南開大學
Nankai University

南　开　大　学

网络空间安全学院

编译原理实验报告

第二次作业

沈梦菊 11911563 信息安全

王婧嵘 1911569 信息安全法学

年级：2019 级

指导教师：王刚

2021 年 10 月 11 日

摘 要

我们设计了 SysY 上下文的无关文法，文法涉及 SysY 变量声明、变量定义、常量变量、全局变量、函数声明、函数定义、语句块定义、条件语句循环语句以及算数表达式的定义。另外还进行了 arm 汇编编程的练习。

关键字： SysY，**上下文无关文法**,arm **汇编**

目录

1 引言

编译器可以将源语言编译为目标语言，再经由汇编器、链接器最后生成可执行文件。我们打算借助 Lex、Yacc 等工具为 SysY 语言编写一个编译器。为此，我们首先需要定义 SysY 语言的文法特性，并使用上下文无关文法将其描述出来。编译器得到的目标语言是汇编代码，为了能够检查自己编译器得到结果的正确与否，最终保证它可以通过汇编器链接器生成可执行代码。我们也进行手工编写汇编代码的练习来熟悉 arm 汇编编程。

2 编译器定义

定义的编译器支持 int 数据类型，支持变量声明与变量定义，可以声明单变量、变量列表或者数组。支持常量与全局变量的定义。由于支持全局变量与局部变量，那么就存在作用域的问题，作用域规则模仿 C 语言，例如：局部变量作用域可以屏蔽同名全局变量作用域等。不过具体的实现方式以及在哪个阶段实现还没有思考清楚。

编译器支持赋值语句、支持加减等运算操作、支持条件比较语句、支持循环语句、支持函数声明与函数定义。具体针对各种语言特性的上下文无关文法如下。

2.1 上下文无关文法定义

SysY 程序主体由声明以及定义语句组成，秉持自顶向下的原则，我们先从变量定义声明开始，然后是函数定义声明。函数由语句块组成，所以我们紧接着描述语句块以及语句，最后就是表达式的定义。

2.2 变量定义以及声明

下面生产式中用到的 identifier 以及 ConstExpr 是非终结符，对应生产式已经很清楚的在 SysY 手册中定义了，这里就不加以赘述了。

$$
\begin{aligned}
\textbf{编译单元}\ ComplieUnit &\longrightarrow ComplieUnit\ Decl|\epsilon \\
\textbf{声明}\ Decl &\longrightarrow VarDecl|FuncDecl|ConstVarDecl \\
\textbf{变量声明}\ VarDecl &\longrightarrow Type\ Array|Type\ Varlist \\
\textbf{变量类型}\ Type &\longrightarrow int \\
\textbf{变量列表}\ Varlist &\longrightarrow identifier, Varlist|identifier \\
&\quad | \ identifier = identifier, Varlist \\
&\quad | \ identifier = identifier \\
&\quad | \ identifier = ConstExpr, Varlist \\
&\quad | \ identifier = ConstExpr \\
\textbf{数组}\ Array &\longrightarrow ArrayVar = ArrayValue \\
ArrayVar &\longrightarrow Identifier\ [ConstExpr\ ]ArrayVar|\ [ConstExpr\ ]|\epsilon \\
ArrayValue &\longrightarrow \{ArrayValue\}|num, ArrayValue|num|\epsilon \\
\textbf{常量声明}\ ConstVarDecl &\longrightarrow const\ Type\ ConstVarlist|\ const\ Type\ ConstArray \\
ConstVarlist &\longrightarrow identifier = ConstExpr, ConstVarlist|identifier = ConstExpr \\
ConstArray &\longrightarrow ArrayVar = ArrayValue
\end{aligned}
$$

2.3 函数定义以及声明

$$
\begin{aligned}
\textbf{函数声明}\ FuncDecl &\longrightarrow returnType\ identifier\ FuncParam\ Block \\
\textbf{返回值类型}\ returnType &\longrightarrow void \mid int \\
\textbf{形参列表}\ FuncParam &\longrightarrow Type\ identifier\ , FuncParam|Type\ identifier|\varepsilon
\end{aligned}
$$

