浙江大学

本科实验报告

课程名称：	计算机组成与设计
姓　　名：	王若鹏
学　　院：	信息与电子工程学院
专　　业：	电子科学与技术
学　　号：	3170105582
指导教师：	刘　鹏
上课时间：	周四 3, 4, 5 节

2019 年 11 月 26 日

专业：电子科学与技术
姓名：王若鹏
学号：3170105582
日期：2019.11.26
地点：教 7-202

浙江大学实验报告

课程名称：计算机组成与设计　指导老师：刘 鹏　成绩：_____

实验名称：RISC-V 编译器的设计　实验类型：设计型

一、实验目的

1、熟悉 RISC-V 指令集的实现

2、为一部分 RISC-V 指令集建立一个模拟器和非流水线处理器

二、实验任务与要求

给定 6 个文件: asm.c, riscv-small.c, Makefile, simple.s, simple.out, sim_simple.output.

1) asm.c：为简化的 RISC-V ISA 设计的汇编程序
2) riscv-small.c：一个全功能的非流水线模拟器
3) Makefile：生成二进制文件 asm（汇编程序）和 sim（模拟器）的 unix Makefile，分别来自源文件 asm.c 和 riscv small.c。
4) simple.s：一个汇编程序例子
5) simple.out：通过编译器 asm.c 运行 simple.s 的输出结果
6) sim_simple.output：通过模拟器 sim 运行 simple.out 的输出结果

要求：构建一个汇编程序和一个模拟器。遵照给定代码的格式，编写补全 asm.c 和 riscv small.c 文件，实现汇编程序和模拟器的功能。

三、实验主要仪器与设备

装有 VSCode、gcc、Linux 的电脑一台

四、实验原理

4.1 简化的 RISC-V ISA

为了模拟 RISC-V ISA 并实现 32 位体系结构。另外，为了保持模拟器的简单性，只需

要支持一个由 10 条指令组成的精简版 RISC-V ISA，如下表所示：

name	*format type*	*instruction format*					
beq	*SB-type*	*imm[12|10:5]*	*rs2*	*rs1*	*000*	*imm[4:1|11]*	*1100011*
lw	*I-type*	*imm[11:0]*		*rs1*	*010*	*rd*	*0000011*
sw	*S-type*	*imm[11:5]*	*rs2*	*rs1*	*010*	*imm[4:0]*	*0100011*
addi	*I-type*	*imm[11:0]*		*rs1*	*000*	*rd*	*0010011*
add	*R-type*	*0000000*	*rs2*	*rs1*	*000*	*rd*	*0110011*
sub	*R-type*	*0100000*	*rs2*	*rs1*	*000*	*rd*	*0110011*
sll	*R-type*	*0000000*	*rs2*	*rs1*	*001*	*rd*	*0110011*
srl	*R-type*	*0000000*	*rs2*	*rs1*	*101*	*rd*	*0110011*
or	*R-type*	*0000000*	*rs2*	*rs1*	*110*	*rd*	*0110011*
and	*R-type*	*0000000*	*rs2*	*rs1*	*111*	*rd*	*0110011*

