ECE 570
High-Performance Computer Architecture
Assignment #1
Winter 2015

[20 pts]

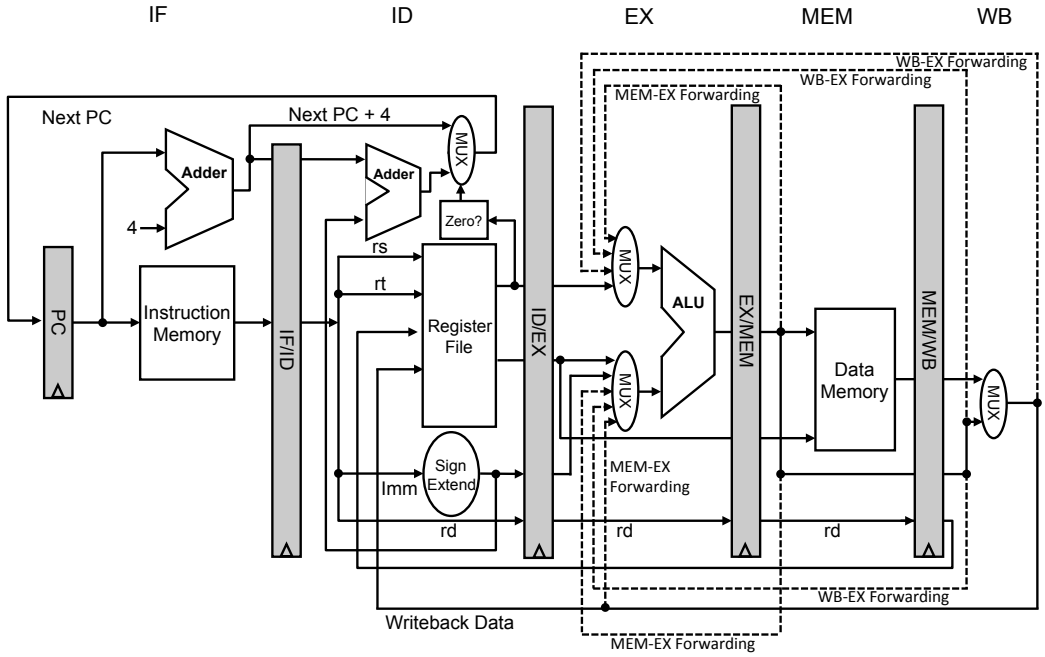
1- For this problem, consider the following two code sequences executing on the 5-stage datapath with normal forwarding and bypassing shown below:

Code 1

```
…
LD     R4, 0(R2)
ADD    R1, R4, R2
…
```

Code 2

```
…
SUB    R4, R3, R2
BNEZ   R4, Loop
…
```



(a) The first code is an example of a "load followed by its immediate use". Show and explain the correct execution timing that would allow the datapath to properly execute Code 1.

(b) The second code is an example where *MEM-to-ID forwarding* (also know as branch forwarding) will benefit on the 5-stage pipeline.

(i) Show and explain the execution timing as to why the datapath would not properly execute the above code sequence without the MEM-to-ID forwarding.

(ii) Make the necessary modifications to the datapath to implement the MEM-to-ID forwarding and allow Code 2 to run with minimal stalling and show the execution timing for the modified design. Clearly explain your design.

[20 pts]

2- Consider the following code, which represents Y = a×X+Y operation for a vector length of 100. Assume the pipeline latencies from Figure 3.2 in the text and a 1-cycle delay branch that is resolved in the ID stage. In addition, as discussed in Problem #1, the pipeline uses MEM-to-ID forwarding to forward the result of an ALU operation from the MEM stage to the ID stage.

```
        DADDIU    R4, R1, #800  ; R1 = X, R4 = upper bound for X
foo:    L.D       F2, 0(R1)     ; load X(i)
        MUL.D     F4, F2, F0    ; F0 = a, perform a*x(i)
```

```
L.D       F6, 0(R2)     ; load Y(i)
ADD.D     F6, F4, F6    ; perform a*X(i) + Y(i)
S.D       F6, 0(R2)     ; store Y(i)
DADDIU    R1, R1, #8    ; increment X index
DADDIU    R2, R2, #8    ; increment Y index
DSUBU     R3, R4, R1    ; test if done
BNEZ      R3, foo       ; loop if not done
```

(a) Show how this loop would execute without any scheduling. Maximize the performance of this code by applying both instruction reordering (also know as pipeline scheduling) and delay branch techniques. Ignoring the startup delays and assuming the loop executes 100 times, determine the number of cycles required to execute the code *before* and *after* the optimizations. Do not be concerned about what happens after the loop.