2.4 语句块

非终结符可用表格列出

2.4.1 语句块

语句块，也即复合语句（compound statement），由一对花括号和它们包含的语句组成，被视为一条语句 Stmt，用非终结符 Block 表示。为方便定义上下文无关文法，我们将这一对花括

号中间包含的语句成为语句块项，用非终结符 BlockItem 表示。

则可得到语句块的上下文无关定义：

$$\textbf{语句块}\ Block \longrightarrow \{\ BlockItem\ \}$$

2.4.2 语句块项

语句块项代表语句块的一对花括号中间的语句，由非终结符 BlockItem 表示。根据 C++ 语言规范，语句块项既可以是声明的组合，也可以是语句的组合，亦可以是声明和语句的组合，还可以为空。因此，可以使用递归定义的方法，确定语句块项的定义如下：

$$\textbf{语句块项}\ BlockItem \longrightarrow Decl\ BlockItem \mid Stmt\ BlockItem \mid \varepsilon$$

2.4.3 语句

语句由非终结符 Stmt 表示，本文中定义的语句包括赋值语句、分支语句(if…else 和 switch)、循环语句（while、do-while 和 for)、表达式语句、跳转语句（continue、break 和 return)、复合语句（语句块）和空语句。

(1) 赋值语句

赋值语句由非终结符 LValStmt 表示。根据赋值语句的结构，赋值语句将表达式 Expr 的值赋给左值 LVal，因此可如下定义：

$$\textbf{赋值语句}\ LValStmt \longrightarrow LVal\ =\ Expr\ ;$$

(2) 分支语句

分支语句由非终结符 CoDStmt 表示。SysY 语言有两种形式的条件语句，分别是：if 分支和 switch 分支语句，具体结构如下所示：

```
//if语句格式，if语句后也可以没有else分支
if(boolean-expression)
{
    Statement1;
}
else
{
    Statement2;
}

//switch语句格式
switch(expression)
{
```

```
    case expression1:
    Statement1;
    (break;)
    case expression2:
    Statement2;
    (break;)
    case expression3:
    Statement3;
    (break;)
    ...
    default:
    Statementn;
    (break;)
}
```

对于 switch 语句中的 case 语句，要进行递归定义，因此用非终结符 CaseStmt 表示 case 语句。由此可得定义如下：

$$
\begin{aligned}
\textbf{分支语句}\ CoDStmt \longrightarrow\ & if\ (\ Expr\ )\ Stmt \\
& |\ if\ (\ Expr\ )\ Stmt\ else\ Stmt \\
& |\ if\ (\ Expr\ )\ Stmt\ else\ if\ (\ Expr\ )\ Stmt \\
& |\ switch\ (\ Expr\ )\ \{\ CaseStmt\ default\ :\ Stmt\ ;\ \} \\
& |\ switch\ (\ Expr\ )\ \{\ default\ :\ Stmt\ ;\ CaseStmt\ \} \\
\text{case }\textbf{语句}\ CaseStmt \longrightarrow\ & CaseStmt\ case\ Expr\ :\ Stmt\ ;\ |\ case\ Expr\ :\ Stmt\ ;
\end{aligned}
$$

(3) 循环语句

循环语句由非终结符 LoopStmt 表示，SysY 语言中循环语句主要有三种形式：while 循环、do-while 循环和 for 循环，其具体格式如下：

```
//while语句格式
while(Expression)
{
    Statement1;
}

//do-while语句格式
do{
    Statement2;
}while(Expression)

//for语句格式
for(initialization;Expression;Expression)
{
```

```
    Statement3;
}
```

由于 for 语句中的 initialization 部分设置初始值的格式不能简单用 Stmt 表示，故用非终结符 Forinit 表示该语句。同理，用非终结符 ForExpr 表示 for 循环中的表达式。

具体定义如下：

$$
\begin{aligned}
\textbf{循环语句}\ LoopStmt &\longrightarrow while\ (\ Expr\ )\ Stmt \\
&\quad \mid do\ Stmt\ while\ (\ Expr\ )\ ; \\
&\quad \mid for\ (\ Forinit\ ;\ ForExpr\ ;\ ForExpr\ )\ Stmt \\
\text{for\_initialization}\ \ Forinit &\longrightarrow VarDecl\ LValStmt\ ; \\
\text{for\_expression}\ ForExpr &\longrightarrow Expr \mid \varepsilon
\end{aligned}
$$

