



南开大学
Nankai University

南开大学

计算机学院和网络空间安全学院

编译系统原理实验报告

定义你的编译器、汇编编程 & 熟悉辅助工具

朱子轩 2110853 计算机科学与技术

刘修铭 2112492 信息安全

年级：2021 级

指导教师：王刚

[HTTPS://GITLAB.EDUXIJI.NET/NKU2023/NKU2023_COMPILER](https://gitlab.eduxiji.net/nku2023/nku2023_compiler)

2023 年 9 月 28 日

摘要

为了将来能够更好的完成自己的编译器，在本次实验中，朱子轩和刘修铭两人确定了编译器支持的 SysY 语言特性，参考 SysY 中的巴克斯瑙尔范式定义，使用上下文无关文法描述了 SysY 语言的子集。

接着，根据确定的 SysY 语言特性设计了两个小程序并编写了其对应的 ARM 汇编程序并加以优化，进一步掌握了 ARM 汇编。

最后，使用汇编器将自己编写的程序生成可执行程序加以验证。经过调试运行，得到了预期结果，说明编写的程序准确无误。

关键字：SysY, CFG, ARM 汇编

目录

一、 分工	1
二、 定义编译器	2
(一) 编译器支持的 SysY 语言特性	2
(二) CFG 描述	2
1. 终结符号集合 V_T	2
2. 非终结符号集合 V_N	4
3. 开始符号 S	4
4. 产生式集合 \mathcal{P}	4
三、 汇编编程	7
(一) 圆的周长与面积	7
1. SysY 程序	7
2. ARM 汇编程序	7
(二) 斐波那契数列	9
1. SysY 程序	9
2. ARM 汇编程序	9
四、 思考题	12

一、 分工

在 CFG 设计阶段，刘修铭负责完成终结符号集合与开始符号集合部分的编写，并与朱子轩一同完成非终结符号集合与产生式集合的编写。

在 ARM 汇编编程部分，由刘修铭设计两个 SysY 程序，并负责圆的周长与面积程序的编写，由朱子轩负责斐波那契数列程序的编写，期间遇到的问题由二人共同讨论解决。

最后的思考题部分，由二人共同讨论完成。

NIJUB

二、 定义编译器

(一) 编译器支持的 SysY 语言特性

为了深入学习编译原理，了解相关知识，我们将尝试实现以下 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：

- 函数、语句块
 - 函数：函数声明、函数调用
 - 变量、常量作用域：在函数中、语句块（嵌套）中包含变量、常量声明的处理，break、continue 语句
- 数组：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

(二) CFG 描述

按照讲义说明，使用四元组 $(v_t, v_N, S, \mathcal{P})$ 定义该文法，下面将分别进行各部分的说明。

1. 终结符号集合 V_T

终结符号是该文法所定义的语言的基本符号的集合，通常是源代码中的实际字符、词法分析器生成的记号或单词，以及编程语言中的关键字、运算符、标点符号等。

(1) 标识符 (identifier)

$$\begin{aligned}
 \text{identifier} &\rightarrow \text{identifier_nondigit} \\
 &\quad | \text{identifier identifier_nondigit} \\
 &\quad | \text{identifierdigit} \\
 \text{identifier_nondigit} &\rightarrow _ | a | b | c | d | e | f | g | h | i | j | k | l | m | \\
 &\quad n | o | p | q | r | s | t | u | v | w | x | y | z | \\
 &\quad A | B | C | D | E | F | G | H | I | J | K | L | M | \\
 &\quad N | O | P | Q | R | S | T | U | V | W | X | Y | Z | \\
 \text{digit} &\rightarrow 0 | 1 | 2 | 3 | 4 | 5 | 6 | 7 | 8 | 9
 \end{aligned}$$