(b) Unroll the loop as many times as necessary to schedule it without stalls and show the instruction schedule. Again, assuming the loop executes 100 times, determine the number of cycles required to execute the code *before* and *after* unrolling.

[25 pts]

3- Consider the following code segment within a loop body:

```
if (aa==2)
    aa=0;
if (bb==2)
    bb=0;
if (aa!=bb) {
```

Here is the equivalent MIPS code assuming `aa` and `bb` are assigned to registers `R1` and `R2`, respectively.

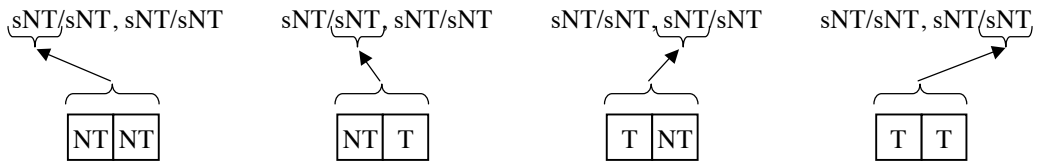
```
    DADDIU    R3, R1,#-2
    BNEZ      R3, L1        ; branch b1 (aa!=2)
    DADD      R1,R0,R0      ; aa=0
L1: DADDIU    R3, R2, #-2
    BNEZ      R3, L2        ; branch b2 (bb!=2)
    DADD      R2,R0,R0      ; bb=0
L2: DSUBU     R3,R1,R2      ; R3=aa-bb
    BEQZ      R3, L3        ; branch b3 (aa=bb)
```

Assume that the following list of 8 values of `aa` and `bb` are to be processed:

	iteration							
value	0	1	2	3	4	5	6	7
`aa`	0	2	0	2	0	2	0	2
`bb`	1	2	1	2	1	2	1	2

(a) Suppose 2-bit BPBs are used to predict the execution of the three branches in this loop. 2-bit BPBs flips when they are wrong two consecutive times and consist of four states: strongly not taken (sNT), weakly not taken (wNT), strongly taken (sT), and weakly taken (wT). Show the trace of predictions and the actual outcome of branches b1, b2, and b3 in the table shown below. The initial values of the 2–bit predictors are shown below. What are the prediction accuracies for b1, b2, and b3. What is the overall prediction accuracy?

(b) Suppose a two-level (2, 2) branch prediction is used predict the execution of the three branches in this loop. That is, in addition to the 2-bit predictor, a 2-bit global register (g) is used. Assume the 2-bit predictors are initialized to sNT and g is initialized to NT, NT, with the LSB representing the most recent branch outcome. Therefore, initial predictions, g, and their meaning are given below



Show the trace of predictions and updated g values for branches b1, b2, and b3 in the table shown below. What are the prediction accuracies for b1, b2, and b3? What is the overall prediction accuracy?

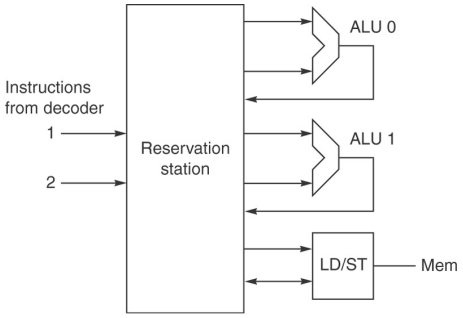
(c) Comment on the performance of 2-bit versus two-level predictors. That is, explain why one predictor performed better than the other.

