

计算机体系结构实验 第四周实验报告

组员：许诗瑶 20023105、刘朝润 20023114、刘晓航 20020070

目录

一、实验要求	2
(一) 5 级流水线功能扩增	2
(二) RISC-V ISA 模拟器使用	2
(三) 5 级流水线路序测试	2
二、实验环境	2
三、 实验内容	2
(一) 5 级流水线功能扩增	2
1、确定新增指令	2
2、浮点乘除	3
3、ECALL 指令及 CSR	5
(二) RISC-V ISA 模拟器使用	6
1、模拟器介绍	6
2、修改模拟器输出	7
四、实验结果	8
(一) 5 级流水线测试手写 Fibonacci	8
(二) 5 级流水线测试手写 Bubble Sort	9
(三) 模拟器测试 C 语言 Fibonacci	11
(四) 模拟器测试 Bubble Sort by C	15
(五) 浮点测试结果	18
五、实验总结	20
(一) 实验分工	20
(二) 实验遇到的问题	20
(三) 附件文件说明	21

一、实验要求

（一）5 级流水线功能扩增

针对上周根据带 printf 测试程序整理出的目标指令集，在原有 5 级流水线上扩增浮点乘除、ECALL 及相关 CSR 指令的功能。

（二）RISC-V ISA 模拟器使用

选择 Spike RISC-V ISA 模拟器用做实验结果的对比，使用模拟器运行带有 printf 的测试程序，并使之方便比较运行时状态。

（三）5 级流水线程序测试

将实现功能扩增后的 5 级流水线运行此前手写 RV 汇编和带 printf 的 C 语言的 Fibonacci 和 Bubble Sort 测试程序，通过与模拟器结果比较调试。

二、实验环境

编程语言：Verilog

IDE：Vivado 2018.3

编译工具链：riscv-gnu-tool

工程版本控制及代码托管：Github 平台

RISC-V ISA 模拟器：Spike

三、实验内容

（一）5 级流水线功能扩增

1、确定新增指令

由于上周对测试程序编译时使用了 C 扩展，编译结果的指令中产生了 16 位长指令。为方便实验，编译时不使用 C 扩展功能，统一指令为 32 位定长，重新编译形成测试程序。

由于通过静态链接后测试程序指令规模庞大（约两万多行），通过**编写 python 程序辅助筛选**整理指令集。其中.py 程序代码及其筛选结果如下所示：

```

1 instruction = []
2 instruction16 = []
3
4 bubble_sort_ins = open("./BubbleSort.txt", "r")
5
6 lines = bubble_sort_ins.readlines()
7
8 for line in lines:
9     temp = line.split()
10    if len(temp) >= 3 and temp[0][len(temp[0]) - 1] == ":" and temp[2] != "format":
11        instruction.append(temp[2])
12        if len(temp[1]) == 4:
13            instruction16.append(temp[2])
14
15 bubble_sort_ins.close()
16
17 bubble_sort_ins = open("./BubbleSort_ass.txt", "r")
18
19 lines = bubble_sort_ins.readlines()
20
21 for line in lines:
22     temp = line.split()
23     if len(temp) >= 3 and temp[0][len(temp[0]) - 1] == ":" and temp[2] != "format":
24         instruction.append(temp[2])
25         if len(temp[1]) == 4:
26             instruction16.append(temp[2])
27
28 bubble_sort_ins.close()
29
30 fibonacci_ins = open("./Fibonacci.txt", "r")
31
32 lines = fibonacci_ins.readlines()
33
34 for line in lines:
35     temp = line.split()
36     if len(temp) >= 3 and temp[0][len(temp[0]) - 1] == ":" and temp[2] != "format":
37         instruction.append(temp[2])
38         if len(temp[1]) == 4:
39             instruction16.append(temp[2])
40
41 fibonacci_ins.close()
42

```

```

66
67 ['andi', 'seqz', 'csrwi', 'ret', 'and', 'srl', 'remu', 'fsd', 'bge', 'div', 'blez', 'snez', 'sll', 'lhu', 'ori', 'sb', 'bgeu', 'jalr', 'bgtz', 'ecall', 'mv', 'sh', 'mul', 'xori', 'neg', 'xor', 'sw',
68 'jr', 'csrwi', 'lri', 'lw', 'jal', 'lb', 'sub', 'divu', 'ori', 'xext.b', 'slt', 'frrm', 'sltu', 'bltu', 'lbu', 'not', 'rem', 'beq', 'sltiu', 'bgez', 'bnez', 'fdiv.d', 'mulhu', 'addi', 'fld', 'blt', '
69 'slli', 'beqz', 'lui', 'bne', 'jrr', 'srli', 'bltz', 'sra', 'add', 'srli', 'auipc', 'lh', 'fmul.d']

```

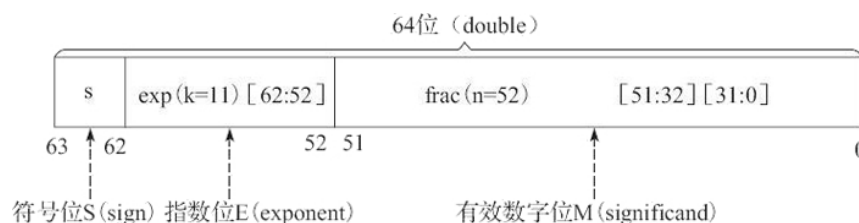
整理涉及到的指令与此前 5 级流水线实现的指令集比较，确定需要新增的指令为：FMUL.D、FDIV.S、FSD、FLD 和 ECALL。明确 5 级流水线需要新增的功能为浮点乘除及 ECALL 指令。

2、浮点乘除

RISC-V 使用的浮点数标准为 IEEE754-2008 标准，我们基于该标准和 RISC-V 官方文档中关于 FD 拓展的描述实现浮点指令 FMUL.D、FDIV.S、FSD 和 FLD 四条指令。该四条指令主要针对双精度浮点数进行处理。

(1) 浮点表示形式

使用 IEEE754 标准中的 64 位浮点数，其格式如下：



其中，frac[31: 0] 存放小数段的低 32 位（即第 0 位存放整个小数段的最低有效位 LSB，第 31 位存放小数段低 32 位的最高有效位 MSB）；frac[51: 32] 存放小数段的高 20 位（即第 32 位存放高 20 位的最低有效位 LSB，第 51 位存放整个小数段的最高有效位 MSB）；第 52 位存放指数段 exp 的最低有效位 LSB，第 62 位存放指数段 exp 的最高有效位 MSB；最高位，即第 63 位存放符号 s。

我们知道，指数可以为正数，也可以为负数。为了处理负指数的情况，实际的指数值按要求需要加上一个偏置（Bias）值作为保存在指数段中的值。因此，这种情况下的指数段被解释为以偏置形式表示的有符号整数。即指数的值为： $E=e^{-\text{Bias}}$ 。

