南開大學
Nankai University

南　开　大　学

计 算 机 学 院

编译原理实验报告

预备工作 2：定义你的编译器、汇编编程 & 熟悉辅助工具（小组合作）

组名：NK_HL

成员：林语盈、贺祎昕

年级：2020 级

专业：计算机科学与技术

指导教师：王刚

2022 年 10 月 11 日

摘要

首先，对 SysY 语言进行了上下文无关文法的描述。然后，针对其语言特性，对三个有代表性的程序进行 ARM 汇编语言的翻译。最后，对工作进行总结。

关键字：ARM 汇编；上下文无关文法

目录

一、 支持的 SysY 语言特性描述

(一) SysY 语言特性

我们描述了暂时想到的全部 SysY 语言特性，包括 int/float 类型、函数、变量/常量声明、语句、表达式等，参考了实验指导书《SysY2022 语言定义》。

(二) 上下文无关文法描述

1. 终结符集合

V_T = { Ident, IntConst, floatConst, const, int, float, void, if, else, while, break, continue, return, (,), ;, [,], +, -, !, *, /, %, <, >, <=, >=, ==, !=, ||, && }

2. 非终结符集合

V_N = {BasicUnit, CompUnit, Decl, ConstDel, ConstDefList, BType, ConstDef, ConstLVal, ConstInitVal, ConstInitValInit, VarDecl, VarDefLsit, VarDef, InitVal, InitValList, FuncDef, FuncType, FuncFParams, FuncFParam, ExpList, Block, BlockItemList, BlockItem, Stmt, Exp, Cond, LVal, PrimaryExp, Number, UnaryExp, FunvRParams, MulExp, AddExp, RelExp, EqExp, LAndExp, LOrExp, ConstExp, UnaryOp, MulOp, AddOp, RelOp, EqOp}

3. 开始符号

S = BasicUnit

4. 产生式

P = {
BasicUnit → Decl | FuncDef
CompUnit → BasicUnit | CompUnit BasicUnit
Decl → ConstDel | VarDecl
ConstDel → **const** BType ConstDefList ;
ConstDefList → ConstDef | ConstDef, ConstDefList
BType → **int** | **float**
ConstDef → ConstLVal = ConstInitVal
ConstLVal → **Ident** | ConstLVal[ConstExp]
ConstInitVal → ConstExp | |ConstInitValList
ConstInitValList → ConstInitVal | ConstInitValList, ConstInitVal
VarDecl → BType VarDefList;
VarDefList → VarDef | VarDefList, VarDef
VarDef → ConstLVal | ConstLVal = InitVal
InitVal → Exp | InitValList
InitValList → InitVal | InitValList, InitVal | ε
FuncDef → FunType **Ident**(FunctionFParams) Block
FuncType → **void** | **int** | **float**
FuncFParams → FuncFParam | FuncFParams, FuncFParam | ε
FuncFParam → BType **Ident** | BType **Ident** [] ExpList

ExpList → [Exp] | ExpList [Exp] | ε

Block → BlockItemList

BlockItemList → BlockItem | BlockItemList BlockItem| ε

BlockItem → Decl | Stmt

Stmt → LVal = Exp; | Exp; | ; | Block | **if**(Cond) Stmt | **if**(Cond) Stmt **else** Stmt | **while**(Cond) Stmt | **break**; | **continue**; | **return**; | **return** Exp;

Exp → AddExp

Cond → LOrExp

LVal → **Ident** | LVal[Exp]

PrimaryExp → (Exp) | LVal | Number

Number → **IntConst** | **floatConst**

UnaryExp → PrimaryExp | **Ident** (FuncRParams) | Ident () | UnaryOp UnaryExp

FuncRParams → Exp | FuncRParams,Exp

MulExp → UnaryExp | MulExp MulOp UnaryExp

AddExp → MulExp | AddExp AddOp MulExp

RelExp → AddExp | RelExp RelOp AddExp

EqExp → RelExp | EqExp EqOp RelExp

LAndExp → EqExp | LAndExp && EqExp

LOrExp → LAndExp | LOrExp || LAndExp

ConstExp → AddExp

UnaryOp → + | - | !

