



南開大學  
Nankai University

南 开 大 学  
网 络 空 间 安 全 学 院  
编译原理实验报告

---

第二次作业

---

付文轩 1911410 信息安全  
马思远 1911452 信息安全与法学  
年级：2019 级  
指导教师：王刚

2021 年 10 月 13 日

## 摘要

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

**关键字：**sysY 语言，上下文无关文法，sysY 程序，ARM 汇编程序

## 目录

一、 sysY 语言文法	1
(一) 基础声明和定义部分	1
(二) 运算部分	1
(三) 函数部分	2
(四) 部分终结符特征	2
二、 程序实例	2
(一) 第一个程序示例	2
1. sysY 语言程序	2
2. ARM 汇编程序	3
3. 实验结果	4
(二) 第二个程序示例	4
1. sysY 语言程序	4
2. ARM 汇编程序	5
3. 实验结果	8
三、 总结	8
四、 实验分工	8
(一) 付文轩	8
(二) 马思远	9

## 一、 sysY 语言文法

对于如下文法定义，需要注意有如下表示方法：

1. 终结符由黑体或者单引号括起标识出
2. 非终结符为斜体或者非黑体（有的地方斜体不明显或没有显示）
3. 文字以符号//开头为注释

定义的文法表示如下，其中 *CompUnit* 为开始符号：

编译单元  $CompUnit \rightarrow CompUnitDecl \mid CompUnitFuncDef \mid Decl \mid FuncDef$

### (一) 基础声明和定义部分

支持的类型:  $type \rightarrow int$  (1)

可以改变的左值  $lval \rightarrow id \mid id\{[ expr ]\}$  (2)

一元表达式  $unary\_expr \rightarrow pri\_exp \mid id(' func\_tps ')\mid unary\_op unary\_expr$  (3)

基本表达式  $pri\_exp \rightarrow ' ( expr ')\mid lval \mid int$  (4)

单目运算符  $unary\_op \rightarrow ' + ' \mid ' - ' \mid ' ! ' (5)$

常数声明  $ConstDecl \rightarrow const type ConstDef \mid ConstDecl, ConstDef$  (6)

常数定义  $ConstDef \rightarrow id\{ '[ const\_expr ] ' \} ' = ' ConstInitVal$  (7)

常数初值  $ConstInitVal \rightarrow const\_expr \{ \{ \} \} \{ InitValList \}$  (8)

变量声明  $VarDecl \rightarrow type VarDef \mid VarDecl, VarDef$  (9)

变量定义  $ConstDef \rightarrow id\{ [ const\_expr ] \} \mid id\{ [ const\_expr ] \} ' = ' ConstInitVal$  (10)

变量初值  $InitVal \rightarrow expr \{ \{ \} \} \{ InitValList \}$  (11)

变量初值列表  $InitValList \rightarrow const\_expr \mid InitValList, const\_expr$  (12)

### (二) 运算部分

运算表达式:  $expr \rightarrow add\_expr$  (13)

逻辑表达式  $cond \rightarrow lor\_expr$  (14)

或运算  $lor\_expr \rightarrow land\_expr \mid lor\_expr \parallel land\_expr$  (15)

与运算  $land\_expr \rightarrow eql\_expr \mid land\_expr \&\& eql\_expr$  (16)

等于运算  $eql\_expr \rightarrow rel\_expr \mid eql\_expr ( == \mid != ) rel\_expr$  (17)

关系运算  $rel\_expr \rightarrow add\_expr \mid rel\_expr ( < \mid > \mid <= \mid >= ) add\_expr$  (18)

加法运算  $add\_expr \rightarrow mul\_expr \mid add\_expr ( + \mid - ) mul\_expr$  (19)

乘法运算  $mul\_expr \rightarrow unary\_expr \mid mul\_expr ( * \mid / \mid \% ) unary\_expr$  (20)

常表达式  $const\_expr \rightarrow add\_expr$  (21)

(22)

## (三) 函数部分

函数声明  $func\_def \rightarrow func\_type\ id\ ('func\_fps')\ block$  (23)

函数返回值类型  $func\_def \rightarrow void \mid int$  (24)

函数参数  $func\_fps \rightarrow func\_fp \mid func\_fp\ ',\ func\_fps$  (25)

单个参数  $func\_fp \rightarrow type\ id \mid type\ id\ '['\ ']' \mid func\_fp\ '['\ expr']'$  (26)

代码块  $block \rightarrow \{'blockitem'\} \mid \{'\ '\}$  (27)

