南开大学

Nankai University

南　　开　　大　　学

网 络 空 间 安 学 院

编译原理实验报告

定义编译器 & **汇编编程**

聂志强 2012307

许友锐 2013749

年级：2020 级

专业：信息安全

指导教师：李忠伟

2022 年 10 月 16 日

摘要

在此次实验中，为了将来完成编译器，我们小组合作确定了我们的编译器支持的 SysY 语言特性，并参考 SysY 中巴克斯瑙尔范式定义，用上下文无关文法描述了 SysY 语言子集。并根据所选 SysY 语言特性设计了改进版斐波拉契数列和阶乘程序，包含二维数、putarray() 和 getint() I/O 操作，函数调用、算数运算等十余种 SysY 语言特性，自主编写等价的 ARM 汇编程序并进行优化，通过解决遇到的栈帧调整、函数调用等困难对 ARM 汇编有了基本掌握，最后用汇编器生成可执行程序，调试通过并得到正确结果。

关键字： CFG, SysY, ARM **汇编**

目录

一、 定义编译器

(一) 编译器支持的 SysY 语言特性

基础 track:

- 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

竞赛 track:

- 数组
- 变量、常量作用域——在语句块中包含变量、常量声明，break、continue 语句
- 函数
- 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
  - 其他

为深入学习编译器原理，我们将在后面的实验中尝试实现以上所有 SysY 语言特性。

(二) CFG 描述 SysY 语言特性

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 V_T，非终结符集合 V_N，开始符号 S，产生式集合 P。

1. 终结符集合 V_T

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。

- 数值常量

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

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

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

2. **非终结符集合** V_N

非终结符即一些语法变量，定义了我们所需要的一些中间状态，是源程序到终结符之间的过渡。除终结符外其他均为非终结符，非终结符及其含义在下面的 CFG 表达式中，这里不再一一列出。

3. **开始符号** S

开始符号：CompUnit

4. **表达式集合** P

- 编译单元：CompUnit → CompUnit Decl|CompUnit|Decl|FuncDef| ϵ
- 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 ']'|ϵ
- 常量初值：ConstInitVal → ConstExp|''ConstValElement''

  ConstValElement → ConstValEnum|ϵ

  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']'|ϵ
- 变量初值：InitVal → Exp|''ValElement''

  ValElement → ValEnum|ϵ

  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|ϵ

  FuncFParams → FuncFParams, FuncFParam|FuncFParam

- 函数形参：FuncFParam → BType Ident OpArray
  OpArray → Array|ϵ
  Array → [ ]|[ ]Arrays;
  Arrays → [Exp]Arrays|[Exp]
- 语句块：Block → {OpBlockItems}
  OpBlockItems → BlockItems|ϵ
  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|ϵ
  OpElse → 'else' Stmt|ϵ
- 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|ϵ
- 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 '%' UnaryExp

- 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、 汇编编程

(一) 斐波拉契数列

1. 斐波那契 SysY 程序

斐波那契 sysY 程序

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

```
        i = i + 1;
    }
    int m[2][2] = {{2, 0}, {2, 2}};
    putarray(4, m); // 2 0 2 2
    return 0;
}
```

2. **斐波那契 arm 汇编程序**

斐波那契 sysY 程序

```
@数据段
@全局变量及常量的声明
        .data
a:
    .word 0
b:
        .word 1
i:
        .word 1
t:
        .word 0
n:
        .word 0

@代码段
    .text
    .align 4
res:
    .asciz "fibo: %d \n"
        .align 4
info:
        .asciz "please input the number of Fibonacci: "
input:
        .asciz "%d"
        .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
```

```
38          adr r0, input
39          sub sp, sp, #4    @留出一个4字节的空间，保存用户输入
40          mov r1, sp
41          bl scanf
42          ldr r2, [sp, #0]
43          ldr r1, addr_n0
44          str r2, [r1]  @保存n到对应地址中
45          add sp, sp, #4
46          mov lr, r8
47
48  .params:
49          mov r0, r2
50          ldr r4, addr_i0
51          ldr r4, [r4]      @ 变量i
52          ldr r3, addr_b0
53          ldr r3, [r3]      @ 变量b
54          ldr r6, addr_a0
55          ldr r6, [r6]      @ 变量a
56
57  .LOOP1:
58          mov r5, r3        @ t=b
59          add r3, r3, r6    @ b=a+b
60          push {r0,r1,r2,r3}
61          adr r0, res       @准备printf的参数
62          mov r1, r3
63          bl printf         @ 调用printf函数
64          pop {r0,r1,r2,r3}
65
66          mov r6, r5        @ a=t
67          add r4, #1        @ i=i+1
68          cmp r4, r0        @ 判断i与n大小关系
69          bne .LOOP1        @ 当i<n时跳转至loop1继续循环
70
71  .Array:
72          add     fp, sp, #0          @ 提升fp指向sp
73          sub     sp, sp, #16         @ 准备16的空间，因为我们有a[2][2]共16字节
74          @ 局部变量a[2][2]的空间位于sp到sp+12的位置，从低地址向高地址增长
75          mov r0, #2
76          str r0, [sp]                    @ a[0][0] = 2
77          str r0, [sp, #8]                    @ a[1][0] = 2
78          str r0, [sp, #12]                   @ a[1][1] = 2
79          mov r0, #0
80          str r0, [sp, #4]                    @ a[0][1] = 0
81
82          mov r1, sp                          @ 准备putarray()第二个参数，r1 = a的
                地址
83          mov r0, #4                                  @ 准备putarray()第一个参数，
                输出4个整数的数组
```

```
84          bl      putarray(PLT)       @ 调用putarray()
85
86          mov r0, #0                          @ return 0; r0存储函数返回值
87          add sp, sp, #16             @ 回收栈空间
88
89          pop             {fp}            @返回地址出栈
90          pop             {lr}
91          bx  lr                              @ 返回
92
93  .end:
94          @mov lr, r7
95          @bx lr                  @ 退出
96          pop {pc}
97
98  @桥接全局变量的地址
99  addr_a0:
100         .word a
101 addr_b0:
102         .word b
103 addr_i0:
104         .word i
105 addr_t0:
106         .word t
107 addr_n0:
108         .word n
109
110         .section    .note.GNU-stack,"",%progbits
```