其中， e 是无符号数，其位表示为 $e^{k-1}\cdots e^1e^0$ ，而 Bias 是一个等于 $2^{k-1}-1$ （单精度是 127，双精度是 1023）的偏置值。由此产生指数的取值范围是：单精度为 $-126\sim+127$ ，双精度为 $-1022\sim+1023$ 。

对小数段 frac，可解释为描述小数值 f ，其中 $0\leq f<1$ ，其二进制表示为 $0.f^{n-1}\cdots f^1f^0$ ，也就是二进制小数点在最高有效位的左边。有效数字定义为 $M=1+f$ 。有时候，这种方式也叫作隐含的以 1 开头的表示法，因为我们可以把 M 看成一个二进制表达式为 $1.f^{n-1}f^{n-2}\cdots f^0$ 的数字。既然我们总是能够调整指数 E ，使得有效数字 M 的范围为 $1\leq M<2$ （假设没有溢出），那么这种表示方法是一种轻松获得一个额外精度位的技巧。同时，由于第一位总是等于 1，因此我们就不需要显式地表示它。

特殊数值表示如下：

形式	指数	小数部分
零	0	0
非正规形式	0	大于0小于1
正规形式	1到 $2^e - 2$	大于等于1小于2
无穷	$2^e - 1$	0
NaN	$2^e - 1$	非0

（2）浮点乘法与除法的实现

浮点乘法的实现原理如下：两个数的符号位相乘位符号位，阶码相加再减去 bias，小数位相乘选取前 52 位，再取后三位为舍入位。

浮点除法的实现原理如下：两个数的符号位相乘位符号位，阶码相减再加上 bias，被除数扩展位数 55 位与除数相除得到结果取前 52 位，后三位为舍入位。

（3）舍入与异常

IEEE754 标准中共有五种舍入模式，分别如下表：

舍入模式	说明
RNE	向距离最近的偶数舍入
RTZ	向 0 的方向舍入
RDN	向负无穷大方向舍入
RUP	向正无穷大方向舍入
RMM	向距离最近的最大数量级数舍入

在指令执行过程中有 3 位 rm 码来确定舍入模式，其中 3 位 rm 码中有 6 种舍入模式，五种与上表相同，还有一种则是根据 csr 种 fflags 寄存器中的 7 位到 5 位设定的模式舍入。

浮点操作中共有五种异常，无效操作、除数为 0，上溢、下溢和不精确。分别为与 csr 寄存器中的 fflags 中的后五位。

无效操作，当乘法中出现 NaN，0*无穷，无穷*0，除法中出现 NaN，0/无穷等会导致无效操作异常。

除数为 0，当浮点除法中的除数为 0。

上溢与下溢

11 位阶码：表示范围 $-1022\sim+1023$

0 0000 0000 0001 ~ 0 0011 1111 1110 $-1022\sim-1$

0 0011 1111 1111 0

0 0100 0000 0001 ~ 0 0111 1111 1110 $1\sim+1023$

阶码 下溢

负数+负数

$000\ 0000\ 0001 + 011\ 1111\ 1110 - 011\ 1111\ 1111 = 0\ 0000\ 0000\ 0000$

$000\ 0000\ 0001 + 011\ 1111\ 1101 - 011\ 1111\ 1111 = 1\ 1111\ 1111\ 1111$

$000\ 0000\ 0001 + 011\ 1111\ 1100 - 011\ 1111\ 1111 = 1\ 1111\ 1111\ 1110$

$000\ 0000\ 0001 + 011\ 1111\ 1011 - 011\ 1111\ 1111 = 1\ 1111\ 1111\ 1101$

• • • • •

$000\ 0000\ 0001 + 000\ 0000\ 0001 - 011\ 1111\ 1111 = 1\ 1100\ 0000\ 0011$

负数-正数

$000\ 0000\ 0001 - 100\ 0000\ 0001 + 011\ 1111\ 1111 = 1\ 1111\ 1111\ 1111$

$000\ 0000\ 0001 - 100\ 0000\ 0010 + 011\ 1111\ 1111 = 1\ 1111\ 1111\ 1110$

$000\ 0000\ 0001 - 100\ 0000\ 0011 + 011\ 1111\ 1111 = 1\ 1111\ 1111\ 1101$

• • • • •

$000\ 0000\ 0001 - 111\ 1111\ 1110 + 011\ 1111\ 1111 = 1\ 1100\ 0000\ 0010$

上溢

正数-负数

$111\ 1111\ 1110 - 011\ 1111\ 1110 + 011\ 1111\ 1111 = 0\ 0111\ 1111\ 1111$

$111\ 1111\ 1110 - 011\ 1111\ 1101 + 011\ 1111\ 1111 = 0\ 1000\ 0000\ 0000$

$111\ 1111\ 1110 - 011\ 1111\ 1100 + 011\ 1111\ 1111 = 0\ 1000\ 0000\ 0001$

• • • • •

$111\ 1111\ 1110 - 000\ 0000\ 0001 + 011\ 1111\ 1111 = 0\ 1011\ 1111\ 1100$

正数+ 正数

$111\ 1111\ 1110 + 100\ 0000\ 0001 - 011\ 1111\ 1111 = 0\ 1000\ 0000\ 0000$

$111\ 1111\ 1110 + 100\ 0000\ 0010 - 011\ 1111\ 1111 = 0\ 1000\ 0000\ 0001$

$111\ 1111\ 1110 + 100\ 0000\ 0011 - 011\ 1111\ 1111 = 0\ 1000\ 0000\ 0010$

• • • • •

$111\ 1111\ 1110 + 111\ 1111\ 1110 - 011\ 1111\ 1111 = 0\ 1011\ 1111\ 1101$

阶码使用无符号表示，将乘除结果的阶码位设为 12 位，

if [11:10] == 11 or [10:0] == 000 0000 0000 是下溢

else if [11:10] == 10 or [10:0] == 111 1111 1111 是上溢

3、ECALL 指令及 CSR

查阅 RISC-V 有关特权指令，涉及到 CSR 相关的寄存器，其中 CSR 寄存器虽有 2^{12} 个，但其中有特殊些个寄存器用于系统环境状态的控制等，如下表所示：

