南開大學
Nankai University

南　　开　　大　　学

计　算　机　学　院

编译原理实验报告

定义你的编译器、汇编编程 & 熟悉辅助工具

马永田 & 李佩诺

年级：2020 级

专业：计算机科学与技术 & 信息安全

指导教师：王刚

2022 年 10 月 15 日

摘要

本次实验进行于“预备工作 1 了解你的编译器 &LLVM IR 编程”的基础之上，通过对编写的 SysY 程序进行分析和手工编译，继续了解 SysY 语言特性以及编译器的使用，最终达到熟悉 SysY 语言的特性并且得到调试通过、能正常运行的正确结果。

关键字： SysY　编译器　ARM 汇编

目录

一、 实验目的

基于“预备工作 1”，继续:

1. 确定你要实现的编译器支持哪些 SysY 语言特性，给出其形式化定义——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。
2. 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。这些程序应该尽可能全面地包含你支持的语言特性。

实验要求

- 确定上机大作业分组，后续实验中不再调整。
- 撰写研究报告，编写的 ARM 汇编要给出 GitLab 项目链接
- 在报告中用一个小节明确、详细地指出两人的分工情况。
- 不能直接提交用 GCC 等编译器生成 C 程序对应的汇编程序，可学习 GCC 生成的其他 C 程序的汇编程序，仿照着编写自己 SysY 程序的汇编程序。

二、 实验环境

本次实验使用 VsCode 远程连接虚拟机环境 Ubuntu20.4:

架构	$x86_64$
CPU 运行模式	32-bit,64-bit
字节序	Little Endian
Address sizes	45bits physical,48 bits virtual
CPU	4
每个核的线程数	1
每个座的核数	2
CPU 系列	6
型号名称	Intel(R) Core(TM) i7-10510U CPU@1.80GHZ

表 1: 实验环境

三、 实验过程

(一) 实验描述

1. 实验步骤

对在预备实验 1 中编写的 SysY 程序进行改进，继续了解其定义与编译器所支持的特性，确定实现的功能并编写好程序后，给出程序子集的 CFG 描述并对其进行手工编译，最后对手工编译得到的 ARM 汇编代码进行测试与分析。

2. 分工简述

本次实验的分工为如下：两人合作探讨对整个实验的理解，多次讨论交流实验细节，马永田同学负责 SysY 源程序的编写、armv5t 汇编代码对应 SysY 语言特性中全局变量、函数、循环语句以及运算符的编写、CFG 描述中终结符、非终结符集合的编写、本次实验报告的部分编写；李佩诺同学完成了 SysY 程序的部分修改、armv5t 汇编代码对应 SysY 语言特性中逻辑判断、部分函数调用的编写、CFG 描述中开始符号、产生式集合的编写以及本次实验报告的部分编写。

(二) 编译器定义

1. CFG 描述

终结符集合 V_T

SysY 语言的终结符特征包括以下三个：

1. 标识符 (identifier)
2. 注释
3. 整型常量 (IntConst)

a. 标识符定义

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

identifier-nondigit → '_'| 'a'| 'b'| 'c'| 'd'| 'e'| 'f' | 'g'| 'h'| 'i'| 'j'| 'k'| 'l'| 'm'
| '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'

b. 注释定义

- 单行注释：以序列‘//’开始，直到换行符结束，不包括换行符。
- 多行注释：以序列‘/*’开始，直到第一次出现‘*/’时结束，包括结束处‘*/’。

c. 整型常量定义

integer-const → 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’

nonzero-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’

