



南开大学
Nankai University

南 开 大 学

计 算 机 学 院

编译原理实验报告

定义编译器 & 汇编编程

付寅聪

朱梓豪

年级：2020 级

专业：计算机科学与技术

指导教师：王刚

2022 年 10 月 15 日

摘要

本次实验小组共同完成。主要工作为对 SysY 语言特性进行了分析以及形式化定义，并依据上下文无关文法进行设计，且进行了 arm 汇编编程。

关键字：形式化定义，汇编，SysY 语言

目录

一、 SysY 语言特性以及形式化定义	1
(一) 语言特性	1
1. 关键字	1
2. 变量	1
3. 常量	1
4. 运算符与表达式	1
5. 语句	1
(二) 形式化定义	1
1. 变量声明	2
2. 常量声明	2
3. 表达式	2
4. 赋值表达式	2
5. 逻辑表达式	2
6. 算数表达式	3
7. 关系表达式	3
8. 语句	3
9. 循环语句	3
10. 分支语句	3
二、 汇编编程	3
(一) 斐波那契数列	3
(二) 平方	6
三、 分工	8

一、SysY 语言特性以及形式化定义

(一) 语言特性

Sysy 语言是 C 语言的一个子集，继承了 C 语言的语法定义和特性。由函数、常变量声明、语句、表达式等多种元素构成。我们将对该语言的各个特征进行分析。

1. 关键字

本实验选取 c 语言关键字的子集，构造 SysY 语言。其中每一个关键字都会在上下文无关文法中被看作是一个终结符，也就是语法树的叶节点。

类型	关键字
数据类型相关	int
语句相关	while, if, else, return
常量相关	const

2. 变量

程序运行期间，会产生一些临时数据，并保存在一些内存单元中，每个内存单元都用一个标识符来标识。这些内存单元我们称之为变量，定义的标识符就是变量名。内存单元中存储的数据就是变量的值。变量可以作左值，常量则只能作为右值。这里我们定义 SysY 支持整型变量。

3. 常量

不可变的值称之为常量。在声明时必须初始化且在程序中不可以改变其值。我们定义的 SysY 语言将支持整型常量。

4. 运算符与表达式

表达式由运算分量和运算符组成。运算分量是运算符操作的对象，运算符指明表达式的类型；表达式的运算结果是表达式的值。出现在赋值运算符左边的分量为左值，代表着一个可以存放数据的存储空间；左值只能是变量，不能是常量或表达式。出现在赋值运算符右边的分量为右值。

我们定义的 SysY 语言将支持算术运算（+、-、*、/、%，其中 +、- 都可以是单目运算符）、关系运算（==，>，<，>=，<=，!=）和逻辑运算（&&（与）、||（或）、！（非））。

5. 语句

我们定义的 SysY 语言将支持表达式语句、分支语句、循环语句。表达式语句为表达式后加上“;”。分支语句由 if-else 组成。循环语句由 while 实现。

(二) 形式化定义

使用 CFG 即上下文无关文法对 SysY 语言进行形式化定义。下面是各符号的含义。

名称	符号	名称	符号
声明语句	decl	标识符	id
标识符列表	idlist	数据类型	type
表达式	expr	一元表达式	unary_expr
赋值表达式	assign_expr	逻辑表达式	logical_expr
算数表达式	math_expr	关系表达式	relation_expr
数字	digit	整数	decimal
常量定义	const_init	语句	stmt
分支语句	selection_stmt	循环语句	loop_stmt

1. 变量声明

变量声明有两种，一种是仅声明变量，另一种是声明且赋值。

$$\begin{aligned} \text{idlist} &\rightarrow \text{idlist, id} \mid \text{id} \\ \text{type} &\rightarrow \text{int} \\ \text{decl} &\rightarrow \text{type idlist} \mid \text{type id} = \text{logical_expr} \mid \text{type id} = \text{unary_expr} \end{aligned}$$

2. 常量声明

常量声明仅有整数常量。

$$\text{const_init} \rightarrow \text{const type id} = \mid \text{unary_expr}$$

3. 表达式

表达式有一元表达式、赋值表达式、逻辑表达式、关系表达式、算数表达式。

$$\text{expr} \rightarrow \text{unary_expr} \mid \text{assign_expr} \mid \text{logical_expr} \mid \text{math_expr} \mid \text{relation_expr}$$

4. 赋值表达式

$$\begin{aligned} \text{digit} &\rightarrow 0 \mid 1 \mid 2 \mid 3 \mid 4 \mid 5 \mid 6 \mid 7 \mid 8 \mid 9 \\ \text{decimal} &\rightarrow \text{digit} \mid \text{decimal digit} \\ \text{unary_expr} &\rightarrow \text{decimal} \mid \text{id} \\ \text{assign_expr} &\rightarrow \text{id} = \text{unary_expr} \mid \text{id} = \text{logical_expr} \end{aligned}$$