Table 1: instruction encoding for a reducing RISC-V ISA

4.2 asm.c

有效的汇编程序是一个 ASCII 文件，其中每一行代表一条指令或一个数据常量。程序集代码行的格式为：

Label <tab> instruction <tab> field0 <tab> field1 <tab> field2 <tab> comments

行中最左边的字段是表示符号地址的 label 字段。有效标签最多包含 6 个字符，可以由字母和数字组成。标签是可选的，标签字段后面的选项卡不是。在可选标签之后是一个标签。然后跟随指令字段，其中指令可以是表 1 中列出的任何汇编语言助记符。在另一个选项卡之后是一系列字段。所有字段都以十进制数字表示。字段的数目取决于指令。

```
lw   rd   rs1  imm     Reg[rd] <- Mem[Reg[rs1] + imm]
sw   rs2  rs1  imm     Mem[Reg[rs1] + imm] <- Reg[rs2]
beq  rs1  rs2  imm     if (Reg[rs1] == Reg[rs2]) PC <- PC+imm
addi rd   rs1  imm     Reg[rd] <- Reg[rs1] + imm
add  rd   rs1  rs2     Reg[rd] <- Reg[rs1] + Reg[rs2]
sub  rd   rs1  rs2     Reg[rd] <- Reg[rs1] - Reg[rs2]
sll  rd   rs1  rs2     Reg[rd] <- Reg[rs1] << Reg[rs2]
srl  rd   rs1  rs2     Reg[rd] <- Reg[rs1] >> Reg[rs2]
and  rd   rs1  rs2     Reg[rd] <- Reg[rs1] & Reg[rs2]
or   rd   rs1  rs2     Reg[rd] <- Reg[rs1] | Reg[rs2]
halt       stop simulation
```

注意，在 beq 指令的情况下，使用 PC 相对寻址（同样，在计算 PC 相对分支目标时，模拟器不应执行左移位）。对于 lw、sw 和 beq 指令，imm 字段可以是十进制值，也可以使用标签。对于标签，汇编程序根据指令是 lw/sw 指令还是 beq 指令执行不同的操作。对于 lw

和 sw 指令，汇编程序插入与标签相对应的绝对地址。对于 beq 指令，汇编程序计算 PC 相对于标签的相对偏移量。

最后一个字段后面是另一个选项卡，然后是任何注释，注释在行的末尾结束。

除了指令外，汇编代码行还可以包括汇编程序的指令。.fill 指令告诉汇编程序将一个数字放入指令通常存储的位置。.fill 指令使用一个字段，可以是数值或符号地址。例如，“.fill 32”将值 32 放在通常存储指令的位置。

4.3 模拟器

在完成全部设计后，输入“make sim”来构建模拟器，这将生成一个可执行的 sim。在提供的程序上运行这个模拟器，发出的命令顺序如下：

make sim

./asm simple.s simple.out

./sim simple.out

要将这些消息放入一个文件中以便能够实际读取它们，可以通过输入以下代码将模拟器的输出重定向到一个文件：

./sim simple.out > sim.output

五、实验步骤

1、编写 asm.c

2、编写 riscv-small.c

3、运行给定测试程序 simple.s，观察输出结果 simple.out，simple.output，与 sim_simple.output 对比，检查程序的正确性

