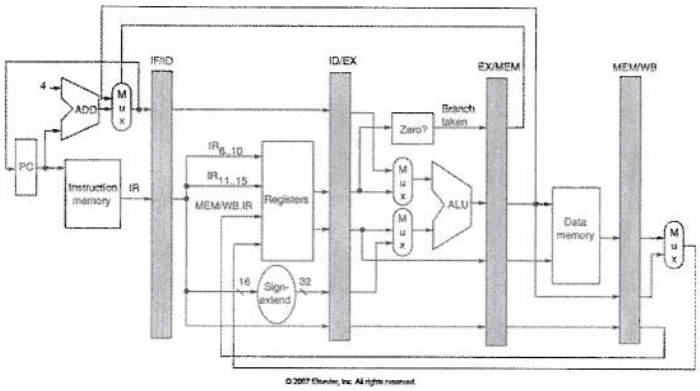
25/25

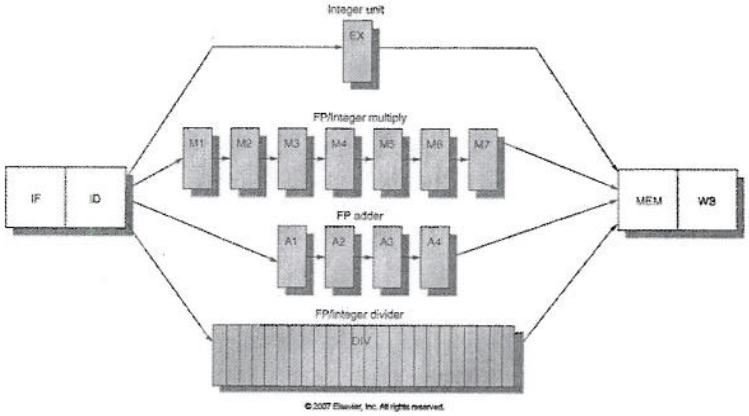
Problem #1 [25 pts]
Consider the following piece of code executing on a 5-stage pipeline and the MIPS FP pipeline shown below:

- Branch is predicted not taken and `BNEZ` is taken.
- Contention for the WB stage is resolved in the ID stage.
- There is no other forwarding besides the normal forwarding and bypassing hardware (not shown).

Clearly show and explain the timing of this instruction sequence (one iteration of the loop plus the `L.D` instruction for the next iteration).

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





Instruction	Clock Cycle																												
	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16	17	18	19	20	21	22	23	24	25	26	27	28	29	30
`L.D F2,0(Rx)`	F	D	E	M	W																									
`MUL.D F2,F0,F2`		F	D	S	E	E	E	E	E	E	E	M	W																	
`ADD.D F8,F0,F2`			F	S	D	S	S	S	S	S	S	E	E	E	E	M	W													
`L.D F4,0(Ry)`					F	S	S	S	S	S	S	D	E	M	W															
`ADD.D F4,F0,F4`												F	D	S	E	E	E	E	M	W										
`ADD.D F10,F8,F2`													F	S	D	E	E	E	E	M	W									
`S.D F4,0(Ry)`															F	D	S	S	S	S	E	M	W							
`DADDI Rx,Rx,#8`																F	S	S	S	S	D	E	M	W						
`DADDI Ry,Ry,#8`																					F	D	E	M	W					
`DSUB R20,R4,Rx`																						F	D	E	M	W				
`BNEZ R20,Loop`																							F	D	S	S	E	M	W	
`L.D F2,0(Rx)`																								F	S	S	D	E	F	D

