选择和循环语句的机器级表示

南京大学
计算机科学与技术系
袁春风
email: cfyuan@nju.edu.cn
2015.6

选择结构的机器级表示

- **if ~ else语句的机器级表示**

```
if (cond_expr)
        then_statement
else
        else_statement
```

```
        c=cond_expr;
        if (!c)
                goto false_label;
        then_statement
        goto done;
false_label:
        else_statement
done:
```

```
        c=cond_expr;
        if (c)
                goto true_label;
        else_statement
        goto done;
true_label:
        then_statement
done:
```

红框处为条件转移指令！

篮框处为无条件转移指令！

If-else语句举例

```
int  get_cont( int *p1, int *p2 ) {

        if  ( p1 > p2 )
              return *p2;
        else
              return *p1;
}
```

p1和p2对应实参的存储地址分别为R[ebp]+8、R[ebp]+12，EBP指向当前栈帧底部，结果存放在EAX。

```
movl  8(%ebp), %eax     //R[eax]←M[R[ebp]+8], 即 R[eax]=p1
movl  12(%ebp), %edx    //R[edx]←M[R[ebp]+12], 即 R[edx]=p2
cmpl  %edx, %eax        //比较 p1 和 p2, 即根据p1-p2结果置标志
jbe   .L1               //若 p1<=p2, 则转 L1 处执行
movl  (%edx), %eax      //R[eax]←M[R[edx]], 即 R[eax]=M[p2]
jmp   .L2               //无条件跳转到 L2 执行
.L1:
movl  (%eax), %eax      // R[eax]←M[R[eax]], 即 R[eax]=M[p1]
.L2
```

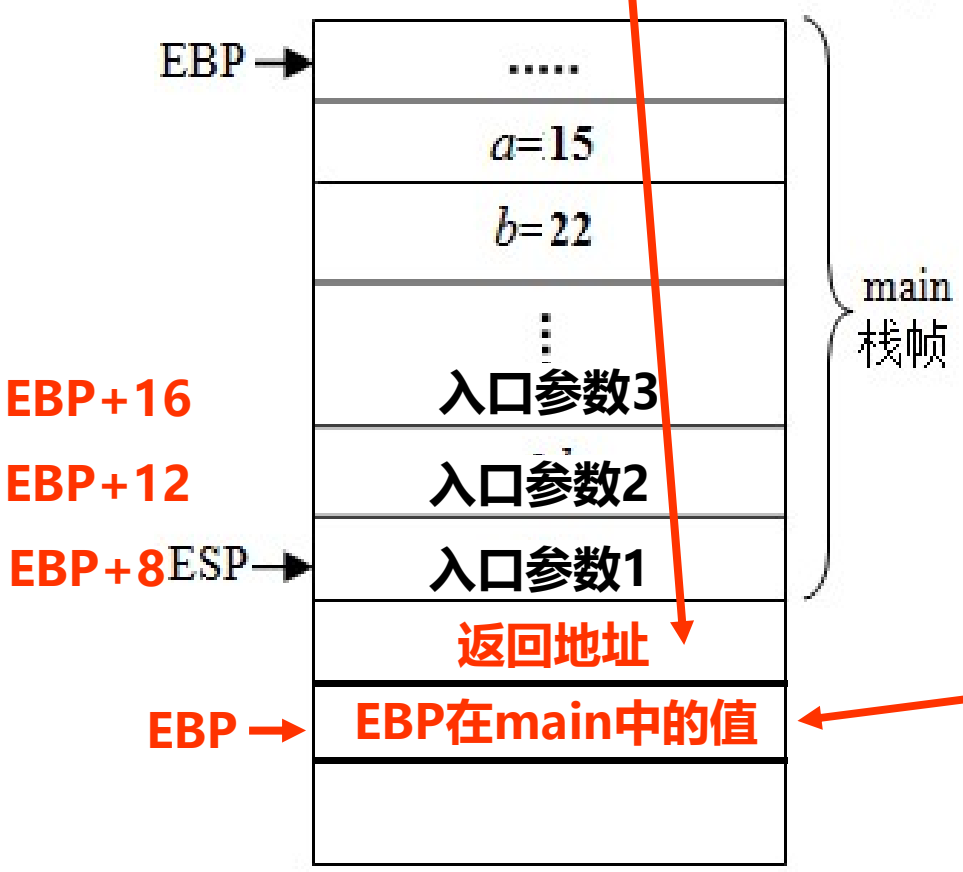
SKIP

入口参数的位置

```
movl  参数3，8(%esp)
.....
movl  参数1, (%esp)
call   add
```

准备入口参数

R[esp]←R[esp]-4
M[R[esp]]←返回地址
R[eip]←add函数首地址



返回地址是什么？

call指令的下一条指令的地址！

- IA-32中，若参数类型是unsigned char、char或unsigned short、short，也都分配4个字节
- 故在被调用函数中，使用R[ebp]+8、R[ebp]+12、R[ebp]+16作为有效地址来访问函数的入口参数
- 每个过程开始两条指令

```
pushl %ebp
movl %esp, %ebp
```

