南开大学

Nankai University

南　　开　　大　　学

计　算　机　学　院

编译原理实验报告

了解编译器及 LLVM IR 编程

王禹曦

年级：2020 级

专业：计算机科学与技术

指导教师：王刚

2022 年 10 月 1 日

目录

一、 实验要求及方法

以一个简单的 C（C++）源程序为例，调整编译器的程序选项获得各阶段的输出，研究它们与源程序的关系

基础示例程序程序

阶乘 C 代码

```c
#include<stdio.h>
int main()
{
    int i, n, f;
    scanf("%d", n);
    i = 2;
    f = 1;
    //阶乘循环
    while (i <= n)
    {
        f = f * i;
        i = i + 1;
    }
        printf("f\n");
}
```

二、 实验内容

(一) 预处理器

在 Ubuntu 操作系统中，使用 gcc 工具对阶乘程序预处理，具体命令为 gcc main.c -E -o main.i，将 main.c 文件进行预处理。预处理后文件由原来的 15 行变成了 753 行。

经过观察发现，预处理操作将头文件内容加入到了源文件，并将注释内容删除掉。

(二) 编译器

1. 词法分析

将源程序转化为了单词序列

通过 clang -E -Xclang -dump-tokens main.c 指令查看 token 序列

2. 语法分析

将词法分析得到的单词序列转换为抽象语法树 AST

通过 clang -E -Xclang -ast-dump main.c 指令查看 AST 树

3. 语义分析

使用语法树和符号表中信息来检查源程序是否与语言定义语义一致，也可通过后序遍历进行类型检查。

4. 中间代码生成

语义分析完成后，要进行中间代码的生成，依次为后来的代码优化和最后机器代码的生成做准备。

具体命令为 clang -S -emit-llvm main.c

5. 代码优化

此步骤会对生成的代码进一步优化，从而提高最终代码的运行效率。

6. 代码生成

形成最终的重定向机器代码，可通过不同的指令生成不同架构的汇编代码，如 gcc main.i -S -o main.S 可用于生成 X86 架构的汇编代码；arm-linux-gnueabihf-gcc main.i -S -o main.S 指令可交叉编译生成 arm 架构下的汇编代码

(三) 汇编器

汇编器会将刚刚生成的汇编代码汇编生成目标可重定向机器码，进而等待下一步链接器加载器加载到内存中，生成最终的机器代码可用 gcc 完成 X86 汇编代码的汇编，arm 架构则需要交叉编译，指令为 arm-linux-gnueabihf-gcc main.S -o main.o

(四) 链接器加载器

链接器、加载器会将目标文件与一些库文件链接在一起，并将相对地址替换为绝对地址，从而形成最终的可执行文件，如命令 gcc main.o -o main

(五) LLVM IR 编程

1. C 程序源代码

C 程序源代码

```c
#include"stdio.h"
float a[5][5]={1.2,2.4,3.6,4.8,5.0};
float max;

//求二维数组元素之和
float sumfunc(float a[][5]){
    int i=0;
    int j;
    float sum=0;
    while(i<5){
        j=0;
        while(j<5){
            sum=sum+a[i][j];
            j=j+1;
        }
        i=i+1;
    }
```

```
18      return sum;
19  }
20
21  /*求二维数组元素最大值*/
22  void maxfunc(float a[][5]){
23      int i=0;
24      int j;
25      max=a[0][0];
26      while(i<5){
27          j=0;
28          while(j<5){
29              if(max<a[i][j]){
30                  max=a[i][j];
31              }
32              j=j+1;
33          }
34          i=i+1;
35      }
36      return;
37  }
38
39  /*
40  int、float
41  变量赋值、运算
42  局部变量和全局变量
43  数组
44  表达式
45  函数调用、函数无返回值、函数有返回值
46  注释
47  while
48  if
49
50  */
51  int main(){
52      printf("元素之和为: %f \n",sumfunc(a));
53      maxfunc(a);
54      printf("元素最大值为: %f \n",max);
55      return 0;
56  }
57
58  /*
59  运行结果:
60  元素之和为: 17.000000
61  元素最大值为: 5.000000
62  */
```

2. 分工

王禹曦：maxfunc() 函数以及其对应 LLVM IR 代码的撰写，其中包含了许多 sysY 语言的特性，如 int、局部变量、二维数组、while、if 语句、全局变量、各种赋值、关系比较语句等。曹倚飞：sumfunc() 函数以及其 LLVM IR 代码的撰写，包含许多 sysY 语言的特性，如 int、float、局部变量、二维数组、while、if 语句、各种赋值语句、关系比较语句等。