(4) 表达式语句

表达式语句由非终结符 ExprStmt 表示，根据表达式的定义可定义表达式语句如下：

$$\textbf{表达式语句}\ ExprStmt \longrightarrow Expr\ ;$$

(5) 跳转语句

跳转语句由非终结符 JmpStmt 表示。C++ 语言中常见的跳转语句有 break、continue 和 return 三种，具体格式如下：

```
//break语句格式
break;

//continue语句格式
continue;

//return语句格式
return (expression);
```

其中 return 语句有可能返回表达式也有可能返回空值，因此用非终结符 returnExpr 表示 return 语句返回的值。根据以上格式，具体定义如下：

$$
\begin{aligned}
\textbf{跳转语句}\ JmpStmt &\longrightarrow break\ ; \\
&\quad \mid continue\ ; \\
&\quad \mid return\ returnExpr\ ; \\
\text{returnExpr}\ \ returnExpr &\longrightarrow Expr \mid \varepsilon \mid void
\end{aligned}
$$

(6) 复合语句

复合语句由非终结符 Block 表示。对复合语句的定义已经在前文详尽叙述了，在此不再赘述。

综上所述，语句 Stmt 中包括赋值语句 LValStmt、分支语句 CoDStmt、循环语句 LoopStmt、表达式语句 ExprStmt 跳转语句 JmpStmt、复合语句 Block 和空语句 ‘;’。由此得到语句 Stmt 的定义如下：

语句 $Stmt \longrightarrow LValStmt \mid CoDStmt \mid LoopStmt \mid ExprStmt \mid JmpStmt \mid Block \mid ;$

2.5 表达式

在 SysY 语言中有算术运算、逻辑运算和关系运算三种类型运算。

算术运算主要有 +、-、-（单目减）、*、/、%、~（按位取反）、& 和 |，运算符优先级为-、~ > *、/ > +、- > &、|，运算结果为数值类型。

逻辑运算主要有!（逻辑非、单目运算）、&&（逻辑与）和 ||（逻辑或），运算符优先级为! > && > ||，运算结果为 bool 类型，0 表示假、1 表示真。

关系运算主要有 ==、!=、<、>、>= 和 <=，运算符优先级为 <、>、>=、<= > ==、!=，运算结果为 bool 类型，0 表示假、1 表示真。

对于三种运算之间的优先级，查阅《c++ 程序设计》后认为，单目运算符 > 双目算术运算符 > 双目关系运算符 > 双目逻辑运算符。而单目运算符!（逻辑非）、-（单目减）和 ~（按位取反）之间的顺序应当认为是 - > ! > ~。

对于表达式的定义，应当按照运算符的优先级进行定义，优先级高的先定义，优先级低的后定义，应尽可能避免二义性的出现。

2.5.1 数值

数值由非终结符 Number 表示。在 SysY 语言中，数值应为 int 类型，int 类型整型数用终结符 IntConst 表示。

因此可得具体定义如下：

表达式 $Number \longrightarrow IntConst$

2.5.2 左值

左值由非终结符 LVal 表示，作为左值的变量应当有唯一一个确定的地址，从而可以将赋值表达式的右值赋给这个地址上的变量。

在 SysY 语言中，左值可以为 int 类型变量或者 int 类型多维数组。而变量实质上是标识符，由终结符 Ident 表示。

由于 int 类型多维数组的维数无法确定，故应用递归定义的方法，因此用非终结符 ArrayLVal 表示多维数组类型左值。

具体定义如下：

$$\textbf{左值}\ LVal \longrightarrow Ident \mid ArrayLVal$$
$$\textbf{多维数组类型左值}\ ArrayLVal \longrightarrow ArrayLVal\ [\ Number\ ] \mid Ident\ [\ Number\ ]$$

2.5.3 基本表达式

基本表达式由非终结符 PrimaryExp 表示，实质上是每个表达式中最先计算的部分，也即其运算级最高。在 SysY 语言中，基本表达式可以有三种，一是用 () 括起来的表达式 Exp，二是左值 LVal，三是数值 Number。

