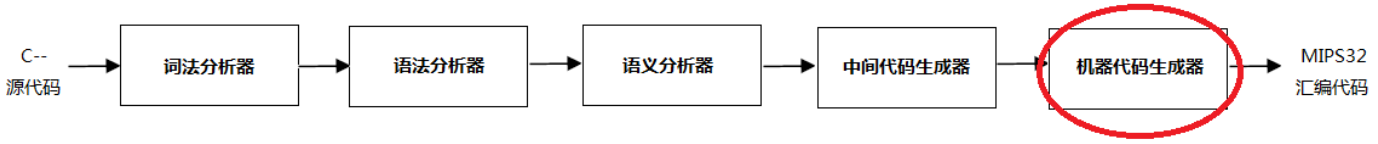
编译原理实习 4　机器代码生成

许畅 陈嘉 朱晓瑞

编译原理课程的实习内容是为一个小型的类 C 语言(C--)实现一个编译器。如果你顺利完成了本课程规定的实习任务，那么不仅你的编程能力将会得到大幅提高，而且你最终会得到一个比较完整的、能将 C--源代码转换成 MIPS 汇编代码的编译器，所得到的汇编代码可以在 SPIM Simulator 上运行。实习总共分为四个阶段：词法和语法分析、语义分析、中间代码生成和目标代码生成。每个阶段的输出是下一个阶段的输入，后一个阶段总是在前一个阶段的基础上完成，如下图所示：



实习 4 是本学期编译课的最后一次实习，其任务是编写一个程序，该程序读入一个 C--源代码文件，在词法分析、语法分析、语义分析与中间代码生成之后，输出最终对应的 MIPS32 指令序列（可以包含伪指令），并在 SPIM Simulator 上进行测试。当你完成本次实习之后，你就拥有了一个自己独立编写的、可以实际运行起来的编译器，回想起这个学期在这个编译器上花费的所有努力，是不是很有成就感呢？

选择 MIPS 作为目标体系结构一方面是因为它属于 RISC 范畴，与 x86 等体系结构相比形式简单而且十分便于我们的处理。另一方面，我们系在大二开设的组成原理课以及之后的组成原理实验课就是以 MIPS 体系结构为例进行讲解和实践的，同学们对于它应该是再熟悉不过了。如果你对于 MIPS 体系结构或者汇编语言有些淡忘也不要紧，我们会提供详细的参考资料。

为了完成本次实习，强烈建议你首先下载并安装 SPIM Simulator 用于对生成的目标代码进行检查和调试，其官方网站的地址为：http://pages.cs.wisc.edu/~larus/spim.html。这是由原 Wisconsin-Madison 的 Jame Larus 教授（现在在微软）领导编写的一个功能强大的 MIPS 汇编语言的汇编器+模拟器，其最新的图形界面版本 QtSPIM 由于使用了 Qt 组件因而可以在 Windows、Linux、Mac 等操作系统上跨平台运行，推荐安装。有关 SPIM 模拟器的使用方法，同样我们会提供参考资料并在指导攻略中说明。

你需要做的就是将上一次实习所得到的中间代码经过与具体体系结构相关的指令选择、寄存器选择以及栈管理后，转换成 MIPS 汇编代码。我们要求你的程序能输出**正确的汇编代码**。“正确的汇编代码”是指汇编代码在 SPIM 上（命令行版本或 Qt 版本均可）运行**结果**正确。因此，以下几个方面**不属于**检查范围：

- 寄存器的使用与指派可以不必遵循 MIPS32 的约定：只要不影响在 SPIM 中的正常运行，你可以随意分配 MIPS 体系结构中的 32 个通用寄存器，而不必在意哪些寄存器存放参数、哪些存放返回值、哪些由调用者负责保存、哪些由被调用者负责保存，等等。

- 栈的管理（包括栈帧中的内容及存放顺序）也不必遵循MIPS32的约定，你甚至可以使用栈以外的方式对过程调用间各种数据的传递进行管理，前提是你输出的目标代码（即MIPS汇编代码）正确。