Number	Privilege	Name	Description
User Trap Setup			
0x000	URW	ustatus	User status register.
0x004	URW	uie	User interrupt-enable register.
0x005	URW	utvec	User trap handler base address.
User Trap Handling			
0x040	URW	uscratch	Scratch register for user trap handlers.
0x041	URW	uepc	User exception program counter.
0x042	URW	ucause	User trap cause.
0x043	URW	utval	User bad address or instruction.
0x044	URW	uip	User interrupt pending.
User Floating-Point CSRs			
0x001	URW	fflags	Floating-Point Accrued Exceptions.
0x002	URW	frm	Floating-Point Dynamic Rounding Mode.
0x003	URW	fcsr	Floating-Point Control and Status Register (frm + fflags).
User Counter/Timers			
0xC00	URO	cycle	Cycle counter for RDCYCLE instruction.
0xC01	URO	time	Timer for RDTIME instruction.
0xC02	URO	instret	Instructions-retired counter for RDINSTRET instruction.
0xC03	URO	hpmcounter3	Performance-monitoring counter.
0xC04	URO	hpmcounter4	Performance-monitoring counter.
		:	
0xC1F	URO	hpmcounter31	Performance-monitoring counter.
0xC80	URO	cycleh	Upper 32 bits of cycle , RV32 only.
0xC81	URO	timeh	Upper 32 bits of time , RV32 only.
0xC82	URO	instreth	Upper 32 bits of instret , RV32 only.
0xC83	URO	hpmcounter3h	Upper 32 bits of hpmcounter3 , RV32 only.
0xC84	URO	hpmcounter4h	Upper 32 bits of hpmcounter4 , RV32 only.
		:	
0xC9F	URO	hpmcounter31h	Upper 32 bits of hpmcounter31 , RV32 only.

其中查阅到 ECALL 指令的作用为，将 mepc 寄存器中的值设为当前 PC 值。

ECALL and EBREAK cause the receiving privilege mode's **epc** register to be set to the address of the **ECALL** or EBREAK instruction itself, *not* the address of the following instruction. **As ECALL and EBREAK cause synchronous exceptions, they are not considered to retire, and should not increment the minstret CSR.**

(二) RISC-V ISA 模拟器使用

1、模拟器介绍

由于带有 printf 指令的反汇编代码共有 2w 多行，而我们实现的五级流水线并没有可以 printf 的外设，因此如何验证最终结果中的寄存器数值正确是一个重要问题。我们可以使用 riscv 指令集模拟器执行相关代码并查看最终寄存器状态来确实最终执行结果是否正确，因此我们选择 spike (<https://github.com/riscv/riscv-isa-sim>) 作为 riscv 指令集模拟器，同时由于我们仅需要模拟程序运行，因此需要使用 riscv-pk (<https://github.com/riscv/riscv-pk>) 作为代理内核。

模拟器安装，首先将 riscv-isa-sim 和 riscv-pk 从 github 上 clone 到本地。首先安装 riscv-pk。

```
cd riscv-pk
mkdir build
cd build
../configure --prefix=/home/lcr/riscv --host=riscv32-unknown-elf --with-arch=rv32imfd --with-abi=ilp32d
make
make install
```

```

安装 riscv-isa-sim
apt-get install device-tree-compiler
cd riscv- isa-sim
mkdir build
cd build
../configure --prefix=/home/lcr/riscv --with-isa=RV32IMFD
make
make install

```

2、修改模拟器输出

由于模拟器仅在交互模式下才可以查看整数和浮点寄存器，无法查看 csr 寄存器状态，因此我们通过修改 riscv-isa-sim 下 riscv 文件下 processor.cc 和 processor.h 文件向日志文件中增加输出寄存器状态。核心代码如下：

```

803 //fibonacci 324643 bubble 326091
804 if (state.counter_add >= 324640) {
805     // XPR
806     fprintf(log_file, "===== XPR =====\n");
807     for(int i = 0; i < 32; i++){
808         fprintf(log_file, "XPR[%d]: 0x%08" PRIx64 "\n", i, state.XPR[i]);
809     }
810 }
811 // FPR
812 fprintf(log_file, "===== FPR =====\n");
813 for(int i = 0; i < 32; i++){
814     fprintf(log_file, "FPR[%d]: 0x%08" PRIx64 "\n", i, state.FPR[i]);
815 }
816 // CSR
817 fprintf(log_file, "===== CSR =====\n");
818 fprintf(log_file, "pc: 0x%08" PRIx64 "\n prv: 0x%08" PRIx64 "\n misa: 0x%08" PRIx64 "\n",
819     state.pc, state.prv, state.misa);
820 fprintf(log_file, "mstatus: 0x%08" PRIx64 "\n mepc: 0x%08" PRIx64 "\n mtval: 0x%08" PRIx64 "\n",
821     state.mstatus, state.mepc, state.mtval);
822 fprintf(log_file, "mscratch: 0x%08" PRIx64 "\n mtvec: 0x%08" PRIx64 "\n mcause: 0x%08" PRIx64 "\n",
823     state.mscratch, state.mtvec, state.mcause);
824 fprintf(log_file, "minstret: 0x%08" PRIx64 "\n mie: 0x%08" PRIx64 "\n mip: 0x%08" PRIx64 "\n",
825     state.minstret, state.mie, state.mip);
826 fprintf(log_file, "medeleg: 0x%08" PRIx64 "\n mideleg: 0x%08" PRIx64 "\n mcounteren: 0x%08" PRIx64 "\n",
827     state.medeleg, state.mideleg, state.mcounteren);
828 fprintf(log_file, "scounteren: 0x%08" PRIx64 "\n sepc: 0x%08" PRIx64 "\n stval: 0x%08" PRIx64 "\n",
829     state.scounteren, state.sepc, state.stval);
830 fprintf(log_file, "sscratch: 0x%08" PRIx64 "\n stvec: 0x%08" PRIx64 "\n satp: 0x%08" PRIx64 "\n",
831     state.sscratch, state.stvec, state.satp);
832 fprintf(log_file, "scause: 0x%08" PRIx64 "\n mtval2: 0x%08" PRIx64 "\n mtinst: 0x%08" PRIx64 "\n",
833     state.scause, state.mtval2, state.mtinst);
834 fprintf(log_file, "hstatus: 0x%08" PRIx64 "\n hedeleg: 0x%08" PRIx64 "\n hedeleg: 0x%08" PRIx64 "\n",
835     state.hstatus, state.hedeleg, state.hedeleg);
836 fprintf(log_file, "hcounteren: 0x%08" PRIx64 "\n htval: 0x%08" PRIx64 "\n htinst: 0x%08" PRIx64 "\n",
837     state.hcounteren, state.htval, state.htinst);
838 fprintf(log_file, "hgatp: 0x%08" PRIx64 "\n vsstatus: 0x%08" PRIx64 "\n vstvec: 0x%08" PRIx64 "\n",
839     state.hgatp, state.vsstatus, state.vstvec);
840 fprintf(log_file, "vsscratch: 0x%08" PRIx64 "\n vsepc: 0x%08" PRIx64 "\n vscause: 0x%08" PRIx64 "\n",
841     state.vsscratch, state.vsepc, state.vscause);
842