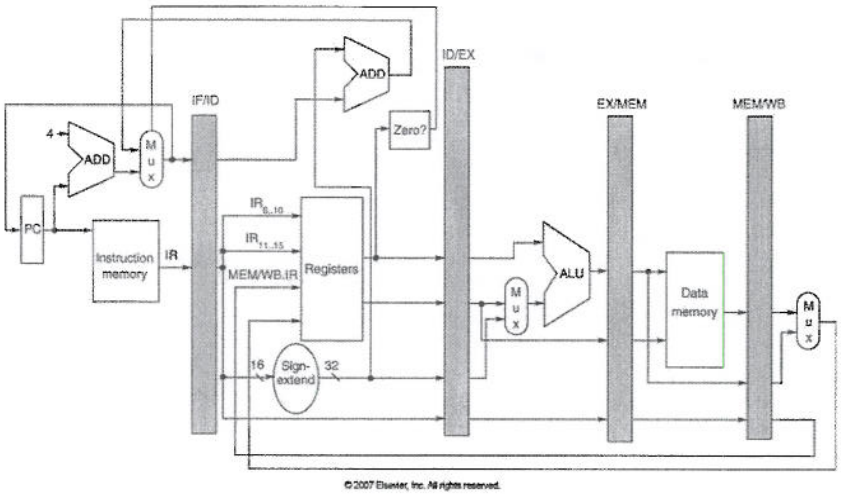
- LD F2, 0(Rx) and MUL.D F2,F0,F2 is a LD followed by immediate use, so MUL.D must still to wait for LD to forward
- stall propagates, ADD.D F8,F0,F2 must wait until MUL.D can forward from M→E
- LD F4, 0(Ry) and ADD.D F4,F0,F4 is a LD followed by immediate use, so ADD.D must still to wait for LD to forward
- stall propagates, SD must wait until ADD.D F4,F0,F4 can forward from WB→EX
- BNEZ stalls to wait for DSUB to forward R20 to register file
- BNEZ forwards BTA in MEM implying that LD cannot be fetched until cycle 29

20/25

Problem #2 [25 pts]
Consider the following example code where *load-branch forwarding* benefits on the datapath shown below.

```
...
LD      R4, 0(R2)
BNEZ    R4, Loop
...
```

(a) Show and explain why the datapath with the normal forwarding or bypassing (not shown) and the ability to handle the problem of load followed by immediate use would not properly execute the above code sequence.
(b) Make the necessary modifications to the datapath shown below with normal forwarding or bypassing (not shown) to allow the code above to run with minimal stalling and show the execution timing for the modified design. Clearly explain your design.



a)

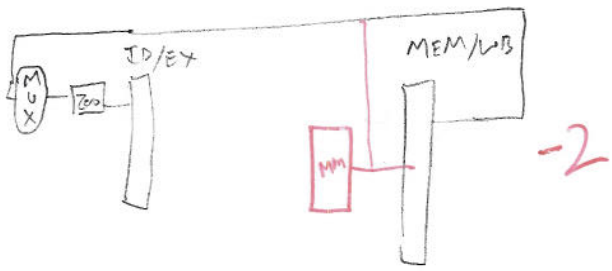
```
LD    R4, 0(R2)     F D E M W
BNEZ  R4, Loop        F D s s E M W
```

The BNEZ must stall, with normal forwarding and bypassing, to wait for LD to write R4 to the register file ✓

b) To minimize the # stalls we must add a forwarding line from the MEM stage (where 0(R2) is known) to the zero comparator in the EX stage -3

we need R4, not address calc. of load

```
LD R4, 0(R2)    F D E M W
BNEZ R4, Loop     F D s E M W
                  X
```



-2

Problem #3 [25 pts] 25/25

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

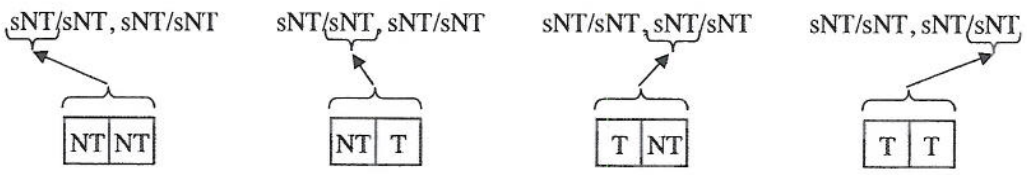
(a) Suppose 2-bit BPBs are used to predict the execution of the three branches in this loop. 2-bit BPBs flips when they are wrong two consecutive times and consist of four states: strongly not taken (sNT), weakly not taken (wNT), strongly taken (sT), and weakly taken (wT). Show the trace of predictions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

