Report

name: Yunxin Yang （杨昀昕）

2023233204

date:2023.10.13

Analysis on the original code structure

simulate()

Function of the total control of the pipeline;

In every loop, run following codes:

```
this->fetch();
this->decode();
this->excecute();
this->memoryAccess();
this->writeBack();
```

After 5 stages finishes, update local registers from last stage to next stage and do branch prediction.

Details of 5 stages as follows:

fetch()

Fetch the instruction according to the PC

decode()

Decode the instruction to opcode,funct3,funct7,rd,rs1,rs2,imm_i and so on and determine the type of instruction.

At last, do the branch prediction.

excecute()

1. do the main calculation logic of the instruction
2. judge if instruction have hazard
    a. Control Hazard disposition of jump and branch instruction
    b. disposition of Data Hazard of stage memoryaccess
    c. disposition of Data Hazard of stage Regwriteback

memoryAccess()

1. do the writemem
2. do the readmem
3. Check for data hazard and forward data

writeBack()

1. writeback the value to the destination register

Description of your design

1. add additional float registers to do the float operation
    a. add 32 global float registers: f0-f31
    b. add local float registers in every stage functions
    c. add needed transformer float registers bewteen every stage, e.g. add floatop1, floatop2, floatout in fRegNew,dRegNew and so on
2. add needed hazard logic for the new float registers
    a. in excecute(): add checking for data hazard and forward data for float registers; as the similar logic of dispositin of integer registers
    b. in memoryAccess: add checking for data hazard and forward data as the similar logic of dispositin of integer registers
3. for every new instruction
    a. in decode(): add decode() logic for each instruction using switch logic

       e.g.

```
case OP_FP: {
// rd+rs1 must have
// 2. funct7
// 3. reg2 and funct3 (option, need to recheck)
  reg1 = rs1;
  dest = rd;
  // (op2,reg2,op2str) and funct3 depends on instruction
  // op1 and op2 or floatop1 floatop1 depends on instruction
  // reg2 = rs2;
  // op2str = FLOATREGNAME[rs2];
  switch (funct7) {
  case 0b1111000: // fmv.w.x
    if (0 == rs2 && 0 == funct3){ // FMV.W.X rd(float), rs1(int)
      op1 = this->reg[rs1];
      instname = "fmv.w.x";
      insttype = FMV_W_X;
    }else{
      this->panic("Unknown rs2 0x%x for funct7 0x%x\n", rs2, funct7);
    }
    break;
  case 0b1110000: // fmv.x.w
    if (0 == rs2 && 0 == funct3){ // fmv.x.w rd(x),rs1(float)  (64位int高32位
      floatop1 = this->floatreg[rs1];
      instname = "fmv.x.w";
      insttype = FMV_X_W;
    }else{
      this->panic("Unknown rs2 0x%x for funct7 0x%x\n", rs2, funct7);
    }
    break;
  case 0b1101000: // fcvt.s.w
    if (0 == rs2){ // fcvt.s.w
      // fcvt.s.w rd(float), rs1(int)
      // converts a 32-bit or 64-bit signed integer, respectively, in intege
      op1 = this->reg[rs1];
      instname = "fcvt.s.w";
      insttype = FCVT_S_W;
    }else{
      this->panic("Unknown rs2 0x%x for funct7 0x%x\n", rs2, funct7);
    }
    break;
   case 0b1100000: // fcvt.w.s
    if (0 == rs2){ // fcvt.w.s
      // fcvt.w.s rd(int), rs1(float)
```