当然，不要求不代表不重要。我们建议你试着去遵守MIPS中的各种约定，否则你的程序生成的目标代码在SPIM中运行时可能会出现一些意想不到的错误。

在进行本次实习之前，请**仔细**阅读所提供的指导攻略，确保你已经大体了解MIPS汇编语言以及SPIM 的使用方法，这些内容是你顺利完成本次实习的基本前提。需要注意的是，由于本次实习内容将会与之前实习中你已经写好的代码进行对接，因此保持一个良好的代码风格，系统地设计代码结构和各模块之间的接口等对于整个实习来讲可以说是相当重要的。

输入格式

程序的输入是一个文本文件，其中包含有C--的源代码。本次实验对输入文件有如下假设：

- 不会出现词法、语法及语义错误
- 不会出现注释、八进制数、十六进制数、浮点类型的常数或者变量
- 整数常量都在16位整数范围内，也就是说你不必花费精力去考虑如果某个常数无法在addi等包含立即数的指令中表示时应该怎么办
- 不会出现结构体类型和高维数组（高于1维的数组）类型的变量
- 任何参数都只能为简单变量，也就是说，数组和结构体不会作为参数传入某个函数中
- 所有变量都具有全局作用域(这里“全局作用域”是指所有变量均不重名，**并非**变量在任何位置都能被访问)，其存储空间都放到该变量所在的函数的活动记录中
- 函数不会返回数组类型和结构体类型的值
- 函数只会进行一次定义（没有函数声明）

本次实习要求你的程序能够接收一个输入文件名和一个输出文件名作为参数。例如，假设你的程序名为cc、输入文件名为test.cmm、输出文件名为out.s，程序和输入文件都位于当前目录下，那么在命令行下运行./cc test.cmm out.s 即可将输出结果写入当前目录下名为out.s的文件。

输出格式

本次实习要求你的程序将运行结果输出到文件。对于每一个输入文件，你的程序应当输出相应的MIPS汇编代码，我们将使用SPIM对你输出的代码的正确性进行测试，任何能被SPIM Simulator执行并且执行结果正确的输出都将被接受。

测试环境

你的程序将在如下环境中被编译并运行：

- GNU Linux Release: Ubuntu 12.04, kernel version 3.2.0-29
- GCC version 4.6.3
- GNU Flex version 2.5.35
- GNU Bison version 2.5
- QtSPIM version 9.1.9

一般而言，只要避免使用过于冷门的特性，使用其他版本的 Linux 或者 GCC 等，也基本上不会出现兼容性方面的问题。注意，实验检查过程中不会去安装或尝试引用各类方便编程的函数库（如 glib 等），因此请不要在你的程序中使用它们。

提交要求

本次实习要求你提交如下内容：

- Flex、Bison 以及 C 语言的可以被正确编译运行的源代码
- 一份 PDF 格式的实验报告，内容主要包括：
  - 你的程序实现了哪些功能？简要说明你是如何实现这些功能的。如果因为你的说明不够充分而导致助教没有对你所实现的功能进行测试，那么后果自负。
  - 你所提交上来的程序应当如何编译？不管你使用了脚本也好，准备了 Makefile 也好甚至是单独地逐条命令手工输入进行编译也好，请**详细说明**具体需要键入哪些命令——无法顺利编译将会使你丢失相应分数，并且如果不去找助教进行修正，后面的正确分也会因你的程序无法运行而全部丢失，请谨记这一点。
  - 你的实验报告长度**不得超过** 3 页！因此，你需要好好考虑一下该往实验报告里写些什么。我们的建议是，实验报告中需要你重点描述的应当是你所提交的工作中的亮点，应当是那些你认为最**个性化**、最具有**独创性**的内容，而那些比较简单的、任何人都可以做出来的内容可以不提或者只简单的提一下，尤其要避免去大段大段地向报告里贴代码。

    为了避免大家通过减小字号来变相加长页数限制，我们规定实验报告中所出现的最小字号不得小于五号字（英文 11 号字）。

样例输入 1

```
int main()
{
    int a=0,b=1,i=0,n;
    n = read();
    while (i < n)
    {
        int c = a+b;
        write(b);
        a = b;
        b = c;
        i = i + 1;
    }
    return 0;
}
```

