



中山大學 软件工程学院  
SUN YAT-SEN UNIVERSITY SCHOOL OF SOFTWARE ENGINEERING

# 编译器构造实验

## Lab 4 – 目标代码生成

2023年秋季学期

Teaching Team: 王焱林、伍嘉栋、黄炎贤、王岩立、张俊鹏

# 课程信息

- 课程类别：专业选修课
- 课程学分：1分
- 开课单位：软件工程学院
- 主要教材
  - 《编译原理实践与指导教程》（非必须）
- 成绩构成
  - 平时40%（4次Lab各10%）+ 大作业60%
- 助教：伍嘉栋Lab1、张俊鹏Lab2、黄炎贤Lab3、王岩立Lab4、共同(期末+其他)

# 特别说明

- 本课程需要和理论课《编译原理》配套学习
- 作业迟交：每天扣10%的作业分
- 作业抄袭：0分

## 4.1 实验内容

## 4.1.1 实验要求

- 将实验三中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之后，转换为MIPS32汇编代码)
- 对作为输入的C--源代码有如下的假设：
  - 假设1：输入文件中不包含任何词法、语法或语义错误（函数也必有return语句）。
  - 假设2：不会出现注释、八进制或十六进制整型常数、浮点型常数或者变量。
  - 假设3：整型常数都在16bits位的整数范围内，也就是说你不必考虑如果某个整型常数无法在addi等包含立即数的指令中表示时该怎么办。

## 4.1.1 实验要求

➤ 对作为输入的C--源代码有如下的假设：

- 假设4：不会出现类型为结构体或高维数组（高于1维的数组）的变量。
- 假设5：没有全局变量的使用，并且所有变量均不重名，变量的存储空间都放到该变量所在的函数的活动记录中。
- 假设6：任何函数参数都只能是简单变量，也就是说数组和结构体不会作为参数传入某个函数中。
- 假设7：函数不会返回结构体或数组类型的值。
- 假设8：函数只会进行一次定义（没有函数声明）。

## 4.1.2 输入和输出

- 输入：一个包含C--源代码的文本文件
- 输出：相应的MIPS32汇编代码
- 测试环境：你的程序将在如下环境中被编译并运行（同实验一）：
  - 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
  - 可以使用其它版本的Linux或者GCC等

## 4.1.3 提交要求

➤ 实验四要求提交如下内容（同实验一）：

- 提交链接：[https://send2me.cn/dlPe\\_ZXw/Ru-hZQH\\_8EzE7A](https://send2me.cn/dlPe_ZXw/Ru-hZQH_8EzE7A)
- 截止时间：2023.12.28 23：59
- Flex、Bison以及C语言的可被正确编译运行的源程序。
- 一份PDF格式的实验报告，内容包括：
  - 程序实现了哪些功能？简要说明如何实现这些功能
  - 程序应该如何被编译？可以使用脚本、makefile或逐条输入命令进行编译，请详细说明应该如何编译程序



# 4.1.4 样例

## ➤ 样例1：

输入：

```
1 int main()
2 {
3     int a = 0, b = 1, i = 0, n;
4     n = read();
5     while (i < n)
6     {
7         int c = a + b;
8         write(b);
9         a = b;
10        b = c;
11        i = i + 1;
12    }
13    return 0;
14 }
```

```
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
```

输出：

该样例程序读入一个整数n，然后计算并输出前n个Fibonacci数的值。将其翻译为一段能

在SPIM Simulator中执行的正确的目标代码可以是这样的：

```
1 .data
2 _prompt: .asciiz "Enter an integer:"
3 _ret: .asciiz "\n"
4 .globl main
5 .text
6 read:
7     li $v0, 4
8     la $a0, _prompt
9     syscall
10    li $v0, 5
11    syscall
12    jr $ra
13
14 write:
15    li $v0, 1
16    syscall
17    li $v0, 4
18    la $a0, _ret
19    syscall
20    move $v0, $0
21    jr $ra
22
23 main:
24    li $t5, 0
25    li $t4, 1
26    li $t3, 0
27    addi $sp, $sp, -4
28    sw $ra, 0($sp)
29    jal read
30    lw $ra, 0($sp)
31    addi $sp, $sp, 4
32    move $t1, $v0
33    move $t2, $t1
34 label1:
35    blt $t3, $t2, label2
36    j label3
37 label2:
38    add $t1, $t5, $t4
39    move $a0, $t4
40    addi $sp, $sp, -4
41    sw $ra, 0($sp)
42    jal write
43    lw $ra, 0($sp)
44    addi $sp, $sp, 4
45    move $t5, $t4
46    move $t4, $t1
47    addi $t1, $t3, 1
48    move $t3, $t1
49    j label1
50 label3:
51    move $v0, $0
52    jr $ra
```