（1）在斐波那契额数列程序中，涉及到二维数组、putarray() I/O 操作、变量的声明与赋值、while 循环、算数运算等 SysY 语言特性，arm 汇编程序中有较多的交互输出语句，使用过程中涉及栈帧的调整和寄存器的保存与恢复。

（2）调用函数时通过寄存器 R0-R3 来传递参数

（3）调用 SysY 运行时库提供的 I/O 函数 putarray()，通过 ARM 指令 mov r1, sp 使 r1 指向数组 m 的首地址【准备第 2 个参数 (输出数组地址)】，通过 ARM 指令 mov r0, #4 使 r0 赋值为 4【准备第 1 个参数 (表示要输出的数组)】，同时 putarray 函数在输出时会在整数之间安插空格。

（4）汇编代码中利用 _bridge 标签，“桥接”了在 C 代码中隐性的全局变量的地址。

（5）将编写的 ARM 汇编程序执行 arm-linux-gnueabihf-gcc fib.s sylib.c -o fib -static 指令，得到正确输出结果如下图所示：

```
nie762174555@ubuntu:~/pre-2$ make test-fib
arm-linux-gnueabihf-gcc fib.s sylib.c -o fib.out -static
qemu-arm ./fib.out
please input the number of Fibonacci: 10
fibo: 1
fibo: 2
fibo: 3
fibo: 5
fibo: 8
fibo: 13
fibo: 21
fibo: 34
fibo: 55
4: 2 0 2 2
TOTAL: 0H-0M-0S-0us
nie762174555@ubuntu:~/pre-2$
```

图 1: 斐波那契数列 ARM 程序运行结果

(二) 阶乘

1. 阶乘 sysY 程序

阶乘 sysY 程序

```
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;
}
```

2. 阶乘 arm 汇编程序

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]! @ Push lr(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r0 and 0 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

    add sp, sp, #4
    ldr lr, [sp], #4           @ 弹出保存的lr
    bx lr                      @ 退出

 @ 桥接全局变量的地址
address_of_input_num:
```

```
59      .word input_num
60  address_of_result:
61      .word result
62  address_of_format:
63      .word format
64
65      .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。以下对于相关 arm 汇编代码的实现做出说明：

（1）arm 汇编代码与 x86 类似，可以分为若干个节，如.data 数据节，.text 代码节等。代码段的最后一般需要声明桥接全局变量的地址。

（2）编写阶乘函数 factorial 的具体代码内容前，需要先使用 str 指令保存 lr 寄存器和函数参数的值，通常 r0-r3 用作保存参数，lr 寄存器用于保存返回地址。函数返回时，需要恢复栈状态，并用 ldr 指令恢复 lr 寄存器的值，最后的返回值保存在 r0 寄存器中。

（3）函数 factorial 的主要代码实现与 if-else 分支语句、递归调用相关。对于分支语句来说，需要使用 cmp,bne 这两个指令和相关代码标签实现，先判断参数 r0 与 0 的值，若不等则跳转到 is_nonzero 代码段，否则跳转到 end 代码段。递归调用需要在 factorial 函数中使用 bl 指令再次调用该函数，此时其参数为 r0 -1。

使用指令 “arm-linux-gnueabihf-gcc fib.s sylib.c -o fib -static” 即可将汇编代码转为相应的可执行文件，文件执行情况如下图所示：

```
wangke@wangke-virtual-machine:~/文档/compile_test/Lab2/fac$ make
arm-linux-gnueabihf-gcc fac1.s sylib.c -o fac1.out -static
qemu-arm ./fac1.out
请输入要计算阶乘的数: 6
6的阶乘为: 720
TOTAL: 0H-0M-0S-0us
```

图 2: 阶乘程序 fac1 运行结果

(三) 使用的 Makefile 文件

Makefile 文件

```
1   .PHONY: test-fib, test-fac, clean
2   test-fib:
3           arm-linux-gnueabihf-gcc fib.s sylib.c -o fib.out -static
4           qemu-arm ./fib.out
5
6   test-fac:
7           arm-linux-gnueabihf-gcc fac1.s sylib.c -o fac1.out -static
8           qemu-arm ./fac1.out
9
10  clean:
11          rm -fr *.out
```

三、 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）?

- 词法分析作为语法分析的调用接口，调用一次返回应该单词。
- 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。
- 为每个阶段构造符号表，以键值对的形式存储 ID 和对应的值。

四、 总结与分工

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，并将完成结果合并后进行讨论与改进，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，聂志强（2012307）负责改进版斐波那契程序的 SysY 编写和 arm 汇编编写，许友锐（2013749）负责阶乘程序的 SysY 和 arm 汇编编写，过程中遇到的函数调用、栈帧转换等困难经过一起讨论得以解决，最后的思考题共同完成。