	0,1	2,2	0,1	2,2	0,1	2,2	0,1	2,2
b1 predicted	sNT	wNT	sNT	wNT	sNT	wNT	sNT	wNT
b1 actual	x T	NT	x T	NT	x T	NT	x T	NT
b2 predicted	sNT	wNT	sNT	wNT	sNT	wNT	sNT	wNT
b2 actual	x T	NT	x T	NT	x T	NT	x T	NT
b3 predicted	sNT	sNT	wNT	sNT	wNT	sNT	wNT	sNT
b3 actual	NT	x T	NT	x T	NT	x T	NT	x T

b1 = 4/8
b2 = 4/8 ✓
b3 = 4/8

total = 12/24 = 50%

(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



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

Problem #3 (cont.)

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?

g=NT,NT

aa,bb	b1 prediction	g	b2 prediction	g	b3 prediction	g
0,1	sNT/sNT, sNT/sNT	NT, T x	sNT/sNT, sNT/sNT	T, T x	sNT/sNT, sNT/sNT	T, NT
2,2	wNT/sNT, sNT/sNT	NT, NT	sNT/wNT, sNT/sNT	NT, NT	sNT/sNT, sNT/sNT	NT, T x
0,1	wNT/sNT, sNT/sNT	T, T x	sNT/wNT, sNT/sNT	T, T x	wNT, sNT, sNT/sNT	T, NT
2,2	wNT/wNT, sNT/sNT	NT, NT	sNT/wNT, sNT/wNT	NT, NT	wNT/sNT, sNT/sNT	NT, T x
0,1	wNT/wNT, sNT/sNT	T, T x	sNT/wNT, sNT/wNT	T, T x	sT/sNT, sNT/sNT	T, NT
2,2	wNT/sT, sNT/sNT	NT, NT	sNT/wNT, sNT/sT	NT, NT	sT/sNT, sNT/sNT	NT, T
0,1	wNT/sT, sNT/sNT	T, T	sNT/wNT, sNT/sT	T, T	sT/sNT, sNT/sNT	T, NT
2,2	wNT/sT, sNT/sNT	NT, NT	sNT/wNT, sNT/sT	NT, NT	sT/sNT, sNT/sNT	NT, T

b1 = 5/6
b2 = 5/6
b3 = 6/6
Total = 16/24
= 2/3
= 67%

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

The two-level predictor performed better than the 2-bit predictor because there is a correlation between b1, b2 and b3. That is, b3 is only taken when b1 and b2 not taken. Had the iterations been longer, the two-level predictor would have learned the correct predictions much better, however a 17% improvement is still good.

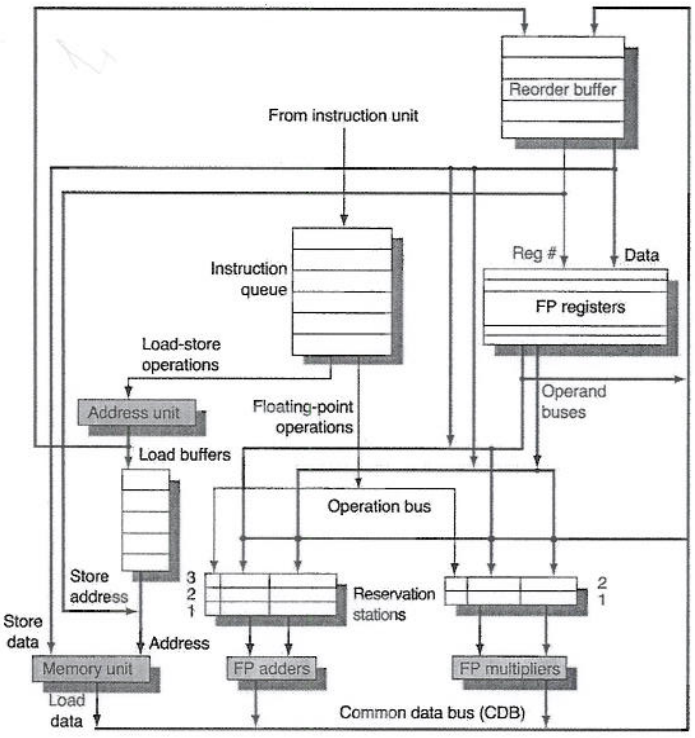
Problem #4 [25 pts] 22/25

Consider the implementation of the Tomasulo's algorithm with Reorder Buffer (ROB) shown below. It consists of four stages: Issue, Execute, Writeback, and Commit. Simulate the execution of the following piece of code using Tomasulo's algorithm and show the content 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 whether the result will be written to. (I left out the Busy field because of lack of space).