代码块中的项  $blockitem \rightarrow Decl \mid Stmt$  (28)

声明  $Decl \rightarrow ConstDecl \mid VarDecl$  (29)

语句  $Stmt \rightarrow labl\ '='\ pri\_exp\ ';' \mid '['\ pri\_exp\ ']' ';' \mid block$  (30)

$\rightarrow if\ ('cond\ ')\ Stmt\ else\ Stmt$  (31)

$\rightarrow if\ ('cond\ ')\ Stmt \mid while\ ('cond\ ')\ Stmt$  (32)

$\rightarrow break\ ';' \mid return\ exp\ ';'$  (33)

$\rightarrow return\ ';' \mid continue\ ';' \mid \epsilon$  (34)

实参  $func\_rps \rightarrow func\_rps, cond \mid expr$  (35)

## (四) 部分终结符特征

*id*

*id* 的规范如下 (identifier)

$$\begin{aligned} identifier &\rightarrow identfitier - nondigit \\ &\mid identifier\ identfitier - nondigit \\ &\mid identfitierdigit \end{aligned}$$

其中 *identfitier-nondigit* 为下划线或者  $[a-zA-Z]$  //这里使用了正则表达式来表示

*number*

*number* 的规范如下 (identifier)

$$number \rightarrow -?([1-9] * [0-9]?) * [1-9]$$

//这里使用了正则表达式来表示

## 二、 程序实例

## (一) 第一个程序示例

## 1. sysY 语言程序

test1.sy

```
1 #include<sylib.h>
2
```

```

3  int testData = 3;
4  const int enter = 10;
5
6  int matrixAdd(int x){
7      int matrix1[2] = {1,2};
8      int matrix2[2] = {0,1};
9      int matrixResult[2] = {0,0};
10     matrix2[0] = x;
11     int indexForX1 = 0;
12     while(indexForX1 < 2){
13         matrixResult[indexForX1] = matrixResult[indexForX1] + matrix1[
            indexForX1] + matrix2[indexForX1];
14         indexForX1 = indexForX1 + 1;
15     }
16
17     int finalResult;
18     finalResult = matrixResult[0] + matrixResult[1];
19     return finalResult;
20 }
21
22 void main(){
23     int n;
24     n = getint();
25     int result;
26     result = matrixAdd(testData);
27     result = result - n;
28     putint(result);
29     putchar(enter);
30 }