```

```

833     state.sscratch, state.stvec, state.satp);
834     fprintf(log_file, "scause: 0x%08" PRIx64 "\n mtval2: 0x%08" PRIx64 "\n mtinst: 0x%08" PRIx64 "\n",
835             state.scause, state.mtval2, state.mtinst);
836     fprintf(log_file, "hstatus: 0x%08" PRIx64 "\n hideleg: 0x%08" PRIx64 "\n hedeleg: 0x%08" PRIx64 "\n",
837             state.hstatus, state.hideleg, state.hedeleg);
838     fprintf(log_file, "hcounteren: 0x%08" PRIx64 "\n htval: 0x%08" PRIx64 "\n htinst: 0x%08" PRIx64 "\n",
839             state.hcounteren, state.htval, state.htinst);
840     fprintf(log_file, "hgap: 0x%08" PRIx64 "\n vsstatus: 0x%08" PRIx64 "\n vstvec: 0x%08" PRIx64 "\n",
841             state.hgap, state.vsstatus, state.vstvec);
842     fprintf(log_file, "vsscratch: 0x%08" PRIx64 "\n vsepc: 0x%08" PRIx64 "\n vscause: 0x%08" PRIx64 "\n",
843             state.vsscratch, state.vsepc, state.vscause);
844     fprintf(log_file, "vstval: 0x%08" PRIx64 "\n vsatp: 0x%08" PRIx64 "\n dpc: 0x%08" PRIx64 "\n",
845             state.vstval, state.vsatp, state.dpc);
846     fprintf(log_file, "dscratch0: 0x%08" PRIx64 "\n dscratch1: 0x%08" PRIx64 "\n tselect: 0x%08" PRIx64 "\n",
847             state.dscratch0, state.dscratch1, state.tselect);
848     // reg_t tdata2[num_triggers];
849     for(int i = 0; i < state.num_triggers; i++){
850         fprintf(log_file, "tdata2[%d]: 0x%08" PRIx64 "\n", i, state.tdata2[i]);
851     }
852     // uint8_t pmpcfg[max_pmp];
853     for(int i = 0; i < state.max_pmp; i++){
854         fprintf(log_file, "pmpcfg[%d]: 0x%08" PRIx64 "\n", i, state.pmpcfg[i]);
855     }
856     // reg_t pmpaddr[max_pmp];
857     for(int i = 0; i < state.max_pmp; i++){
858         fprintf(log_file, "pmpaddr[%d]: 0x%08" PRIx64 "\n", i, state.pmpaddr[i]);
859     }
860     fprintf(log_file, "fflags: 0x%08" PRIx64 "\n frm: 0x%08" PRIx64 "\n",
861             state.fflags, state.frm);
862     }
863     fprintf(log_file, "%d\n", state.counter_add);
864     state.counter_add = state.counter_add + 1;
865
866     last_pc = state.pc;
867     last_bits = bits;
868     executions = 1;
869 } else {
870     executions++;
871 }

```

四、实验结果

依然使用 Fibonacci 和 Bubble Sort 的手写汇编和 C 语言编译的 4 个程序为测试程序，其中对于手写 RV 汇编的两个测试程序使用 5 级流水线测试，带有 printf 的两个由 C 语言编译而来的测试程序，先使用模拟器运行出结果，再依照结果对使用 5 级流水线运行比较调试。

但本周仅实现了先使用模拟器运行出结果，虽然 C 语言编译而来的代码行数为 2w+行，但模拟器实际运行过程中，由于不断地分支体跳转等，实际运行的代码量约有 32w 行，代码量巨大，对 5 级流水线测试带 printf 的程序仍在调试中，因此本报告中仅展示对功能扩增后的 5 级流水线浮点测试结果。

（一）5 级流水线测试手写 Fibonacci

使用人为手写 RV 汇编指令的 Fibonacci 测试程序，代码如下：

其中，寄存器 a2 存储需要取第几个 Fibonacci 值，a3、a4 分别对应 i-2 和 i-1 的值，a5 为计数器用于计数 i，Fibonacci 第 i 个值存在 a6 当中。

Fibonacci_ass.o: 文件格式 elf32-littlescv

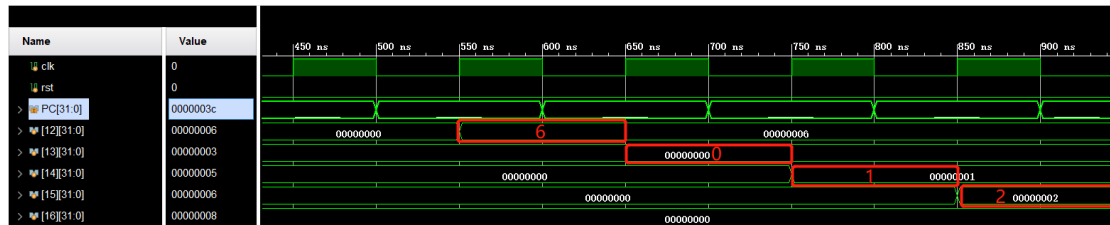
Disassembly of section .text:

```
00000000 <main>:
0: 00600613    li    a2,6
4: 00000693    li    a3,0
8: 00100713    li    a4,1
c: 00200793    li    a5,2
10: 00060813    mv    a6,a2
14: 02e60263    beq   a2,a4,38 <.L1>
18: 00070813    mv    a6,a4
1c: 00f60e63    beq   a2,a5,38 <.L1>

00000020 <.L0>:
20: 00e68833    add   a6,a3,a4
24: 00c7da63    bge   a5,a2,38 <.L1>
28: 00178793    addi  a5,a5,1
2c: 00070693    mv    a3,a4
30: 00080713    mv    a4,a6
34: fedff06f    j     20 <.L0>

00000038 <.L1>:
38: 01002023    sw    a6,0(zero) # 0 <main>
3c: 0008067    ret
```

前 4 条指令为 a2 至 a5 寄存器赋值，中间结果如下：



最终计算出来第 6 个 Fibonacci 值为 5，a6 中显示：



(二) 5 级流水线测试手写 Bubble Sort

使用人为手写 RV 汇编指令的 Bubble Sort 测试程序，代码如下：

其中.L0 为外层循环，.L1 为内层循环。a4 为外层循环变量，从 0 自增一，直至不小于数组大小（a2）。a5 对应内层循环控制变量，初始为 a4 值，自增直至不小于 a2。

