# 编译原理与技术课程设计第三次提交

## 翻译方案

邱喆(5110379077) 张凯源(5110379046)

# 一、翻译方案

## 1) 表达式求值

对于表达式，DLang 采用基于堆栈的求值方法，在经过前面的分析得到正确的语法树之后，DLang 采用类似后缀表达式求值的方法：每次将栈顶的若干元素弹出（个数取决于操作的种类）到寄存器，之后将对应的汇编操作语句写好，然后将结果压栈。

示例：

a + b * c

翻译后应为：

将 a，b，c 从相应的内存地址取出并按照 a，b，c 的顺序压栈

```
popl      %eax
popl      %ebx
imull     %ebx,  %eax
pushl     %eax
popl      %eax
popl      %ebx
addl      %ebx,  %eax
pushl     %eax
```

## 2) 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
```

## 3) 循环语句

| | |
|---|---|
| while ( condition ) {<br>    #do something<br>} | CONDITION:<br>    # calculate condition<br>    testl %eax, %eax<br>    jge OUT<br>    # do something<br>    jmp CONDITION<br>OUT: |
| do{<br>   //do A<br>}while(a) | START_OF_LOOP:<br>    #do A<br>    #calculate condition<br>    testl %eax, %eax<br>    jl START_OF_LOOP<br>OUT: |
| for( exprA ; exprB ; exprC ){<br>    //do something A<br>} | #do expr A<br>START_OF_LOOP:<br>   #calculate condition B<br>   testl %eax, %eax<br>   jge OUT<br>   #do something<br>   #calculate exprC<br>   jmp START_OF_LOOP<br>OUT |
| int a[];<br>foreach(i in a){<br>  //.....<br>} | **First transform it to a regular for** |

**Complex Example:**

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

| | |
|---|---|
| | jmp UPDATE_CONDITION:<br>#do C<br>UPDATE_CONDITION:<br>#calculate exprC<br>jmp START_OF_LOOP<br>OUT |

## 4) 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。

| | |
|---|---|
| int foo( int a , int b , int c ){<br>return a + b + c ;<br>}<br>int main(){<br>return foo(1,2,3);<br>} | Foo:<br>pushl %ebp<br>movl %esp, %ebp<br># argument a<br>movl 8(%esp), %eax<br># argument b<br>movl 12(%esp), %ebx<br># argument c<br>movl 16(%esp), %ecx<br>addl %ebx, %eax<br>addl %ecx, %eax<br>#move to %eax to return<br>movl %eax, %eax<br>leave<br>ret<br><br>Main:<br>pushl %ebp<br>movl %esp, %ebp<br>#save registers<br>pushl %eax<br>pushl %ebx<br>pushl %ecx<br>pushl %edx<br>pushl %esi<br>pushl %edi |

|  |  |
|---|---|
|  | # send arguments<br>pushl $3<br>pushl $2<br>pushl $1<br>call Foo<br>movl %eax, %eax<br>leave<br>ret |

## 5) 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。

## 6) 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)
```

## 7) 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];<br>int main(){<br>    a[2]=a[3];<br>} | .data<br>    A:<br>    .long 0<br>    .long 0 |

| | |
|---|---|
| | .long 0<br>.long 0<br>.text<br>Main:<br>pushl %ebp<br>movl %esp,%ebp<br>movl A, %eax<br>#lvalue<br>leal (%eax,2,4), %ebx<br>#rvalue<br>movl (%eax,3,4), %ecx<br>#assign<br>movl %ecx, (%ebx) |

### 8) 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
```