5. 逻辑表达式

$$\begin{aligned} \text{logical_expr} &\rightarrow \text{unary_expr} \mid \\ &\quad \text{!(logical_expr)} \mid \\ &\quad \text{logical_expr} \mid \mid \text{logical_expr} \mid \\ &\quad \text{logical_expr} \&\& \text{logical_expr} \end{aligned}$$

6. 算术表达式

$$\begin{aligned} \text{math_expr} \rightarrow & \text{unary_expr} \mid -\text{unary_expr} \mid \\ & \text{math_expr} + \text{math_expr} \mid \\ & \text{math_expr} - \text{math_expr} \mid \\ & \text{math_expr} * \text{math_expr} \mid \\ & \text{math_expr} / \text{math_expr} \mid \\ & \text{math_expr} \% \text{math_expr} \mid \end{aligned}$$

7. 关系表达式

$$\begin{aligned} \text{unary_expr} \rightarrow & \text{unary_expr} == \text{unary_expr} \mid \\ & \text{unary_expr} != \text{unary_expr} \mid \\ & \text{unary_expr} > \text{unary_expr} \mid \\ & \text{unary_expr} < \text{unary_expr} \mid \\ & \text{unary_expr} >= \text{unary_expr} \mid \\ & \text{unary_expr} <= \text{unary_expr} \mid \end{aligned}$$

8. 语句

循环语句、分支语句等组成语句。

$$\begin{aligned} \text{stmt} \rightarrow & \text{loop_stmt} \mid \text{selection_stmt} \mid \\ & \text{expr}; \text{stmt} \mid \\ & \text{expr}; \mid \end{aligned}$$

9. 循环语句

$$\text{loop_stmt} \rightarrow \text{while}(\text{expr})\{\text{stmt}\} \mid$$

10. 分支语句

$$\begin{aligned} \text{selection_stmt} \rightarrow & \text{if}(\text{expr})\{\text{stmt}\} \mid \\ & \text{if}(\text{expr})\{\text{stmt}\}\text{else}\{\text{stmt}\} \end{aligned}$$

二、 汇编编程

(一) 斐波那契数列

源程序

```
1 #include <iostream>
2 using namespace std;
```

```

3  int main()
4  {
5      int a, b, i, t, n;
6      a = 0;
7      b = 1;
8      i = 1;
9      cin >> n;
10     cout << "a:" << a << endl;
11     cout << "b:" << b << endl;
12     cout << "we are going to loop now! " << endl;
13     while (i < n)
14     {
15         t = b;
16         b = a + b;
17         cout << b << endl;
18         a = t;
19         i = i + 1;
20     }
21     return 0;
22 }

```

手写汇编代码

```

1  @处理器寄存器被指定为R0、R1等。
2  @MOVE指令的源位于左侧，目标位于右侧。
3  @伪处理程序中的堆栈从高地址增长到低地址。因此，push会导致堆栈指针的递减。pop
   会导致堆栈指针的增量。
4  @寄存器 sp(stack pointer) 用于指向堆栈。
5  @寄存器 fp(frame pointer) 用作帧指针。帧指针充当被调用函数和调用函数之间的
   锚。
6  @当调用一个函数时，该函数首先将 fp 的当前值保存在堆栈上。然后，它将 sp 寄存器
   的值保存在 fp 寄存器中。然后递减 sp 寄存器来为本地变量分配空间。
7  @fp 寄存器用于访问本地变量和参数，局部变量位于帧指针的负偏移量处，传递给函数
   的参数位于帧指针的正偏移量。
8  @当函数返回时，fp 寄存器被复制到 sp 寄存器中，这将释放用于局部变量的堆栈，函
   数调用者的 fp 寄存器的值由pop从堆栈中恢复。
9  @因为在调用者中使用了 bl 调用子函数的时候，会将当前 PC 的值保存在 LR 中，这时
   将 LR 中的值载入到 PC 中，可以使得程序运行位置返回调用者中
10     .arch armv7-a @处理器架构
11     .arm
12     @r0是格式化字符串，r1是对应的printf对应的第二个参数
13     @代码段
14     @主函数
15         .text @代码段
16         .global main
17         .type main, %function
18     main:
19         push {fp, lr} @将fp的当前值保存在堆栈上，然后将sp寄存器的值保存在fp
           中，lr中存储的是pc的保存在lr中