MulOp → * | / | %

AddOp → + | -

RelOp → < | > | <= | >=

EqOp → ==| !=

}

5. 对终结符的进一步定义

Ident 标识符 **Ident** → id-nondigit | Ident id-digit | Ident id-nondigit

id-nondigit = {a, b, c, d, e, f, g, h, i, j, k, l, m, n, o, p, q, r, s, t, u, v, w, x, y, z,

A, B, C, D, E, F, G, H, I, J, K, L, M, N, O, P, Q, R, S, T, U, V, W, X, Y, Z,

_}

id-digit = {0, 1, 2, 3, 4, 5, 6, 7 ,8, 9}

Ident → identifier-nondigit | Ident identifier-nondigit | Ident identifier-digit

identifier-nondigit → a | b | c | d | e | f | g | h | i | j | k | l | m | n | o | p | q | r | s | t | u | v | w | x | y | z | A | B | C | D | E | F | G | H | I | J | K | L | M | N | O | P | Q | R | S | T | U | V | W | X | Y | Z

identifier-digit → 0 | 1 | 2 | 3 | 4 | 5 | 6 | 7 | 8 | 9

IntConst 整形常量 **IntConst** → decimal-const | octal-const | hexadecimal-const

decimal-const → nonzero-digit | decimal-const digit

octal-const → 0 | octal-const octal-digit

hexadecimal-const → hexadecimal-prefix hexadecimal-digit | hexadecimal-const hexadecimal-digit

hexadecimal-prefix → 0x | 0X
nonzero-digit → 1 | 2 | 3 | 4 | 5 | 6 | 7 | 8 | 9
octal-digit → 0 | 1 | 2 | 3 | 4 | 5 | 6 | 7
hexadecimal-digit → 0 | 1 | 2 | 3 | 4 | 5 | 6 | 7 | 8 | 9 | a | b | c | d | e | f | A | B | C | D | E | F

floatConst 浮点数常量 **floatConst** → decimal-floating-constant | hexadecimal-floating-constant

decimal-floating-constant → fractional-constant exponent-partopt floating-suffixopt | digit-sequence exponent-part floating-suffixopt

hexadecimal-floating-constant → hexadecimal-prefix hexadecimal-fractional-constant | binary-exponent-part floating-suffixopt | hexadecimal-prefix hexadecimal-digit-sequence | binary-exponent-part floating-suffixopt

fractional-constant → digit-sequenceopt . digit-sequence | digit-sequence .

exponent-part → e signopt digit-sequence | E signopt digit-sequence

sign→ + | -

digit-sequence→digit|digit-sequence digit

hexadecimal-fractional-constant→ hexadecimal-digit-sequenceopt . |hexadecimal-digit-sequence |hexadecimal-digit-sequence .

binary-exponent-part→ p signopt digit-sequence |P signopt digit-sequence

hexadecimal-digit-sequence→ hexadecimal-digit |hexadecimal-digit-sequence hexadecimal-digit

floating-suffix→f | l | F | L

二、 汇编程序的编写和验证

(一) SysY 程序样例

1. 程序 1

如下所示

p1.c

```c
/*
    斐波那契数列值计算
*/
#include <stdio.h>
//斐波那契
int main() {
        int a, b, i, t, n;

        a = 0;
        b = 1;
        i = 1;
        scanf("%d", &n);
        printf("%d", a);
        printf("%d", b);

        while (i < n) {
```

```
                t = b;
                b = a + b;
                printf("%d", b);
                a = t;
                i = i + 1;
        }
}
```