4、自主编写测试程序 mytest.c，对其他指令进行测试，进一步验证程序的正确性

六、实验记录与结果分析

6.1 asm.c

编写 asm.c 文件，先通过 strcmp 函数先识别不同的指令符号，然后按照要求生成 32 位的二进制指令。arg0、arg1、arg2 用于从左到右读取信息，根据不同的指令格式进行立即数和寄存器序号的生成及各个字段的移位处理。补全的代码如下：

```
/* STUDENT CODE START HERE. */

    else if (!strcmp(opcode, "lw")) { //I
        if (isNumber(arg2))
            immediateValue = atoi(arg2);
        else
            immediateValue = get_label_address(arg2);
        num = ((immediateValue & 0xFFF) << LW_IMM_SHIFT) | (atoi(arg1) << RS1_SHIFT) | LW_FUNC3<<FUNCT3_SHIFT | (atoi(arg0) << RD_
    }
    else if (!strcmp(opcode, "sw")) { //S
        if (isNumber(arg2))
            immediateValue = atoi(arg2);
        else
            immediateValue = get_label_address(arg2);
        num = ((immediateValue & 0xFE0) << LW_IMM_SHIFT) | (atoi(arg0) << RS2_SHIFT) | (atoi(arg1) << RS1_SHIFT) | SW_FUNC3<<FUNCT
    }
    else if (!strcmp(opcode, "beq")) { //SB
        if (isNumber(arg2))
            immediateValue = atoi(arg2);
        else
            immediateValue = get_label_address(arg2);
        num = ((immediateValue & 0x1000) << 19) | ((immediateValue & 0x7E0) << 20) | (atoi(arg1) << RS2_SHIFT) | (atoi(arg0) << R
    }
    else if (!strcmp(opcode, "addi")) { //I
        num = ((atoi(arg2) & 0xFFF) << ADDI_IMM_SHIFT) | (atoi(arg1) << RS1_SHIFT) | ADDI_FUNC3<<FUNCT3_SHIFT | (atoi(arg0) << RD_
    }
    else if (!strcmp(opcode, "sub")) { //R
        num = (SUB_FUNC7 << FUNCT7_SHIFT) | (atoi(arg2) << RS2_SHIFT) | (atoi(arg1) << RS1_SHIFT) | SUB_FUNC3<<FUNCT3_SHIFT | (ato
    }
    else if (!strcmp(opcode, "sll")) { //R
        num = (SLL_FUNC7 << FUNCT7_SHIFT) | (atoi(arg2) << RS2_SHIFT) | (atoi(arg1) << RS1_SHIFT) | SLL_FUNC3<<FUNCT3_SHIFT | (ato
    }
    else if (!strcmp(opcode, "srl")) { //R
        num = (SRL_FUNC7 << FUNCT7_SHIFT) | (atoi(arg2) << RS2_SHIFT) | (atoi(arg1) << RS1_SHIFT) | SRL_FUNC3<<FUNCT3_SHIFT | (ato
    }
    else if (!strcmp(opcode, "and")) { //R
        num = (AND_FUNC7 << FUNCT7_SHIFT) | (atoi(arg2) << RS2_SHIFT) | (atoi(arg1) << RS1_SHIFT) | AND_FUNC3<<FUNCT3_SHIFT | (ato
    }
    else if (!strcmp(opcode, "or")) { //R
        num = (OR_FUNC7 << FUNCT7_SHIFT) | (atoi(arg2) << RS2_SHIFT) | (atoi(arg1) << RS1_SHIFT) | OR_FUNC3<<FUNCT3_SHIFT | (atoi(
    }

/* STUDENT CODE END. */
```

6.2 riscv-small.c

编写 riscv-small.c 文件，首先对 PC 加 4，然后根据指令格式，确定 rs2、rs1、目标寄存器 rd、立即数 imm、funct7、funct3 的值。由于取到的 imm 是未经处理过的，且 I、SB、S 型对立即数的格式不同，所以对立即数 imm 按照不同类型的指令格式分别进行移位处理，得到真正的立即数 immField。补全的代码如下：

```
/* STUDENT CODE START HERE. */

    state->pc += 4;
    rs2 = (state->mem[word_pc] >> RS2_SHIFT) & REG_MASK; //24-20
    rs1 = (state->mem[word_pc] >> RS1_SHIFT) & REG_MASK; //19-15
    rd = (state->mem[word_pc] >> RD_SHIFT) & REG_MASK; //11-7
    imm = (state->mem[word_pc]) & 0xFFF00F80; //immediate number
    func7 = (state->mem[word_pc] >> FUNCT7_SHIFT) & FUNC7_MASK; //31-25
    func3 = (state->mem[word_pc] >> FUNCT3_SHIFT) & FUNC3_MASK; //14-12

    if (opcode == LW_OP||opcode == ADDI_OP) { //I
        immField = (imm >> 20);
    }
    else if (opcode == BEQ_OP) { //SB
        immField = ((imm & 0x1 << 31) >> 19) | ((imm & 0x3F << 25) >> 20) | ((imm & 0xF << 8) >> 7) | ((imm & 0x1 << 7) << 4);
    }
    else if (opcode == SW_OP) { //S
        immField = ((imm & 0x7F << 25) >> 20) | ((imm & 0x1F << 7) >> 7);
    }

/* STUDENT CODE END. */
```