因此可得具体定义如下：

$$\textbf{基本表达式}\ PrimaryExp \longrightarrow (\ Exp\ ) \mid LVal \mid Number$$

2.5.4 一元表达式

一元表达式由非终结符 UnaryExp 表示，可以理解为经过一步运算得到的表达式，其优先级仅次于基本表达式。

在 SysY 语言中，一元表达式可以有三种，一是基本表达式 PrimaryExp 自身，二是函数返回值，三是经过单目运算得到的表达式。

在函数返回值中，函数的实参应是表达式 Exp 类型。由于函数的实参的数量不定，因此应用递归定义的方法，故用非终结符 FuncRParams 表示函数实参表，具体定义如下：

$$\textbf{函数实参表}\ FuncRParams \longrightarrow Exp\ ,\ FuncRParams \mid Exp$$

在单目运算中，我们定义了!（逻辑非）、-（单目减）和 ~（按位取反）三种单目运算符，三者之间的优先级为 - > ! > ~，故应依次定义以上三个单目运算。

用非终结符 MinusExp 表示单目减表达式，用非终结符 NotExp 表示逻辑非表达式，用非终结符 NegExp 表示按位取反表达式。

$$\textbf{单目减表达式}\ MinusExp \longrightarrow -\ UnaryExp$$
$$\textbf{逻辑非表达式}\ NotExp \longrightarrow !\ MinusExp \mid MinusExp$$
$$\textbf{按位取反表达式}\ NegExp \longrightarrow \sim NotExp \mid NotExp$$

因此可得一元表达式的具体定义如下：

$$\textbf{一元表达式}\ UnaryExp \longrightarrow PrimaryExp \mid Ident\ (\ FuncRParams\ ) \mid NegExp$$

2.5.5 乘除模表达式

乘除模表达式由非终结符 MulExp 表示，计算 *、/和% 表达式的值。在 SysY 语言中，计算完一元表达式的值之后就应当计算乘除模表达式，也即其运算顺序仅次于一元表达式。

由乘除模运算均是左结合，可得具体定义如下：

$$\begin{aligned}\textbf{乘除模表达式}\ MulExp \longrightarrow\ & UnaryExp \\ & \mid MulExp\ *\ UnaryExp \\ & \mid MulExp\ /\ UnaryExp \\ & \mid MulExp\ \%\ UnaryExp\end{aligned}$$

2.5.6 加减表达式

加减表达式由非终结符 AddExp 表示，计算 + 和-表达式的值。SysY 语言中，由于加减运算的优先级仅次于乘除模运算，因此计算完乘除模表达式的值之后就应当计算加减表达式，也即其运算顺序仅次于乘除模表达式。

由加减运算均是左结合，可得具体定义如下：

$$\textbf{加减表达式}\ AddExp \longrightarrow MulExp \mid AddExp\ +\ MulExp \mid AddExp\ -\ MulExp$$

2.5.7 关系表达式

关系表达式由非终结符 RelExp 表示，计算 <、>、<= 和 >= 表达式的值。SysY 语言中，由于 <、>、<= 和 >= 运算符的优先级仅次于加减运算，因此计算完加减表达式的值之后就应当计算关系表达式，也即其运算顺序低于加减表达式。

由 <、>、<= 和 >= 运算符是左结合，可得具体定义如下：

$$\begin{aligned}\textbf{关系表达式}\ RelExp \longrightarrow\ & AddExp \\ & \mid RelExp\ <\ AddExp \\ & \mid RelExp\ >\ AddExp \\ & \mid RelExp\ >=\ AddExp \\ & \mid RelExp\ <=\ AddExp\end{aligned}$$

2.5.8 相等性表达式

相等性表达式由非终结符 EqExp 表示，计算 == 和!= 表达式的值。SysY 语言中，由于 == 和!= 运算的优先级低于 <、>、<= 和 >= 运算，因此计算完关系表达式的值之后就应当计算相等性表达式，也即其运算顺序低于关系表达式。

由 == 和!= 均是左结合，可得具体定义如下：

$$\textbf{相等性表达式}\ EqExp \longrightarrow RelExp \mid EqExp\ ==\ RelExp \mid EqExp\ !=\ RelExp$$