非终结符集合 $\mathbf{V}_S$

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		

开始符号 S

编译单元 CompUnit

产生式集合 P

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 {','ConstDef }';'
基本类型	BType	→ 'int'
		| 'float'
常数定义	ConstDef	→ Ident { '[' ConstExp ']' } '=' ConstInitVal
常量初值	ConstInitVal	→ ConstExp
		| '{' [ConstInitVal { ',' ConstInitVal }] '} '
变量声明	VarDecl	→ BType VarDef { ',' VarDef } ';'
变量定义	VarDef	→ Ident { '[' ConstExp ']' }
		| Ident { '[' ConstExp ']' } '=' InitVal
变量初值	InitVal	→ Exp
		| '{' [InitVal { ',' InitVal }] '} '

c. 函数

函数定义	FuncDef	→ FuncType Ident '(' [FuncFParams] ')' Block
函数类型	FuncType	→ 'void'
		| 'int'
		| 'float'
函数形参表	FuncFParams	→ FuncFParam { ',' FuncFParam }
函数形参	FuncFParam	→ BType Ident ['[' ']' { '[' 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 ['else' Stmt]
		| 'while' '(' Cond ')' Stmt
		| 'break' ';' | 'continue' ';'
		| '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 { '[' Exp ']'}
基本表达式	PrimaryExp	→ '(' Exp ')' | LVal | Number
数值	Number	→ IntConst | floatConst
一元表达式	UnaryExp	→ PrimaryExp | Ident '(' [FuncRParams] ')'
		| UnaryOp UnaryExp
单目运算符	UnaryOp	→ '+'
		| '−'
		| '!'
函数实参表	FuncRParams	→ Exp { ',' Exp }
乘除模表达式	MulExp	→ UnaryExp
		| MulExp ('*' | '/' | '%') UnaryExp
加减表达式	AddExp	→ MulExp
		| AddExp ('+' | '−') MulExp
关系表达式	RelExp	→ AddExp
		| RelExp ('<' | '>' | '<=' | '>=') AddExp
相等性表达式	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

2. SysY 程序

SysY 语言是编译系统设计赛要实现的编程语言，由 C 语言的一个子集扩展而成，该文件中有且仅有一个名为 main 的主函数定义，还可以包含若干全局变量声明、常量声明和其他函数定义。在本次实验中，我们编写的 SysY 程序包括以下特性：全局变量声明、函数、语句（while 语句、if 语句、赋值语句等等）、算术运算、关系运算、逻辑运算等等，具体代码与标注如下所示：

SysY 程序代码

```
//全局变量
int globla_var;
int n;
//函数
int mul(int a,int b)
{
        return a*b;
}
int main(){
    n=getint();
//while语句
    while(n>=6)
    {
        n=getint();
    }
//for循环 求阶乘
    globla_var = 1;
    for(int i = 1; i <= n; i++)
    {
//函数调用
        globla_var = mul(i,globla_var);
    }
//运算符
    globla_var = globla_var * 4;
    globla_var = globla_var + 2;
    globla_var = globla_var | 1;
//逻辑判断
    if((globla_var>=65&&globla_var<=90)||(globla_var>=97&&globla_var<=122)){
            putch(globla_var);
    } else {
        putint(globla_var);
    }
    return 0;
}
```

(三) armv5t 汇编编程

确定好 SysY 语言源程序后，我们对其进行手动翻译，我们选择翻译成的 arm 汇编语言版本是 armv5t（与指导书一致），以下为代码与其**具体对应特性的注释**:

armv5t 汇编代码

```
        .arch armv5t
@ comm section
        .comm n, 4 @ global variables
        .comm globla_var, 4 @ global variables
        .text
```

```
        .align 2
@ text section code
        .global mul
        mul: @ int mul(int a, int b)
        str fp, [sp, #-4]! @ sp = sp - 4, push fp
        mov fp, sp
        sub sp, sp, #12     @ allocate space for local variable
        str r0, [fp, #-8]   @ r0 = [fp, #-8] = a
        str r1, [fp, #-12]  @ r1 = [fp, #-12] = b
        mul r0, r1, r0      @ r0 = r1 * r0 = a * b
        add sp, fp, #0      @ release the stack frame of function mul
        ldr fp, [sp], #4    @ post-index mode, pop fp, sp = sp + 4
        bx lr @ recover sp fp pc
_str0:
        .ascii "\012\000"
        .align 2
        .global main

main:
        push {fp, lr}
        add fp, sp, #4
        bl getint(PLT)          @ n = getint()

@ while statement
.L1:
        mov r1, #6
        cmp r0, r1
        blt .L2                 @ if n < 6 go to .L2
        bl getint(PLT)          @ n = getint()
        b .L1

@ for statement to Calculating factorial
.L2:                            @ Initialize the variables
        mov r5, r0              @ r5 = n
        mov r2, r0              @ r2 = n
        mov r1, #1              @ r1 = i = 1
        mov r0, #1              @ r0 = global_var = 1

.L3:                            @ for cycle block
        cmp r2, r1              @ compare i and n
        blt .L4                 @ if i > n go to .L4
        bl mul                  @ call mul(global,i)
        add r1, #1              @ i++
        b .L3                   @ go for

@ Arithmetic operations
.L4:
        mov r2, #4
```

```
        mul r2, r0, r2          @ global_var = global_var * 4
        mov r0, r2

        mov r2, #2
        add r2, r0, r2          @ global_var = global_var + 4
        mov r0, r2

        mov r2, #1
        orr r2, r0, r2          @ global_var = global_var | 4
        mov r0, r2

@ if statement and Logical operations
        mov r3, r0      @ r3 = global_var
        cmp r3, #64
        ble .L5         @ if global_var <= 64 go to .L5(Enter else block)
        cmp r3, #90
        ble .L6         @ if global_var <= 90 go to .L6 (Enter if block)
.L5:
        cmp     r3, #96
        ble     .L7     @ if global_var <= 96 go to .L7 (Enter else block)
        cmp     r3, #122
        bgt     .L7     @ if global_var >122 go to .L7 (Enter else block)

.L6:                    @ if block
        bl      putch(PLT)
        b .L8
.L7:                    @ else block
        bl      putf(PLT)
.L8:
        ldr r0, _bridge+8
        bl      putf(PLT)
        mov r0, #0
        pop {fp, pc}    @ return 0
        _bridge:
        .word globla_var
        .word n
        .word _str0
        .section .note.GNU-stack,"",%progbits @ do you know what's the use of
            this :-)
```

(四) 实验结果与分析

1. 测试结果

通过以下代码对编写的 armv5t 代码进行交叉编译和运行。

```
arm-linux-gnueabihf-gcc main.S sylib.c -o main.out
qemu-arm ./main.out
```

得到下图结果:

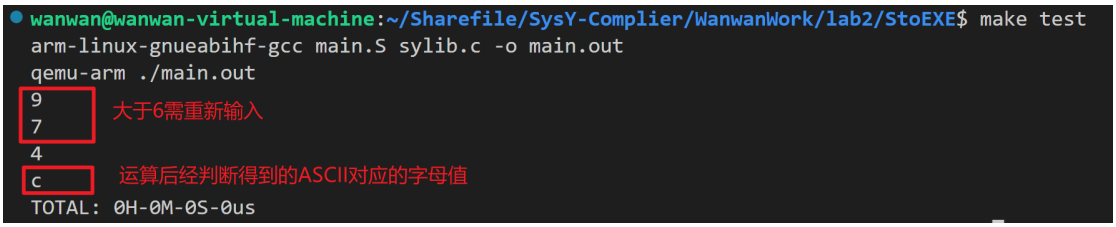

图 1: 汇编代码结果验证

经验证结果正确。

2. 代码对比

通过使用 arm-linux-gnueabihf-gcc 编译出的 arm 汇编代码版本为 armv7-a，与手动翻译的 armv5t 汇编代码相比, 除了版本不同导致反汇编器在函数识别上的不同以外, 机器翻译出的 arm 汇编程序多了很多内容，例如开头部分多了如下代码

```
1       .arch armv7-a
2       .eabi_attribute 28, 1
3       .eabi_attribute 20, 1
4       .eabi_attribute 21, 1
5       .eabi_attribute 23, 3
6       .eabi_attribute 24, 1
7       .eabi_attribute 25, 1
8       .eabi_attribute 26, 2
9       .eabi_attribute 30, 6
10      .eabi_attribute 34, 1
11      .eabi_attribute 18, 4
12      .file   "main.c"
```