BubbleSort_ass.o: file format elf32-littliscv

Disassembly of section .text:

```
00000000 <main>:
0:      00900613      li      a2,9
4:      00000713      li      a4,0

00000008 <.L0>:
8:      00070793      mv      a5,a4

0000000c <.L1>:
c:      00279813      slli    a6,a5,0x2
10:     00082283      lw      t0,0(a6)
14:     00482303      lw      t1,4(a6)
18:     0062c663      blt     t0,t1,24 <.L3>
1c:     00582223      sw      t0,4(a6)
20:     00682023      sw      t1,0(a6)

00000024 <.L3>:
24:     00178793      addi    a5,a5,1
28:     fec7c2e3      blt     a5,a2,c <.L1>
2c:     00170713      addi    a4,a4,1
30:     fcc74ce3      blt     a4,a2,8 <.L0>
34:     fcdff0ef      jal     ra,0 <main>
```

初始时，数据存储器 RAM 中的值如下所示，按从小到大的地址分别对应从大到小的数组值：

Name	Desig...	Block ...	Name	Value	Data Type
TOP	TOP	Verilog...	ram[65534:0][7:0]	00,00,00,00,00,00,...	Array
> if_unit	IF	Verilog...	> [25][7:0]	00	Array
> id_unit	ID	Verilog...	> [24][7:0]	04	Array
ex_unit	EX	Verilog...	> [23][7:0]	00	Array
mem_unit	MEM	Verilog...	> [22][7:0]	00	Array
ram	RAM	Verilog...	> [21][7:0]	00	Array
wb_unit	WB	Verilog...	> [20][7:0]	05	Array
gbl	gbl	Verilog...	> [19][7:0]	00	Array
			> [18][7:0]	00	Array
			> [17][7:0]	00	Array
			> [16][7:0]	06	Array
			> [15][7:0]	00	Array
			> [14][7:0]	00	Array
			> [13][7:0]	00	Array
			> [12][7:0]	07	Array
			> [11][7:0]	00	Array
			> [10][7:0]	00	Array
			> [9][7:0]	00	Array
			> [8][7:0]	08	Array
			> [7][7:0]	00	Array
			> [6][7:0]	00	Array
			> [5][7:0]	00	Array
			> [4][7:0]	09	Array
			> [3][7:0]	00	Array
			> [2][7:0]	00	Array
			> [1][7:0]	00	Array
			> [0][7:0]	0a	Array
			> [31:0]	65535	Array

冒泡排序后，数据存储器 RAM 中的值如下所示，可发现经过排序后，数组被完全倒置：

Name	Design...	Block ...	Name	Value	Data Type
TOP	TOP	Verilog...	ram[65534:0][7:0]	00,00,00,00,00,00,...	Array
> if_unit	IF	Verilog...	> [25][7:0]	00	Array
> id_unit	ID	Verilog...	> [24][7:0]	07	Array
> ex_unit	EX	Verilog...	> [23][7:0]	00	Array
> mem_unit	MEM	Verilog...	> [22][7:0]	00	Array
> ram	RAM	Verilog...	> [21][7:0]	00	Array
> wb_unit	WB	Verilog...	> [20][7:0]	06	Array
> gtbl	gtbl	Verilog...	> [19][7:0]	00	Array
			> [18][7:0]	00	Array
			> [17][7:0]	00	Array
			> [16][7:0]	05	Array
			> [15][7:0]	00	Array
			> [14][7:0]	00	Array
			> [13][7:0]	00	Array
			> [12][7:0]	04	Array
			> [11][7:0]	00	Array
			> [10][7:0]	00	Array
			> [9][7:0]	00	Array
			> [8][7:0]	03	Array
			> [7][7:0]	00	Array
			> [6][7:0]	00	Array
			> [5][7:0]	00	Array
			> [4][7:0]	02	Array
			> [3][7:0]	00	Array
			> [2][7:0]	00	Array
			> [1][7:0]	00	Array
			> [0][7:0]	01	Array
			> [31:0]	65535	Array

(三) 模拟器测试 C 语言 Fibonacci

使用模拟器运行带有 printf 语句的 Fibonacci 和 BubbleSort 程序。

```
lcr@DESKTOP-JDL9JAK:~/riscv/bin$ ./spike -l ../riscv32-unknown-elf/bin/pk Fibonacci.out
bbl loader
The 5 of fobonacci is: 3.
lcr@DESKTOP-JDL9JAK:~/riscv/bin$ mv exe.log Fibonacci_exe.log
lcr@DESKTOP-JDL9JAK:~/riscv/bin$ ./spike -l ../riscv32-unknown-elf/bin/pk BubbleSort.out
bbl loader
the first element is 1.
lcr@DESKTOP-JDL9JAK:~/riscv/bin$ mv exe.log BubbleSort_exe.log
```

结果分别放在 Fibonacci_exe.log 和 BubbleSort_exe.log 下。

Fibonacci 最终各个寄存器状态结果如下，其中 XPR 为 32 位寄存器（32 个）、FPR 为 64 位浮点寄存器（32 个），其余为特殊的 CSR 寄存器：

===== XPR =====

```
XPR[0]: 0x00000000
XPR[1]: 0xffffffff80000f90
XPR[2]: 0xffffffff80216e30
XPR[3]: 0x00026b08
XPR[4]: 0x00000000
XPR[5]: 0x7ffffd90
XPR[6]: 0x00000000
XPR[7]: 0x00000000
XPR[8]: 0xffffffff80010000
XPR[9]: 0xffffffff80012c50
XPR[10]: 0xffffffff80012c50
```

XPR[11]: 0x00000000
XPR[12]: 0x00000000
XPR[13]: 0x00000000
XPR[14]: 0x00000000
XPR[15]: 0x00000000
XPR[16]: 0x00000000
XPR[17]: 0x00000000
XPR[18]: 0x00000000
XPR[19]: 0x00000000
XPR[20]: 0x00000000
XPR[21]: 0x00000000
XPR[22]: 0x00000000
XPR[23]: 0x00000000
XPR[24]: 0x00000000
XPR[25]: 0x00000000
XPR[26]: 0x00000000
XPR[27]: 0x00000000
XPR[28]: 0x00000000
XPR[29]: 0x00000000
XPR[30]: 0x00000000
XPR[31]: 0x00000000

===== FPR =====

FPR[0]: 0xffffffff00000000
FPR[1]: 0xffffffff00000000
FPR[2]: 0xffffffff00000000
FPR[3]: 0xffffffff00000000
FPR[4]: 0xffffffff00000000
FPR[5]: 0xffffffff00000000
FPR[6]: 0xffffffff00000000
FPR[7]: 0xffffffff00000000
FPR[8]: 0xffffffff00000000
FPR[9]: 0xffffffff00000000
FPR[10]: 0xffffffff00000000
FPR[11]: 0xffffffff00000000
FPR[12]: 0xffffffff00000000
FPR[13]: 0xffffffff00000000
FPR[14]: 0xffffffff00000000
FPR[15]: 0xffffffff00000000
FPR[16]: 0xffffffff00000000
FPR[17]: 0xffffffff00000000
FPR[18]: 0xffffffff00000000
FPR[19]: 0xffffffff00000000
FPR[20]: 0xffffffff00000000
FPR[21]: 0xffffffff00000000

