南開大學
Nankai University

南　　开　　大　　学

计　算　机　学　院

编译原理实验报告

预备工作二——定义编译器 & 汇编编程

王娇妹 2012679

王浩 2013287

年级：2020 级

专业：计算机科学与技术

指导教师：王刚

2022 年 10 月 16 日

目录

一、 实验内容

(一) 实验描述

基于“预备工作 1”，继续:

1. 确定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的 SysY 语言子集。

2. 设计 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) 分工

对于 Sysy 语言特性的上下文无关文法定义，王娇妹同学写了文法的第 1、2、3 部分，王浩同学写了文法的第 4、5 部分，并且双方经过交流讨论，解决了错误部分，汇总在一起。

对于 arm 汇编，王娇妹和王浩同学共同编写了一个包含了全局变量、int 型有参返回值函数、break 语句、if 语句、while 语句、return 语句、加减乘除取模的算数运、== 和！= 的比较运算的 C 程序，然后将它翻译为 arm 汇编。并且王娇妹负责了手动编写 arm 汇编程序与编译器生成的 arm 汇编程序之间的比较工作，并对不同版本的 arm 架构下指令集的差异做了简单分析与说明。

大家各自撰写自己完成部分的 LaTex 代码，并由王浩完成最后的排版。笔者（王浩）认为，此次实验中王娇妹与王浩的贡献比重大约为 7：3。

二、 sysy 语言特性的上下文无关文法定义

(一) 终结符集合 V_T

终结符 V_T 包括 **Ident**、**IntConst**、**floatConst**，以及所有由单引号括起来的串。

(二) 非终结符集合 V_N

编译单元 CompUnit
声明 Decl
常量声明 ConstDecl
基本类型 BType
常数定义 ConstDef
常量初值 ConstInitVal
变量声明 VarDecl
变量定义 VarDef
变量初值 InitVal
函数定义 FuncDef
函数类型 FuncType
函数形参表 FuncFParams
函数形参 FuncFParam
语句块 Block

语句块项 BlockItem
语句 Stmt
表达式 Exp
条件表达式 Cond
左值表达式 LVal
基本表达式 PrimaryExp
数值 Number
一元表达式 UnaryExp
单目运算符 UnaryOp
函数实参表 FuncRParams
乘除模表达式 MulExp
加减表达式 AddExp
关系表达式 RelExp
相等性表达式 EqExp
逻辑与表达式 LAndExp
逻辑或表达式 LOrExp
常量表达式 ConstExp

(三) 开始符号 S

编译单元 CompUnit

(四) 产生式集合 P

以下按照语言特性分别给出该语言各个部分的产生式。

1. 编译单元

编译单元

$$
\begin{aligned}
CompUnit \rightarrow\ & CompUnit\ Decl \\
\mid\ & CompUnit\ FuncDef \\
\mid\ & Decl \\
\mid\ & FuncDef
\end{aligned}
$$

声明语句

$$
\begin{aligned}
Decl \rightarrow\ & ConstDecl\ \text{';'} \\
\mid\ & VarDecl\ \text{';'}
\end{aligned}
$$

2. 常量与变量

常量/变量基本类型

$$
\begin{aligned}
BType \rightarrow\ & \text{'int'} \\
\mid\ & \text{'float'}
\end{aligned}
$$

常量声明

$$
\begin{aligned}
ConstDecl \rightarrow\ & \text{'const'}\ BType\ ConstDef \\
|\ & ConstDecl\ \text{','}\ ConstDef
\end{aligned}
$$

常数定义

$$
\begin{aligned}
ConstDef \rightarrow\ & \textbf{Ident}\{'['ConstExp']'\}\ \text{'='}\ ConstInitVal \\
|\ & \text{'float'}
\end{aligned}
$$

常量初值

$$
\begin{aligned}
ConstInitVal \rightarrow\ & ConstExp \\
|\ & \text{'\{' '\}'} \\
|\ & \text{'\{'}InitValList\text{'\}'} \\
InitValList \rightarrow\ & InitValList\ \text{','}\ ConstExp \\
|\ & ConstExp
\end{aligned}
$$

变量声明

$$
\begin{aligned}
VarDecl \rightarrow\ & BType\ VarDef \\
|\ & VarDecl\ \text{','}\ VarDef
\end{aligned}
$$

变量定义

$$
\begin{aligned}
VarDecl \rightarrow\ & \textbf{Ident}\ \{\text{'['}\ ConstExp\ \text{']'}\} \\
|\ & \textbf{Ident}\ \{\text{'['}\ ConstExp\ \text{']'}\}\ \text{'='}\ InitVal \\
|\ &
\end{aligned}
$$

