CSC 462/562 Computer Architecture
Homework #4 answer key

1.
    a. 1 stall after fld, 6 stalls after fmult.d, 2 stalls after the fadd.d, 1 stall after the addiw, and the branch delay slot.
    b.
```
Loop: fld         f1, 0(x1)      // 1 stall after this instruction
      fmult.d     f3, f1, f2     // 5 stalls after this instruction
      addiw       x1, x1, 8
      fadd.d      f5, f4, f3     // 1 stall after this instruction
      bne         x1, x2, Loop
      fsd         f5, -8(x1)
```
    c. We need to fill the remaining 5 stalls after the fmult.d, so we unroll the loop for 6 total iterations.
```
Loop: fld         f1, 0(x1)
      fld         f6, 8(x1)
      fld         f9, 16(x1)
      fld         f12, 24(x1)
      fld         f15, 32(x1)
      fld         f18, 40(x1)
      fmult.d     f3, f1, f2
      fmult.d     f7, f6, f2
      fmult.d     f10, f9, f2
      fmult.d     f13, f12, f2
      fmult.d     f16, f15, f2
      fmult.d     f19, 18, f2
      addiw       x1, x1, 48
      fadd.d      f5, f4, f3
      fadd.d      f8, f4, f7
      fadd.d      f11, f4, f10
      fadd.d      f14, f4, f13
      fadd.d      f17, f4, f16
      fadd.d      f19, f4, f18
      fsd         f5, -48(x1)
      fsd         f8, -40(x1)
      fsd         f11, -32(x1)
      fsd         f14, -24(x1)
      fsd         f17, -16(x1)
      bne         x1, x2, Loop
      fsd         f19, -8(x1)
```
    d. Original loop: 6 instructions in 17 cycles for a CPI = 17 / 6 = 2.83, revised code has no stalls so CPI = 1, speedup = 2.83.

2.

a. 1 stall after the second fld, 2 stalls after the fadd.d, 1 stall after the third fld, 5 stalls after the fmult.d, 1 stall after the last addiw, and the branch delay.

b. There are several ways to schedule the code. Here is one.

```
Loop:    fld       f1, 0(x1)
         fld       f2, 0(x2)
         fld       f4, 0(x3)
         fadd.d    f3, f1, f2
         addiw     x1, x1, 8
         addiw     x2, x2, 8
         addiw     x3, x3, 8
         fmult.d   f5, f4, f3
         addiw     x4, x4, 8     // 3 stalls after this instruction
         bne       x4, x5, Loop
         fsd       f5, -8(x4)
```

c. If we unroll the loop for 2 total iterations, we will have enough instructions to schedule away all stalls.

```
Loop:    fld       f1, 0(x1)
         fld       f2, 0(x2)
         fld       f6, 8(x1)
         fld       f7, 8(x2)
         fadd.d    f3, f1, f2
         fadd.d    f8, f6, f7
         fld       f4, 0(x3)
         fld       f9, 8(x3)
         fmult.d   f5, f4, f3
         fmult.d   f10, f8, f4
         addiw     x1, x1, 16
         addiw     x2, x2, 16
         addiw     x3, x3, 16
         addiw     x4, x4, 16
         fsd       f5, -16(x1)
         bne       x4, x5, Loop
         fsd       f10, -8(x1)
```

d. Original code executed 11 instructions in 22 cycles for a CPI of 2, the new code has a CPI of 1, so the speedup is exactly 2.0.

3. First, the branch prediction automatically incurs a 1 cycle penalty because it is not available for the next instruction until the end of the second stage of the branch instruction. On an accurate prediction then, the penalty is 1. On a buffer miss or miss-prediction, the penalty is based on how long it takes to compute the branch along with updating the buffer (2 extra cycles of penalty). As the branch is determined in an integer execution unit, this happens once the branch is executed, which occurs after it is issued, so this adds another 2 cycle penalty at least + 2 cycles to update the buffer. The number of cycles for the branch to execute varies as 40%: no further delay, 30%: 1 cycle delay, 20%: 2 cycle delay, 10%: 3 cycle delay. This amounts to a wait time to compute the branch of .40 * 0 + .30 * 1 +

