实验预备工作实验报告

徐云凯 1713667

一、 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(V_T, V_N, P, S)，下面依次给出定义。

1. 终结符集合V_T

该语言终结符包含：

标识符	*Ident*
数值常量	*IntConst*

对于标识符（$identifier$）：

$$
\begin{aligned}
identifier &\rightarrow identifier\_nondigit \\
&\mid identifier\ identifier\_nondigit \\
&\mid identifier\ identifier\_digit \\
identifier\_nondigit &\rightarrow \text{'\_'} \mid \text{'a'} \mid \text{'b'} \mid \text{'c'} \mid \text{'d'} \mid \text{'e'} \mid \text{'f'} \mid \text{'g'} \mid \text{'h'} \mid \text{'i'} \mid \text{'j'} \mid \text{'k'} \mid \text{'l'} \mid \text{'m'} \\
&\mid \text{'n'} \mid \text{'o'} \mid \text{'p'} \mid \text{'q'} \mid \text{'r'} \mid \text{'s'} \mid \text{'t'} \mid \text{'u'} \mid \text{'v'} \mid \text{'w'} \mid \text{'x'} \mid \text{'y'} \mid \text{'z'} \\
&\mid \text{'A'} \mid \text{'B'} \mid \text{'C'} \mid \text{'D'} \mid \text{'E'} \mid \text{'F'} \mid \text{'G'} \mid \text{'H'} \mid \text{'I'} \mid \text{'J'} \mid \text{'K'} \mid \text{'L'} \mid \text{'M'} \\
&\mid \text{'N'} \mid \text{'O'} \mid \text{'P'} \mid \text{'Q'} \mid \text{'R'} \mid \text{'S'} \mid \text{'T'} \mid \text{'U'} \mid \text{'V'} \mid \text{'W'} \mid \text{'X'} \mid \text{'Y'} \mid \text{'Z} \\
identifier\_digit &\rightarrow \text{'0'} \mid \text{'1'} \mid \text{'2'} \mid \text{'3'} \mid \text{'4'} \mid \text{'5'} \mid \text{'6'} \mid \text{'7'} \mid \text{'8'} \mid \text{'9'}
\end{aligned}
$$

对于数值常量（$IntConst$）：

$$
\begin{aligned}
integer\_const &\rightarrow decimal\_const \\
&\mid octal\_const \\
&\mid hexadecimal\_const \\
decimal\_const &\rightarrow nonzero\_digit \\
&\mid decimal\_const\ digit \\
octal\_const &\rightarrow \text{'0'} \mid octal\_const\ octal\_digit \\
hexadecimal\_const &\rightarrow hexadecimal\_prefix\ hexadecimal\_digit \\
&\mid hexadecimal\_const\ hexadecimal\_digit \\
hexadecimal\_prefix &\rightarrow \text{'0}x\text{'} \mid \text{'0}X\text{'} \\
nonzero\_digit &\rightarrow \text{'1'} \mid \text{'2'} \mid \text{'3'} \mid \text{'4'} \mid \text{'5'} \mid \text{'6'} \mid \text{'7'} \mid \text{'8'} \mid \text{'9'} \\
octal\_digit &\rightarrow \text{'0'} \mid \text{'1'} \mid \text{'2'} \mid \text{'3'} \mid \text{'4'} \mid \text{'5'} \mid \text{'6'} \mid \text{'7'} \\
hexadecimal\_digit &\rightarrow \text{'0'} \mid \text{'1'} \mid \text{'2'} \mid \text{'3'} \mid \text{'4'} \mid \text{'5'} \mid \text{'6'} \mid \text{'7'} \mid \text{'8'} \mid \text{'9'}
\end{aligned}
$$

| 'a' | 'b' | 'c' | 'd' | 'e' | 'f' | 'A' | 'B' | 'C' | 'D' | 'E' | 'F'

2. 非终结符集合V_N

编译单元	CompUnit	表达式	Exp
声明	Decl	条件表达式	Cond
常量声明	ConstDecl	左值表达式	LVal
基本类型	BType	基本表达式	PrimaryExp
常数定义	ConstDef	数值	Number
常量初值	ConstInitVal	一元表达式	UnaryExp
变量声明	VarDecl	单目运算符	UnaryOp
变量定义	VarDef	函数实参表	FuncRParams
变量初值	InitVal	乘除模表达式	MulExp
函数定义	FuncDef	加减表达式	AddExp
函数类型	FuncType	关系表达式	RelExp
函数形参表	FuncFParams	相等性表达式	EqExp
函数形参	FuncFParam	逻辑与表达式	LAndExp
语句块	Block	逻辑或表达式	LOrExp
语句块项	BlockItem	常量表达式	ConstExp
语句	Stmt		

3. 开始符号S

编译单元	CompUnit

4. 产生式集合P

以下按照语言特性分别给出该语言各个部分的产生式。

a. 编译单元

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

b. 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

c. 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 ']'

d. 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 ';'

e. 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

二、 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
```

```
16      x:
17          .long   1
18          .globl  n
19          .bss
20          .align 4
21          .type   n, @object
22          .size   n, 4
23      n:
24          .zero   4
25
26      # 常量
27          .section .rodata
28      STR0:
29          .string "%d"
30      STR1:
31          .string "%d\n"
32
33      # 主函数
34          .text
35          .globl main
36          .type main, @function
37      main:
38      # scanf("%d", &n);
39          pushl $n
40          pushl $STR0
41          call scanf
42          addl $8, %esp
43      L1:
44      # end if n<=0
45          movl n, %eax
46          cmpl $0, %eax
47          jle L2
48      # calculate x * n
49          pushl x
50          pushl n
51          call mult
52          addl $8, %esp
53      # x = x * n;
54          movl %eax, x
55      # n--;
56          subl $1, n
57          jmp L1
58      L2:
59      # printf("%d\n", x);
60          pushl x
61          pushl $STR1
62          call printf
63          addl $8, %esp
64      # return 0;
65          xorl %eax, %eax
66          ret
67      # 可执行堆栈段
68          .section .note.GNU-stack,"",@progbits
69
```