```

本程序的功能：输入一个整数，输出结果是数组内部所有数求和后减去输入整数得到的数值，其中数组内部的数求和后是一个定值（因为调用函数时使用的参数是固定值）

这个 sysY 程序涉及到的 sysY 语言特性有：全局变量的声明、常量的声明、函数声明、函数调用、参数传递、数组定义、算数运算、对 sysY 库的调用等。

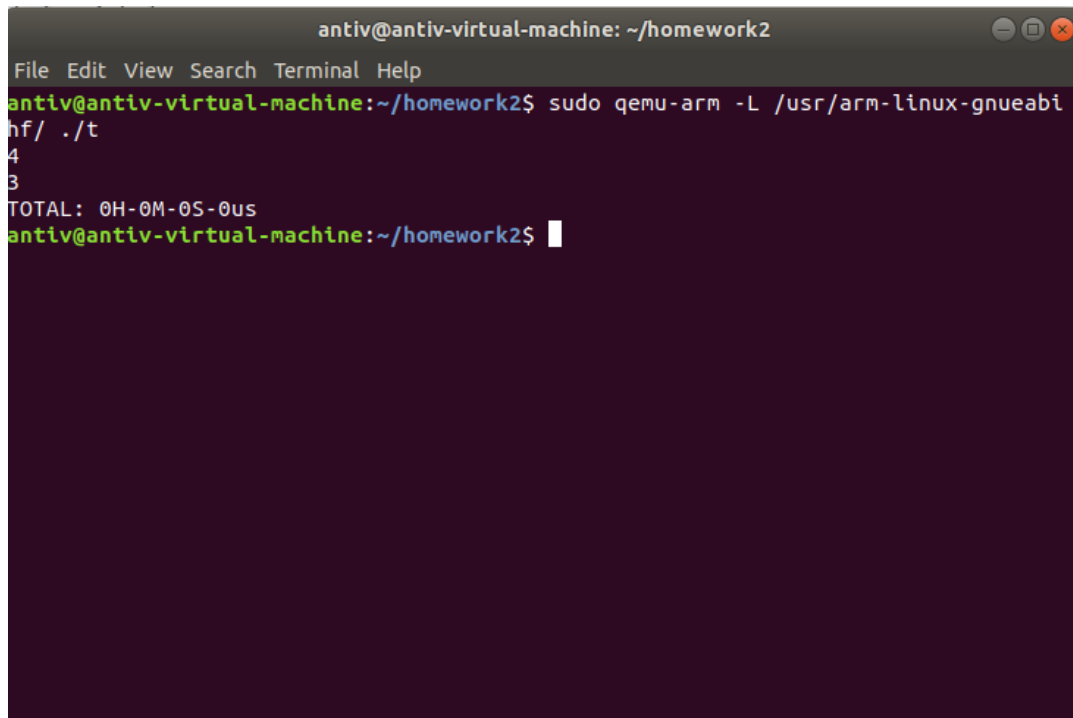
## 2. ARM 汇编程序

由于 ARM 汇编代码较长，这里只放 gitlab 中的链接，其中.S 文件即为 ARM 汇编代码源程序 <https://gitlab.eduxiji.net/nku2021-zjdoudui/byyl/-/tree/master/lab2/codeForF>

这个汇编代码还有比较大的优化空间，如其中有一些寄存器其实可以进行重复使用，有的在存放到栈空间后直接跟着就是读取操作，尤其是在之后相关内存地址上的内容并没有涉及到其他操作，那么在这一步中的存取就是一些冗余的操作，可以考虑忽略存取，直接使用寄存器内部的值。

同时需要注意，在使用 linux 中的 arm gcc 和 qemu 之前，需要将官方使用的 sysY 库文件 [1] 放到对应文件夹下，并指出编译时使用的文件位置。其中.c 和.h 文件需要放到/usr/include/下，.a 和.so 文件需要放到/usr/lib/目录下；使用 gcc 时也需要指出相关位置，可以使用的指令为：arm-linux-gnueabi-gcc test.c -o t -I /usr/include/ -L /usr/lib/ -lsysy

## 3. 实验结果



```
antiv@antiv-virtual-machine: ~/homework2
File Edit View Search Terminal Help
antiv@antiv-virtual-machine:~/homework2$ sudo qemu-arm -L /usr/arm-linux-gnueabi
hf/ ./t
4
3
TOTAL: 0H-0M-0S-0us
antiv@antiv-virtual-machine:~/homework2$
```

图 1: 程序示例 1 结果

从实验结果中可以看见输入的是 4，输出的是 3，结果正确。

## (二) 第二个程序示例

## 1. sysY 语言程序

冒泡排序算法

```
1 #include <stdio.h>
2 void DataSwap(int* data1, int* data2)
3 {
4     int temp = *data1;
5     *data1 = *data2;
6     *data2 = temp;
7 }
8
9
10
11 int main()
12 {
13     int pDataArray[10] = { 3,4,6,7,1,2,8,9,5,0 };
14     for (int i = 0; i < 10- 1; i++)    // 走iDataNum-1趟
15         for (int j = 0; j < 10- i - 1; j++)
16             if (pDataArray[j] > pDataArray[j + 1])
17                 DataSwap(&pDataArray[j], &pDataArray[j + 1]);
```

```
18     for (int i = 0; i < 10; i++)
19         printf("%d ", pDataArray[i]);
20 }
```

本程序的功能非常直观，大家都是清楚的。就是一个最简单的冒泡排序的算法，是所有排序算法当中最简单的一个算法，本质上就是实现一个双重循环。在实现冒泡排序之外，我还实现了一个c风格的数据交换函数，利用指针进行传参。这个程序本身非常简单，但是当写成汇编代码的时候还是比较困难的，尤其是汇编代码的循环结构看起来对人眼非常不友好。所以调试的时候耗费了很多时间。