对于同名标识符，我们约定：

- 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。

(2) 数值常量

$$\begin{aligned}
 \text{integer_const} &\rightarrow \text{decimal_const} \\
 &\quad \text{octal_const} \\
 &\quad \text{hexadecimal_const} \\
 \text{decimal_const} &\rightarrow \text{nonzero_digit} \\
 &\quad \text{decimal_const digit} \\
 \text{octal_const} &\rightarrow 0 | \text{octal_const octal_digit} \\
 \text{hexadecimal_const} &\rightarrow \text{hexadecimal_prefix hexadecimal_digit} \\
 &\quad | \text{hexadecimal_const hexadecimal_digit} \\
 \text{hexadecimal_prefix} &\rightarrow '0x' | '0X' \\
 \text{nonzero_digit} &\rightarrow 1 | 2 | 3 | 4 | 5 | 6 | 7 | 8 | 9 \\
 \text{octal_digit} &\rightarrow 0 | 1 | 2 | 3 | 4 | 5 | 6 | 7 \\
 \text{hexadecimal_digit} &\rightarrow 0 | 1 | 2 | 3 | 4 | 5 | 6 | 7 | 8 | 9 | \\
 &\quad a | b | c | d | e | f | A | B | C | D | E | F
 \end{aligned}$$

(3) 基本符号

$$\{ ;, [,], (,), \{, \}, //, /*, */ \}$$

(4) 关键字

$$\{ int, void, const, if, while, break, continue, return, else, Ident, IntConst \}$$

(5) 运算符

$$\{ +, -, *, /, \%, =, ==, !=, <, >, <=, >=, !, \&\&, || \}$$

2. 非终结符号集合 V_N

非终结符表示语法规则中的可扩展部分或占位符，是一种抽象的符号，可以进一步扩展、重复或嵌套。编译器使用非终结符来定义编程语言的语法，通过组合非终结符和终结符来构建语法树或分析源代码。实验中涉及的部分非终结符号如下表所示：

符号含义	符号表示	符号含义	符号表示
变量类型	Type	一维数组	SingleArray
数组特征	ArrayLike	数字列表	DigitList
数组符号	ArrayVal	初始化值	InitVal
变量具体定义	Def	定义列表	DefList
常量声明	ConstDecl	变量声明	ValDecl
变量与常量声明	Decl	括号等	Factor
单目运算	Single	乘除取模	Term
加减运算	AddSub	比较	Comp
等于和不等	Eq	表达式	Expr
表达式列表	ExprList	函数返回类型	FuncType
函数参数	FuncParam	函数参数	FuncParams
函数定义	FuncDef	语句块	Block
声明或语句	BlockItem	左值表达式	LVal
语句	Stmt	编译单元	CompUnit

3. 开始符号 S

开始符号: *CompUnit*

4. 产生式集合 \mathcal{P}

- 变量和常量的声明、定义和初始化，包括数组等

$$\begin{aligned}
Type &\rightarrow \mathbf{int} \mid \mathbf{float} \\
SingleArray &\rightarrow [\mathbf{digit}] \\
ArrayLike &\rightarrow ArrayLike \ SingleArray \mid SingleArray \\
DigitList &\rightarrow DigitList, \ digit \mid digit \\
ArrayVal &\rightarrow ArrayVal, \ \{DigitList\} \mid \{DigitList\} \\
InitVal &\rightarrow Expr \mid \{ArrayVal\} \mid \{\} \mid \{DigitList\} \\
Def &\rightarrow \mathbf{id} = InitVal \mid \mathbf{id} \ ArrayLike = InitVal \\
DefList &\rightarrow DefList, \ Def \mid Def \\
ConstDecl &\rightarrow \mathbf{const} \ Type \ DefList \\
ValDecl &\rightarrow \ Type \ DefList \\
Decl &\rightarrow ConstDecl \mid ValDecl
\end{aligned}$$

