595015h)],0
0059204A   jl         _start+26h (0592056h)
0059204C   mov        dword ptr [r (0595019h)],1
@C0001:
00592056   ……
```

6.5 与转移指令功能类似的指令

6.5.1 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)→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

6.5.1 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

6.5.2 字节指令

语句格式：set***　opd

功　　能：在条件“***”成立时，

(opd)←1，否则 (opd)←0 。

opd 一般为 一个字节寄存器

```
cmp    eax,   ebx
setg   cl
seta   cl
sete   cl
```

6.5.2 字节指令

- 使用单个标志位设置

sete/setz、setc、sets、seto、setp

条件：ZF=1　　CF=1　　SF=1　　OF=1　PF=1

setne/setnz、setnc、setns、setno、setnp

条件：ZF=0　　CF=0　　SF=0　　OF=0　　PF=0

- 使用多个标志位组合设置

seta、setb、setg、setl

setae、setbe、setge、setle

第6章 总结

- 简单条件转移指令
- 有（无）符号数条件转移指令
- 无条件转移指令
- 分支向无分支的转换

分支指令的执行过程

选择分支指令时注意事项

编写分支程序的技巧

第6章 总结

对 C 语言程序编写的启示

- 比较转移时，正确定义变量：有符号数、无符号数

  注意类型的转换：同时有无、有符号类型变量时，处理的结果是什么？

- 对switch 语句的理解

  如何正确地写出 switch case 语句？

- 程序优化

  如何避免分支转移？

  将多个变量的地址组合成一个表

  将多个函数的地址组合成一个表