变量初值

$$
\begin{aligned}
InitVal \rightarrow\ & Exp \\
|\ & \text{'\{' '\}'} \\
|\ & \text{'\{'}[\ InitVal\ \{\ \text{','}\ InitVal\ \}\ ]\text{'\}'}
\end{aligned}
$$

这里的变量初值不需要是常量值。

3. 函数

函数定义

$$
\begin{aligned}
FuncDef \rightarrow\ & FuncType\ \textbf{Ident}\ \text{'('}\ \text{')'}\ Block \\
|\ \ & FuncType\ \textbf{Ident}\ \text{'('}\ FuncFParams\ \text{')'}\ Block
\end{aligned}
$$

函数类型

$$
\begin{aligned}
FuncType \rightarrow\ & \text{'void'} \\
|\ \ & \text{'int'} \\
|\ \ & \text{'float'}
\end{aligned}
$$

函数形参表

$$
\begin{aligned}
FuncFParams \rightarrow\ & FuncFParam \\
|\ \ & FuncFParams\ \text{','}\ FuncFParam
\end{aligned}
$$

函数形参

$$
\begin{aligned}
FuncFParams \rightarrow\ & Btype\ \textbf{Ident} \\
|\ \ & Btype\ \textbf{Ident}\ \text{'['}\ \text{']'} \\
|\ \ & FuncFParam\ \text{'['}\ Exp\ \text{']'} \\
|\ \ & \epsilon
\end{aligned}
$$

4. 语句

语句块

$$Block \rightarrow \text{'\{'}\ \ \{\ \text{BlockItem}\ \}\ \text{'\}'}$$

语句块项

$$
\begin{aligned}
BlockItem \rightarrow\ & Decl \\
|\ \ & Stmt
\end{aligned}
$$

语句

$$
\begin{aligned}
Stmt \rightarrow\ & \epsilon \\
|\ & Lval\ \text{'='}\ Exp\ \text{';'} \\
|\ & Exp\ \text{';'} \\
|\ & Block \\
|\ & \text{'if'}\ \text{'('}\ Cond\ \text{')'}\ Stmt \\
|\ & \text{'if'}\ \text{'('}\ Cond\ \text{')'}\ Stmt\ \text{'else'}\ Stmt \\
|\ & \text{'while'}\ \text{'('}\ Cond\ \text{')'}\ Stmt \\
|\ & \text{'break'}\ \text{';'} \\
|\ & \text{'continue'}\ \text{';'} \\
|\ & \text{'return'}\ \text{';'} \\
|\ & \text{'return'}\ Exp\ \text{';'}
\end{aligned}
$$

5. 表达式

表达式

$$Exp \rightarrow AddExpr$$

条件表达式

$$Cond \rightarrow LOrExp$$

左值表达式

$$
\begin{aligned}
Lval \rightarrow\ & \textbf{Ident} \\
|\ & Lval\ \text{'['}\ Exp\ \text{']'}
\end{aligned}
$$

基本表达式

$$
\begin{aligned}
PrimaryExp \rightarrow\ & \text{'('}\ Exp\ \text{')'} \\
|\ & Lval \\
|\ & Number
\end{aligned}
$$

数值

$$Number \rightarrow \textbf{IntConst}$$

一元表达式

$$
\begin{aligned}
UnaryExp \rightarrow\ & PrimaryExp \\
|\ & \textbf{Ident}\ \text{'('}\ \text{')'} \\
|\ & \textbf{Ident}\ \text{'('}\ FuncRParams\ \text{')'} \\
|\ & UnaryOp\ UnaryExp
\end{aligned}
$$

单目运算符

$$
\begin{aligned}
UnaryOp \rightarrow\ & \text{'+'} \\
|\ & \text{'-'} \\
|\ & \text{'!'}
\end{aligned}
$$

函数实参表

$$
\begin{aligned}
FuncRParams \rightarrow\ & Exp \\
|\ & FuncRparams\ \text{','}\ Exp
\end{aligned}
$$

乘除模表达式

$$
\begin{aligned}
MulExp \rightarrow\ & MulExp\ \text{'\%'}\ UnaryExp \\
|\ & MulExp\ \text{'*'}\ UnaryExp \\
|\ & MulExp\ \text{'/'}\ UnaryExp
\end{aligned}
$$

加减表达式

$$
\begin{aligned}
AddExp \rightarrow\ & MulExp \\
|\ & AddExp\ \text{'+'}\ MulExp \\
|\ & AddExp\ \text{'-'}\ MulExp
\end{aligned}
$$