b. in excecute(): add execution logic for each instruction

   e.g.

```
1182    case FMV_W_X:
1183      writeFloatReg = true;
1184      uint32_t low_32_bits;
1185      low_32_bits = op1 & 0xFFFFFFFF;
1186      // TODO: why can not together give value
1187      // uint32_t low_32_bits = op1 & 0xFFFFFFFF;
1188      uint32_t* place;
1189      place = (uint32_t*)(&floatout);
1190      *place = low_32_bits;
1191      break;
1192    case FMV_X_W:
1193      writeReg = true;
1194      // uint32_t low_32_bits;
1195      // low_32_bits = ;
1196      float real_floatop1;
1197      real_floatop1 = cutDoubleReg2low32bi
1198      bool negative_sign;
1199      negative_sign = (real_floatop1 < 0);
1200      out = 0;
1201      *(uint32_t*)(&out) = *(uint32_t *)(&real_floatop1);
1202      if (negative_sign) *((uint32_t*)(&out)+1) = 0xFFFFFFFF;
1203      break;
1204    case FCVT_S_W:
1205      // fcvt.s.w rd(float), rs1(int)
1206      // converts a 32-bit or 64-bit signed integer, respectively, in integer register r
1207      writeFloatReg = true;
1208      saveFloat2DoubleReg2low32bit((float)op1, floatout);
1209      break;
1210    case FCVT_W_S:
1211      // fcvt.w.s rd(int), rs1(float)
1212      // converts float to signed int
1213      writeReg = true;
1214      out = cutDoubleReg2low32bit2Float(floatop1);
1215      break;
1216    case FADD_S:
1217      // fadd.s rd(float), rs1(float), rs2(float)
```

```
float real_floatop1
uint32_t low_32_bits;
low_32_bits = ;
```

c. in memoryAccess(): add storing and reading float logic to and from memory

d. in writeBack(): add writeback logic for float registers as the similar logic of dispositin of integer registers

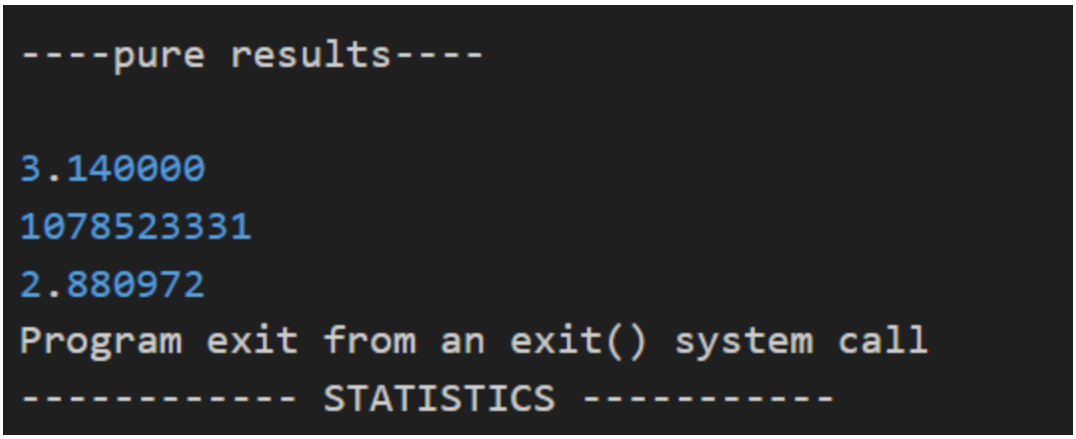
Evaluation and analysis on your results, test case(s), etc.

test case of fmv.w.x, fmv.x.w, fsqrt.s

```
int main() {
    float floatout = 0;
```

```
    int64_t intValue = 0x4048f5c3;//1078523331 in decimal, 3.14 in float if bits not changed
    float floatin = 0;

    asm volatile(
        "fmv.w.x %[result], %[input1]"
        : [result] "=f" (floatout)
        : [input1] "r" (intValue)
    );
    print_f(floatout);
    print_s("\n");

    asm volatile(
        "fmv.x.w %[result], %[input1]"
        : [result] "=r" (intValue)
        : [input1] "f" (floatout)
    );
    print_d(intValue);
    print_s("\n");

    floatin = 8.3;
    asm volatile(
        "fsqrt.s %[result], %[input1]"
        : [result] "=f" (floatout)
        : [input1] "f" (floatin)
    );
    print_f(floatout);
    print_s("\n");


    exit_proc();
}
```