p1.c

```
; Function Attrs: noinline nounwind optnone uwtable
define dso_local i32 @main() #0 {
  %1 = alloca i32, align 4
  %2 = alloca i32, align 4
  %3 = alloca i32, align 4
  %4 = alloca i32, align 4
  %5 = alloca i32, align 4
  %6 = alloca i32, align 4
  store i32 0, i32* %1, align 4
  store i32 0, i32* %2, align 4
  store i32 1, i32* %3, align 4
  store i32 1, i32* %4, align 4
  %7 = call i32 (i8*, ...) @__isoc99_scanf(i8* getelementptr inbounds ([3 x
      i8], [3 x i8]* @.str, i64 0, i64 0), i32* %6)
  %8 = load i32, i32* %2, align 4
  %9 = call i32 (i8*, ...) @printf(i8* getelementptr inbounds ([3 x i8], [3 x
       i8]* @.str, i64 0, i64 0), i32 %8)
  %10 = load i32, i32* %3, align 4
  %11 = call i32 (i8*, ...) @printf(i8* getelementptr inbounds ([3 x i8], [3
      x i8]* @.str, i64 0, i64 0), i32 %10)
  br label %12

12:                                               ; preds = %16, %0
  %13 = load i32, i32* %4, align 4
  %14 = load i32, i32* %6, align 4
  %15 = icmp slt i32 %13, %14
  br i1 %15, label %16, label %26

16:                                               ; preds = %12
  %17 = load i32, i32* %3, align 4
  store i32 %17, i32* %5, align 4
  %18 = load i32, i32* %2, align 4
  %19 = load i32, i32* %3, align 4
  %20 = add nsw i32 %18, %19
  store i32 %20, i32* %3, align 4
  %21 = load i32, i32* %3, align 4
  %22 = call i32 (i8*, ...) @printf(i8* getelementptr inbounds ([3 x i8], [3
      x i8]* @.str, i64 0, i64 0), i32 %21)
  %23 = load i32, i32* %5, align 4
```

```
  store i32 %23, i32* %2, align 4
  %24 = load i32, i32* %4, align 4
  %25 = add nsw i32 %24, 1
  store i32 %25, i32* %4, align 4
  br label %12

26:                                               ; preds = %12
  %27 = load i32, i32* %1, align 4
  ret i32 %27
}
```

2. 程序 2

如下所示

p2.c

```
/*
包含的SysY语言特性:
    float 数据类型
    变量声明, 赋值语句
    for循环
    if语句
    算术运算（加减乘除、按位与或等）、逻辑运算（逻辑与或等）、关系运算（不等、等于、大于、小于等）
    数组和指针
*/
#include <stdio.h>
#include <stdlib.h>
int main() {
        //数组 指针
        float a[10];
        float* p = (float*)malloc(10 * sizeof(float));
        for (int i = 0; i < 10; i++) {
                a[i] = i;
                p[i] = 10-i;
                if (a[i] == p[i]) {
                        a[i] = -1;
                        p[i] = -1;
                }
        }
}
```

