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

史文天 乔诣昊

年级：2021 级

专业：信息安全，计算机科学与技术

指导教师：王刚

2023 年 10 月 9 日

摘要

确定编译器实现的功能，完成 CFG 设计工作，完善适合多种书写方式的变量和常量的声明及初始化。在 arm 汇编编程部分，乔诣昊负责改进版斐波那契程序的 SysY 编写和 arm 汇编编写，史文天负责阶乘程序的 SysY 和 arm 汇编编写，最后的共同完成思考题。

关键字：CFG，SysY，arm 汇编

目录

一、 问题描述

确定你要实现的编译器支持哪些 SvsY 语言特性，给出其形式化定义一-学习教材第 2 章及第 2 章讲义中的 2.2 节、参考 SysY 中巴克斯瑙尔范式定义，用上下文无关文法描述你的 SysY 语言子集。

设计几个 SysY 程序（如“预备工作 1”给出的阶乘或斐波那契），编写等价的 ARM 汇编程序，用汇编器生成可执行程序，调试通过、能正常运行得到正确结果。

二、 实验平台

设备名称	具体型号
虚拟机	VMware Workstation Pro
操作系统	ubuntu-22.04.2

表 1: 实验平台

三、 实验分工

- 史文天：CFG 的设计，阶乘程序的 SysY 编写和 arm 汇编编写，思考题讨论
- 乔诣昊：CFG 的设计，斐波那契程序的 SysY 编写和 arm 汇编编写，思考题讨论

四、 实验过程

(一) 定义编译器

1. 支持的 SysY 语言特性

- 数据类型：int
- 变量声明、常量声明，常量、变量的初始化
- 语句：赋值（=）、表达式语句、语句块、if、while、return
- 表达式：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））
- 注释
- 输入输出（实现连接 SysY 运行时库）
- 函数：函数声明、函数调用
- 变量、常量作用域——在语句块中包含变量、常量声明，break、continue 语句
- 数组（一维、二维、…）的声明和数组元素访问
- 浮点数：浮点数常量识别、变量声明、存储、运算

- 代码优化: 寄存器分配优化方法; 基于数据流分析的强度削弱、代码外提、公共子表达式删除、无用代码删除等
- 自动向量化

2. CFG 描述

我们利用 CFG 对所选 SysY 语言特性子集进行形式化定义，CFG 主要包括终结符集合 VT，非终结符集合 VN,开始符号 S,产生式集合 P。参考网站：https://buaa-se-compiling.github.io/miniSysY-tutorial/miniSysY.html

终结符集合 终结符是由单引号引起的字符串或者是标识符 (Ident) 和数值常量 (IntConst)。对于标识符，与 C 语言类似，具体见下上下文无关文法。

- 标识符（identifier）

identifier → identifier_nondigit |identifier identifier_nondigt | 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
digit → 0 | 1 | 2 | 3 | 4 | 5 | 6 | 7 | 8 | 9
全局变量和局部变量的作用域可以重叠，重叠部分局部变量优先；同名局部变量的作用域不能重叠；SysY 语言中变量名可以与函数名相同。

- 数字（number）

number 可以表示八进制、十进制、十六进制数字，文法如下:
Number → 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
digit → 0 | nonzero-digit
hexadecimal_digit → 0 | 1 | 2 | 3 | 4 | 5 | 6 | 7 | 8 | 9 | a | b | c | d | e | f | A | B | C | D | E | F

- 运算符：{ =, +, -, !, *, /, %, <, >, <=, >=, ==, !=, &&, || }
- 关键字：{void, int, const, Ident, if, while, break, continue, return, else, IntConst}
- 基本符号：{ ;, [,], {, }, (,), //, /*, */ }

非终结符集合 非终结符是一些表示源程序到终结符之间的中间状态语法变量，除了终结符之外，所有的符号都可以是非终结符。后面会有详细说明，此处不再列出。

开始符号 S 开始符号：CompUnit

表达式集合 P

- 编译单元：CompUnit → CompUnit Decl|CompUnit|Decl|FuncDef|
- 声明：Decl → ConstDecl|VarDecl
- 基本类型：BType → 'int'
- 浮点类型