```

```
20      sub sp, sp, #4 @在栈中开辟一块大小为4的内存地址，用于存储即将输入的数据
21      ldr r0, =_cin
22      mov r1, sp @将sp的值传输给r1寄存器，使scanf传入的值存储在栈上，即栈顶的值是n
23      bl scanf
24      ldr r6, [sp, #0] @取出sp指针指向的地址中的内容，即栈顶中的内容（输入的n的值）
25      add sp, sp, #4 @恢复栈顶，释放内存空间
26
27      @测试是否写入
28      @ldr r0, =_bridge3
29      @mov r1, r2
30      @bl printf
31
32      mov r4, #0 @a = 0
33      mov r5, #1 @b = 1
34      mov r7, #1 @i = 1
35      @r4中存a的值，r5中存b的值，r7中存i的值，r6中存n的值
36      ldr r0, =_bridge
37      mov r1, r4 @将r4中的值即a的值赋予r1
38      bl printf @打印a的值
39      ldr r0, =_bridge2
40      mov r1, r5 @将r5中的值即b的值赋予r1
41      bl printf @打印b的值
42      ldr r0, =_bridge4
43      bl printf
44
45      @输出进行调试
46      @ldr r0, =_bridge3
47      @mov r1, r6
48      @bl printf
49      @ldr r0, =_bridge3
50      @mov r1, r7
51      @bl printf
52
53      Loop:
54      @输出进行调试
55      @ldr r0, =_bridge4
56      @bl printf
57      @ldr r0, =_bridge3
58      @mov r1, r6
59      @bl printf
60      @ldr r0, =_bridge3
61      @mov r1, r7
62      @bl printf
63
64      cmp r6, r7
```

```

65     ble RETURN @比较r7和r6（即i和n）的大小用于跳转
66     mov r8, r5 @t = b @r8为临时变量的寄存器
67     add r5, r5, r4 @b = a + b
68     ldr r0, =_bridge3
69     mov r1, r5 @将r5中的值即b的值赋予r1
70     bl printf @cout << b << endl;
71     mov r4, r8 @a = t
72     add r7, r7, #1 @i = i + 1
73     b Loop
74 RETURN:
75     pop {fp, lr} @上下文切换
76     bx lr @return 0
77 .data @数据段
78 _cin:
79     .asciz "%d"
80
81 _bridge:
82     .asciz "a:%d\n"
83
84 _bridge2:
85     .asciz "b:%d\n"
86
87 _bridge3:
88     .asciz "%d\n"
89
90 _bridge4:
91     .asciz "We are going to loop now! \n"
92
93 .section .note.GNU-stack,"",%progbits @ do you know what's the use of this
    :-)

```

```

root@LAPTOP-QVUA3NIU:~# qemu-arm ./Fibonacci
8
a:0
b:1
We are going to loop now!
1
2
3
5
8
13
21

```

图 1: 斐波那契数列运行结果

(二) 平方

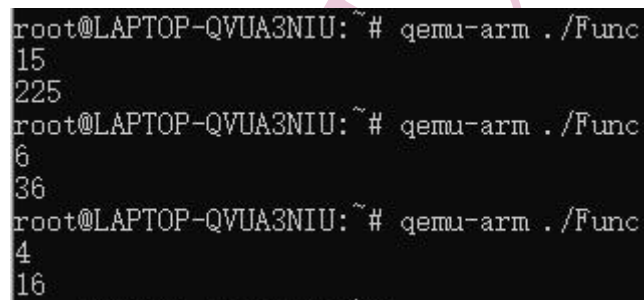
源程序


```
1 #include <iostream>
2 using namespace std;
3 int square(int a){
4     int m;
5     m = a * a;
6     return m;
7 }
8 int main()
9 {
10     int a, s_a;
11     cin >> a;
12     s_a = square(a);
13     cout << s_a << endl;
14     return 0;
15 }
```

手写汇编代码

```
1 .arch armv7-a
2 .arm
3
4 .text @代码段
5 .global square
6 square: @function int square(int a)
7     str fp, [sp, #-4]! @pre-index mode, sp = sp -4, push fp
8     mov fp, sp
9     sub sp, sp, #8 @为本地变量开辟空间
10    str r0, [fp, #-8] @r0 = [fp, #-8] = a
11    mul r1, r0, r0
12    mov r0, r1
13    add sp, fp, #0
14    ldr fp, [sp], #4
15    bx lr
16
17 .text @代码段
18 .global main
19 .type main, %function
20 main:
21     push {fp, lr}
22     sub sp, sp, #4
23     ldr r0, =_cin
24     mov r1, sp
25     bl scanf
26     ldr r0, [sp, #0] @取出输入的内容放入r0中
27     add sp, sp, #4
28     bl square
29     mov r1, r0
30     ldr r0, =_cout
```

```
31     bl printf
32     mov r0, #0
33     pop {fp, lr}
34     bx lr
35
36
37
38 .data @数据段
39 _cin:
40     .asciz "%d"
41
42 _cout:
43     .asciz "%d\n"
44
45
46 .section .note.GNU-stack,"",%progbits @ do you know what's the use of this
47     :-)
Footer
```



```
root@LAPTOP-QVUA3NIU:~# qemu-arm ./Func
15
225
root@LAPTOP-QVUA3NIU:~# qemu-arm ./Func
6
36
root@LAPTOP-QVUA3NIU:~# qemu-arm ./Func
4
16
```

图 2: 平方运行结果

三、 分工

朱梓豪负责了形式化定义的变量声明、表达式、逻辑表达式、算数表达式、循环语句以及斐波那契函数的汇编。

付寅聪负责了形式化定义的常量声明、赋值表达式、关系表达式、语句、分支语句以及平方运算的汇编。