p2.ll

```
; 首部参考其自动生成的.ll文件
; ModuleID = 'try.c'
source_filename = "try.c"
target datalayout = "e-m:e-p270:32:32-p271:32:32-p272:64:64-i64:64-f80:128-n8:16:32:64-S128"
```

```
target triple = "x86_64-pc-linux-gnu"

; Function Attrs: noinline nounwind optnone uwtable

define dso_local i32 @main() #0 {
  ; 数组声明 16字节对齐
  ; float a[10];
  %1 = alloca [10 x float], align 16

  ; 指针声明 8字节对齐
  ; float* p = (float*)malloc(10 * sizeof(float));
  %2 = alloca float*, align 8

  ; 调用全局函数@malloc动态分配空间
  %3 = call noalias i8* @malloc(i64 40) #2
  ; The 'bitcast' instruction converts value to type ty2 without changing
     any bits.
  %4 = bitcast i8* %3 to float*
  store float* %4, float** %2, align 8

  ; for循环中的 int i=0; 4字节对齐
  %5 = alloca i32, align 4
  ; i初始化为0
  store i32 0, i32* %5, align 4
  %6 = alloca i32, align 4
  br label %7

7:                                                ; preds = %42, %0
  %8 = load i32, i32* %5, align 4
  ; 利用slt 判断循环条件i<10是否符合
  %9 = icmp slt i32 %8, 10
  ; 符合则跳转到label10  否则跳转至label45结束循环
  br i1 %9, label %10, label %45

10:                                               ; preds = %7
  ; (循环条件符合) 开始循环体部分
  %11 = load i32, i32* %5, align 4
  ; sitofp 将有符号整型i转换为浮点数
  %12 = sitofp i32 %11 to float
  %13 = load i32, i32* %5, align 4
  %14 = sext i32 %13 to i64
  %15 = getelementptr inbounds [10 x float], [10 x float]* %1, i64 0, i64 %14
  ; a[i]=i
  store float %12, float* %15, align 4
  %16 = load i32, i32* %5, align 4
  %17 = sub nsw i32 10, %16
  %18 = sitofp i32 %17 to float
  %19 = load float*, float** %2, align 8
```

```
  %20 = load i32, i32* %5, align 4
  %21 = sext i32 %20 to i64
  %22 = getelementptr inbounds float, float* %19, i64 %21
  ; p[i]=i
  store float %18, float* %22, align 4

  %23 = load i32, i32* %5, align 4
  %24 = sext i32 %23 to i64
  %25 = getelementptr inbounds [10 x float], [10 x float]* %1, i64 0, i64 %24
  ; load a[i]到%26
  %26 = load float, float* %25, align 4
  %27 = load float*, float** %2, align 8
  %28 = load i32, i32* %5, align 4
  %29 = sext i32 %28 to i64
  %30 = getelementptr inbounds float, float* %27, i64 %29
  ; load p[i]到%26
  %31 = load float, float* %30, align 4
  ; if条件判断 a[i]==p[i]
  %32 = fcmp oeq float %26, %31
  ; 条件成立, 跳转至label%33 否则跳转至label%41
  br i1 %32, label %33, label %41

33:                                               ; preds = %10
  ; a[i]==p[i]条件成立
  %34 = load i32, i32* %5, align 4
  %35 = sext i32 %34 to i64
  %36 = getelementptr inbounds [10 x float], [10 x float]* %1, i64 0, i64 %35
  ; a[i]=-1
  store float -1.0, float* %36, align 4
  %37 = load float*, float** %2, align 8
  %38 = load i32, i32* %5, align 4
  %39 = sext i32 %38 to i64
  %40 = getelementptr inbounds float, float* %37, i64 %39
  ; p[i]=-1
  store float -1.0, float* %40, align 4
  br label %41

41:                                               ; preds = %33, %10
  ; 分支条件不符合
  br label %42

42:                                               ; preds = %41
  ; 取i
  %43 = load i32, i32* %5, align 4
  ; i自增 i++
  %44 = add nsw i32 %43, 1
  store i32 %44, i32* %5, align 4
  br label %7
```

```
100
101 45:                                               ; preds = %7
102   ; (循环条件不符合) 退出循环
103   ret i32 0
104 }
105
106 ; 尾部参考其自动生成的.ll文件
107 ; Function Attrs: nounwind
108 declare dso_local noalias i8* @malloc(i64) #1
109
110 !llvm.module.flags = !{!0}
111 !llvm.ident = !{!1}
112
113 !0 = !{i32 1, !"wchar_size", i32 4}
114 !1 = !{!"clang version 10.0.0-4ubuntu1 "}
```