样例输出 1

这段程序读入一个整数 n，然后计算并输出前 n 个 Fibonacci 数。将其翻译为一段能在 SPIM 中执行的正确的目标代码可以是这样的：

```
.data
_prompt: .asciiz "Enter an integer:"
_ret: .asciiz "\n"
.globl main
.text
read:
    li $v0, 4
    la $a0, _prompt
    syscall
    li $v0, 5
    syscall
    jr $ra

write:
    li $v0, 1
    syscall
    li $v0, 4
    la $a0, _ret
    syscall
    move $v0, $0
    jr $ra

main:
    li $t5, 0
    li $t4, 1
    li $t3, 0
    addi $sp, $sp, -4
    sw $ra, 0($sp)
    jal read
    lw $ra, 0($sp)
    addi $sp, $sp, 4
    move $t1, $v0
    move $t2, $t1
label1:
    blt $t3, $t2, label2
    j label3
label2:
    add $t1, $t5, $t4
    move $a0, $t4
    addi $sp, $sp, -4
    sw $ra, 0($sp)
    jal write
    lw $ra, 0($sp)
    addi $sp, $sp, 4
    move $t5, $t4
    move $t4, $t1
    addi $t1, $t3, 1
    move $t3, $t1
    j label1
label3:
    move $v0, $0
    jr $ra
```

该段汇编代码在命令行 SPIM 中的运行结果为（输入 7，输出前 7 个 fibonacci 数）：

```
SPIM Version 8.0 of January 8, 2010
Copyright 1990-2010, James R. Larus.
All Rights Reserved.
See the file README for a full copyright notice.
Loaded: /usr/lib/spim/exceptions.s
Enter an integer:7
1
1
2
3
5
8
13
```

样例输入 2

```
int fact(int n)
{
     if (n == 1)
         return n;
     else
         return (n*fact(n-1));
}

int main()
{
     int m, result;
     m = read();
     if (m > 1)
         result = fact(m);
     else
         result = 1;
     write(result);
     return 0;
}
```

样例输出 2

这段程序读入一个整数 n，然后计算并输出 n!。将其翻译为一段能在 SPIM 中执行的正确的目标代码可以是这样的：

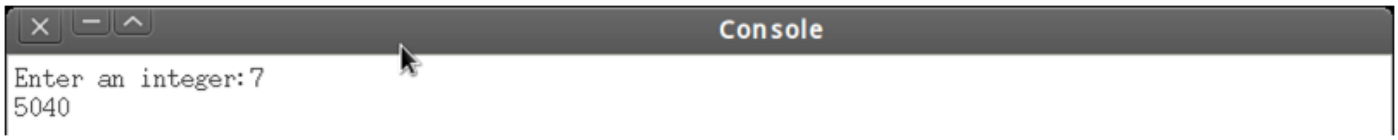
```
.data
_prompt: .asciiz "Enter an integer:"
_ret: .asciiz "\n"
.globl main
.text
read:
    li $v0, 4
    la $a0, _prompt
    syscall
    li $v0, 5
    syscall
    jr $ra

write:
    li $v0, 1
    syscall
    li $v0, 4
    la $a0, _ret
    syscall
    move $v0, $0
    jr $ra

main:
    addi $sp, $sp, -4
    sw $ra, 0($sp)
    jal read
    lw $ra, 0($sp)
    addi $sp, $sp, 4
    move $t1, $v0
    li $t3, 1
    bgt $t1, $t3, label6
    j label7
label6:
    move $a0, $t1
    addi $sp, $sp, -4
    sw $ra, 0($sp)
    jal fact
    lw $ra, 0($sp)
    addi $sp, $sp, 4
    move $t2, $v0
    j label8
label7:
    li $t2, 1
label8:
    move $a0, $t2
    addi $sp, $sp, -4
    sw $ra, 0($sp)
    jal write
    lw $ra, 0($sp)
    addi $sp, $sp, 4
    move $v0, $0
    jr $ra

fact:
    li $t4, 1
    beq $a0, $t4, label1
    j label2
label1:
    move $v0, $a0
    jr $ra
label2:
    addi $sp, $sp, -8
    sw $a0, ($sp)
    sw $ra, 4($sp)
    sub $a0, $a0, 1
    jal fact
    lw $a0, ($sp)
    lw $ra, 4($sp)
    addi $sp, $sp, 8
    mul $v0, $v0, $a0
    jr $ra
```

该段汇编程序在 QtSPIM 中的运行结果如下（输入7，输出7!=5040）：



除了上面给出的样例程序外，你的程序要能够将其它符合本次试验假设的 C--源代码翻译为目标代码。我们将通过检查目标代码是否能在模拟器上运行得到正确结果来判断你的程序的正确性。