实验预备工作实验报告

徐云凯 1713667

# 设计语言

本C++语言子集参考SysY语言。对于该语言的上下文无关文法描述，下面依次给出定义。

1. 终结符集合

该语言终结符包含：

标识符

数值常量

对于标识符（）：

对于数值常量（）：

1. 非终结符集合

编译单元 CompUnit

声明 Decl

常量声明 ConstDecl

基本类型 BType

常数定义 ConstDef

常量初值 ConstInitVal

变量声明 VarDecl

变量定义 VarDef

变量初值 InitVal

函数定义 FuncDef

函数类型 FuncType

函数形参表 FuncFParams

函数形参 FuncFParam

语句块 Block

语句块项 BlockItem

语句 Stmt

表达式 Exp

条件表达式 Cond

左值表达式 LVal

基本表达式 PrimaryExp

数值 Number

一元表达式 UnaryExp

单目运算符 UnaryOp

函数实参表 FuncRParams

乘除模表达式 MulExp

加减表达式 AddExp

关系表达式 RelExp

相等性表达式 EqExp

逻辑与表达式 LAndExp

逻辑或表达式 LOrExp

常量表达式 ConstExp

1. 开始符号

编译单元 CompUnit

1. 产生式集合

以下按照语言特性分别给出该语言各个部分的产生式。

1. 编译单元

|  |  |  |
| --- | --- | --- |
| 编译单元 | CompUnit → | CompUnit Decl |
|  |  | | CompUnit FuncDef |
|  |  | | Decl |
|  |  | | FuncDef |
| 声明 | Decl → | ConstDecl ';' |
|  |  | | VarDecl ';' |

1. 常量与变量

该部分中部分表述在SysY语言EBNF中的描述不符合普通上下文无关文法要求，补充定义了InitValList非终结符以完善其上下文无关文法定义。

|  |  |  |
| --- | --- | --- |
| 常量声明 | ConstDecl → | 'const' BType ConstDef |
|  |  | | ConstDecl ',' ConstDef |
| 基本类型 | BType → | 'int' |
| 常数定义 | ConstDef → | Ident { '[' ConstExp ']' } '=' ConstInitVal |
| 常量初值 | ConstInitVal → | ConstExp |
|  |  | | '{' '}' |
|  |  | | '{' InitValList '}' |
| 变量声明 | VarDecl → | BType VarDef | VarDecl ',' VarDef |
| 变量定义 | VarDef → | Ident { '[' ConstExp ']' } |
|  |  | | Ident { '[' ConstExp ']' } '=' InitVal |
| 变量初值 | InitVal → | Exp |
|  |  | | '{' '}' |
|  |  | | '{' InitValList '}' |
|  | InitValList→ | ConstExp | InitValList, ConstExp |

1. 函数

|  |  |  |
| --- | --- | --- |
| 函数定义 | FuncDef → | FuncType Ident '(' ')' Block |
|  |  | | FuncType Ident '(' FuncFParams ')' Block |
| 函数类型 | FuncType → | 'void' | 'int |
| 函数形参表 | FuncFParams → | FuncFParam | FuncFParams ',' FuncFParam |
| 函数形参 | FuncFParam → | BType Ident | BType Ident '[' ']' |
|  |  | | FuncFParam '[' Exp ']' |

1. 语句

|  |  |  |
| --- | --- | --- |
| 语句块 | Block → | '{' { BlockItem } '}' |
| 语句块项 | BlockItem → | Decl | Stmt |
| 语句 | Stmt → | ; | LVal '=' Exp ';' |
|  |  | | Exp ';' |
|  |  | | Block |
|  |  | | 'if' '( Cond ')' Stmt |
|  |  | | 'if' '( Cond ')' Stmt 'else' Stmt |
|  |  | | 'while' '(' Cond ')' Stmt |
|  |  | | 'break' ';' |
|  |  | | 'continue' ';' |
|  |  | | 'return' ';' |
|  |  | | 'return' Exp ';' |