output：



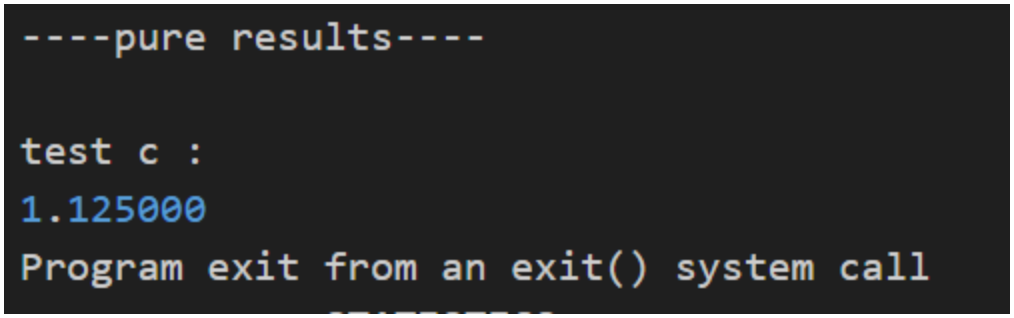
analysis:

1. first `fmv.w.x` move low 32bits of `int64_t intValue = 0x4048f5c3` to floatout, and it turns out `3.14` right
2. `fmv.x.w` move back from floatout to intValue, it turns out `1078523331` right
3. `fsqrt.s` calculate the sqrt of 8.3, it turns out 2. 880972 right

test case of flw, fsw

```
int main() {
    float floatValue = 1.125;
    print_s("test c :\n");
    print_f(floatValue);
    print_s("\n");
    exit_proc();
}
```

output:



analisys: print_f uses assemble code `flw, fsw` , and it prints float number 1.125 right

assemble code for print_f(partly):

```
00000000000101e0 <main>:
   101e0:   fe010113            add   sp,sp,-32
   101e4:   00113c23            sd ra,24(sp)
   101e8:   00813823            sd s0,16(sp)
   101ec:   02010413            add   s0,sp,32
   101f0:   000107b7            lui   a5,0x10
   101f4:   76c7a787            flw   fa5,1900(a5) # 1076c <__errno+0x1c>
   101f8:   fef42627            fsw   fa5,-20(s0)
   101fc:   000107b7            lui   a5,0x10
   10200:   75878513            add   a0,a5,1880 # 10758 <__errno+0x8>
   10204:   060000ef            jal   10264 <print_s>
   10208:   fec42507            flw   fa0,-20(s0)
   1020c:   130000ef            jal   1033c <print_f>
   10210:   000107b7            lui   a5,0x10
   10214:   76878513            add   a0,a5,1896 # 10768 <__errno+0x18>
   10218:   04c000ef            jal   10264 <print_s>
```

testcase for fcvt.w.s and fcvt.s.w

```
int main() {
    float a = 3.14159;
    int b = a;
    print_d(b);
    print_s("\n");
    b = 10;
    a = b;
    print_f(a);
    print_s("\n");
    exit_proc();
}
```

output:

```
----pure results----

3
10.000000
Program exit from an exit() system call
------------ STATISTICS -----------
```

analysis:

code `int b = a;` and `a = b;` uses fcvt.w.s and fcvt.s.w

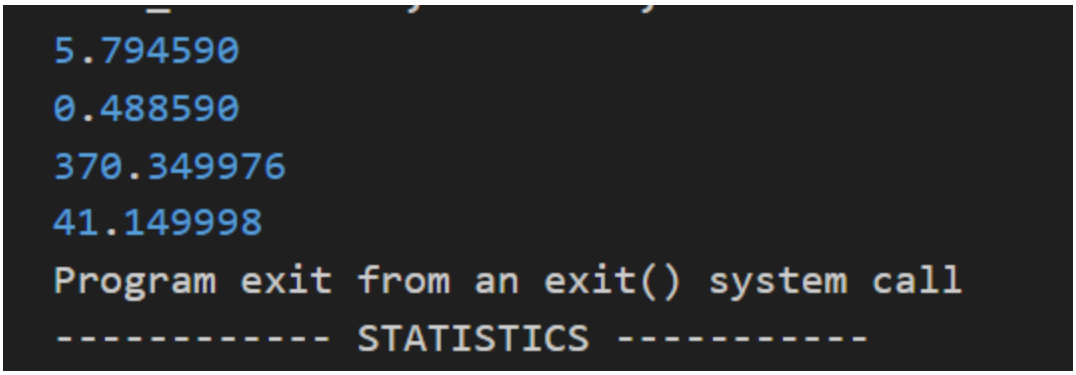
```
 10200:   c00797d3            fcvt.w.s a5,fa5,rtz
 10204:   fef42423            sw a5,-24(s0)
 10208:   fe842783            lw a5,-24(s0)
 1020c:   00078513            mv a0,a5
 10210:   054000ef            jal   10264 <print_d>
 10214:   000107b7            lui   a5,0x10
 10218:   78878513            add   a0,a5,1928 # 10788 <__errno+0xc>
 1021c:   074000ef            jal   10290 <print_s>
 10220:   00a00793            li a5,10
 10224:   fef42423            sw a5,-24(s0)
 10228:   fe842783            lw a5,-24(s0)
 1022c:   d007f7d3            fcvt.s.w fa5,a5
```

and the output 3.0 and 10 right.

testcase for fadd.s,fsub.s,fmul.s,fdiv.s

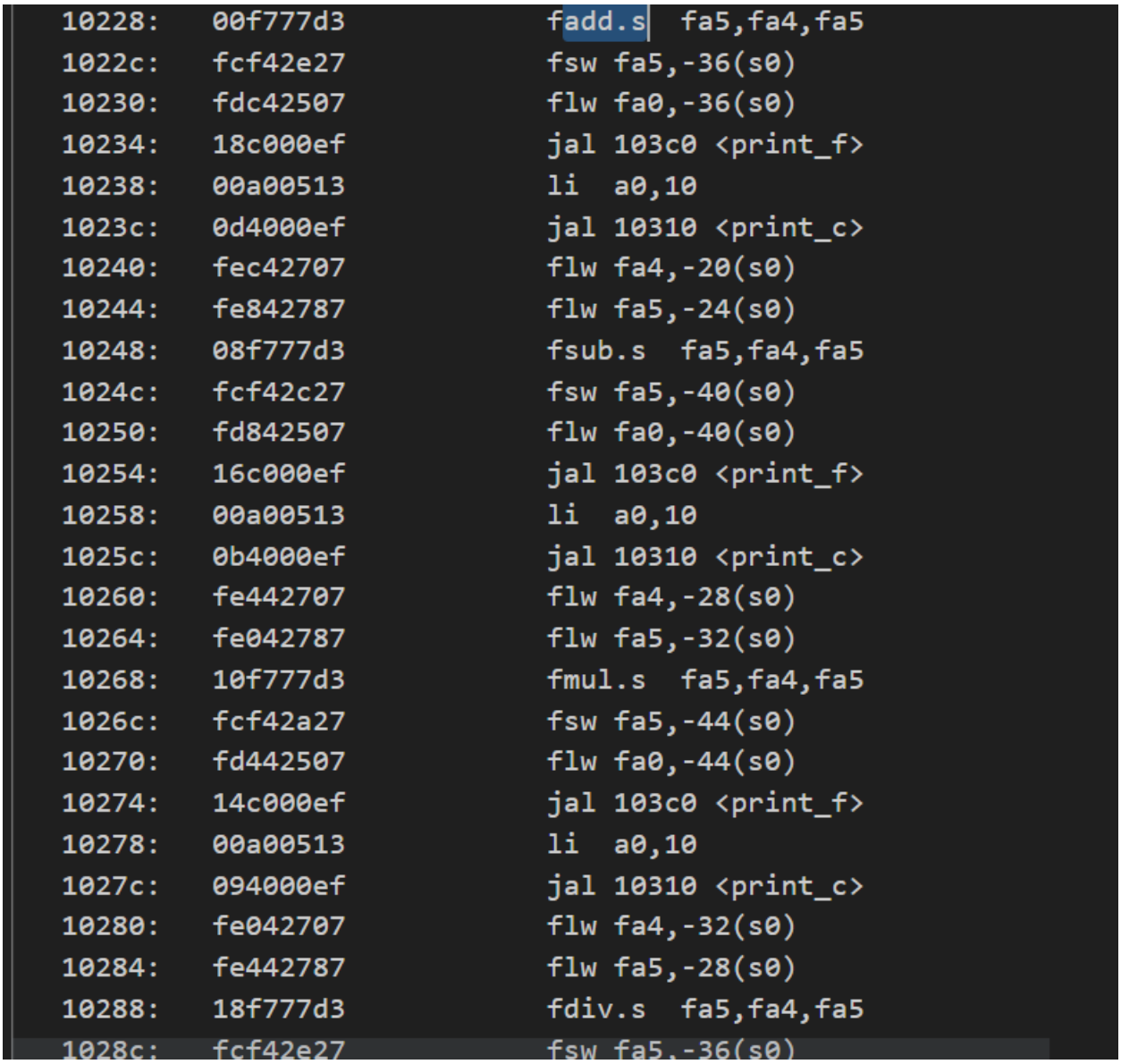
```
// sample test_float
int main() {
    float a = 3.14159, b = 2.653, c = 3.0, d = 123.45;
    float x = a + b;
    print_f(x);
    print_c('\n');
    float y = a - b;
    print_f(y);
    print_c('\n');
    float z = c * d;
    print_f(z);
    print_c('\n');

    x = d / c;
    print_f(x);
    print_c('\n');

    exit_proc();
}
```