2.5.9 按位与或表达式

按位与或表达式由非终结符 BitExp 表示，计算按位与 & 和按位或 | 表达式的值。SysY 语言中，由于按位与或运算的优先级仅低于 == 和!= 运算，因此计算完相等性表达式的值之后就应当计算按位与或表达式，也即其运算顺序低于相等性表达式。

由按位与或运算均是左结合，可得具体定义如下：

$$\textbf{按位与或表达式}\ BitExp \longrightarrow EqExp \mid BitExp\ '|'\ EqExp \mid BitExp\ \&\ EqExp$$

由于按位或的运算符 | 和用于合并产生式左部相同的候选式的符号或 | 相同，故使用'|' 表示按位或的运算符 |。

2.5.10 逻辑与表达式

逻辑与表达式由非终结符 LAndExp 表示，计算逻辑与 && 表达式的值。SysY 语言中，由于逻辑与 && 运算的优先级仅低于按位与或运算，因此计算完按位与或表达式的值之后才应当计算逻辑与表达式，也即其运算顺序低于按位与或表达式。

由逻辑与是左结合，可得具体定义如下：

$$\textbf{逻辑与表达式}\ LAndExp \longrightarrow BitExp \mid LAndExp\ \&\&\ BitExp$$

2.5.11 逻辑或表达式

逻辑或表达式由非终结符 LOrExp 表示，计算逻辑或 || 表达式的值。SysY 语言中，由于逻辑或 || 运算的优先级仅低于逻辑与 && 运算，因此计算完逻辑与表达式的值之后才应当计算逻辑或表达式，也即其运算顺序低于逻辑与表达式。

由逻辑或是左结合，可得具体定义如下：

$$\textbf{逻辑或表达式}\ LOrExp \longrightarrow LAndExp \mid LOrExp\ ||\ LAndExp$$

2.5.12 条件表达式

条件表达式由非终结符 Cond 表示，计算逻辑或 ||、逻辑与 && 和逻辑非! 表达式的值，因此用逻辑或表达式来定义条件表达式。

因此可得具体定义如下：

条件表达式 $Cond \longrightarrow LOrExp$

2.5.13 表达式

表达式在 SysY 中代表 int 型表达式，由非终结符 Exp 表示。由于 int 类型的限定，表达式 Exp 不能为逻辑表达式和关系表达式，只能为算术表达式。且又由于在算术表达式中按位与或运算符的优先级最低，故表达式 Exp 应定义为按位与或表达式 BitExp，具体定义如下：

表达式 $Exp \longrightarrow BitExp$

2.6 SysY 程序示例

使用 SysY 中的语言特性，我们编写了几个 SysY 程序来熟悉特性，并且针对该程序编写其对应的汇编程序。

2.7 第一个 SysY 程序示例

fibo.sy

```
int a=1;//global variable
int b=1;
const int Top=1000000;
int main()
{
int c=a+b;
putint(a);
putint(b);
while(c<Top)
{
putint(c);
a=b;
b=c;
c=a+b;
}
return 0;
}
```

这里使用全局变量与常量，编写了一个简单的循环，并调用 SysY 库函数 putin() 来显示输出。

2.8 第二个 SysY 程序示例

在此使用局部变量、全局变量和常量等，编写 factorial.sy 以实现阶乘功能，调用 SysY 库函数 putint()、putch() 和 getint() 来完成输入输出。factorial.sy 中涉及算术运算、关系运算、while 循环、复合语句、赋值语句、变量声明、if 分支语句等语言特性。

factorial.sy

```
int i = 2;//全局变量
int f = 1;

int main()
{
        int n;
        getint(n);//输入整数n
        if(n<i)//若n<i则输出提示信息"404!"，并返回
        {
                putint(404);
                putch(33);
                return 0;
        }

        while (i <= n)//计算阶乘
        {
                f = f * i;
                i = i + 1;
        }
        putint(f);//输出结果

        return 0;
}
```