3. 中间代码

IR 代码

```
@a = dso_local global [5 x [5 x float]] [[5 x float] [float 0
    x3FF3333340000000, float 0x4003333340000000, float 0x400CCCCCC0000000,
    float 0x4013333340000000, float 5.000000e+00], [5 x float]
    zeroinitializer, [5 x float] zeroinitializer, [5 x float] zeroinitializer
    , [5 x float] zeroinitializer], align 16
@max = common dso_local global float 0.000000e+00, align 4
@.str = private unnamed_addr constant [23 x i8] c"\E5\85\83\E7\B4\A0\E4\B9\8B
    \E5\92\8C\E4\B8\BA\EF\BC\9A%f \0A\00", align 1
@.str.1 = private unnamed_addr constant [26 x i8] c"\E5\85\83\E7\B4\A0\E6\9C
    \80\E5\A4\A7\E5\80\BC\E4\B8\BA\EF\BC\9A%f \0A\00", align 1



;sumfunc()函数
define dso_local float @sumfunc([5 x float]* %0) #0 {
    ;声明变量a、i、j、sum
    %a = alloca [5 x float]*, align 8                    ;局部变量a
    %i = alloca i32, align 4                             ;int i
    %j = alloca i32, align 4                             ;int j
    %sum = alloca float, align 4                         ;float sum

    ;初始化a、i、sum
    store [5 x float]* %0, [5 x float]** %a, align 8     ;a=函数传入的参数
    store i32 0, i32* %i, align 4                        ;i=0
    store float 0.000000e+00, float* %sum, align 4       ;sum=0.0

    br label %while1.cond

while1.cond:                                          ;preds = %while1.body2,
    %1
    %2 = load i32, i32* %i, align 4
    %cmpi = icmp slt i32 %2, 5
                                                          ;while(i<5)
    br i1 %cmpi, label %while1.body1, label %return

while1.body1:                                         ;preds = %while1.cond
    store i32 0, i32* %j, align 4
                                                          ;j=0
```

```
    br label %while2.cond

while2.cond:                                      ; preds = %while1.body1, %while2.body
    %3 = load i32, i32* %j, align 4
    %cmpj = icmp slt i32 %3, 5
                                                  ; while(j<5)
    br i1 %cmpj, label %while2.body, label %while1.body2

while2.body:                                      ; preds = %while2.cond
    %4 = load float, float* %sum, align 4
                                                  ; sum
    %5 = load i32, i32* %i, align 4
                                                  ; i
    %6 = load i32, i32* %j, align 4
                                                  ; j
    %7 = load [5 x float]*, [5 x float]** %a, align 8
                                                  ; a

    %8 = sext i32 %5 to i64
    %9 = getelementptr inbounds [5 x float], [5 x float]* %7, i64 %8
    %10 = sext i32 %6 to i64
    %11 = getelementptr inbounds [5 x float], [5 x float]* %9, i64 0, i64 %10
    %12 = load float, float* %11, align 4
                                                  ; a[i][j]

    %add = fadd float %4, %12
                                                  ; sum+a[i][j]
    store float %add, float* %sum, align 4
                                                  ; sum=sum+a[i][j]

    %incj = add nsw i32 %6, 1
    store i32 %incj, i32* %j, align 4
                                                  ; j=j+1
    br label %while2.cond

while1.body2:                                     ; preds = %while2.cond
    %13 = load i32, i32* %i, align 4
    %inci = add nsw i32 %13, 1
    store i32 %inci, i32* %i, align 4
                                                  ; i=i+1
    br label %while1.cond

return:                                           ; preds = %while1.cond
    %14 = load float, float* %sum, align 4
    ret float %14
}
```

```
67
68  ;maxfunc()函数
69  define dso_local void @maxfunc([5 x float]* %0) #0 {
70      ;声明变量a、i、j
71      %a = alloca [5 x float]*, align 8                    ;局部变量a
72      %i = alloca i32, align 4                             ;i
73      %j = alloca i32, align 4                             ;j
74
75      ;初始化a、i
76      store [5 x float]* %0, [5 x float]** %a, align 8     ;a=函数传入的
            参数
77      store i32 0, i32* %i, align 4                        ;i=0
78
79      ;初始化max
80      %2 = load [5 x float]*, [5 x float]** %a, align 8
81      %3 = getelementptr inbounds [5 x float], [5 x float]* %2, i64 0
82      %4 = getelementptr inbounds [5 x float], [5 x float]* %3, i64 0, i64 0
83      %5 = load float, float* %4, align 4                  ;a[0][0]
84      store float %5, float* @max, align 4                 ;max=a
            [0][0]
85      br label %while1.cond
86
87  while1.cond:                                             ;preds = %while1.body2,
        %1
88      %6 = load i32, i32* %i, align 4
89      %cmpi = icmp slt i32 %6, 5
                                                             ;while(i<5)
90      br i1 %cmpi, label %while1.body1, label %return
91
92  while1.body1:                                            ;preds = %while1.cond
93      store i32 0, i32* %j, align 4
                                                             ;j=0
94      br label %while2.cond
95