output:



analysis:

the assemble code of the c code involves fadd.s,fsub.s,fmul.s,fdiv.s, and it turns out all right.



```
10228:   00f777d3          fadd.s  fa5,fa4,fa5
1022c:   fcf42e27          fsw fa5,-36(s0)
10230:   fdc42507          flw fa0,-36(s0)
10234:   18c000ef          jal 103c0 <print_f>
10238:   00a00513          li  a0,10
1023c:   0d4000ef          jal 10310 <print_c>
10240:   fec42707          flw fa4,-20(s0)
10244:   fe842787          flw fa5,-24(s0)
10248:   08f777d3          fsub.s  fa5,fa4,fa5
1024c:   fcf42c27          fsw fa5,-40(s0)
10250:   fd842507          flw fa0,-40(s0)
10254:   16c000ef          jal 103c0 <print_f>
10258:   00a00513          li  a0,10
1025c:   0b4000ef          jal 10310 <print_c>
10260:   fe442707          flw fa4,-28(s0)
10264:   fe042787          flw fa5,-32(s0)
10268:   10f777d3          fmul.s  fa5,fa4,fa5
1026c:   fcf42a27          fsw fa5,-44(s0)
10270:   fd442507          flw fa0,-44(s0)
10274:   14c000ef          jal 103c0 <print_f>
10278:   00a00513          li  a0,10
1027c:   094000ef          jal 10310 <print_c>
10280:   fe042707          flw fa4,-32(s0)
10284:   fe442787          flw fa5,-28(s0)
10288:   18f777d3          fdiv.s  fa5,fa4,fa5
1028c:   fcf42e27          fsw fa5,-36(s0)
```

Bug report in original code

the system call `print_f` can not run

1. need int register a0 to pass parameters

work solution:

1. change `print_f` to

```
void print_f(float num){
  asm("fmv.x.w a0,fa0");
```

```
  asm("li a7,6;" "scall");
}
```

2. modify `Simulator::handleSystemCall()`:

```
// turn printf("%f", (float)arg1); into below
printf("%f", *(float*)(&arg1));
```

Suggestion

Suggest TAs can announce the bug in lab0 including code bug or error in books to ALL distinctly. TAs can set the fix note as top in piazza to notice anyone to save all's debug time (debuging the orginal code and fixing the unknown bug is such a torture…)