接着根据运算要求，把运算结果存入 rd 寄存器中，此处代码较为容易。但是要注意 sw 指令，mem 中的数要除以 4（右移 2 位）。

```
153   /* STUDENT CODE START HERE. */
154       else if (opcode == BEQ_OP) { //SB
155           if (state->reg[rs1] == state->reg[rs2])
156               state->pc += immField;
157       }
158       else if (opcode == LW_OP) { //I
159           state->reg[rd] = state->mem[(state->reg[rs1]+immField)>>2];
160       }
161       else if (opcode == SW_OP) { //S
162           state->mem[(state->reg[rs1]+immField)>>2] = state->reg[rs2];
163       }
164       else if (opcode == REG_REG_OP) { //R
165           if (func3 == ADD_FUNC3 && func7 == ADD_FUNC7) {
166               state->reg[rd] = state->reg[rs1] + state->reg[rs2];
167           }
168           else if (func3 == SUB_FUNC3 && func7 == SUB_FUNC7) {
169               state->reg[rd] = state->reg[rs1] - state->reg[rs2];
170           }
171           else if (func3 == SLL_FUNC3 && func7 == SLL_FUNC7) {
172               state->reg[rd] = state->reg[rs1] << state->reg[rs2];
173           }
174           else if (func3 == SRL_FUNC3 && func7 == SRL_FUNC7) {
175               state->reg[rd] = state->reg[rs1] >> state->reg[rs2];
176           }
177           else if (func3 == OR_FUNC3 && func7 == OR_FUNC7) {
178               state->reg[rd] = state->reg[rs1] | state->reg[rs2];
179           }
180           else if (func3 == AND_FUNC3 && func7 == AND_FUNC7) {
181               state->reg[rd] = state->reg[rs1] & state->reg[rs2];
182           }
183       }
184   /* STUDENT CODE END. */
```

6.3 案例测试 simple.s

按照要求在终端运行测试案例 simple.s 输出结果如下。将生成的 sim.output 与 sim_simple.output 相对比，运行结果完全正确。

```
问题  输出  调试控制台  终端                                  1: zsh

Wrp-MacBookPro% make all
make: Nothing to be done for `all'.
Wrp-MacBookPro% make sim
make: `sim' is up to date.
Wrp-MacBookPro% ./asm simple.s simple.out
Wrp-MacBookPro% ./sim simple.out
memory[0]=01c02a83
memory[1]=02002b03
memory[2]=02402b83
memory[3]=016a8c33
memory[4]=417c0cb3
memory[5]=03902423
memory[6]=0000003f
memory[7]=00000032
memory[8]=0000001e
memory[9]=00000014
memory[10]=00000000

state after cycle 0:
        pc=4
        memory:
                mem[0] 0x1c02a83        (29371011)
                mem[1] 0x2002b03        (33565443)
                mem[2] 0x2402b83        (37759875)
                mem[3] 0x16a8c33        (23759923)
                mem[4] 0x417c0cb3       (1098648755)
                mem[5] 0x3902423        (59778083)
                mem[6] 0x3f     (63)
                mem[7] 0x32     (50)
                mem[8] 0x1e     (30)
                mem[9] 0x14     (20)
                mem[10] 0x0     (0)
        registers:
                reg[0] 0x0      (0)
                reg[1] 0x0      (0)
                reg[2] 0x0      (0)
                reg[3] 0x0      (0)
```

6.4 自主测试 mytest.s

对其他在 simple.s 里未使用过的指令（beq、addi、sll、srl、and、or），自主编写测试样例如下，本样例可以测试所有未曾测试的指令：

mytest.s

```
1       lw  21  0   op1
2       lw  22  0   op2
3       lw  23  0   op3
4       beq 21  21  label
5       addi    21  22  50
6   label   sll 24  21  23
7       beq 21  21  8
8       addi    21  22  50
9       sll 24  21  23
10      srl 25  21  23
11      and 26  21  22
12      or  27  21  22
13      addi    28  22  10
14      sw  29  0   answer
15  done    halt
16  op1 .fill   50
17  op2 .fill   30
18  op3 .fill   4
19  answer  .fill   0
```