关系表达式

$$
\begin{aligned}
RelExp \rightarrow\ & AddExp \\
|\ & RelExp\ \text{'<'}\ AddExp \\
|\ & RelExp\ \text{'<='}\ AddExp \\
|\ & RelExp\ \text{'>='}\ AddExp
\end{aligned}
$$

相等性表达式

$$
\begin{aligned}
EqExp &\rightarrow RelExp \\
&\mid EqExp \text{ '==' } RelExp \\
&\mid EqExp \text{ '!=' } RelExp
\end{aligned}
$$

逻辑表达式

$$
\begin{aligned}
LAndExp &\rightarrow EqExp \\
&\mid LAndExp \text{ '\&\&' } EqExp \\
LOrExp &\rightarrow LAndExp \\
&\mid LOrExp \text{ '||' } LAndExp
\end{aligned}
$$

常量表达式

$$ConstExp \rightarrow AddExp$$

三、 arm 汇编

此 test.c 程序包含了全局变量、int 型有参返回值函数，break 语句，if 语句，while 语句，return 语句，加减乘除取模的算数运算，== 和！= 的比较运算。

test

```c
#include<stdio.h>
int num = 60;
int avg(int a, int b) {
    if (a == b)
        return a;
    int sum = a + b;
    sum = sum / 2;
    return sum;
}

int main() {
    int x = 11;
    int y = 6;
    x = x * y;
    int avg_x_y = avg(x, y);
    while (avg_x_y != num) {
    if (avg_x_y % num == 0)
        break;
    num = num - 1;
    }
    printf("%d\n", num);
```

```
22      return 0;
23  }
```

在学习和翻译 arm 汇编代码的过程中，我发现实验文档提供的 arm 架构学习链接、实验文档中给出的代码示例、以及我机器上编译产生的 arm 汇编代码，在很多地方有差异。比如 arm 架构学习网站中使用 r11 寄存器作为栈帧寄存器，代码示例直接使用 fp，而我机器上是使用 r7 作为栈帧寄存器。

经过查阅资料，这些差异与指令集架构、使用的编译器等有关。代码示例是基于 armv5t 架构的，而我在编译 C 代码时采用了默认架构 armv7-a。此外，arm 中不同的交叉编译器也有差别，在函数执行开头的处理和对栈帧寄存器的使用等方面有差异，例如 arm-linux-gnueabihf-gcc 用的 r7 和 arm-linux-gnueabi-gcc 用的 r11，我使用的是前者。

我使用 arm-linux-gnueabihf-gcc 交叉编译器，架构 armv7-a，根据 C 代码生成 arm 汇编代码，对编译生成的 arm 汇编代码进行了阅读分析，然后在此基础上重新将 C 代码手动翻译成 arm 汇编代码。

对比我翻译的汇编和电脑执行命令产生的汇编，发现实际编译产生的代码比我写的要多一些，而且还有一些看起来似乎没有意义的片段。

例如，根据上面 C 代码，avg 函数中，如果 a 和 b 不相等，就要取平均值，这里需要跳转到.L2 进行求平均值操作。

我翻译的 arm 汇编代码

```
1  .L2:
2          add     r3, r3, r2
3          asr     r3, #1
```

在第 5、6 行，将 r3 寄存器中的值存到 r7 寄存器中的地址（偏移量为 12）中，然后又取出来放回 r3 中，并且后面也没有用到 [r7, #12] 中存储的数据的地方。在 10、11 行，又进行了一次这样的操作。

电脑编译的 arm 汇编代码

```
1  .L2:
2          ldr     r2, [r7, #4]
3          ldr     r3, [r7]
4          add     r3, r3, r2
5          str     r3, [r7, #12]
6          ldr     r3, [r7, #12]
7          lsrs    r2, r3, #31
8          add     r3, r3, r2
9          asrs    r3, r3, #1
10         str     r3, [r7, #12]
11         ldr     r3, [r7, #12]
```

