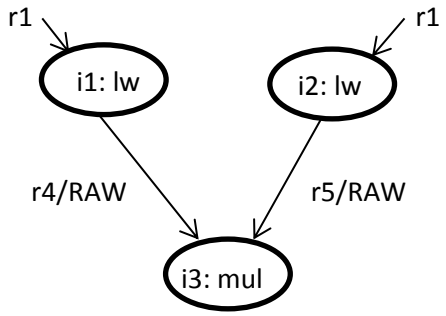
CPSC 3300 – Fall 2015 – Homework 6
Due at class time on Friday, October 30

1. For the MIPS instruction sequence below, complete the data dependency diagram. (Destination register is listed first except for sw instruction; sw writes into memory rather than a register.) (15 pts.)

```
i1: lw   r4, 0( r1 )  // reg[4] ← memory[ reg[1] + 0 ]
i2: lw   r5, 4( r1 )  // reg[5] ← memory[ reg[1] + 4 ]
i3: mul  r6, r4, r5   // reg[6] ← reg[4] * reg[5]
i4: sub  r8, r6, r7   // reg[8] ← reg[6] - reg[7]
i5: sw   r8, 8( r1 )  // memory[ reg[1] + 8 ] ← reg[8]
i6: add  r1, r1, r2   // reg[1] ← reg[1] + reg[2]
```



2. Draw the dependency diagram for the following MIPS code (15 pts.)

```
i1: add  r3, r4, r6    // reg[3] ← reg[4] + reg[6]
i2: sub  r5, r3, r2    // reg[5] ← reg[3] - reg[2]
i3: lw   r7, 0( r5 )   // reg[7] ← memory[ reg[5] + 0 ]
i4: add  r5, r5, r8    // reg[5] ← reg[5] + reg[8]
i5: sw   r7, 0( r5 )   // memory[ reg[5] + 0 ] ← reg[7]
```

3. For the following MIPS instruction sequence, complete the pipeline cycle diagram for the standard 5-stage pipeline without forwarding. Assume register file writes occur in the first half cycle and reads in the second half cycle. (15 pts.)

```
i1: lw   r1, 0( r5 )   // reg[1] ← memory[ reg[5] + 0 ]
i2: lw   r2, 4( r1 )   // reg[2] ← memory[ reg[1] + 4 ]
i3: addi r2, r2, 1     // reg[2] ← reg[2] + 1
```

```
i1:lw    IF    ID    EX    MEM    WB
i2:lw          IF
i3:addi
```

4. For the following MIPS instruction sequence, complete the pipeline cycle diagram for the standard 5-stage pipeline with forwarding. Assume register file writes occur in the first half cycle and reads in the second half cycle. (15 pts.)

```
i1: lw   r1, 0( r5 )  // reg[1] ← memory[ reg[5] + 0 ]
i2: lw   r2, 4( r1 )  // reg[2] ← memory[ reg[1] + 4 ]
i3: addi r2, r2, 1    // reg[2] ← reg[2] + 1
```

```
i1:lw   IF   ID   EX   MEM  WB
i2:lw        IF
i3:addi
```

5. Based on the lecture notes, show the pipeline cycle diagram for the standard 5-stage pipeline with forwarding, extended with an integer ALU and floating-point ALU (which are in the same execute stage but which are designated by integer instructions executing as 'E' and FP instructions executing as 'X'). Only one instruction can be in the execute stage at a time. Once the pipeline is full, what is the number of cycles between successive stores from F4? (20 pts.)

```
ld    F0,0(R1)   // 2-cycle latency, F0 <- memory[ R1 + 0 ]
ld    F6,8(R1)   // 2-cycle latency, F6 <- memory[ R1 + 8 ]
addf  F4,F2,F0   // 4-cycle execution, F4 <- F2 + F0
addf  F8,F6,F0   // 4-cycle execution, F8 <- F6 + F0
st    F4,0(R1)   // 2-cycle latency, memory[ R1 + 0 ] <- F4
st    F8,8(R1)   // 2-cycle latency, memory[ R1 + 8 ] <- F8
sub   R1,R1,16    // 1-cycle execution, R1 <- R1 - 16
bne   R1,R2,loop  // 1-cycle execution, branch to loop if R1 != R2
```

6. Repeat 5 but with the provision that an independent instruction can start into the execute stage on each cycle. All instructions must complete in program order, so single-cycle instructions cannot enter the memory stage until all previous instruction have passed through the memory stage. (20 pts.)

E.g.,

```
addf  F4,F2,F0   // 4-cycle execution, F4 <- F2 + F0          F D X X X X M W
addf  F8,F6,F0   // 4-cycle execution, F8 <- F6 + F0            F D X X X X M W
```

and

```
addf  F4,F2,F0   // 4-cycle execution, F4 <- F2 + F0          F D X X X X M W
sub   R1,R1,16    // 1-cycle execution, R1 <- R1 - 16            F D E - - - M W
```