3 arm 汇编编程

3.1 第一个汇编程序

fibo.s

```
.section .data
.global a
.type a,%object
.size a,4
a:
.word 1

```

```
.global b
.type b,%object
.size b,4
b:
.word 1

.section .rodata
.global Top
.type Top,%object
.size Top,4
Top:
.word 1000000

str1:
.ascii "%d %d \0"
str2:
.ascii "%d \0"

.section .text

.global fibonacci
.type fibonacci,%function
fibonacci:
push {lr}

ldr r2,_bridge@a,get memory address of a
ldr r2,[r2]
ldr r1,_bridge+4@b
ldr r1,[r1]
ldr r0,_bridge+8@str1
push {r1,r2}
bl printf @printf("%d %d ",b,a)
pop {r1,r2}
ldr r4,_bridge+16
ldr r4,[r4]
.l1:
add r3,r1,r2@c=a+b
mov r2,r1@b=a
mov r1,r3@a=c
ldr r0,_bridge+12@str2
push {r0,r1,r2,r3}
bl printf @printf("%d ",b+a)
pop {r0,r1,r2,r3}
cmp r3,r4
blt .l1
```

```
53
54
55 pop {pc}
56
57 .global main
58 .type main,%function
59 main:
60 push {lr}
61 bl fibonacci
62 pop {pc}
63
64
65 _bridge:
66 .word a
67 .word b
68 .word str1
69 .word str2
70 .word Top
```

对应第一个 SysY 运行程序，这里也实现一个 fibonacci 数列的输出。使用了 arm 汇编中的全局变量与常量，局部变量使用寄存器，没有用到栈，由于目前还不是很清楚 sysY 库 IO 函数的使用，所以这里还是使用了 c 语言中的 printf 函数来展示输出。

使用 arm-linux-gnueabihf-gcc fibo.s -o fibo 生成可执行文件，qemu-arm -L /usr/arm-linux-gnueabihf/ ./fibo 运行可执行文件。输出结果如下图

```
shen@DESKTOP-9515K9S:~/hw2$ arm-linux-gnueabihf-gcc fibo.s -o fibo
shen@DESKTOP-9515K9S:~/hw2$ qemu-arm -L /usr/arm-linux-gnueabihf/ ./fibo
1 1 2 3 5 8 13 21 34 55 89 144 233 377 610 987 1597 2584 4181 6765 10946 17711 28657 46368 75025 121393 196418 317811 514229
shen@DESKTOP-9515K9S:~/hw2$
```

3.2 第二个汇编程序

factorial.s

```
1 .section .data
2 .global a
3 .type a,%object
4 .size a,4
5 a:
6 .word 1
7
8 .global b
9 .type b,%object
10 .size b,4
11 b:
```

```
.word 1

.section .rodata
.global Top
.type Top,%object
.size Top,4
Top:
.word 1000000

str1:
.ascii "%d %d \0"
str2:
.ascii "%d \0"

.section .text

.global fibonacci
.type fibonacci,%function
fibonacci:
push {lr}

ldr r2,_bridge@a,get memory address of a
ldr r2,[r2]
ldr r1,_bridge+4@b
ldr r1,[r1]
ldr r0,_bridge+8@str1
push {r1,r2}
bl printf @printf("%d %d ",b,a)
pop {r1,r2}
ldr r4,_bridge+16
ldr r4,[r4]
.l1:
add r3,r1,r2@c=a+b
mov r2,r1@b=a
mov r1,r3@a=c
ldr r0,_bridge+12@str2
push {r0,r1,r2,r3}
bl printf @printf("%d ",b+a)
pop {r0,r1,r2,r3}
cmp r3,r4
blt .l1


pop {pc}

```

```
57 .global main
58 .type main,%function
59 main:
60 push {lr}
61 bl fibonacci
62 pop {pc}
63
64
65 _bridge:
66 .word a
67 .word b
68 .word str1
69 .word str2
70 .word Top
```

4 总结

4.1 内容总结

我们使用了上下文无关文法来定义我们想要实现的 SysY 语言特性，为了熟悉编译器的输出（目标语言：汇编语言）进行了 arm 编程的练习。

4.2 分工总结

在分工上，沈梦菊与王婧嵘一起讨论设计了上下文无关文法，分部分撰写了文法的文档部分；都进行了 arm 汇编编程的练习。

参考文献

[1] SysY 语言定义

[2] arm 汇编文档:https://azeria-labs.com/arm-data-types-and-registers-part-2/