- 表达式

$$\begin{aligned}
Factor &\rightarrow (\ Expr \) \mid Id \ [Expr] \mid digit \mid id \mid FuncId(Expr) \\
Single &\rightarrow + \ Factor \mid - \ Factor \mid !Factor \\
Term &\rightarrow Term * \ Single \mid Term / \ Single \mid Term \% \ Single \mid Single \\
AddSub &\rightarrow AddSub + \ Term \mid AddSub - \ Term \mid Term \\
Comp &\rightarrow Comp > \ AddSub \mid Comp < \ AddSub \\
&\mid Comp >= \ AddSub \mid Comp <= \ AddSub \mid AddSub \\
Eq &\rightarrow Eq \ != \ Comp \mid Eq \ == \ Comp \mid Comp \\
Expr &\rightarrow Expr \ \&\& \ Eq \mid Expr \ || \ Eq \mid Eq \\
ExprList &\rightarrow ExprList, \ Expr \mid Expr
\end{aligned}$$

- 函数声明和定义

$$\begin{aligned}
FuncType &\rightarrow \mathbf{void} \mid Type \\
FuncParam &\rightarrow FuncType \ id \\
FuncParams &\rightarrow FuncParams, \ FuncParams \mid FuncParam \\
FuncDef &\rightarrow FuncType \ id \ () \ Block \mid FuncType \ id \ (FuncParams) \ Block
\end{aligned}$$

- 语句块

$$\begin{aligned}Block &\rightarrow \{\} \mid \{ BlockItem \} \\BlockItem &\rightarrow Decl \mid Stmt \\LVal &\rightarrow id \ ArrayLike \mid id \\Stmt &\rightarrow Lval = Expr; \mid Expr; \mid Block \\&\mid \textbf{if} (expr) \mid Stmt \\&\mid \textbf{if} (expr) \mid Stmt \textbf{ else } Stmt \\&\mid \textbf{while} (expr) Stmt \\&\mid \textbf{break}; \\&\mid \textbf{continue}; \\&\mid \textbf{return } Expr; \mid \textbf{return}; \\CompUnit &\rightarrow Decl \mid FuncDef\end{aligned}$$

MINUB

三、 汇编编程

为了验证上述的语言规则，我们编写了如下两个程序用来测试。

(一) 圆的周长与面积

1. SysY 程序

圆的周长与面积 SysY 程序

```
1 #include <stdio.h>
2 const float pi = 3.14;
3 int main() {
4     float D = 2.0;
5     printf("perimeter: %.2f\n", pi * D);
6     printf("area: %.2f\n", pi * D * D / 4.0);
7     return 0;
8 }
```

2. ARM 汇编程序

圆的周长与面积 ARM 汇编程序

```
1 .arch armv7-a
2 .fpu vfpv3-d16
3 .eabi_attribute 28, 1
4 .eabi_attribute 20, 1
5 .eabi_attribute 21, 1
6 .eabi_attribute 23, 3
7 .eabi_attribute 24, 1
8 .eabi_attribute 25, 1
9 .eabi_attribute 26, 2
10 .eabi_attribute 30, 6
11 .eabi_attribute 34, 1
12 .eabi_attribute 18, 4
13 @ 数据段
14 .data
15 .align 1
16 pi: .float 3.14
17 D: .float 2.0
18 four: .float 4.0
19 perimeter_format: .asciz "perimeter: %.2f\n"
20 area_format: .asciz "area: %.2f\n"
21 @ 代码段
22 .text
23 .align 1
24 .global main
25 .type main, %function
26 main:
```

```

27  @ 保存返回地址
28  push {fp, lr}
29  @ 加载pi和D的值
30  ldr r0, =pi
31  ldr r1, =D
32  vldr.32 s0, [r0] @ 将pi加载到s0
33  vldr.32 s1, [r1] @ 将D加载到s1
34  @ 计算周长
35  vmul.f32 s2, s0, s1 @ s2 = s0 * s1
36  @ 输出周长
37  ldr r0, =perimeter_format @ 准备printf的参数r0
38  vcvtf64.f32 d7, s2
39  vmov r2, r3, d7
40  bl printf
41  @ 加载pi和D的值
42  ldr r0, =pi
43  ldr r1, =D
44  vldr.32 s0, [r0] @ 将pi加载到s0
45  vldr.32 s1, [r1] @ 将D加载到s1
46  @ 计算面积
47  vmul.f32 s3, s1, s1 @ 计算D*D
48  vmul.f32 s3, s3, s0 @ 计算pi*D*D
49  ldr r0, =four @ 加载4.0的地址到r0中
50  vldr s4, [r0] @ 将4.0加载到s4中
51  vdiv.f32 s3, s3, s4 @ 计算pi*D*D/4
52  @ 输出面积
53  ldr r0, =area_format @ 准备printf的参数r0
54  vcvtf64.f32 d7, s3
55  vmov r2, r3, d7
56  bl printf
57  mov r0, #0 @ 设置返回值为0
58  @ 恢复堆栈并返回
59  pop {fp, pc}
60  @ 函数原型
61  .extern printf
62  .section .note.GNU-stack,"",%progbits