以下是我手动翻译 test.c 程序的全部 arm 汇编代码：

手动翻译的 arm 汇编

```
1          .text
2          .global num @全局变量num
3          .data
4          .align  2
```

```
5         .type   num, %object
6         .size   num, 4
7 num:
8         .word   60 @num = 60
9         .text
10        .align  1
11        .global avg @函数avg
12        .syntax unified
13        .type   avg, %function
14 avg:
15        push    {r7} @r7是栈帧寄存器
16        sub     sp, sp, #20 @为局部变量分配空间
17        add     r7, sp, #0 @r7存储sp
18        str     r0, [r7, #4] @参数a, [r7, #4] = r0
19        str     r1, [r7] @参数b, [r7] = r1
20        ldr     r2, [r7, #4] @参数a
21        ldr     r3, [r7] @参数b
22        cmp     r2, r3 @比较两个参数ab大小
23        bne     .L2 @不相等,跳转到.L2
24        ldr     r3, [r7, #4]
25        b       .L3 @跳转到.L3
26 .L2: @a和b不相等, 求平均值
27        add     r3, r3, r2 @求和, sum = a + b
28        asr     r3, #1 @算数右移1位, 除以二
29 .L3:
30        mov     r0, r3 @r0做返回值
31        add     r7, #20 @r7 = r7 + 20
32        mov     sp, r7 @sp = r7, 恢复sp
33        ldr     r7, [sp], #4 @r7 = [sp + #4], sp变化, +#4
34        bx      lr
35        .size   avg, .-avg
36        .global __aeabi_idivmod
37        .section        .rodata
38        .align  2
39 .LC0:
40        .ascii  "%d\012\000" @字符串, printf函数中用到
41        .text
42        .align  1
43        .global main
44        .syntax unified
45        .thumb
46        .thumb_func
47        .fpu vfpv3-d16
48        .type   main, %function
49 main:
50        push    {r7, lr} @r7是栈帧寄存器,lr是链接地址寄存器
51        sub     sp, sp, #16 @为局部变量分配空间
52        add     r7, sp, #0 @r7 = sp
```

```
53          mov     r3, #11 @局部变量x = 11
54          str     r3, [r7, #4]
55          mov     r3, #6 @局部变量y = 6
56          str     r3, [r7, #8]
57          ldr     r3, [r7, #4] @r3 = 11
58          ldr     r2, [r7, #8] @r2 = 6
59          mul     r3, r2, r3 @ x = x * y
60          str     r3, [r7, #4]
61          ldr     r1, [r7, #8] @参数b
62          ldr     r0, [r7, #4] @参数a
63          bl      avg(PLT) @函数avg (a, b)
64          str     r0, [r7, #12] @函数返回值r0, 存到[r7, #12]
65          ldr r2, [r3] @num
66          ldr r3, [r7, #12] @把avg_x_y取出来
67          b       .L5 @跳转到.L5
68  .L8:
69          ldr     r3, .L11
70  .LPIC0:
71          add     r3, pc
72          ldr     r2, [r3]
73          ldr     r3, [r7, #12] @r3 = [r7, #12],avg函数的返回值
74          mov     r1, r2 @num
75          mov     r0, r3 @avg_x_y
76          bl      __aeabi_idivmod(PLT) @求模运算
77          mov     r3, r1 @r3 = r1
78          cmp     r3, #0 @比较,avg_x_y % num是否为0
79          beq     .L10 @如果r3 == 0,跳出while循环
80          ldr     r3, .L11+4
81  .LPIC1:
82          add     r3, pc
83          ldr     r3, [r3]
84          subs    r3, r3, #1 @num = num -1
85          ldr     r2, .L11+8
86  .LPIC2:
87          add     r2, pc
88          str     r3, [r2]
89  .L5:
90          ldr     r3, .L11+12
91  .LPIC3:
92          add     r3, pc
93          ldr     r3, [r3]
94          ldr     r2, [r7, #12]
95          cmp     r2, r3 @比较r2和r3
96          bne     .L8 @不相等, 就跳到.L8
97          b       .L7
98  .L10:
99          nop
100 .L7:
```

```
101         ldr     r3, .L11+16
102 .LPIC4:
103         add     r3, pc
104         ldr     r3, [r3]
105         mov     r1, r3
106         ldr     r3, .L11+20
107 .LPIC5:
108         add     r3, pc
109         mov     r0, r3 @打印的参数
110         bl      printf(PLT) @printf函数
111         movs    r3, #0 @r3 = #0
112         mov     r0, r3
113         adds    r7, r7, #16 @r7 = r7 +16
114         mov     sp, r7 @恢复sp
115         pop     {r7, pc}
116 .L12:
117         .align  2
118 .L11:
119         .word   num-(.LPIC0+4)
120         .word   num-(.LPIC1+4)
121         .word   num-(.LPIC2+4)
122         .word   num-(.LPIC3+4)
123         .word   num-(.LPIC4+4)
124         .word   .LC0-(.LPIC5+4)
125         .size   main, .-main
```