BACK

switch-case语句举例

```
int sw_test(int a, int b, int c)
{
   int result;
   switch(a) {
   case 15:
      c=b&0x0f;
   case 10:
      result=c+50;
      break;
   case 12:
   case 17:
      result=b+50;
      break;
   case 14:
      result=b
      break;
   default:
      result=a;
   }
   return result;
}
```

```
  movl  8(%ebp), %eax
  subl  $10, %eax
  cmpl  $7, %eax
  ja    .L5
  jmp     *.L8( , %eax, 4)
.L1:
  movl  12(%ebp), %eax
  andl  $15, %eax
  movl  %eax, 16(%ebp)
.L2:
  movl  16(%ebp), %eax
  addl    $50, %eax
  jmp  .L7
.L3:
  movl  12(%ebp), %eax
  addl    $50, %eax
   jmp    .L7
.L4:
   movl  12(%ebp), %eax
   jmp  .L7
.L5:
   addl  $10, %eax
.L7:
```

R[eax]=a-10=i

if (a-10)>7 转 L5

转.L8+4*i 处的地址

跳转表在目标文件的只读节中，按4字节边界对齐。

```
   .section  .rodata
   .align 4
.L8                    a=
   .long    .L2    10
   .long    .L5    11
   .long    .L3    12
   .long    .L5    13
   .long    .L4    14
   .long    .L1    15
   .long    .L5    16
   .long    .L3    17
```

循环结构的机器级表示

- do~while循环的机器级表示

```
do  loop_body_statement
      while (cond_expr);
```

```
loop :
    loop_body_statement
    c=cond_expr;
    if (c) goto loop;
```

红色处为条件转移指令！

- while循环的机器级表示

```
while (cond_expr)
      loop_body_statement
```

```
    c=cond_expr;
    if (!c) goto done;
loop :
    loop_body_statement
    c=cond_expr;
    if (c) goto loop;
done :
```

- for循环的机器级表示

```
for (begin_expr; cond_expr; update_expr)
        loop_body_statement
```

```
        begin_expr;
        c=cond_expr;
        if (!c) goto done;
    loop :
        loop_body_statement
        update_expr;
        c=cond_expr;
        if (c) goto loop;
    done :
```

循环结构与递归的比较

递归函数nn_sum仅为说明原理，实际上可直接用公式，为说明循环的机器级表示，这里用循环实现。

```
int  nn_sum ( int n)
{
        int i;
        int result=0;
        for (i=1; i <=n; i++)
            result+=i;
        return result ;
}
```

```
movl  8(%ebp), %ecx
movl  $0, %eax
movl  $1, %edx
cmpl  %ecx, %edx
jg    .L2
.L1:
addl  %edx, %eax
addl  $1, %edx
cmpl  %ecx, %edx
jle   .L1
.L2
```

i 和 result 分别分配在EDX和EAX中。

通常复杂局部变量被分配在栈中，而这里都是简单变量

SKIP

过程体中没用到被调用过程保存寄存器。因而，该过程栈帧中仅需保留EBP，即其栈帧仅占用4字节空间，若考虑栈帧按16B对齐，也仅用16字节，而递归方式则用了16n字节，是n倍关系！每次递归调用都要执行16条指令，一共多了n次过程调用，因而，递归方式比循环方式至少多执行16n条指令。由此看出，为提高程序性能，能用非递归方式执行则最好用非递归方式。

递归过程调用举例

```
int  nn_sum ( int n)
{
        int result;
        if  (n<=0 )
            result=0;
        else
            result=n+nn_sum(n-1);
        return  result ;
}
```

```
 pushl   %ebp
 movl    %esp, %ebp
 pushl    %ebx
 subl    $4, %esp
 movl    8(%ebp), %ebx
 movl    $0, %eax
 cmpl    $0, %ebx
 jle     .L2
 leal    -1(%ebx), %eax
 movl    %eax, (%esp)
 call    nn_sum
 addl    %ebx, %eax
.L2
 addl    $4, %esp
 popl    %ebx
 popl    %ebp
 ret
```

BACK

时间开销：每次递归执行16条指令，共16n条指令

空间开销：一次调用增加16B栈帧，共16n

逆向工程举例

```
int function_test( unsigned x)
{
    int result=0;
    int i ;
    for ( ____①____ ; ____②____ ; ____③____ ) {
            ____________④____________ ;
     }
     return result;
}
```

```
movl   8(%ebp), %ebx
movl   $0, %eax
movl   $0, %ecx
.L12:
leal   (%eax,%eax), %edx
movl   %ebx, %eax
andl   $1, %eax
orl    %edx, %eax
shrl   %ebx
addl   $1, %ecx
cmpl   $32, %ecx
jne    .L12
```

① 处为i=0，② 处为i≠32，③ 处为i++。

入口参数 x 在EBX中，返回参数 result 在EAX中。LEA实现“2*result”，即：将result左移一位；第6和第7条指令则实现“x&0x01”；第8条指令实现“result=(result<<1) | (x & 0x01)”，第9条指令实现“x>>=1”。综上所述，④ 处的C语言语句是：

“result=(result<<1) | (x & 0x01); x>>=1;”。