3. 程序 3

如下所示

p3.c

```c
/*
sysY语言特性:
    int 数据类型
    变量声明, 赋值语句
    while循环
    算术运算（加减乘除、按位与或等）、逻辑运算（逻辑与或等）、关系运算（不等、等于、大于、小于等）
    函数
*/
#include<stdio.h>
int function(int a,int b){
    return a*b;
}
int main()
{
        int i, n, f;
        scanf("%d",&n);
        i = 2;
        f = 1;
        while (i <= n)
        {
                f = function(f,i);
                i = i + 1;
        }
        printf("%d\n",f);
    return 0;
}
```

p3.ll

```
; 首部参考其自动生成的.ll文件
...
; function函数 输入为整数（i32）
define dso_local i32 @function(i32 %0, i32 %1) #0 {
  %3 = mul nsw i32 %0, %1 ;%3 = %0 * %1
  ret i32 %3 ;return %3
}

;注意: 一切变量名和lable必须依次顺序编号!
; main函数
define dso_local i32 @main() #0 {
  %1 = alloca i32, align 4  ;声明变量i
  %2 = alloca i32, align 4  ;声明变量n
  %3 = alloca i32, align 4  ;声明变量f
  %4 = call i32 (i8*, ...) @__isoc99_scanf(i8* getelementptr inbounds ([3 x
      i8], [3 x i8]* @.str, i64 0, i64 0), i32* %2);调用函数scanf("%d",&n);
  store i32 2, i32* %1, align 4 ;i = 2;
  store i32 1, i32* %3, align 4 ;f = 1;
  br label %5

5:                                                ;while循环退出条件
  %6 = load i32, i32* %1, align 4 ;读i
  %7 = load i32, i32* %2, align 4 ;读n
  %8 = load i32, i32* %3, align 4 ;读f
  %9 = icmp sle i32 %6, %7 ; %9为 i <= n 的比较结果
  br i1 %9, label %10, label %13 ;如果为%9真, 那么执行 label%10, 否则执行
      label%16

10:                                               ;while循环体
  %11 = call i32 @function(i32 %6, i32 %8) ;调用function函数
  store i32 %11, i32* %3, align 4 ;将函数的返回值存入f
  %12 = add nsw i32 %6, 1
  store i32 %12, i32* %1, align 4 ;i = i + 1
  br label %5 ;跳转到退出条件判断

13:
  %14 = call i32 (i8*, ...) @printf(i8* getelementptr inbounds ([4 x i8], [4
      x i8]* @.str.1, i64 0, i64 0), i32 %8);输出f
  ret i32 0 ;main函数返回0
}

;函数声明
declare dso_local i32 @__isoc99_scanf(i8*, ...) #1

declare dso_local i32 @printf(i8*, ...) #1

; 尾部参考了其自动生成的.ll文件
```

```
45 ...
```

(二) 编写等价汇编程序

1. p1.S

如下所示

p1.ll

```
1         .global main
2 main:
3         push    {r7}
4         sub     sp, sp, #20
5         mov     r7, sp
6         movs    r3, #0
7         str     r3, [r7, #4]
8         movs    r3, #1
9         str     r3, [r7, #8]
10        movs    r3, #1
11        str     r3, [r7, #12]
12 .while:
13        ldr     r2, [r7, #12]
14        ldr     r3, [r7, #16]
15        cmp     r3, r2
16        blt     .exit_while
17        ldr     r3, [r7, #8]
18        str     r3, [r7, #20]
19        ldr     r2, [r7, #8]
20        ldr     r3, [r7, #4]
21        add     r3, r3, r2
22        str     r3, [r7, #8]
23        ldr     r3, [r7, #20]
24        str     r3, [r7, #4]
25        ldr     r3, [r7, #12]
26        adds    r3, r3, #1
27        str     r3, [r7, #12]
28 .exit_while:
29        movs    r3, #0
30        mov     r0, r3
31        adds    r7, r7, #20
32        adds    sp, sp, #20
```