  float-const -> float-dec-const | float-oct-const | float-hex-const float-dec-const -> int-dec-const node frac-dec-const float-oct-const -> int-oct-const node frac-oct-const float-hex-const -> int-hex-const node frac-hex-const node -> . frac-dec-const -> digit | frac-dec-const digit frac-oct-const -> oct-digit | frac-oct-const oct-digit frac-hex-const -> hex-digit | frac-hex-const hex-digit
- 常量声明：ConstDecl → 'const' BType ConstDefList ';' ConstDefList → ConstDefList, ConstDef|ConstDef
- 常数定义：ConstDef → Ident Dim '=' ConstInitVal Dim → Dim '[' ConstExp ']'|
- 常量初值:ConstInitVal → ConstExp|"ConstValElement" ConstValElement → ConstValEnum| ConstValEnum → ConstValEnum,ConstInitVal|ConstInitVal
- 变量声明：VarDecl → BType VarDefList ';' VarDefList → VarDefList, VarDef|VarDef
- 变量定义：VarDef → Ident Dim|Ident Dim '=' InitVal Dim → Dim '['ConstExp']'|
- 变量初值：InitVal → Exp|"ValElement"ValElement → ValEnum| ValEnum → ValEnum, InitVal|InitVal
- 函数定义：FuncDef → FuncType Ident '('FuncFParamList')' Block
- 函数类型：FuncType → 'void'|'int'
- 函数形参表：FuncFParamList → FuncFParams| FuncFParams → FuncFParams, FuncFParam|FuncFParam
- 函数形参:FuncFParam → BType Ident OpArray OpArray → Array| Array → [ ]|[ ]Arrays; Arrays → [Exp]Arrays|[Exp]
- 语句块：Block → OpBlockItems OpBlockItems → BlockItems| BlockItems → BlockItems BlockItem|BlockItem
- 语句块项：BlockItem → Decl|Stmt
- 语句：Stmt → LVal '=' Exp ';' | OpExp ';' | Block | 'if' (Cond) Stmt OpElse | 'while' (Cond) Stmt | 'break' ';' | 'continue' ';' | 'return' OpExp ';' OpExp → Exp| OpElse → 'else' Stmt|
- 表达式：Exp → AddExp（SysY 表达式是 int 型表达式）
- 条件表达式：Cond → LOrExp
- 左值表达式：LVal → Ident OpArr OpArr → Arrays|

- 基本表达式：PrimaryExp → ’(’ Exp ’)’|LVal|Number
- 数值：Number → IntConst
- 一元表达式：UnaryExp → PrimaryExp|Ident ’(’ OpFuncRParams ’)’| UnaryOp UnaryExp
- 单目运算符：UnaryOp → ’+’|’-’|’!’ • 函数实参表：FuncRParams → FuncRParams,Exp|Exp
- 乘除模表达式：MulExp → UnaryExp | MulExp ’*’ UnaryExp | MulExp ’/’ UnaryExp | MulExp ’
- 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) SysY 程序设计

1. 斐波那契数列

斐波那契 sysY 程序

```
#include<stdio.h>
int main(){
        int a,b,i,t,n;
        a = 0;
        b = 1;
        i = 1;
        printf("input the number:");
        scanf("%d",&n);
        printf("%d\n",b);
        while(i<n){
                t = b;
                b = a + b;
                printf("%d\n",b);
                a = t;
                i = i + 1;
        }
        return 0;
}
```