```

Makefile

```

1  .PHONY: debug,gdb,out,port
2  SOURCENAME=./circle.s
3  EXENAME=./circle
4  debug:
5      arm-linux-gnueabi-gcc $(SOURCENAME) -o $(EXENAME) -static -g
6      qemu-arm -g 1234 $(EXENAME)
7  gdb:
8      arm-linux-gnueabi-gdb -ex 'target remote localhost:1234' $(EXENAME)
9  out:
10     arm-linux-gnueabi-gcc $(SOURCENAME) -o $(EXENAME) -static -g

```

```

11 port:
12     lsof -i:1234
13 run:
14     $(EXENAME)

```

利用已经编写好的 Makefile 文件, 使用 make out 指令得到可执行文件, 接着借助 make run 指令运行, 如图1所示, 可以看到程序正确运行, 输出了正确结果。

此处编写时主要遇到的问题时 printf 函数的传参问题。我们配置了可以用于单步调试的 arm-gdb 进行寄存器值的查询, 再通过查询相关文档及代码, 我们成功解决了单精度浮点数的传参问题, 输出正确结果。

```

● lxmlu2002@lxmlu2002-Ubuntu:~/compiler/lab2/SysY$ make run
./circle
perimeter: 6.28
area: 3.14

```

图 1: 圆的周长与面积 ARM 汇编程序运行结果

在圆的周长与面积程序编写中, 我们使用到了单精度浮点数据类型、变量与常量的声明、数值运算等 SysY 语言特性。

(二) 斐波那契数列

1. SysY 程序

斐波那契数列 SysY 程序

```

1 #include <stdio.h>
2 const int n = 5;
3 int main() {
4     int f[5] = {1, 1, 0, 0, 0};
5     int i = 0;
6     while (i < n) {
7         if (i < 2) {
8             i = i + 1;
9         }
10        else {
11            f[i] = f[i - 1] + f[i - 2];
12            i = i + 1;
13        }
14    }
15    printf("f[5]=%d\n", f[4]);
16    return 0;
17 }