编译验证正确。

2. p2.S

如下所示

p2.ll

```
        .global main
main:
        @ args = 0, pretend = 0, frame = 56
        @ frame_needed = 1, uses_anonymous_args = 0
        push    {r7, lr}
        sub     sp, sp, #56
        add     r7, sp, #0
        ldr     r2, .L7
.LPIC0:
        add     r2, pc
        ldr     r3, .L7+4
        ldr     r3, [r2, r3]
        ldr     r3, [r3]
        str     r3, [r7, #52]

        mov     r3,#0
        movs    r0, #40
        bl      malloc(PLT)
        mov     r3, r0
        str     r3, [r7, #8]

        movs    r3, #0
        str     r3, [r7, #4]
        b       .for

.else:
.add_i:
        ldr     r3, [r7, #4]
        adds    r3, r3, #1
        str     r3, [r7, #4]
.for:
        ldr     r3, [r7, #4]
        cmp     r3, #10
        bge     .exit_for

        ldr     r3, [r7, #4]
        vmov    s15, r3 @ int
        vcvt.f32.s32    s15, s15
        ldr     r3, [r7, #4]
        lsls    r3, r3, #2
        add     r2, r7, #56
        add     r3, r3, r2
        subs    r3, r3, #44
        vstr.32 s15, [r3]

        ldr     r3, [r7, #4]
        rsb     r1, r3, #10
        ldr     r3, [r7, #4]
```

```
49          lsls    r3, r3, #2
50          ldr     r2, [r7, #8]
51          add     r3, r3, r2
52
53          vmov    s15, r1 @ int
54          vcvt.f32.s32    s15, s15
55          vstr.32 s15, [r3]
56
57          ldr     r3, [r7, #4]
58          lsls    r3, r3, #2
59          add     r2, r7, #56
60          add     r3, r3, r2
61          subs    r3, r3, #44
62          vldr.32 s14, [r3]
63
64          ldr     r3, [r7, #4]
65          lsls    r3, r3, #2
66          ldr     r2, [r7, #8]
67          add     r3, r3, r2
68          vldr.32 s15, [r3]
69
70          vcmp.f32        s14, s15
71          vmrs    APSR_nzcv, FPSCR
72          bne     .else
73
74  .if:
75          ldr     r3, [r7, #4]
76          lsls    r3, r3, #2
77          add     r2, r7, #56
78          add     r3, r3, r2
79          subs    r3, r3, #44
80          mov     r2, #0
81          movt    r2, 49024
82          str     r2, [r3]        @ float
83
84          ldr     r3, [r7, #4]
85          lsls    r3, r3, #2
86          ldr     r2, [r7, #8]
87          add     r3, r3, r2
88          mov     r2, #0
89          movt    r2, 49024
90          str     r2, [r3]        @ float
91
92          b .add_i
93
94  .exit_for:
95          movs    r3, #0
96          ldr     r1, .L7+8
```

```
.LPIC1:
        add     r1, pc
        ldr     r2, .L7+4
        ldr     r2, [r1, r2]
        ldr     r1, [r2]
        ldr     r2, [r7, #52]
        eors    r1, r2, r1
        mov     r2, #0
        beq     .L6
        bl      __stack_chk_fail(PLT)
.L6:
        mov     r0, r3
        adds    r7, r7, #56
        mov     sp, r7
        pop     {r7, pc}
.L7:
        .word   _GLOBAL_OFFSET_TABLE_-(.LPIC0+4)
        .word   __stack_chk_guard(GOT)
        .word   _GLOBAL_OFFSET_TABLE_-(.LPIC1+4)
        .size   main, .-main
        .ident  "GCC: (Ubuntu 9.4.0-1ubuntu1~20.04.1) 9.4.0"
        .section        .note.GNU-stack,"",%progbits
```

编译验证正确。

3. p3.S

如下所示

p3.S

```
.arch armv5t
@ comm section save global variable without initialization
.comm i, 4 @ 全局变量
.comm f, 4
.comm n, 4
.text
.align 2@ 设置对齐
@ rodata section save constant
.section .rodata

 _str0:
 .ascii "%d\0" @ 定义字符串
.align 2
 _str1:
 .ascii "%d\n"
.align 2

 .text