这一部分是给 ARM cpu 的声明，.arch 指明体系架构类型。接下来的几行.eabi_attrbute 指定了一些接口，EABI 嵌入式应用二级制接口是 ARM 指定的一套接口规范, 此处即 EABI 的 option, 最后一行指明文件名称。

此外还会多出如下代码:

```
1       .syntax unified
2       .thumb
3       .thumb_func
4       .fpu vfpv3-d16
```

.syntax unified 为语法选择,ARM 和 THUMB 指令支持两种略有不同的语法. 默认值 divided 为旧样式, 下面的指令使用 ARM 和 THUMB 各自独立的语法; unified 为新样式，下面的指令使用 ARM 和 THUMB 通用格式。而.thumb 则是表示使用 thumb 模式, 等价于 gcc -mthumb。.fpu 则指明 Floating Point Unit(浮点运算单元的运算模式)，与 gcc “-mfpu”命令行选项作用相同, 例如 softvfp 为软浮点，fpv5-d16 or fpv5-sp-d16 cortex-M7 为单精度硬件浮点等。

此外也可以看到在 armv5t 指令集中寻址使用的是 _brigde, 而在 armv7 指令集中则是使用 _GLOBAL_OFFSET_TABLE_ 这样一个全局偏移地址进行计算来实现的寻址。

使用编译器生成的代码还会自带部分注释, 例如函数定义中会有如下部分说明函数的一些相关信息:

```
@ args = 0, pretend = 0, frame = 8
@ frame_needed = 1, uses_anonymous_args = 0
```