```

2. ARM 汇编程序

斐波那契数列 ARM 汇编程序

```

1  .arch armv7-a
2  .text
3  .global n
4  .section .rodata
5  .align 2
6  n:
7  .word 5
8  .align 2
9  _str0:
10 .ascii "f[5]=%d\n"
11 .align 2
12 .text
13 .global main
14 .type main, %function
15 main:
16     push {fp, lr}
17     mov fp, sp
18     add r7, sp, #0
19     @ 初始化数组和 i
20     sub sp, sp, #24
21     add r3, sp, #4
22     mov r4, #1
23     str r4, [fp, #-20]
24     str r4, [fp, #-16]
25     mov r4, #0
26     str r4, [fp, #-12]
27     str r4, [fp, #-8]
28     str r4, [fp, #-4]
29     ldr r4, [fp, #-24]
30     ldr r6, _bridge
31     ldr r6, [r6]
32 .L0: @ while 循环
33     cmp r6, r4 @ r6 为 n , r4 为 i
34     blt .L3
35 .L1: @ if
36     cmp r4, #1
37     bgt .L2
38     add r4, r4, #1
39     b .L1
40 .L2: @ else
41     sub r0, r4, #1
42     lsls r0, r0, #2
43     add r0, r3
44     ldr r1, [r0] @ r1 = f[i-1]
45     sub r0, r4, #2
46     lsls r0, r0, #2
47     add r0, r3

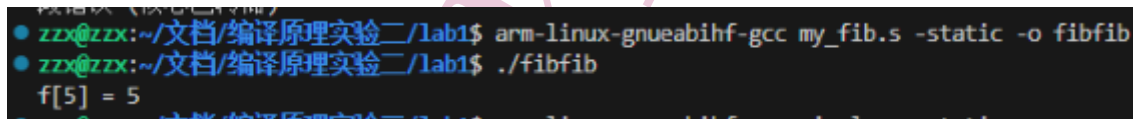
```

```

48     ldr r2, [r0] @ r2 = f[i-1]
49     add r1, r2 @ r1 = r1 + r2
50     mov r0, r4
51     lsls r0, r0, #2
52     add r0, r3
53     str r1, [r0]
54     add r4, r4, #1
55     b .L0
56 .L3: @ 输出结果
57     ldr r0, _bridge+4
58     ldr r1, [sp, #20]
59     bl printf
60     mov sp, r7
61     mov r0, #0
62     pop {fp, pc}
63 .L4:
64     .align 2
65 _bridge:
66     .word n
67     .word _str0
68     .section .note.GNU-stack,"",%progbits

```

利用 `arm-linux-gnueabi-gcc my_fib.s -static -o fibfib` 指令得到可执行文件,接着利用 `./fibfib` 指令运行,如图2所示,可以看到程序正确运行,输出了 `f[5] = 5` 的正确结果。



```

ZZX@ZZX:~/文档/编译原理实验二/lab1$ arm-linux-gnueabi-gcc my_fib.s -static -o fibfib
ZZX@ZZX:~/文档/编译原理实验二/lab1$ ./fibfib
f[5] = 5

```

图 2: 斐波那契数列 ARM 汇编程序运行结果

在斐波那契数列程序编写中,我们使用到了数组、变量与常量的声明、if 条件判断、while 循环、数值运算、参数赋值等 SysY 语言特性。

四、 思考题

词法分析 (Lexical Analysis):

- 设计词法分析器, 将 SysY 源代码分解成词法单元 (tokens), 如标识符、关键字、运算符、常量等。
- 使用有限状态自动机 (Finite State Automaton) 或正则表达式等工具来识别和生成词法单元。
- 构造符号表 (Symbol Table), 用 Key-Value 的形式来表示词法单元, 例如词法单元的类型和值。

语法分析 (Syntax Analysis):

- 使用上下文无关文法 (Context-Free Grammar) 来定义 SysY 语言的语法规则。
- 设计语法分析器, 将词法单元组织成语法树 (或抽象语法树, AST)。
- 使用递归下降分析、LL(1) 分析器等技术来实现语法分析器。
- 在语法树中记录语法结构的信息, 如表达式的运算符优先级、控制流结构等。

语义分析 (Semantic Analysis):

- 设计语义分析器, 检查语法树上的语义错误, 并构建符号表。
- 符号表用于跟踪变量、函数、类型等的声明信息。
- 进行类型检查, 确保表达式和操作的类型匹配。
- 检查变量的作用域、函数的参数传递、数组访问越界等语义规则。

中间表示 (Intermediate Representation, IR):

- 设计中间表示 (IR), 用于在语法分析和代码生成之间传递信息。
- IR 可以是三地址码、抽象语法树、静态单赋值 (SSA) 形式等。
- 中间表示应包括操作数、操作符和控制流信息。

代码生成 (Code Generation):

- 设计代码生成器, 将中间表示翻译成目标汇编语言或机器码。
- 根据目标架构生成汇编指令。
- 实施寄存器分配、指令调度、循环优化、内联函数等代码优化技术。
- 处理函数调用、栈操作、内存访问等底层细节。

目标代码输出:

- 使用汇编指令的结构体来表示目标代码。