```

```
20
21  .global function
22 function: @ function
23     str fp, [sp, #-4]! @ pre-index mode, sp = sp - 4, push fp
24     mov fp, sp
25     sub sp, sp, #12 @ allocate space for local variable
26     str r0, [fp, #-8] @ r0 = [fp, #-8] = f
27     str r1, [fp, #-12] @ r1 = [fp, #-12] = i
28     mul r2, r0, r1 @计算,函数的参数输入位于r0、r1,结果存于r2寄存器
29     mov r0,r2
30
31     add sp, fp, #0
32     ldr fp, [sp], #4 @ post-index mode, pop fp, sp = sp + 4
33     bx lr @ recover sp fp pc
34
35 .global main
36 main:
37         @ main函数
38         push    {fp, lr} @ 非叶函数,需要额外存储lr
39     add fp, sp, #4
40
41     ldr r1, _bridge+8 @ r1 = &n
42     ldr r0, _bridge+12 @ *r0 = "%d\0"
43         bl      __isoc99_scanf @ 调用函数读入n,scanf("%d", &n)
44
45         movs    r3, #2 @ i=2
46     ldr r4,_bridge
47     str r3,[r4]
48         movs    r3, #1 @ f=1
49     ldr r4,_bridge+4
50     str r3,[r4]
51         b       .L4 @ 无条件跳转, 进入while循环判断
52 .L5: @ while循环体
53         ldr     r3, _bridge
54     ldr r0, [r3] @ r0=变量i
55         ldr     r3, _bridge+4
56     ldr r1, [r3] @ r1=变量f
57         bl      function @ 参数存入r0r1,调用函数
58     ldr r4,_bridge+4
59     str r0,[r4] @ 函数返回值存于r0,将其赋给f
60         ldr     r3, _bridge @ 将变量i赋给r3
61     ldr r2, [r3]
62         adds    r2, r2, #1 @ r2=i+1
63     ldr r4,_bridge
64     str r2,[r4] @ i=r2
65
66 .L4: @ while循环退出条件
67     ldr r3, _bridge
```

```
68      ldr r0, [r3] @ r0=变量i
69          ldr     r3, _bridge+8 @ 变量n
70          ldr     r2, [r3] @ 变量n
71          cmp     r0, r2 @ 比较i与n
72          ble     .L5 @ 若i <= n,重新进入循环体
73          @ 退出循环
74          @ r1 = f
75      ldr r3, _bridge+4
76      ldr r1, [r3] @ r1=变量f
77      ldr r0, _bridge+16 @ *r0 = "%d\n"
78          bl      printf @ printf("%d\n", f);
79      mov r0, #0
80      pop {fp, pc} @ return 0
81
82  _bridge: @ 变量桥梁
83  .word i
84  .word f
85  .word n
86  .word _str0
87  .word _str1
88
89  .section .note.GNU-stack,"",%progbits
```

编译验证正确。

综上，三个程序基本覆盖了主要的 sysY 语言特性。

三、 总结

本次作业由林语盈、贺祎昕共同合作完成。

(一) 组内成员分工

对于构造上下文无关文法的任务，终结符、非终结符、终结符定义部分由二人共同完成；产生式部分，按本文中出现的顺序，贺祎昕负责 Block 及之前的定义，林语盈负责其后的定义与全部定义的整体修改。对于汇编程序编写的任务，贺祎昕负责 p1、p2 两个程序的编写和验证，林语盈负责 p3 程序的编写和验证。

(二) 完成工作总述

本次作业能够圆满完成实验要求，具体完成情况如下：

1. 我们确定了编译器支持的语言特性，并参考教材及《SysY2022 语言定义》，用上下文无关文法对其进行详细的描述；
2. 我们一共设计了 3 个 SysY 程序，对每个程序给出其 C 语言描述，并对每个程序手动翻译为汇编程序，再利用汇编器对翻译后的程序生成可执行程序验证正确性；
3. 经验证，编写的所有程序均能够运行得到正常结果。

四、 预备工作 3 源码链接

贺祎昕 林语盈