96  while2.cond:                                             ;preds = %while1.body1, %
        while2.body2
97      %7 = load i32, i32* %j, align 4
98      %cmpj = icmp slt i32 %7, 5
                                                             ;while(j<5)
99      br i1 %cmpj, label %while2.body1, label %while1.body2
100
101 while2.body1:                                            ;preds = %while2.cond
102     br label %if.cond
103
104 if.cond:                                                 ;preds = %while2.body1
105     %8 = load float, float* @max, align 4                ;max
106     %9 = load [5 x float]*, [5 x float]** %a, align 8    ;a
107     %10 = load i32, i32* %i, align 4
```

```
    %11 = sext i32 %10 to i64                                    ;i
    %12 = load i32, i32* %j, align 4
    %13 = sext i32 %12 to i64                                    ;j

    %14 = getelementptr inbounds [5 x float], [5 x float]* %9, i64 %11
    %15 = getelementptr inbounds [5 x float], [5 x float]* %14, i64 0, i64
        %13
    %16 = load float, float* %15, align 4                        ;a[i][j]

    %cmpmax = fcmp olt float %8, %16
    br i1 %cmpmax, label %if.body, label %while2.body2

if.body:                                                         ;preds = %if.cond
    ;%17 = load float, float* @max, align 4                      ;max
    %17 = load [5 x float]*, [5 x float]** %a, align 8           ;a
    %18 = load i32, i32* %i, align 4
    %19 = sext i32 %18 to i64                                    ;i
    %20 = load i32, i32* %j, align 4
    %21 = sext i32 %20 to i64                                    ;j

    %22 = getelementptr inbounds [5 x float], [5 x float]* %17, i64 %19
    %23 = getelementptr inbounds [5 x float], [5 x float]* %22, i64 0, i64
        %21
    %24 = load float, float* %23, align 4                        ;a[i][j]

    store float %24, float* @max, align 4                        ;max=
        a[i][j]
    br label %while2.body2

while2.body2:                                                    ;preds = %if.cond, %
    if.body
    %25 = load i32, i32* %j, align 4
    %incj = add nsw i32 %25, 1
    store i32 %incj, i32* %j, align 4                            ;j=j
        +1
    br label %while2.cond

while1.body2:                                                    ;preds = %while2.cond
    %26 = load i32, i32* %i, align 4
    %inci = add nsw i32 %26, 1
    store i32 %inci, i32* %i, align 4                            ;j=j
        +1
    br label %while1.cond

return:                                                          ;preds = %while1.cond
    ret void
}
```

```
150
151
152 ;main函数
153 define dso_local i32 @main() #0 {
154
155     ;返回变量retval=0
156     %retval = alloca i32, align 4
157     store i32 0, i32* %retval, align 4
158
159     ;调用函数sumfunc()
160     %1 = call float @sumfunc([5 x float]* getelementptr inbounds ([5 x [5 x
        float]], [5 x [5 x float]]* @a, i64 0, i64 0))
161     ;把sumfunc()返回值改成double型
162     %2 = fpext float %1 to double
163
164     ;调用函数printf(), 输出sumfunc()返回值
165     %3 = call i32 (i8*, ...) @printf(i8* getelementptr inbounds ([23 x i8],
        [23 x i8]* @.str, i64 0, i64 0), double %2)
166
167     ;调用函数maxfunc()
168     call void @maxfunc([5 x float]* getelementptr inbounds ([5 x [5 x float
        ]], [5 x [5 x float]]* @a, i64 0, i64 0))
169
170     ;调用函数printf(), 输出max
171     %4 = load float, float* @max, align 4
172     %5 = fpext float %4 to double
173     %6 = call i32 (i8*, ...) @printf(i8* getelementptr inbounds ([26 x i8],
        [26 x i8]* @.str.1, i64 0, i64 0), double %5)
174
175     ;return 0
176     %7 = load i32, i32* %retval, align 4
177     ret i32 %7
178 }
179
180     declare dso_local i32 @printf(i8*, ...) #1
181
182     attributes #0 = { noinline nounwind optnone uwtable "correctly-rounded-
        divide-sqrt-fp-math"="false" "disable-tail-calls"="false" "frame-
        pointer"="all" "less-precise-fpmad"="false" "min-legal-vector-width"=
        "0" "no-infs-fp-math"="false" "no-jump-tables"="false" "no-nans-fp-
        math"="false" "no-signed-zeros-fp-math"="false" "no-trapping-math"="
        false" "stack-protector-buffer-size"="8" "target-cpu"="x86-64" "
        target-features"="+cx8,+fxsr,+mmx,+sse,+sse2,+x87" "unsafe-fp-math"="
        false" "use-soft-float"="false" }
183     attributes #1 = { "correctly-rounded-divide-sqrt-fp-math"="false" "
        disable-tail-calls"="false" "frame-pointer"="all" "less-precise-fpmad
        "="false" "no-infs-fp-math"="false" "no-nans-fp-math"="false" "no-
        signed-zeros-fp-math"="false" "no-trapping-math"="false" "stack-
```

```
protector-buffer-size"="8" "target-cpu"="x86-64" "target-features"="+
cx8,+fxsr,+mmx,+sse,+sse2,+x87" "unsafe-fp-math"="false" "use-soft-
float"="false" }
```

4. 结果验证

通过命令 llvm-as a.ll -o a.bc，将 LLVM IR 代码编译，生成.bc 文件。
下一步通过 lli main.bc 命令执行.bc 文件，结果与 C 程序运行结果相同。

三、 总结

高级语言的编译、汇编是很复杂的，通过分层次地对代码分析，可获得最终的可执行文件机器代码。