这个 sysY 程序涉及到的 sysY 语言特性有：数组的声明、常量的声明、函数声明、函数调用、参数传递、算数运算、循环结构、选择结构等。

## 2. ARM 汇编程序

### 冒泡排序算法汇编

```
1     .arch armv5t
2     .data
3
4     .section .rodata
5     .align 2
6 .LC0:
7     .string "val = %d\n"
8
9     .section .text
10    .align 2
11
12
13    .global DataSwap
14
15 DataSwap:
16    push {r11}
17    add r11,sp,#0
18    sub sp,sp,#12
19    //start
20    ldr r2,[r0]
21    ldr r3,[r1]
22    str r2,[r1]
23    str r3,[r0]
24    //end
25    add sp,r11,#0
26    pop {r11}
27    bx lr
28
29    .global main
30
31 main:
32    push {r11, lr}
```

```
33      add    r11, sp, #0
34      sub    sp, sp, #16
35      //start
36
37      mov r0,#3      //array initial
38      push {r0}
39      mov r0,#4
40      push {r0}
41      mov r0,#6
42      push {r0}
43      mov r0,#7
44      push {r0}
45      mov r0,#1
46      push {r0}
47      mov r0,#2
48      push {r0}
49      mov r0,#8
50      push {r0}
51      mov r0,#9
52      push {r0}
53      mov r0,#5
54      push {r0}
55      mov r0,#0
56      push {r0}
57
58      mov r4,sp      //array adress
59
60      mov r5,#0
61 loop1:
62      cmp r5,#9
63      bge end1
64
65      mov r6,#0
66 loop2:
67      mov r1,#9
68      sub r0,r1,r5
69      cmp r6,r0
70      bge end2
71      add r0,r4,r6,LSL #2
72      add r1,r0,#4
73      push {r0,r1}
74      ldr r0,[r0]
75      ldr r1,[r1]
76      cmp r0,r1
77      ble endif
78      pop {r1,r0}
79      bl DataSwap
80
```



```
81 endif:
82     //end loop2
83     add r6,r6,#1
84     b loop2
85
86 end2:
87     //end loop1
88     add r5,r5,#1
89     b loop1
90
91
92
93
94 end1:
95     mov r1,#0
96 looppr:
97     cmp r1,#10
98     bge endpr
99     push {r1}
100    add r1,r4,r1,LSL #2
101
102    ldr r1,[r1]
103    ldr r0,=.LC0
104    bl printf
105    pop {r1}
106    add r1,r1,#1
107    b looppr
108
109 endpr:
110    //end
111    sub sp, r11, #0
112    pop {r11, pc}
```

这是第一次学着写 arm 汇编代码，中间遇到了很多的问题，首先在标准化输出的时候就遇到了非常多的问题。在一开始连循环输出都做不到。但是在经过查阅资料之后还是解决了大部分的问题。即使这样代码的性能也还是很差，例如在数组定义时直接把所有的数据都压入了栈中，显得非常不美观。然后在寄存器的使用选取上也没有经过优化，经常有寄存器的重复使用。

另一方面与机器生成的代码进行对比之后发现二者之间的差别还是很大。机器生成的代码效率很高但是可读性很差。虽然我的代码效率不好但是可读性尚且。还是应当多多学习机器生成代码的规律来进行接下来的工作。

### 3. 实验结果

```
msy@msy-virtual-machine:~/lab/lab2$ qemu-arm a.out
val = 0
val = 5
val = 9
val = 8
val = 1
val = 2
val = 7
val = 6
val = 3
val = 4
```

图 2: 排序前

这张是对数组的直接输出，是排序之前的结果，以做对比

```
msy@msy-virtual-machine:~/lab/lab2$ qemu-arm a.out
val = 0
val = 1
val = 2
val = 3
val = 4
val = 5
val = 6
val = 7
val = 8
val = 9
```

图 3: 排序前

这是经过冒泡排序之后的输出，可以看到经过排序数组的输出顺序正确了，结果正确。

## 三、 总结

本次实验我们做了：

1. 使用了上下文无关文法设计了想要实现的 sysY 语言部分特性
2. 了解并尝试编写了 ARM 汇编语言的相关代码
3. 通过以上的我们对于编译器的运作有了更加深刻认识。虽然只是人来翻译，但是却在更高的层次上了解了编译器的底层机制，以及翻译方法。也许目前的我们还不能实现一个真正的编译器，但是这次实验还是让我们有了一个深入思考编译器工作方式的机会。

## 四、 实验分工

两个人共同讨论、编写了上下文无关文法，并分别设计、编写了自己的 sysY 语言代码和 ARM 汇编语言代码

### (一) 付文轩

1. 编写实验报告中文法设计的函数部分、基础声明和定义部分（常数定义之前）、部分终结符特征

2. 设计并编写了第一个程序示例代码，并编写相关报告

## (二) 马思远

1. 编写实验报告中文法设计的运算、基础声明和定义部分（常数定义之后）
2. 设计并编写了第二个程序示例代码，并编写相关报告

NIJU

## 参考文献

- [1] 2020. <https://gitlab.eduxiji.net/windcome/sysruntime/library/-/tree/master>.

NIKU