FPR[22]: 0xffffffff00000000
FPR[23]: 0xffffffff00000000
FPR[24]: 0xffffffff00000000
FPR[25]: 0xffffffff00000000
FPR[26]: 0xffffffff00000000
FPR[27]: 0xffffffff00000000
FPR[28]: 0xffffffff00000000
FPR[29]: 0xffffffff00000000
FPR[30]: 0xffffffff00000000
FPR[31]: 0xffffffff00000000

===== CSR =====

pc: 0xffffffff80004fb4
prv: 0x00000001
misa: 0x40141128
mstatus: 0x5800060a0
mepc: 0x800002b0
mtval: 0x00000000
mscratch: 0x80013f40
mtvec: 0x80000004
mcause: 0x00000000
minstret: 0x0004e0b5
mie: 0x00000008
mip: 0x00000000
medeleg: 0x0000b109
mideleg: 0x00000222
mcounteren: 0xffffffff
scounteren: 0xffffffff
sepc: 0x0001fe1c
stval: 0x00000000
sscratch: 0x00000000
stvec: 0x800027c0
satp: 0x80080215
scause: 0x00000008
mtval2: 0x00000000
mtinst: 0x00000000
hstatus: 0x100000000
hideleg: 0x00000000
hedeleg: 0x00000000
hcounteren: 0x00000000
htval: 0x00000000
htinst: 0x00000000
hgap: 0x00000000
vsstatus: 0x100000000
vstvec: 0x00000000

vsscratch: 0x00000000
vsepc: 0x00000000
vscause: 0x00000000
vstval: 0x00000000
vsatp: 0x00000000
dpc: 0x00000000
dscratch0: 0x00000000
dscratch1: 0x00000000
tselect: 0x00000000
tdata2[0]: 0x00000000
tdata2[1]: 0x00000000
tdata2[2]: 0x00000000
tdata2[3]: 0x00000000
pmpcfg[0]: 0x0000001f
pmpcfg[1]: 0x00000000
pmpcfg[2]: 0x00000000
pmpcfg[3]: 0x00000000
pmpcfg[4]: 0x00000000
pmpcfg[5]: 0x00000000
pmpcfg[6]: 0x00000000
pmpcfg[7]: 0x00000000
pmpcfg[8]: 0x00000000
pmpcfg[9]: 0x00000000
pmpcfg[10]: 0x00000000
pmpcfg[11]: 0x00000000
pmpcfg[12]: 0x00000000
pmpcfg[13]: 0x00000000
pmpcfg[14]: 0x00000000
pmpcfg[15]: 0x00000000
pmpaddr[0]: 0xffffffff
pmpaddr[1]: 0x00000000
pmpaddr[2]: 0x00000000
pmpaddr[3]: 0x00000000
pmpaddr[4]: 0x00000000
pmpaddr[5]: 0x00000000
pmpaddr[6]: 0x00000000
pmpaddr[7]: 0x00000000
pmpaddr[8]: 0x00000000
pmpaddr[9]: 0x00000000
pmpaddr[10]: 0x00000000
pmpaddr[11]: 0x00000000
pmpaddr[12]: 0x00000000
pmpaddr[13]: 0x00000000
pmpaddr[14]: 0x00000000

pmpaddr[15]: 0x00000000
fflags: 0x00000000
frm: 0x00000000

(四) 模拟器测试 Bubble Sort by C

Bubble Sort 的结果为:

===== XPR =====

XPR[0]: 0x00000000
XPR[1]: 0xffffffff80000f90
XPR[2]: 0xffffffff80216e30
XPR[3]: 0x00026be8
XPR[4]: 0x00000000
XPR[5]: 0x7ffffd90
XPR[6]: 0x00000000
XPR[7]: 0x00000000
XPR[8]: 0xffffffff80010000
XPR[9]: 0xffffffff80012c50
XPR[10]: 0xffffffff80012c50
XPR[11]: 0x00000000
XPR[12]: 0x00000000
XPR[13]: 0x00000000
XPR[14]: 0x00000000
XPR[15]: 0x00000000
XPR[16]: 0x00000000
XPR[17]: 0x00000000
XPR[18]: 0x00000000
XPR[19]: 0x00000000
XPR[20]: 0x00000000
XPR[21]: 0x00000000
XPR[22]: 0x00000000
XPR[23]: 0x00000000
XPR[24]: 0x00000000
XPR[25]: 0x00000000
XPR[26]: 0x00000000
XPR[27]: 0x00000000
XPR[28]: 0x00000000
XPR[29]: 0x00000000
XPR[30]: 0x00000000
XPR[31]: 0x00000000

===== FPR =====

FPR[0]: 0xffffffff00000000
FPR[1]: 0xffffffff00000000
FPR[2]: 0xffffffff00000000

FPR[3]: 0xffffffff00000000
FPR[4]: 0xffffffff00000000
FPR[5]: 0xffffffff00000000
FPR[6]: 0xffffffff00000000
FPR[7]: 0xffffffff00000000
FPR[8]: 0xffffffff00000000
FPR[9]: 0xffffffff00000000
FPR[10]: 0xffffffff00000000
FPR[11]: 0xffffffff00000000
FPR[12]: 0xffffffff00000000
FPR[13]: 0xffffffff00000000
FPR[14]: 0xffffffff00000000
FPR[15]: 0xffffffff00000000
FPR[16]: 0xffffffff00000000
FPR[17]: 0xffffffff00000000
FPR[18]: 0xffffffff00000000
FPR[19]: 0xffffffff00000000
FPR[20]: 0xffffffff00000000
FPR[21]: 0xffffffff00000000
FPR[22]: 0xffffffff00000000
FPR[23]: 0xffffffff00000000
FPR[24]: 0xffffffff00000000
FPR[25]: 0xffffffff00000000
FPR[26]: 0xffffffff00000000
FPR[27]: 0xffffffff00000000
FPR[28]: 0xffffffff00000000
FPR[29]: 0xffffffff00000000
FPR[30]: 0xffffffff00000000
FPR[31]: 0xffffffff00000000