.20 * 2 + .10 * 3 = 1.0 cycles. To this, we add 3 cycles to fetch the instruction and prediction and issue the instruction plus 2 cycles to update the buffer, or a buffer miss and miss-prediction penalty of 6.0. On a correct prediction, the penalty is 1.0. Misses occur 10% of the time and conditional branches are miss-predicted 20% of the time. This gives us a CPI of 1 + (1.0 * 90% * 80% + 6.0 * 10%) * (19% + 4%) + 6.0 * 20% * 19% (only conditional branches can be miss-speculated) = 1.532, which is far from ideal. We see a much lower impact of branches in the RISC-V pipeline.

4.

a. First, this promotes WAR hazards in that an instruction issued later may complete and save to a register that the waiting instruction has yet to read. Second, by reading both operands in one cycle, it prevents other functional units from reading operands, so the operand read stage is serialized, much like it was in the pipeline. Another drawback, not specifically caused by having to wait and read both operands in one clock cycle, is that the functional unit remains busy and if another instruction needs the same functional unit, it cannot be issued which stalls the instruction fetch/issue stages.
b. Let's assume the hardware can read 2 registers from the register file per cycle. If we allowed a functional unit to read its operand as soon as it became available, how do we police which functional unit(s) gets to read from the register file in any given cycle? If functional unit 1 has two values available, functional unit 2 has one available and functional unit 3 has one available, which functional unit(s) gets to read in the next cycle? Unit 1 only? Units 2 and 3? One from unit 1 and unit 2's? This would require some complex logic that would have to be handled in hardware quickly enough to accommodate the two operand reads in the same cycle.
c. The Tomasulo approach gets around this problem by having a functional unit forward its result over the CDB when its available and all instructions waiting at reservation stations that need the result simply read it off the CDB as it comes across. Notice that the CDB only permits 1 write per cycle so at best, a reservation station is reading 1 datum off the CDB per cycle. However, if the data are in the register file, it can read up to two of those data per cycle if both are available and no other functional unit is using the register file.

5. Scoreboard:

	Fetch	Issue	Read ops	Exec	Write
fld	1	2	3	4	5
fmult.d	2	3	6	7	14
fadd.d	3	4	15	16	20
fsd	4	5	21	22	n/a
addiw	5	6	7	8	9
bne	6	7	10	11	n/a
fld	12	13	14	15	16
fmult.d	13	15	17	18	25
fadd.d	14	21	26	27	31
fsd	15	22	32	33	n/a
addiw	22	23	24	25	26
bne	23	24	27	28	n/a

Tomasulo:	Fetch	Issue	Exec	Write
fld	1	2	3	4
fmult.d	2	3	5	12
fadd.d	3	4	13	17
fsd	4	5	18	n/a
addiw	5	6	7	8
bne	6	7	9	n/a
fld	10	11	12	13
fmult.d	11	12	14	21
fadd.d	12	13	22	26
fsd	13	14	27	n/a
addiw	14	15	16	18
bne	15	16	19	n/a

6.

Scoreboard:	Fetch	Issue	Read Ops	Execute	Write
fld	1	2	3	4	5
fld	2	3	4	5	6
fmult.d	3	4	7	8	15
fadd.d	4	5	16	17	21
addiw	5	6	8	9	10
addiw	6	7	9	10	11
bne	7	11	12	13	n/a
fld	14	15	17	18	19
fld	15	16	18	19	20
fmult.d	16	17	21	22	29
fadd.d	17	22	30	31	35
addiw	22	23	24	25	26
addiw	23	24	25	26	27
bne	24	27	28	29	n/a

Tomasulo:	Fetch	Issue	Exec	Write
fld	1	2	3	4
fld	2	3	4	5
fmult.d	3	4	6	13
fadd.d	4	5	14	18
addiw	5	6	7	8
addiw	6	7	8	9
bne	7	9	10	n/a
fld	11	12	13	14
fld	12	13	14	15
fmult.d	13	14	16	23
fadd.d	14	15	24	28
addiw	15	16	17	19
addiw	16	17	18	20
bne	17	21	22	n/a