编译原理与技术 课程设计第二次提交

邱喆(5110379077) 张凯源(5110379046)

一、翻译方案

1) 条件语句

```
if ( condition ) then {
    //do A
}else{
    //do B
}
```

可以翻译为

```
    # calculate condition here, save result
    testl %eax, %eax
    jge ELSE
    # do A
ELSE:
    # do B
```

2) 循环语句

while (condition) { #do something }	CONDITION: # calculate condition testl %eax, %eax jge OUT: # do something OUT:
do{ //do A }while(a)	START_OF_LOOP: #do A #calculate condition testl %eax, %eax jl START_OF_LOOP OUT:
for(exprA ; exprB ; exprC){ //do something A	#do expr A START_OF_LOOP:

<table>
<tr><td><pre>}</pre></td><td><pre>   #calculate condition B
   testl %eax, %eax
   jge OUT
   #do something
   #calculate exprC
OUT</pre></td></tr>
<tr><td><pre>int a[];
foreach(i in a){
  //.....
}</pre></td><td>First transform it to a regular for</td></tr>
</table>

Complex Example:

<table>
<tr><td><pre>for (exprA ; exprB ; exprC){
    //do a
    break;
    //do b
    continue;
    //do C
}</pre></td><td><pre>#do expr A
START_OF_LOOP:
   #calculate condition B
   testl %eax, %eax
   jge OUT
   #do A
   jmp OUT
   #do B
   jmp UPDATE_CONDITION:
   #do C
UPDATE_CONDITION:
   #calculate exprC
OUT</pre></td></tr>
</table>

3) 函数调用与参数传递

函数调用用 x86 的 call 指令完成

```
    call Main
Main:
```

函数传递的参数按照语言中声明的顺序逆序，即从右往左依此压栈。Int 类型的返回值通过%eax 寄存器返回。调用者保存所有寄存器状态。

<table>
<tr><td><pre>int foo(int a , int b , int c){
  return a + b + c ;
}
int main(){
return foo(1,2,3);
}</pre></td><td><pre>Foo:
    pushl %ebp
    movl %esp, %ebp
    # argument a
    movl 8(%esp), %eax
    # argument b
    movl 12(%esp), %ebx</pre></td></tr>
</table>

	`` # argument c movl 16(%esp), %ecx addl %ebx, %eax addl %ecx, %eax #move to %eax to return movl %eax, %eax leave ret Main: pushl %ebp movl %esp, %ebp #save registers pushl %eax pushl %ebx pushl %ecx pushl %edx pushl %esi pushl %edi # send arguments pushl $3 pushl $2 pushl $1 call Foo movl %eax, %eax leave ret ```

4) 函数声明

函数声明采用在汇编中添加标签。函数的参数获得、返回值处理等，参见上一节以及对应实例。

5) 左值与右值

在对象赋值等操作中，需要获得变量的左值。左值的获取本质上是一个取地址的操作，通过类似于指针解引用的操作。

比如，

```
int a;
int main(){
a=2;
```

```
}
```

汇编：

```
.data:
    intA
    .long 0
.text:
    Main:
    leal intA, %eax #now %eax contains the address of A
    movl $2, (%eax)
```

6) 数组引用

数组引用的左值、右值，使用 x86 对应的伸缩地址引用。获得内存地址后，根据左值右值需要，分别使用 movl 和 leal 操作：

```
int a[4];
int main(){
    a[2]=a[3];
}
```

```
.data
    A:
    .long 0
    .long 0
    .long 0
    .long 0
.text
    Main:
    pushl %ebp
    movl %esp,%ebp
    movl A, %eax
    #lvalue
    leal (%eax,2,4), %ebx
    #rvalue
    movl (%eax,3,4), %ecx
    #assign
    movl %ecx, (%ebx)
```

7) 结构引用

维护结构中每个 field 对应的 offset 之后，类似于数组处理。

二、快速排序算法-汇编实现

```
.data
strtag1:
    .ascii "%d "
```

```
strtag2:
    .ascii "\n"

.text
.globl _my_qsort
_my_qsort:
    pushl  %ebp
    movl   %esp, %ebp
    movl   (%edi, %esi, 4), %eax
#;;;i = begin
    movl   %esi, %ebx
#;;; j = end
    movl   %edx, %ecx
start_loop:
#;;; while (i <= j)
    cmpl   %ebx, %ecx
    jl  end_loop

#;;; while (a[i] <= pivot)
loop1:
    cmpl   (%edi, %ebx, 4), %eax
    jle end1
    incl   %ebx
    jmp start_loop
end1:
#;;; while (a[j] > pivot)
loop2:
    cmpl   (%edi, %ecx, 4), %eax
    jge end2
    decl   %ecx
    jmp start_loop
end2:
    cmpl   %ebx, %ecx
    jl  no_swap
    pushl  %eax
    pushl  %esi
    movl   (%edi, %ebx, 4), %eax
    movl   (%edi, %ecx, 4), %esi
    movl   %esi, (%edi, %ebx, 4)
    movl   %eax, (%edi, %ecx, 4)
    popl   %esi
    popl   %eax

    incl   %ebx
```

```
    decl   %ecx

no_swap:
    jmp start_loop

end_loop:

#;;; if (begin < j)
    cmpl   %esi, %ecx
    jle skip1
    pushl  %edx
    pushl  %ebx
    pushl  %ecx
    movl   %ecx, %edx
    call   _my_qsort
    popl   %ecx
    popl   %ebx
    popl   %edx
skip1:
#;;; if (i < end)
    cmpl   %ebx, %edx
    jle skip2
    pushl  %ebx
    pushl  %ecx
    pushl  %esi
    movl   %ebx, %esi
    call   _my_qsort
    popl   %esi
    popl   %ecx
    popl   %ebx
skip2:
    leave
    ret

.globl main
main:
    pushl  %ebp
    movl   %esp, %ebp
    subl   $0x40, %esp
    leal   0x4(%esp), %ebx
    movl   $strtag1, %edi
    movl   $0, %eax
read_loop:
    cmpl   $10, %eax
```

```
    jge for_loop_over1
    leal   (%ebx, %eax, 4), %esi
    pushl  %eax
    pushl  %esi
    pushl  %edi
    call   __isoc99_scanf
    popl   %edi
    popl   %esi
    popl   %eax
    addl   $1, %eax
    jmp read_loop
for_loop_over1:
    movl   %ebx, %edi
    pushl  %ebx
    movl   $0, %esi
    movl   $9, %edx
    call   _my_qsort
    popl   %ebx

    movl   $0, %eax
    movl   $strtag1, %edi
write_loop:
    cmpl   $10, %eax
    jge for_loop_over2
    movl   (%ebx, %eax, 4), %esi
    pushl  %eax
    pushl  %esi
    pushl  %edi
    call   printf
    popl   %edi
    popl   %esi
    popl   %eax
    addl   $1, %eax
    jmp write_loop
for_loop_over2:

    movl   $0, %eax
    leave
    Ret
```