===== CSR =====

pc: 0xffffffff80004fb4
prv: 0x00000001
misa: 0x40141128
mstatus: 0x5800060a0
mepc: 0x800002b0
mtval: 0x00000000
mscratch: 0x80013f40
mtvec: 0x80000004
mcause: 0x00000000
minstret: 0x0004e65d
mie: 0x00000008
mip: 0x00000000
medeleg: 0x0000b109
mideleg: 0x00000222

mcounteren: 0xffffffff
scounteren: 0xffffffff
sepc: 0x0001fed4
stval: 0x00000000
sscratch: 0x00000000
stvec: 0x800027c0
satp: 0x80080215
scause: 0x00000008
mtval2: 0x00000000
mtinst: 0x00000000
hstatus: 0x10000000
hideleg: 0x00000000
hedeleg: 0x00000000
hcounteren: 0x00000000
htval: 0x00000000
htinst: 0x00000000
hgap: 0x00000000
vsstatus: 0x10000000
vstvec: 0x00000000
vsscratch: 0x00000000
vsepc: 0x00000000
vscause: 0x00000000
vstval: 0x00000000
vsatp: 0x00000000
dpc: 0x00000000
dscratch0: 0x00000000
dscratch1: 0x00000000
tselect: 0x00000000
tdata2[0]: 0x00000000
tdata2[1]: 0x00000000
tdata2[2]: 0x00000000
tdata2[3]: 0x00000000
pmpcfg[0]: 0x0000001f
pmpcfg[1]: 0x00000000
pmpcfg[2]: 0x00000000
pmpcfg[3]: 0x00000000
pmpcfg[4]: 0x00000000
pmpcfg[5]: 0x00000000
pmpcfg[6]: 0x00000000
pmpcfg[7]: 0x00000000
pmpcfg[8]: 0x00000000
pmpcfg[9]: 0x00000000
pmpcfg[10]: 0x00000000
pmpcfg[11]: 0x00000000

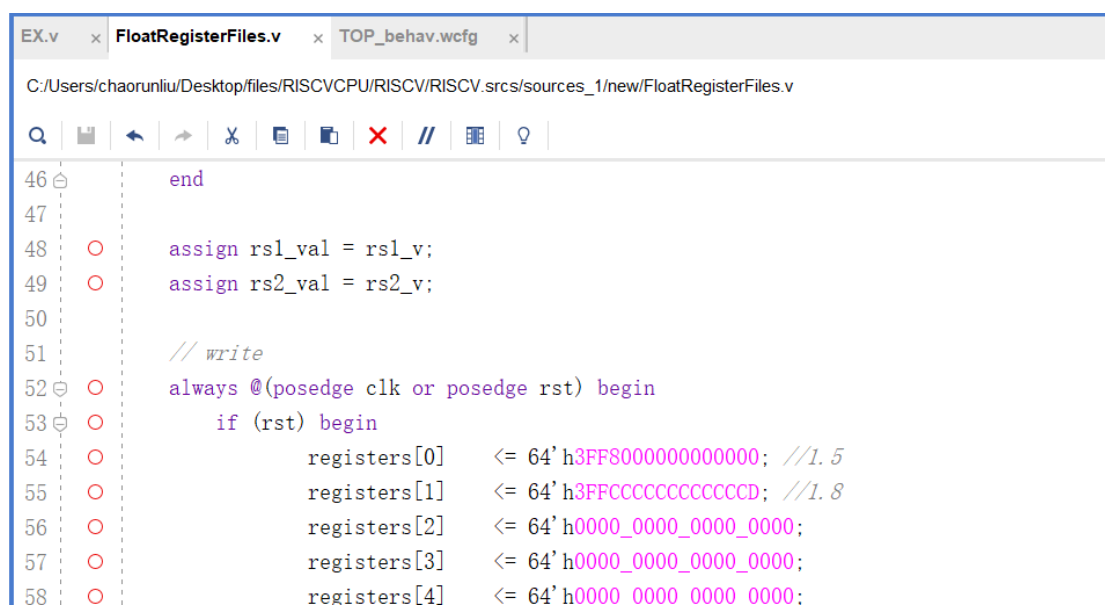
```

pmpcfg[12]: 0x00000000
pmpcfg[13]: 0x00000000
pmpcfg[14]: 0x00000000
pmpcfg[15]: 0x00000000
pmpaddr[0]: 0xffffffff
pmpaddr[1]: 0x00000000
pmpaddr[2]: 0x00000000
pmpaddr[3]: 0x00000000
pmpaddr[4]: 0x00000000
pmpaddr[5]: 0x00000000
pmpaddr[6]: 0x00000000
pmpaddr[7]: 0x00000000
pmpaddr[8]: 0x00000000
pmpaddr[9]: 0x00000000
pmpaddr[10]: 0x00000000
pmpaddr[11]: 0x00000000
pmpaddr[12]: 0x00000000
pmpaddr[13]: 0x00000000
pmpaddr[14]: 0x00000000
pmpaddr[15]: 0x00000000
fflags: 0x00000000
frm: 0x00000000