1. 表达式

|  |  |  |
| --- | --- | --- |
| 表达式 | Exp → | AddExp |
| 条件表达式 | Cond → | LOrExp |
| 左值表达式 | LVal → | Ident | LVal '[' Exp ']' |
| 基本表达式 | PrimaryExp → | '(' Exp ')' | LVal | Number |
| 数值 | Number → | IntConst |
| 一元表达式 | UnaryExp → | PrimaryExp |
|  |  | | Ident '(' ')' |
|  |  | | Ident '(' FuncRParams ')' |
|  |  | | UnaryOp UnaryExp |
| 单目运算符 | UnaryOp → | '+' | '−' | '!' |
| 函数实参表 | FuncRParams → | Exp | FuncRParams ',' Exp |
| 乘除模表达式 | MulExp → | UnaryExp |
|  |  | | MulExp '\*' UnaryExp |
|  |  | | MulExp '/' UnaryExp |
|  |  | | MulExp '%' UnaryExp |
| 加减表达式 | AddExp → | MulExp |
|  |  | | AddExp '+' MulExp |
|  |  | | AddExp '-' MulExp |
| 关系表达式 | RelExp → | AddExp |
|  |  | | RelExp '<' AddExp |
|  |  | | RelExp '>' AddExp |
|  |  | | RelExp '<=' AddExp |
|  |  | | RelExp '>=' AddExp |
| 相等性表达式 | EqExp → | RelExp |
|  |  | | EqExp '==' RelExp |
|  |  | | EqExp '!=' RelExp |
| 逻辑与表达式 | LAndExp → | EqExp | LAndExp '&&' EqExp |
| 逻辑或表达式 | LOrExp → | LAndExp | LOrExp '||' LAndExp |
| 常量表达式 | ConstExp → | AddExp |

# 编写汇编程序

对于预备工作1中的阶乘程序，将其稍加改写以充分体现语言特性，如使用全局变量，将乘法分拆为函数等，随后对照c语言代码编写其汇编语言版本。（完整的C语言源码与汇编源码已附后）

在编写汇编程序的过程中，先参考实验指导中的汇编程序演示，将全局变量，常量与函数分别写好，然后在main程序段中写好循环与对mult，scanf，printf等函数的调用。但是在第一次汇编运行时出现多个错误。

首先是对于初始值不为0的全局变量不能放置于.bss标签后，应当在前方单独声明全局标识符，并使用.long标签对32位整型变量进行初始化。另外，全局变量的声明必须加上.global标识符以保证后续代码能够正常调用该变量，这一点在实验指导中并未出现。

最后，对比gcc给出的汇编代码，gcc的汇编更长，加入了更多的编译器标记。在代码流程上，其主体循环基本一致，最后的结束返回部分包含了一个堆栈完整性检查，使得有两个ret指令，分别是正常返回（return 0）和堆栈有问题时的返回。

# 附录

Makefile

.PHONY: asm, my, clean

default:

cc -O0 -m32 -o main.out main.c

asm:

cc -O0 -m32 -S -o main.S main.c

my:

cc -O0 -m32-march=i386 -o mymain.out mymain.S

qemu-i386 mymain.out

clean:

-rm \*.out

main.c

#include <stdio.h>

int x = 1;

int n = 0;

int mult(int *a*, int *b*){

    return *a* \* *b*;

}

void main(){

    scanf("%d", &n);

    while(n>0){

        x = mult(x, n);

        n--;

    }

    printf("%d\n", x);

}

mymain.S

# 函数 mult

    .text

    .globl mult

    .type mult, @function

mult:

    movl 4(%esp), %eax

    imull 8(%esp), %eax

    ret

# 全局变量

    .globl  x

    .data

    .align 4

    .type   x, @object

    .size   x, 4

x:

    .long   1

    .globl  n

    .bss

    .align 4

    .type   n, @object

    .size   n, 4

n:

    .zero   4

# 常量

    .section .rodata

STR0:

    .string "%d"

STR1:

    .string "%d\n"

# 主函数

    .text

    .globl main

    .type main, @function

main:

# scanf("%d", &n);

    pushl $n

    pushl $STR0

    call scanf

    addl $8, %esp

L1:

# end if n<=0

    movl n, %eax

    cmpl $0, %eax

    jle L2

# calculate x \* n

    pushl x

    pushl n

    call mult

    addl $8, %esp

# x = x \* n;

    movl %eax, x

# n--;

    subl $1, n

    jmp L1

L2:

# printf("%d\n", x);

    pushl x

    pushl $STR1

    call printf

    addl $8, %esp

# return 0;

    xorl %eax, %eax

    ret

# 可执行堆栈段

    .section .note.GNU-stack,"",@progbits