5. 目标代码生成

本章实验为**实验四**，其任务是在词法分析、语法分析、语义分析和中间代码生成程序的基础上，将C−−源代码翻译为MIPS32指令序列（可以包含伪指令），并在SPIM Simulator上运行。当你完成实验四之后，你就拥有了一个自己独立编写、可以实际运行的编译器。

选择MIPS作为目标体系结构是因为它属于RISC范畴，与x86等体系结构相比形式简单便于我们处理。如果你对于MIPS体系结构或汇编语言不熟悉并不要紧，我们会提供详细的参考资料。

需要注意的是，由于本次实验的代码会与之前实验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对于整个实验来讲相当重要。

5.1 实验内容

5.1.1 实验要求

为了完成实验四，我们建议你下载并安装SPIM Simulator用于对生成的目标代码进行检查和调试，SPIM Simulator的官方下载地址为：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MIPS32汇编语言的汇编器和模拟器，其最新的图形界面版本QtSPIM由于使用了Qt组件因而可以在各大操作系统平台如Windows、Linux、Mac等上运行，推荐安装。我们会在后面介绍有关SPIM Simulator的使用方法。

你需要做的就是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。我们要求你的程序能输出正确的汇编代码。“正确”是指该汇编代码在SPIM Simulator（命令行或Qt版本均可）上运行结果正确。因此，以下几个方面不属于检查范围：

1) 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 Simulator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应该存放参数、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。

2) 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MIPS32汇编代码）能运行正确。

当然，不检查并不代表不重要。我们建议你试着去遵守MIPS32中的各种约定，否则你的程序生成的目标代码在SPIM Simulator中运行时可能会出现一些意想不到的错误。

另外，实验四对作为输入的C−−源代码有如下的假设：

1) **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）。

2) **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。

3) **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) **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。

5) **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。

6) **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。

7) **假设7**：函数不会返回结构体或数组类型的值。

8) **假设8**：函数只会进行一次定义（没有函数声明）。

在进行实验四之前，请阅读后面的实验指导部分，以确保你已经了解MIPS32汇编语言以及SPIM Simulator的使用方法，这些内容是你顺利完成实验四的前提。

5.1.2 输入格式

你的程序的输入是一个包含C−−源代码的文本文件，你的程序需要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test1.cmm、输出文件名为out1.s，程序和输入文件都位于当前目录下，那么在Linux命令行下运行./cc test1.cmm out1.s即可将输出结果写入当前目录下名为out1.s的文件中。

5.1.3 输出格式

实验四要求你的程序将运行结果输出到文件。对于每个输入文件，你的程序应当输出相应的MIPS32汇编代码。我们将使用SPIM Simulator对你输出的汇编代码的正确性进行测试，任何能被SPIM Simulator执行并且结果正确的输出都将被接受。

5.1.4 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）：

1) GNU Linux Release: Ubuntu 20.04, kernel version 5.13.0-44-generic;

2) GCC version 7.5.0;

3) GNU Flex version 2.6.4;

4) GNU Bison version 3.5.1;

5) QtSPIM version 9.1.9。

一般而言，只要避免使用过于冷门的特性，使用其它版本的Linux或者GCC等，也基本上不会出现兼容性方面的问题。注意，实验四的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。

5.1.5 提交要求

实验四要求提交如下内容（同实验一）：

1) Flex、Bison以及C语言的可被正确编译运行的源程序。

2) 一份PDF格式的实验报告，内容包括：

   a) 你的程序实现了哪些功能？简要说明如何实现这些功能。清晰的说明有助于助教对你的程序所实现的功能进行合理的测试。

   b) 你的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你的程序。无法顺利编译将导致助教无法对你的程序所实现的功能进行任何测试，从而丢失相应的分数。

   c) 实验报告的长度不得超过三页！所以实验报告中需要重点描述的是你的程序中的亮点，是你认为最个性化、最具独创性的内容，而相对简单的、任何人都可以做的内容则可不提或简单地提一下，尤其要避免大段地向报告里贴代码。实验报告中所出现的最小字号不得小于五号字（或英文11号字）。

5.1.6 样例

实验四无选做要求，因此下面只列举**必做内容样例**。请仔细阅读样例，以加深对实验要求以及输出格式要求的理解。

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 }
```

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
```

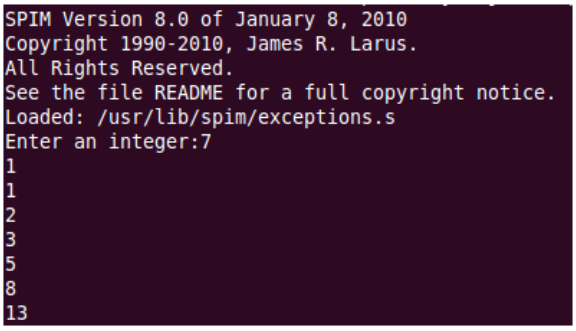

图15. 样例1汇编代码的运行结果。

该汇编代码在命令行SPIM Simulator中的运行结果如图15所示（输入7，则输出前7个Fibonacci数）。

样例2:

输入:

```
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;
}
```

输出:

该样例程序读入一个整数n，然后计算并输出n!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:

```
.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
```

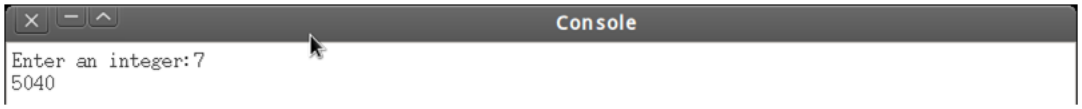

图16. 样例2汇编代码的运行结果。

```
16    syscall
17    li $v0, 4
18    la $a0, _ret
19    syscall
20    move $v0, $0
21    jr $ra
22
23  main:
24    addi $sp, $sp, -4
25    sw $ra, 0($sp)
26    jal read
27    lw $ra, 0($sp)
28    addi $sp, $sp, 4
29    move $t1, $v0
30    li $t3, 1
31    bgt $t1, $t3, label6
32    j label7
33  label6:
34    move $a0, $t1
35    addi $sp, $sp, -4
36    sw $ra, 0($sp)
37    jal fact
38    lw $ra, 0($sp)
39    addi $sp, $sp, 4
40    move $t2, $v0
41    j label8
42  label7:
43    li $t2, 1
44  label8:
45    move $a0, $t2
46    addi $sp, $sp, -4
47    sw $ra, 0($sp)
48    jal write
49    lw $ra, 0($sp)
50    addi $sp, $sp, 4
51    move $v0, $0
52    jr $ra
53
54  fact:
55    li $t4, 1
56    beq $a0, $t4, label1
57    j label2
58  label1:
59    move $v0, $a0
60    jr $ra
61  label2:
62    addi $sp, $sp, -8
63    sw $a0, ($sp)
64    sw $ra, 4($sp)
65    sub $a0, $a0, 1
66    jal fact
67    lw $a0, ($sp)
68    lw $ra, 4($sp)
69    addi $sp, $sp, 8
70    mul $v0, $v0, $a0
71    jr $ra
```

该汇编程序在QtSPIM中的运行结果如图16所示（输入7，输出5040）。

除了上面给的两个样例以外，你的程序要能够将其它符合假设的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SPIM Simulator上运行并得到正确结果来判断你的程序的正确性。