输出测试结果至 mytest.output，进行分析：

（1）运行至 PC=12，beq 条件成立，直接跳转至 PC＝label＝20，符合预期；运行至 PC=24，beq 条件成立，直接跳转至 PC = 24 + 8 = 32，符合预期。跳转指令的下一行 addi 均没有执行，说明跳转成功。

（2）观察最后的运行结果，符合预期：

reg[21] = 50；reg[22] = 30；reg[23] = 4

reg[24] = reg[21] << reg[23] = 50 << 4 = 800

reg[25] = reg[21] >> reg[23] = 50 >> 4 = 3

reg[26] = reg[21] & reg[22] = 110010 & 011110 = 010010 = 18

reg[27] = reg[21] | reg[22] = 110010 | 011110 = 111110 = 62

reg[28] = reg[22] + 10 = 30 + 10 = 40

```
reg[20] 0x0 (0)
reg[21] 0x32    (50)
reg[22] 0x1e    (30)
reg[23] 0x4 (4)
reg[24] 0x320   (800)
reg[25] 0x3 (3)
reg[26] 0x12    (18)
reg[27] 0x3e    (62)
reg[28] 0x28    (40)
reg[29] 0x0 (0)
```

综上：RISC-V ISA 模拟器**成功实现**。

附录：代码

1、asm.c

```
/* STUDENT CODE START HERE. */
        else if (!strcmp(opcode, "lw")) { //I
            if (isNumber(arg2))
                immediateValue = atoi(arg2);
            else
                immediateValue = get_label_address(arg2);
            num = ((immediateValue & 0xFFF) << LW_IMM_SHIFT) | (atoi(arg1) <<
RS1_SHIFT) | LW_FUNC3<<FUNCT3_SHIFT | (atoi(arg0) << RD_SHIFT) | LW_OP;
        }
        else if (!strcmp(opcode, "sw")) { //S
            if (isNumber(arg2))
                immediateValue = atoi(arg2);
            else
                immediateValue = get_label_address(arg2);
            num = ((immediateValue & 0xFE0) << LW_IMM_SHIFT) | (atoi(arg0) <<
RS2_SHIFT) | (atoi(arg1) << RS1_SHIFT) | SW_FUNC3<<FUNCT3_SHIFT | ((immediateValue
& 0x1F) << RD_SHIFT) | SW_OP;
        }
        else if (!strcmp(opcode, "beq")) { //SB
            if (isNumber(arg2))
                immediateValue = atoi(arg2);
            else
                immediateValue = get_label_address(arg2)-address;
            num = ((immediateValue & 0x1000) << 19) | ((immediateValue & 0x7E0) << 20) |
(atoi(arg1) << RS2_SHIFT) | (atoi(arg0) << RS1_SHIFT) | BEQ_FUNC3<<FUNCT3_SHIFT |
((immediateValue & 0x1E) << 7) | ((immediateValue & 0x800) >> 4) | BEQ_OP;
        }
        else if (!strcmp(opcode, "addi")) { //I
            num = ((atoi(arg2) & 0xFFF) << LW_IMM_SHIFT) | (atoi(arg1) << RS1_SHIFT) |
ADDI_FUNC3<<FUNCT3_SHIFT | (atoi(arg0) << RD_SHIFT) | ADDI_OP;
        }
        else if (!strcmp(opcode, "sub")) { //R
            num = (SUB_FUNC7 << FUNCT7_SHIFT) | (atoi(arg2) << RS2_SHIFT) |
(atoi(arg1) << RS1_SHIFT) | SUB_FUNC3<<FUNCT3_SHIFT | (atoi(arg0) << RD_SHIFT) |
REG_REG_OP;
        }
        else if (!strcmp(opcode, "sll")) { //R
            num = (SLL_FUNC7 << FUNCT7_SHIFT) | (atoi(arg2) << RS2_SHIFT) |
(atoi(arg1) << RS1_SHIFT) | SLL_FUNC3<<FUNCT3_SHIFT | (atoi(arg0) << RD_SHIFT) |
```

```
REG_REG_OP;
        }
        else if (!strcmp(opcode, "srl")) { //R
            num = (SRL_FUNC7 << FUNCT7_SHIFT) | (atoi(arg2) << RS2_SHIFT) |
(atoi(arg1) << RS1_SHIFT) | SRL_FUNC3<<FUNCT3_SHIFT | (atoi(arg0) << RD_SHIFT) |
REG_REG_OP;
        }
        else if (!strcmp(opcode, "and")) { //R
            num = (AND_FUNC7 << FUNCT7_SHIFT) | (atoi(arg2) << RS2_SHIFT) |
(atoi(arg1) << RS1_SHIFT) | AND_FUNC3<<FUNCT3_SHIFT | (atoi(arg0) << RD_SHIFT) |
REG_REG_OP;
        }
        else if (!strcmp(opcode, "or")) { //R
            num = (OR_FUNC7 << FUNCT7_SHIFT) | (atoi(arg2) << RS2_SHIFT) | (atoi(arg1)
<< RS1_SHIFT) | OR_FUNC3<<FUNCT3_SHIFT | (atoi(arg0) << RD_SHIFT) | REG_REG_OP;
        }

  /* STUDENT CODE END. */
```

