



北京理工大学
BEIJING INSTITUTE OF TECHNOLOGY

本科生《编译原理》课程实践报告

题 目： 目标代码生成实验

学 院： 徐特立学院

专业名称： 计算机科学与技术

姓 名： 陈照欣-1120191086

实验目的

- (1) 了解编译器指令生成和寄存器分配的基本算法;
- (2) 掌握目标代码生成的相关技术和方法, 设计并实现针对 x86/MIPS/RISCV/ARM 的目标代码生成模块;
- (3) 掌握编译器从前端到后端各个模块的工作原理, 目标代码生成模块与其他模块之间的交互过程。

实验内容

基于 BIT-MiniCC 构建目标代码生成模块, 该模块能够基于中间代码选择合适的目标指令, 进行寄存器分配, 并生成相应平台汇编代码。如果生成的是 MIPS 或者 RISC-V 汇编, 则要求汇编代码能够在 BIT-MiniCC 集成的 MIPS 或者 RISC-V 模拟器中运行。需要注意的是, config.xml 的最后一个阶段“ncgen”的“skip”属性配置为“false”, “target”属性设置为“mips”、“x86”或者“riscv”中的一个。如果生成的是 X86 汇编, 则要求使用 X86 汇编器生成 exe 文件并运行。

实验步骤

全局维护一个变量 stringBuilder 用来存储生成的汇编代码。

由于本次实验提供的测试用例的输入输出函数, Mars_PrintfInt, Mars_PrintfStr, Mars_GetInt 在 x86 中并未定义。所以需要通过借用对应的 c 语言函数, 又由于 x86 中对应的 printf 和 scanf 的调用处理较为特殊, 所以在此处对于这一类函数调用进行特殊处理。所以在完成 lab8 时对 lab7 中间代码部分增加了内置函数的识别。

本实验中新建了一个类 MyQuat, 与 lab7 中建立的结构类似, 只是元素由 AST 节点换成了 String 类。在生成目标代码前先使用 LoadIC 转换, 方便后续的读取。

首先由于我生成的四元式中的函数调用形式为: 先用 arg 表示要传入的变量, 再使用 call 来进行函数调用, 所以, 通过维护一个 ParamStack, 将遇到 arg 四元式时, 将内容压入, 即要传入的参数, 当遇到 call 四元式时, 将 ParamStack 中的变量均弹出, 即可进行函数调用。所以, 对于 Mars_PrintfInt, Mars_PrintfStr, Mars_GetInt 这三个特殊函数, 对应的输出字符串首先定义在 x86 代码的.data 段, 以便后面调用。

```

if(q.getOp().equals("arg")){
    String param = q.getRes();
    ParamStack.push(param);
}
if(q.getOp().equals("call")){
    if(q.getOpnd1().equals("Mars_PrintStr")){
        while(ParamStack.size()!=0){
            print_scanf.add(ParamStack.pop());
        }
    }
}

```

对于四元式中频繁出现的@1, @2 之类的临时变量, 本次实验选择将其都定义为局部变量, 不进行寄存器分配。我们通过维护符号表, 将 TemporaryValue 统统加入对应函数的符号表中, 这样在访问到 func 四元式时, 即可访问对应的函数的符号表, 将其中的变量通过 local 关键字进行定义。

同时, 对于数组的处理与临时变量类似, 不过需要将多维数组展开成一维数组进行定义。

```

if(mq.getOp().equals("var")){
    stringBuilder.append("local "+mq.getRes()+":dword\n");
} else if (mq.getOp().equals("arr")) {
    String limit = mq.getOpnd1();
    String limit1 = limit.replace( target: "int<", replacement: "");
    limit1 = limit1.replace( target: ">", replacement: "");
    String limit2 = limit1;
    while(limit2.charAt(0)=='0' ||
        limit2.charAt(0)=='1' ||
        limit2.charAt(0)=='2' ||
        limit2.charAt(0)=='3' ||
        limit2.charAt(0)=='4' ||
        limit2.charAt(0)=='5' ||
        limit2.charAt(0)=='6' ||
        limit2.charAt(0)=='7' ||
        limit2.charAt(0)=='8' ||
        limit2.charAt(0)=='9')
        limit2 = limit1.substring(1,limit1.length());
    limit2 = limit2.substring(1,limit2.length());
    if(limit2.equals(""))
        stringBuilder.append("local "+mq.getRes()+":["+limit1+"]:dword\n");
    else {
        limit2 = limit2.replace( target: "<", replacement: "");
        limit2 = limit2.replace( target: ">", replacement: "");
        stringBuilder.append("local "+mq.getRes()+"[8+4*"+limit2+"]:dword\n");
    }
}