[20 pts]

4- Consider the following code sequence executing on the microarchitecture shown below. Assume that the ALUs can perform all arithmetic and branch operations, and that the centralized Reservation Station (RS) can dispatch at most one operation to each functional unit per cycle (i.e., one instruction to each ALU plus one instruction to LD/ST unit). If more than two ALU instructions can be dispatched, then instructions are dispatched in program order. Assume that dispatching instructions from RS to functional units requires one cycle and once instructions are in the function units they have the latencies shown on right. Also, assume functional unit results can be fully bypassed to subsequent instructions, i.e., when a result is available at cycle *t*, any instructions dependent on the result (and have all their operands available) can be dispatched at cycle *t*+1. All functional units are fully pipelined.

```
Loop:   L.D    F2, 0(Rx)
        DIV.D  F8, F2, F0
        MUL.D  F2, F6, F2
        L.D    F4, 0(Ry)
        ADD.D  F4, F0, F4
        ADD.D  F10, F8, F2
        DADDI  Rx, Rx, #8
        DADDI  Ry, Ry, #8
        S.D    F4, 0(Ry)
        DSUB   R20, R4, Rx
        BNEZ   R20, Loop
```

Latencies	Cycles
DIV.D	10
MUL.D	4
L.D	3
ADD.D	2
DADDI, S.D, BNEZ, DSUB	1



(a) Suppose all of the instructions from the code sequence above are present in the RS without register renaming at cycle 0. Indicate all the RAW, WAR, and WAW hazards and show how the RS should dispatch these instructions using a timing table similar to the one shown below. The first `L.D` instruction dispatched at cycle 1 is shown.

Cycle	ALU0	ALU1	LD/ST
1			`L.D F2,0(Rx)`
2			
3			
4			
5			
6			
7			
8			
9			
10			
11			
12			
13			
14			
15			
16			
17			
18			
19			

(b) Now rewrite the code using register renaming. Assume the free list contains rename registers T0, T1, T2, T2, etc. Also, assume these registers can be used to rename both FP and integer registers. Suppose the code with registers renamed is resident in the RS at cycle 0. Show how the RS should dispatch these instructions out-of-order to obtain the optimal performance.

Cycle	ALU0	ALU1	LD/ST
1			
2			
3			
4			
5			
6			
7			
8			
9			
10			
11			
12			
13			
14			
15			
16			
17			
18			
19			

(c) Part (b) assumes that the centralized RS contains all the instructions in the code sequence. But in reality, the entire code sequence of interest is not present in the RS, and thus the RS must choose to dispatch what it has. Suppose the RS is initially empty. In cycle 0, the first two register-renamed instructions of the code sequence appear in the RS. Further assume that the front-end (decoder and register renaming logic) will

continually supply two new instructions per clock cycle. Show the cycle-by-cycle order of dispatch of the RS. The contents of the RS for the Cycle 0 and Cycle 1 are shown below.

Cycle 0	Cycle 1	Cycle 2	Cycle 3
L.D T0,0(Rx) DIV.D T1,T0,F0	DIV.D T1,T0,F0 MUL.D T2,F6,T0 L.D T3,0(Ry)		

Cycle 4	Cycle 5	Cycle 6	Cycle 7

Cycle 8	…	Cycle 13	Cycle 14

Cycle	ALU0	ALU1	LD/ST
1			
2			
3			
4			
5			
6			
7			
8			
9			
10			
11			
12			
13			
14			
15			
16			
17			
…			

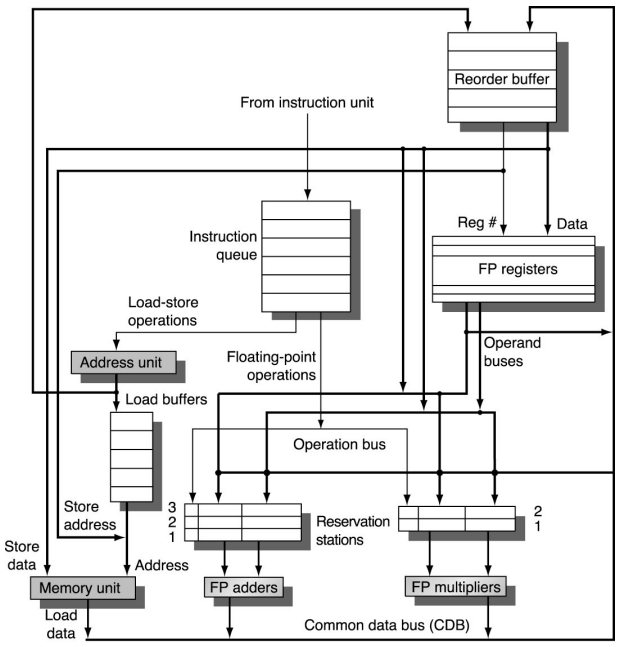
[20 pts]