2、riscv-small.c

```
/* STUDENT CODE START HERE. */
        state->pc += 4;
        rs2 = (state->mem[word_pc] >> RS2_SHIFT) & REG_MASK; //24-20
        rs1 = (state->mem[word_pc] >> RS1_SHIFT) & REG_MASK; //19-15
        rd = (state->mem[word_pc] >> RD_SHIFT) & REG_MASK; //11-7
        imm = (state->mem[word_pc]) & 0xFFF00F80; //immediate number
        func7 = (state->mem[word_pc] >> FUNCT7_SHIFT) & FUNC7_MASK; //31-25
        func3 = (state->mem[word_pc] >> FUNCT3_SHIFT) & FUNC3_MASK; //14-12

        if (opcode == LW_OP||opcode == ADDI_OP) { //I
            immField = (imm >> 20);
        }
        else if (opcode == BEQ_OP) { //SB
            immField = ((imm & 0x1 << 31) >> 19) | ((imm & 0x3F << 25) >> 20) | ((imm &
0xF << 8) >> 7) | ((imm & 0x1 << 7) << 4);
        }
        else if (opcode == SW_OP) { //S
            immField = ((imm & 0x7F << 25) >> 20) | ((imm & 0x1F << 7) >> 7);
        }
  /* STUDENT CODE END. */

/* STUDENT CODE START HERE. */
```

```
        else if (opcode == BEQ_OP) { //SB
            if (state->reg[rs1] == state->reg[rs2])
                state->pc += immField-4;
        }
        else if (opcode == LW_OP) { //I
            state->reg[rd] = state->mem[(state->reg[rs1]+immField)>>2];
        }
        else if (opcode == SW_OP) { //S
            state->mem[(state->reg[rs1]+immField)>>2] = state->reg[rs2];
        }
        else if (opcode == REG_REG_OP) { //R
            if (func3 == ADD_FUNC3 && func7 == ADD_FUNC7) {
                state->reg[rd] = state->reg[rs1] + state->reg[rs2];
            }
            else if (func3 == SUB_FUNC3 && func7 == SUB_FUNC7) {
                state->reg[rd] = state->reg[rs1] - state->reg[rs2];
            }
            else if (func3 == SLL_FUNC3 && func7 == SLL_FUNC7) {
                state->reg[rd] = state->reg[rs1] << state->reg[rs2];
            }
            else if (func3 == SRL_FUNC3 && func7 == SRL_FUNC7) {
                state->reg[rd] = state->reg[rs1] >> state->reg[rs2];
            }
            else if (func3 == OR_FUNC3 && func7 == OR_FUNC7) {
                state->reg[rd] = state->reg[rs1] | state->reg[rs2];
            }
            else if (func3 == AND_FUNC3 && func7 == AND_FUNC7) {
                state->reg[rd] = state->reg[rs1] & state->reg[rs2];
            }
        }

/* STUDENT CODE END. */
```