```

（五）浮点测试结果

将浮点寄存器中 ft0 和 ft1 置为 1.5 和 1.8。



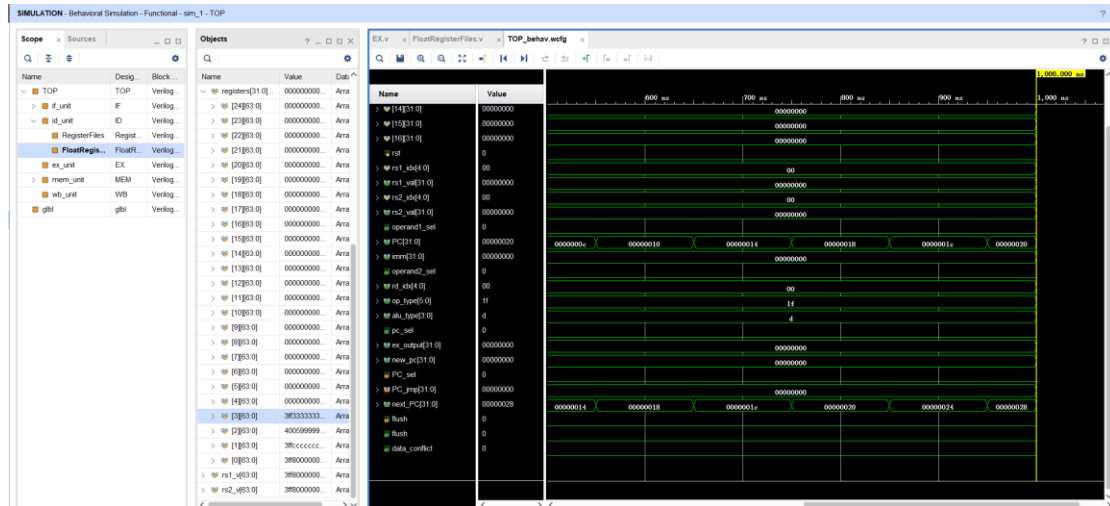
The screenshot shows a Verilog code editor with the file name 'FloatRegisterFiles.v'. The code is as follows:

```

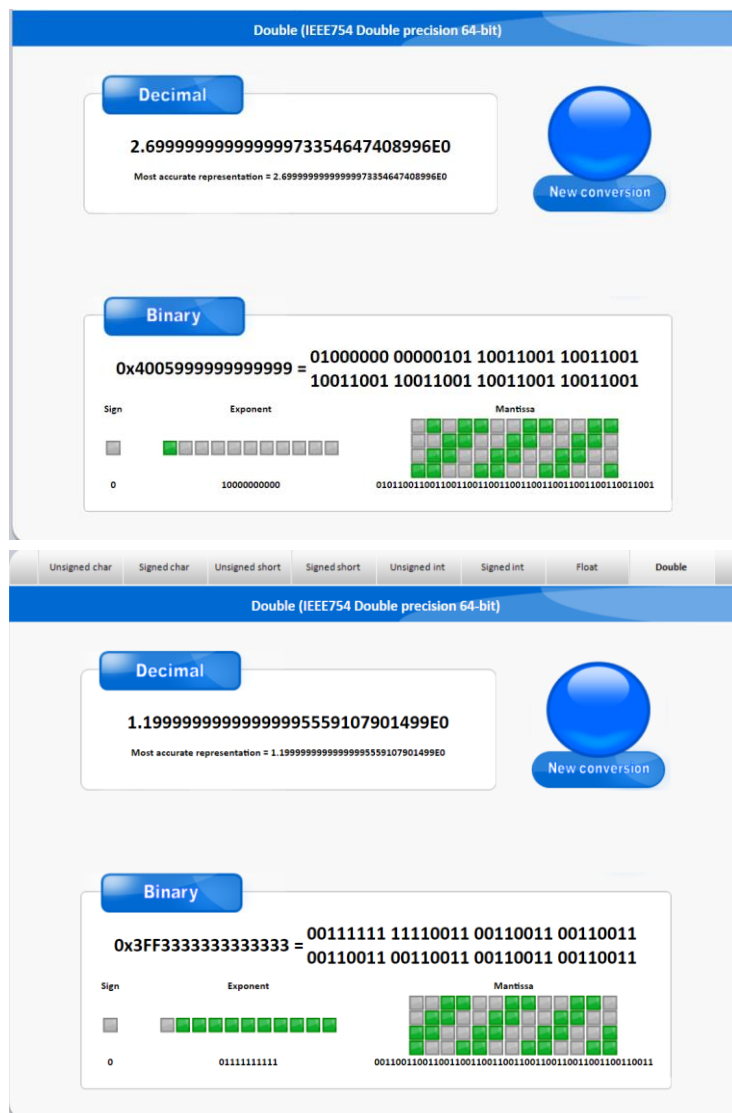
46 end
47
48 assign rs1_val = rs1_v;
49 assign rs2_val = rs2_v;
50
51 // write
52 always @(posedge clk or posedge rst) begin
53     if (rst) begin
54         registers[0] <= 64'h3FF8000000000000; //1.5
55         registers[1] <= 64'h3FFCCCCCCCCCD; //1.8
56         registers[2] <= 64'h0000_0000_0000_0000;
57         registers[3] <= 64'h0000_0000_0000_0000;
58         registers[4] <= 64'h0000_0000_0000_0000;

```

然后运行指令 `fmul.d f2, f1, f0` 和 `fdiv.d f3, f1, f0`。仿真结果如下：



f2 和 f3 的值分别为 0x4005999999999999 和 0x3FF3333333333333，转成 10 进制如下，结果正确。



五、实验总结

（一）实验分工

许诗瑶：ECALL 功能的实现、模拟器输出修改、5 级流水线测试调试

刘朝润：浮点乘除功能的实现和测试、模拟器的调试使用和输出修改、5 级流水线测试调试

刘晓航：指令集细节整理

（二）实验遇到的问题

1、本周工作集中于浮点乘除的实现和模拟器的调试使用，其中浮点乘除实现的困难在于，浮点运算中的五个舍入模式和五个异常，上下溢出比较麻烦，花费了较多的时间和精力。

2、带有 printf 的 C 语言程序经过静态链接编译后代码规模量大，指令行数高达 2w+行，但实际经过模拟器运行时，由于分支跳转等，真正运行的代码行数高达 32w+行。修改模拟器代码使之能输出结果后，由于每运行一行指令输出当前状态，最终输出文件大小有 120+G。于是再修改代码，手动加入计数变量，选择性地输出我们想要时的输出结果，并且方便获得中间结果与 5 级流水线结果实时测试比较。

3、模拟器调试使用时发现，模拟器运行的指令和反汇编的结果不一致，发现原因在于模拟器为用户模式，运行中对 ecall 之后新增了一些指令，用于保护寄存器等，如下图所示：

```

278392
core 0: 0x0001ff1c (0x00000073) ecall
278393
core 0: exception trap_user_ecall, epc 0x0001ff1c
core 0: 0x800027c0 (0x14011173) csrwr sp, sscratch, sp
278394
core 0: 0x800027c4 (0x00011463) bnez sp, pc + 8
278395
core 0: 0x800027cc (0xec010113) addi sp, sp, -320
278396
core 0: 0x800027d0 (0x00112223) sw ra, 4(sp)
278397
core 0: 0x800027d4 (0x00312623) sw gp, 12(sp)
278398
core 0: 0x800027d8 (0x00412823) sw tp, 16(sp)
278399
core 0: 0x800027dc (0x00512a23) sw t0, 20(sp)
278400
core 0: 0x800027e0 (0x00612c23) sw t1, 24(sp)
278401
core 0: 0x800027e4 (0x00712e23) sw t2, 28(sp)
278402
core 0: 0x800027e8 (0x02812023) sw s0, 32(sp)
278403
core 0: 0x800027ec (0x02912223) sw s1, 36(sp)
278404
core 0: 0x800027f0 (0x02a12423) sw a0, 40(sp)
278405
core 0: 0x800027f4 (0x02b12623) sw a1, 44(sp)
278406
core 0: 0x800027f8 (0x02c12823) sw a2, 48(sp)
278407
core 0: 0x800027fc (0x02d12a23) sw a3, 52(sp)
278408
core 0: 0x80002800 (0x02e12c23) sw a4, 56(sp)
278409
core 0: 0x80002804 (0x02f12e23) sw a5, 60(sp)
278410
core 0: 0x80002808 (0x05012023) sw a6, 64(sp)
278411
core 0: 0x8000280c (0x05112223) sw a7, 68(sp)
278412
core 0: 0x80002810 (0x05212423) sw s2, 72(sp)
278413
core 0: 0x80002814 (0x05312623) sw s3, 76(sp)
278414
core 0: 0x80002818 (0x05412823) sw s4, 80(sp)

```

（三）附件文件说明

RISCV 文件夹：Vivado 环境下项目工程代码

test_program：实验测试程序代码及运行结果

riscv-isa-sim.zip：模拟器源码（含我们的修改）