图15. 样例1汇编代码的运行结果。

## 4.1.4 样例

### ➤ 样例2:

样例2:

输入:

```
1  int fact(int n)
2  {
3      if (n == 1)
4          return n;
5      else
6          return (n * fact(n - 1));
7      }
8
9  int main()
10 {
11     int m, result;
12     m = read();
13     if (m > 1)
14         result = fact(m);
15     else
16         result = 1;
17     write(result);
18     return 0;
19 }
```



图16. 样例2汇编代码的运行结果。

## 4.1.5 参考代码（挖空）

- 一共挖了五个空，包括在寄存器分配、操作数装载、中间代码翻译
- 1) 在存在空闲寄存器的情况下，为变量描述符分配寄存器，可以参考不存在空闲寄存器的情况

```
249 /*
250  * 为变量描述符分配寄存器，load用于指示是否需要装载寄存器，
251  * 形如 x = y op z 的表达式中，为x分配寄存器就不需要装载，而为y和z分配时都需要
252  */
253 int allocateReg(VarDes var, FILE* fp, int load) {
254     // 查找是否有空闲寄存器
255     int i = 8;
256     for (; i < 26; i++)
257         if (regs[i]->var == NULL)
258             break;
259     // 存在空闲寄存器
260     if (i >= 8 && i < 26) {
261         // TODO
262     }
```


## 4.1.5 参考代码（挖空）

- 2) 完成地址操作数的装载，参考值操作数的装载


```
345 // 根据操作数的类型完成装载
346 int handleOp(Operand op, FILE* fp, int load) {
347     if (op->kind == VARIABLE_OP || op->kind == TEMP_VAR_OP || op->kind == CONSTANT_OP)
348         return getReg(op, fp, load);
349     else if (op->kind == GET_VAL_OP) {
350         int reg = getReg(op->opr, fp, load);
351         fprintf(fp, "    lw %s, 0(%s)\n", regs[reg]->name, regs[reg]->name);
352         return reg;
353     }
354     else if (op->kind == GET_ADDR_OP) {
355         // TODO
356     }
357 }
```

## 4.1.5 参考代码（挖空）


- 3) 将中间代码翻译成目标代码并输入到文件， ASSIGN

```
415  case ASSIGN_IR: {  
416 // TODO  
417 }
```

- 4) 将中间代码翻译成目标代码并输入到文件， SUB

```
437  case SUB_IR: {  
438 // TODO  
439 }
```

- 5) 将中间代码翻译成目标代码并输入到文件， DIV

```
459  case DIV_IR: {  
460 // TODO  
461 }
```

## 4.2 实验指导

## 4.2.1 QtSPIM简易教程

### ➤ 安装:

- 命令行版: `sudo apt-get install spim`
- GUI版: 访问 SPIM Simulator 官方 <http://pages.cs.wisc.edu/~larus/spim.html> 下载安装

### ➤ 使用:

- 命令行版: `spim -file [汇编代码文件名]`
- GUI版: 可以参照 Project\_4.pdf 中 Page103-104 内容

## 4.2.2 MIPS32汇编代码书写

- SPIM Simulator 不仅是一个MIPS32的模拟器，也是一个MIPS32的汇编器。
- 想要让 SPIM Simulator 正常模拟，首先需要为它准备符合格式的 MIPS32 汇编代码文本文件。非操作系统内核的汇编代码文件必须以 .s 或者 .asm 作为文件的后缀名。
- 汇编代码由若干代码段和若干数据段组成，其中代码段以 .text 开头，数据段以 .data 开头。
- 汇编代码中的注释以 # 开头。



## 4.2.2 MIPS32汇编代码书写

数据段可以为汇编代码中所要用到的常量和全局变量申请空间，其格式为：

```
name: storage_type value(s)
```

其中**name**代表内存地址（标签）名，**storage\_type**代表数据类型，**value**代表初始值。常见的**storage\_type**有表7所列的几类。

下面是三个例子：

```
1  var1: .word 3          # create a single integer variable with
2                          # an initial value of 3
3  array1: .byte 'a','b'  # create a 2-element character array with
4                          # its elements initialized to a and b
5  array2: .space 40       # allocate 40 consecutive bytes, with storage
6                          # uninitialized; could be used as a 40-element
7                          # character array, or a 10-element integer array
```

## 4.2.2 目标代码生成提示

- 实验四需要在实验三的基础上完成。
- 在动手写代码之前，建议先熟悉 SPIM Simulator 的使用方法，然后写几个简单的 MIPS32 汇编程序使用 SPIM Simulator 运行一下，以确定是否已经清楚MIPS32代码应该如何书写。



中山大學

SUN YAT-SEN UNIVERSITY

软件工程学院

SCHOOL OF SOFTWARE ENGINEERING

完结撒花 ッ (@^▽^@)ノ