Assume the following: (1) Dual issue, wrtiteback, and commit, i.e., two instructions can be issued, forwarded to the CDB, and committed per cycle; (2) add/sub latency is 1 cycle and multiply/divide latency is 2 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 6 cycles to complete!*

```
MUL.D  F0,F2,F4
SUB.D  F8,F6,F2
DIV.D  F10,F0,F6
ADD.D  F6,F8,F2
```



Cycle 1

	Op	Dest	V_j	Q_j	V_k	Q_k
1	Sub	1	7.8		3.5	
2						
3						

Add/Sub

	Op	Dest	V_j	Q_j	V_k	Q_k
4	Mpy	0	3.5		10.0	
5						

Mult/Div

	Commited Dest	Data
0	N	0
1	N	8
2		
3		

ROB

	ROB Busy	#	Data
0	Y	0	
2			3.5
4			10.0
6			7.8
8	Y	1	6.0
10			

Register File

Value=4.3, Dest=1 (ROB#)

not reading from previous inst, DIV.D

Cycle 2

	Op	Dest	V_j	Q_j	V_k	Q_k
1	ADD	3	4.3			2
2						
3						

Adder

Tag	Op	Dest	V_j	Q_j	V_k	Q_k
4	Mpy	0	3.5		10.0	
5	Div	2		0	7.8	

Mult/Div

	Commited Dest	Data
0	N	0
1	N	8	4.3
2	N	10	
3	N	6	

ROB

	ROB Busy	#	Data
0	Y	0	
2			3.5
4			10.0
6	Y	3	7.8
8	Y	1	6.0
10	Y	2	

Register File

Cycle 3

	Op	Dest	V_j	Q_j	V_k	Q_k
1	ADD	3	4.3			2
2						
3						

Adder

	Op	Dest	V_j	Q_j	V_k	Q_k
4						
5	Div	2	35		7.8	

Mult/Div

	Commited Dest	Data
0	N	0	35
1	N	8	4.3
2	N	10	
3	N	6	7.8

ROB

	ROB Busy	#	Data
0	Y	0	
2			3.5
4			10.0
6	Y	3	7.8
8	Y	1	6.0
10	Y	2	

Register File

Cycle 4

	Op	Dest	V_j	Q_j	V_k	Q_k
1	ADD	3	4.3			2
2						
3						

Adder

	Op	Dest	V_j	Q_j	V_k	Q_k
4						
5	Div	2	35		7.8	

Mult/Div

Value = 4.49

	Commited Dest	Data
0	Y	0	35
1	Y	8	4.3
2	N	10	
3	N	6	7.8

ROB

not at head of ROB

	ROB Busy	#	Data
0			35.0
2			3.5
4			10.0
6	Y	3	7.8
8			4.7
10	Y	2	

Cycle 5

	Op	Dest	V_j	Q_j	V_k	Q_k
1	ADD	3	4.3		4.49	
2						
3						

	Op	Dest	V_j	Q_j	V_k	Q_k
4						
5						

Mult/Div

	Commited Dest	Data
0	Y	0	35
1	Y	8	4.3
2	N	10	4.49
3	N	6	7.8

ROB

	ROB Busy	#	Data
0			35.0
2			3.5
4			10.0
6	Y	3	7.8
8			4.3
10	Y	2	

Only commit when head of ROB

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

	Commited		
		Dest	Data
0	Y	0	35
1	Y	8	4.3
2	Y	10	4.49
3	Y	6	8.79

	ROB		
	Busy	#	Data
0			35.0
2			3.5
4			10.0
6			8.79
8			43
10			4.49

W
N
need extra cycle to commit
−1