完整机器编译 armv7-a 汇编代码

```
        .arch armv7-a
        .eabi_attribute 28, 1
        .eabi_attribute 20, 1
        .eabi_attribute 21, 1
        .eabi_attribute 23, 3
        .eabi_attribute 24, 1
        .eabi_attribute 25, 1
        .eabi_attribute 26, 2
        .eabi_attribute 30, 6
        .eabi_attribute 34, 1
        .eabi_attribute 18, 4
        .file   "main.c"
        .text
        .comm   globla_var,4,4
        .comm   n,4,4
        .align  1
        .global mul
        .arch armv7-a
        .syntax unified
        .thumb
        .thumb_func
        .fpu vfpv3-d16
        .type   mul, %function
mul:
        @ args = 0, pretend = 0, frame = 8
        @ frame_needed = 1, uses_anonymous_args = 0
        @ link register save eliminated.
        push    {r7}
        sub     sp, sp, #12
        add     r7, sp, #0
        str     r0, [r7, #4]
        str     r1, [r7]
        ldr     r3, [r7, #4]
        ldr     r2, [r7]
        mul     r3, r2, r3
        mov     r0, r3
        adds    r7, r7, #12
        mov     sp, r7
        @ sp needed
        ldr     r7, [sp], #4
        bx      lr
```

```
        .size   mul, .-mul
        .align  1
        .global main
        .syntax unified
        .thumb
        .thumb_func
        .fpu vfpv3-d16
        .type   main, %function
main:
        @ args = 0, pretend = 0, frame = 8
        @ frame_needed = 1, uses_anonymous_args = 0
        push    {r4, r7, lr}
        sub     sp, sp, #12
        add     r7, sp, #0
        ldr     r4, .L13
.LPIC0:
        add     r4, pc
        bl      getint(PLT)
        mov     r2, r0
        ldr     r3, .L13+4
        ldr     r3, [r4, r3]
        str     r2, [r3]
        b       .L4
.L5:
        bl      getint(PLT)
        mov     r2, r0
        ldr     r3, .L13+4
        ldr     r3, [r4, r3]
        str     r2, [r3]
.L4:
        ldr     r3, .L13+4
        ldr     r3, [r4, r3]
        ldr     r3, [r3]
        cmp     r3, #5
        bgt     .L5
        ldr     r3, .L13+8
        ldr     r3, [r4, r3]
        mov     r2, r3
        movs    r3, #1
        str     r3, [r2]
        movs    r3, #1
        str     r3, [r7, #4]
        b       .L6
.L7:
        ldr     r3, .L13+8
        ldr     r3, [r4, r3]
        ldr     r3, [r3]
        mov     r1, r3
```

```
90          ldr     r0, [r7, #4]
91          bl      mul(PLT)
92          mov     r2, r0
93          ldr     r3, .L13+8
94          ldr     r3, [r4, r3]
95          str     r2, [r3]
96          ldr     r3, [r7, #4]
97          adds    r3, r3, #1
98          str     r3, [r7, #4]
99  .L6:
100         ldr     r3, .L13+4
101         ldr     r3, [r4, r3]
102         ldr     r3, [r3]
103         ldr     r2, [r7, #4]
104         cmp     r2, r3
105         ble     .L7
106         ldr     r3, .L13+8
107         ldr     r3, [r4, r3]
108         ldr     r3, [r3]
109         lsls    r3, r3, #2
110         ldr     r2, .L13+8
111         ldr     r2, [r4, r2]
112         str     r3, [r2]
113         ldr     r3, .L13+8
114         ldr     r3, [r4, r3]
115         ldr     r3, [r3]
116         adds    r3, r3, #2
117         ldr     r2, .L13+8
118         ldr     r2, [r4, r2]
119         str     r3, [r2]
120         ldr     r3, .L13+8
121         ldr     r3, [r4, r3]
122         ldr     r3, [r3]
123         orr     r3, r3, #1
124         ldr     r2, .L13+8
125         ldr     r2, [r4, r2]
126         str     r3, [r2]
127         ldr     r3, .L13+8
128         ldr     r3, [r4, r3]
129         ldr     r3, [r3]
130         cmp     r3, #64
131         ble     .L8
132         ldr     r3, .L13+8
133         ldr     r3, [r4, r3]
134         ldr     r3, [r3]
135         cmp     r3, #90
136         ble     .L9
137 .L8:
```

```
        ldr     r3, .L13+8
        ldr     r3, [r4, r3]
        ldr     r3, [r3]
        cmp     r3, #96
        ble     .L10
        ldr     r3, .L13+8
        ldr     r3, [r4, r3]
        ldr     r3, [r3]
        cmp     r3, #122
        bgt     .L10
.L9:
        ldr     r3, .L13+8
        ldr     r3, [r4, r3]
        ldr     r3, [r3]
        mov     r0, r3
        bl      putch(PLT)
        b       .L11
.L10:
        ldr     r3, .L13+8
        ldr     r3, [r4, r3]
        ldr     r3, [r3]
        mov     r0, r3
        bl      putint(PLT)
.L11:
        movs    r3, #0
        mov     r0, r3
        adds    r7, r7, #12
        mov     sp, r7
        @ sp needed
        pop     {r4, r7, pc}
.L14:
        .align  2
.L13:
        .word   _GLOBAL_OFFSET_TABLE_-(.LPIC0+4)
        .word   n(GOT)
        .word   globla_var(GOT)
        .size   main, .-main
        .ident  "GCC: (Ubuntu 9.4.0-1ubuntu1~20.04.1) 9.4.0"
        .section        .note.GNU-stack,"",%progbits
```

四、 总结

在本次实验中，主要学习了以下三方面的知识：

- 对上下文无关文法的学习，通过编写 SysY 语言的 CFG 描述，对编译课程理论课简述的相关知识有了更加深刻的理解。
- 继续深入地学习了编译器相关知识，同时在编写程序地过程中对编译器所支持的 SysY 语言特性更为熟悉。
- arm 汇编编程相关知识。通过对 arm 汇编代码的编写，了解了整个程序在汇编阶段时栈的变化情况与寄存器的使用方式，对汇编阶段的认识更为清晰。

另外，小组合作中明确的任务划分和与队友的讨论是本次实验能顺利完成的不可或缺因素，希望在之后的实验中继续掌握更多编译原理相关知识。

(一) 源码链接

实验中的所有源码文件等均已上传到 Github 和 GitLab