5- Consider the implementation of the Tomasulo's algorithm with Reorder Buffer (ROB) shown below. It consists of four stages: Issue, Execute, Write-back, and Commit. Simulate the execution of the following piece of code using Tomasulo's algorithm and show the contents of the RS, ROB, and register file entries for each cycle (shown in the following page).

- An ROB entry contains three fields:
  - Committed – Yes (committed) and No (not committed)
  - Dest – destination register identifier
  - Data – value
- In addition to the Busy and Value fields, a register contains ROB # that indicates the ROB entry that will generate the result.
- In addition to Op, Busy, V_j, V_k, Q_j, and Q_k fields, a RS contains Dest field that indicates the ROB entry where the result will be written to. (I left out Busy field because of lack of space).

Assume the following: (1) Dual issue, write-back, and commit, i.e., two instructions can be issued, forwarded to the CDB, and committed per cycle; (2) add/subtract latency is 1 cycle and multiply/divide latency is 3 cycles; (3) an instruction can begin execution in the same cycle that it is issued, assuming all dependencies are satisfied. Also, forwarded results are immediately available for use in the next cycle. *Note that this code takes exactly 7 cycles to complete!*

```
MUL.D  F0,F2,F4
SUB.D  F8,F2,F6
DIV.D  F0,F0,F6
ADD.D  F6,F8,F2
```



Cycle 1

	Op	Dest	V_j	Q_j	V_k	Q_k
1						
2						
3						

Add/Sub

	Op	Dest	V_j	Q_j	V_k	Q_k
4						
5						

Mult/Div

	Committed Dest	Data
0		
1		
2		
3		

ROB

	ROB Busy	#	Data
0			
2			3.5
4			10.0
6			7.8
8			6.0
10			

Cycle 2

	Op	Dest	V_j	Q_j	V_k	Q_k
1						
2						
3						

Adder

Tag	Op	Dest	V_j	Q_j	V_k	Q_k
4						
5						

Mult/Div

	Committed Dest	Data
0		
1		
2		
3		

ROB

	ROB Busy	#	Data
0			
2			
4			
6			
8			
10			

Register File

Cycle 3

	Op	Dest	V_j	Q_j	V_k	Q_k
1						
2						
3						

Adder

	Op	3 Dest	V_j	Q_j	V_k	Q_k
4						
5						

Mult/Div

	Committed Dest	Data
0		
1		
2		
3		

ROB

	ROB Busy	#	Data
0			
2			
4			
6			
8			
10			

Register File

Cycle 4

	Op	Dest	V_j	Q_j	V_k	Q_k
1						
2						
3						

Adder

	Op	Dest	V_j	Q_j	V_k	Q_k
4						
5						

Mult/Div

	Committed Dest	Data
0		
1		
2		
3		

ROB

	ROB Busy	#	Data
0			
2			
4			
6			
8			
10			

Cycle 5

	Op	Dest	V_j	Q_j	V_k	Q_k
1						
2						
3						

	Op	Dest	V_j	Q_j	V_k	Q_k
4						
5						

Mult/Div

	Committed Dest	Data
0		
1		
2		
3		

ROB

	ROB Busy	#	Data
0			
2			
4			
6			
8			
10			

Cycle 6

	Op	Dest	V_j	Q_j	V_k	Q_k
1						
2						
3						

	Op	Dest	V_j	Q_j	V_k	Q_k
4						
5						

Mult/Div

Committed

	Dest	Data
0		
1		
2		
3		

ROB

	Busy	#	Data
0			
2			
4			
6			
8			
10			

Cycle 7

	Op	Dest	V_j	Q_j	V_k	Q_k
1						
2						
3						

	Op	Dest	V_j	Q_j	V_k	Q_k
4						
5						

Mult/Div

Committed

	Dest	Data
0		
1		
2		
3		

ROB

	Busy	#	Data
0			
2			
4			
6			
8			
10			