```

后续调用时, “[]=”用来实现对数组的访问, “=[]”用来实现对数组的赋值

```

}else if(op.equals("=[]")) {
    String symoff = q.getOpnd1(),symarr = q.getOpnd2(),symtar = q.getRes();
    stringBuilder.append("mov edi, "+symoff+"\n");
    stringBuilder.append("mov eax,dword ptr "+symarr+"[edi*4]\n");
    stringBuilder.append("mov "+symtar+", eax\n");
}else if(op.equals("[]=")) {
    String symoff = q.getOpnd2(),symarr = q.getRes(),symsrc = q.getOpnd1();
    stringBuilder.append("mov eax," +symsrc+"\n");
    stringBuilder.append("mov dword ptr "+symarr+"[edi*4], eax\n");
}

```

实验结果

本次实验能够编译通过的样例为 1_Fibonacci.c, 2_Prime.c, 3_PerfectNumber.c

1_Fibonacci.c

```

Microsoft Visual Studio 调试控制台
Please input a number:
8
This number's fibonacci value is :
21
D:\Documents\Assembly language _and connection technology\test\Debug\test.exe (进程 3612) 已退出, 代码为 0。
要在调试停止时自动关闭控制台, 请启用“工具”->“选项”->“调试”->“调试停止时自动关闭控制台”。
按任意键关闭此窗口。 . . .

```

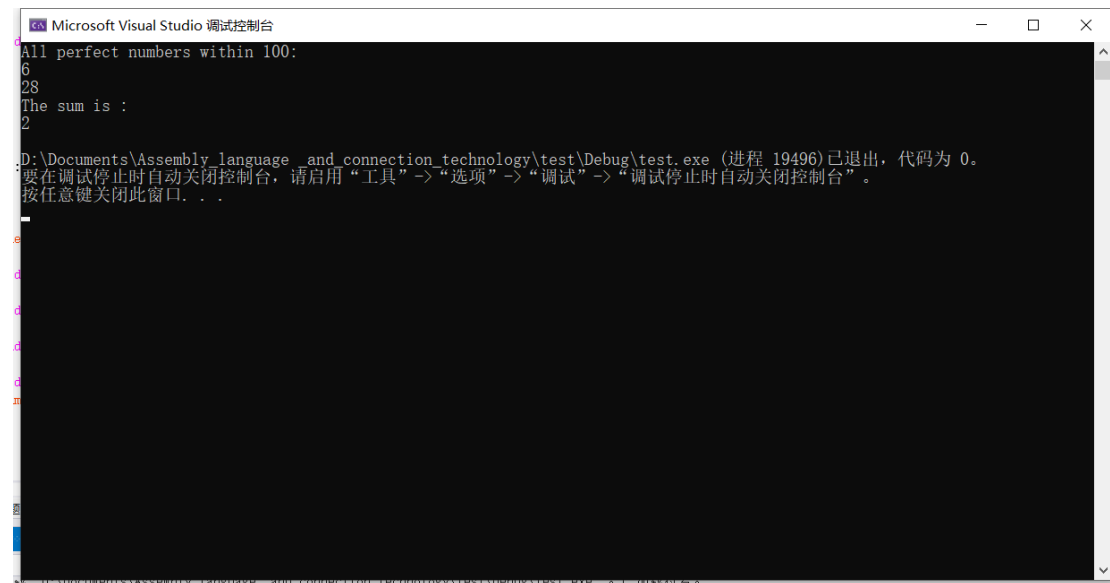
2_Prime.c

```

Microsoft Visual Studio 调试控制台
5
6
7
11
13
17
19
23
29
31
37
41
43
47
53
59
61
67
71
73
79
83
89
97
The number of prime numbers within n is:
25
D:\Documents\Assembly language _and connection technology\test\Debug\test.exe (进程 21092) 已退出, 代码为 0。
要在调试停止时自动关闭控制台, 请启用“工具”->“选项”->“调试”->“调试停止时自动关闭控制台”。
按任意键关闭此窗口。 . . .

```

3_PerfectNumber.c



```
Microsoft Visual Studio 调试控制台
All perfect numbers within 100:
6
28
The sum is :
2
D:\Documents\Assembly language and connection technology\test\Debug\test.exe (进程 19496) 已退出，代码为 0。
要在调试停止时自动关闭控制台，请启用“工具”->“选项”->“调试”->“调试停止时自动关闭控制台”。
按任意键关闭此窗口。...
```

实验总结

在词法分析，语法分析，语义分析，中间代码一系列实验的基础之上，勉强完成了一整套编译器的搭建。从 lab5 开始的无比痛苦到后面思路的逐渐明朗，我对编译原理的一整套流程也有了渐渐清晰的认识。每次在尝试新实验的时候都会因为各种 bug 而去反思先前的实验是否有缺陷，进而理解了每一步在这一阶段的真实用意。

最后一段时间真的很紧，考试周安排很紧张，考好了之后开始备战夏令营，还需要积极完成整套架构。这个过程真的很累，但是在看到自己生成的代码真的能像 C 语言编译器一样运行时，还是有了极大的满足感。

最后感谢计老师的悉心指导，也希望之后自己能够抽出时间完善自己的这一套架构。