斐波那契 arm 汇编程序

```
@数据段
@全局变量及常量的声明
```

```
    .data
a:              @声明变量a
    .word 0     @a的初始值为0
b:              @声明变量b
    .word 1     @b的初始值为1
i:              @声明变量i
    .word 1     @i的初始值为1
t:              @声明变量t
    .word 0     @t的初始值为0
n:              @声明变量n
    .word 0     @n的初始值为0

@代码段
    .text
    .align 4
res:
    .asciz "%d␣\n"  @声明字符串res, 用于printf函数输出
    .align 4
info:
    .asciz "input␣the␣number:"  @声明字符串info, 用于printf函数输出
input:
    .asciz "%d"     @声明字符串input, 用于scanf函数输入
    .align 4

@主函数
        .global main
        .type main, %function
main:
        @mov r7, lr
        push {fp, lr}  @保存返回地址栈基地址

.input:
        adr r0, info    @读取字info字符串地址
        bl printf       @调用printf函数输出
        mov r8, lr
        adr r0, input
        sub sp, sp, #4   @留出一个4字节的空间, 保存用户输入
        mov r1, sp
        bl scanf
        ldr r2, [sp, #0]  @将用户输入的值保存到r2寄存器中
        ldr r1, addr_n0
        str r2, [r1]  @将用户输入的值保存到变量n对应的地址中
        add sp, sp, #4
        mov lr, r8

.params:
        mov r0, r2          @将用户输入的值保存到r0寄存器中
        ldr r4, addr_i0  @将变量i的地址保存到r4寄存器中
```

```
51          ldr r4, [r4]        @将变量i的值保存到r4寄存器中
52          ldr r3, addr_b0     @将变量b的地址保存到r3寄存器中
53          ldr r3, [r3]        @将变量b的值保存到r3寄存器中
54          ldr r6, addr_a0     @将变量a的地址保存到r6寄存器中
55          ldr r6, [r6]        @将变量a的值保存到r6寄存器中
56
57  .output:
58          push {r0,r1,r2,r3}  @将r0、r1、r2、r3寄存器的值保存到栈中
59          adr r0, res         @将字符串res的地址保存到r0寄存器中
60          mov r1, r3          @将变量b的值保存到r1寄存器中
61          bl printf           @调用printf函数输出
62          pop {r0,r1,r2,r3}   @将r0、r1、r2、r3寄存器的值从栈中弹出
63
64  .LOOP1:
65          cmp r4, r0          @比较变量i的值和变量n的值
66          bge .end          @如果i>=n, 则跳转到end,i<n的话则执行下面的循环体
67          mov r5, r3          @将变量b的值保存到r5寄存器中
68          add r3, r3, r6      @计算a+b的值, 并将结果保存到r3寄存器中
69          push {r0,r1,r2,r3}  @将r0、r1、r2、r3寄存器的值保存到栈中
70          adr r0, res         @将字符串res的地址保存到r0寄存器中
71          mov r1, r3          @将变量b的值保存到r1寄存器中
72          bl printf           @调用printf函数输出
73          pop {r0,r1,r2,r3}   @将r0、r1、r2、r3寄存器的值从栈中弹出
74          mov r6, r5          @将变量t的值保存到r6寄存器中
75          add r4, #1          @将变量i的值加1
76          b .LOOP1            @跳回循环开头
77
78  .end:
79          @mov lr, r7
80          @bx lr                  @ 退出
81          pop {pc}
82
83  @桥接全局变量的地址
84  addr_a0:
85          .word a
86  addr_b0:
87          .word b
88  addr_i0:
89          .word i
90  addr_t0:
91          .word t
92  addr_n0:
93          .word n
94
95          .section    .note.GNU-stack,"",%progbits
```

- 在斐波那契额数列程序中，涉及到变量的声明，初始化与赋值、while 循环、算数运算等 SysY 语言特性。

- arm 汇编程序中有较多输入输出语句，使用过程中涉及栈帧的调整和寄存器的保存与恢复。
- 调用函数时通过寄存器 R0-R3 来传递参数。
- 汇编代码中最后“桥接”了在 C 代码中隐性的全局变量的地址。
- 代码的运行流程为用户输入一个数，然后输出斐波那契数列的前 n 个数

这里的测试示例是输入 5，正常运行的话就会输出斐波那契数列的前五个数，即 1，1，2，3，5，通过下图可以发现程序运行正确。

```
qiao@qiao-virtual-machine:~/桌面/编译/2$ make test
arm-linux-gnueabihf-gcc example.S -o example.out
qemu-arm -L /usr/arm-linux-gnueabihf ./example.out
input the number:5
1
1
2
3
5
```

图 1: 测试结果

2. 阶乘

阶乘 sysY 程序

```
#include <stdio.h>
int factorial(int n)
{
    if(n == 0)
    return 1;
    else
    return n*factorial(n-1);
}
int main()
{
    int num,result;
    putf("请输入要计算阶乘的数:");
    num=getint();
    result=factorial(num);
    putf("%d的阶乘为: %d\n",num,result);
    return 0;
}
```

阶乘 arm 汇编程序

```
@数据段
.data
input_num:
    .asciz "请输入要计算阶乘的数:"
```

```
format:
    .asciz "%d"
result:
    .asciz "%d的阶乘为: %d\n"

@代码段
.text
factorial:
    str lr,[sp,#-4]! @Pushlr(返回链接寄存器)到堆栈
    str r0,[sp,#-4]! @Push参数r0到堆栈, 这个是函数的参数

    cmp r0,#0 @对比r0and0的值
    bne is_nonzero @如果r0!=0那么跳转到分支is_nonzero
    mov r0,#1 @如果参数是0, 则r0=1（0的阶乘为1）, 函数退出
    b end

is_nonzero:

    sub r0,r0,#1
    bl factorial @调用factorial函数, 其结果会保存在r0中
    ldr r1,[sp] @从sp指向地址处取回原本的参数保存到r1中
    mul r0,r1

end:
    add sp,sp,#4 @恢复栈状态, 丢弃r0参数
    ldr lr,[sp],#4 @加载源lr的寄存器内容重新到lr寄存器中
    bx lr @退出factorial函数

.global main
main:
    str lr,[sp,#-4]! @保存lr到堆栈中
    sub sp,sp,#4 @留出一个4字节空间, 给用户输入保存

    ldr r0,address_of_input_num @传参, 提示输入num
    bl printf @调用printf

    ldr r0,address_of_format @scanf的格式化字符串参数
    mov r1,sp @堆栈顶层作为scanf的第二个参数
    bl scanf @调用scanf

    ldr r0,[sp] @加载输入的参数num给r0
    bl factorial @调用factorial, 结果保存在r0

    mov r2,r0 @结果赋值给r2, 作为printf第三个参数
    ldr r1,[sp] @读入的整数, 作为printf第二个参数
    ldr r0,address_of_result @作为printf第一个参数
    bl printf @调用printf

```

```
53 add sp,sp,#4
54 ldr lr,[sp],#4 @弹出保存的lr
55 bx lr @退出
56
57 @桥接全局变量的地址
58 address_of_input_num:
59
60     .word input_num
61 address_of_result:
62     .word result
63 address_of_format:
64     .word format
65
66     .section .note.GNU-stack,"",%progbits
```

- 在该计算阶乘的程序中，体现出了 SysY 语言的如下几个特性：函数的编写与调用；变量的声明与赋值；putf 和 getint 等运行时库的调用；if 条件判断语句和基本的运算表达式使用。
- 该程序运行流程为用户输入一个数 n，最后输出这个 n 的阶乘

这里的测试示例是输入 9，正常运行的话就会输出 9 的阶乘，即 362880，通过下图可以发现程序运行正确。

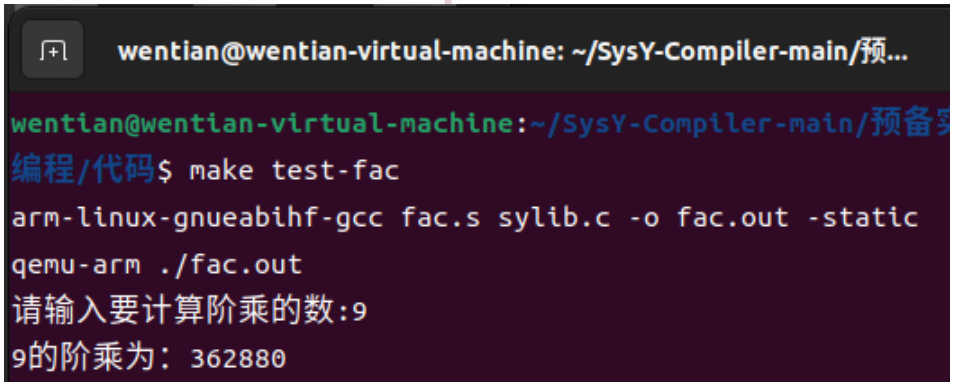

图 2: 测试结果

3. makefile 内容

斐波那契的 makefile

makefile

```
1 .PHONY: test, clean
2 test:
3         arm-linux-gnueabihf-gcc example.S -o example.out
4         qemu-arm -L /usr/arm-linux-gnueabihf ./example.out
5 clean:
6         rm -fr example.out
```

阶乘的 makefile

makefile

```
.PHONY: test-fac,clean
test-fac:
        arm-linux-gnueabihf-gcc fac.s sylib.c -o fac.out -static
        qemu-arm ./fac.out
clean:
        rm -fr *.out
```

五、 思考

如果不是人“手工编译”，而是要实现一个计算机程序（编译器）来将 SysY 程序转换为汇编程序，应该如何做（这个编译器程序的数据结构和算法设计）?

注意: 编译器不能只会翻译一个源程序，而是要有能力翻译所有合法的 SysY 程序。

- 词法分析作为语法分析的调用接口，将源程序中的字符序列转换为单词序列。
- 利用上下文无关文法将不同语言特性的程序语句符号化。
- 利用语法制导翻译，数据结构采用语法分析树的形式，将语义动作嵌入树的节点中。
- 语法分析树的构造采用预测分析法，根据下一输入单词进行首单词比对，选择候选式；对终结符进行匹配 match；对每一个非终结符编写递归函数；左递归改为右递归。
- 设计语法制导定义/翻译模式实现 SysY 程序到汇编程序的翻译。
- 以键值对的形式存储 ID 和对应的值，以这种方式构造符号表。

六、 总结

实验完成过程中，经过预先查阅资料学习确定了本学期构建的编译器实现的特性，共同完成 CFG 设计工作，完善了适合多种书写方式的变量和常量的声明及初始化。在 arm 汇编编程部分，乔诣昊负责改进版斐波那契程序的 SysY 编写和 arm 汇编编写，史文天负责阶乘程序的 SysY 和 arm 汇编编写，最后的思考题共同完成。

七、 git 库

GitLab 链接：https://gitlab.eduxiji.net/nku2023_compiler_principle/labs