ECE 570
High-Performance Computer Architecture
Assignment #1
Winter 2023

[20 pts.]

1- Consider the following two code examples where *ALU-store forwarding* will benefit on the datapath shown below.

<u>Code 1</u>

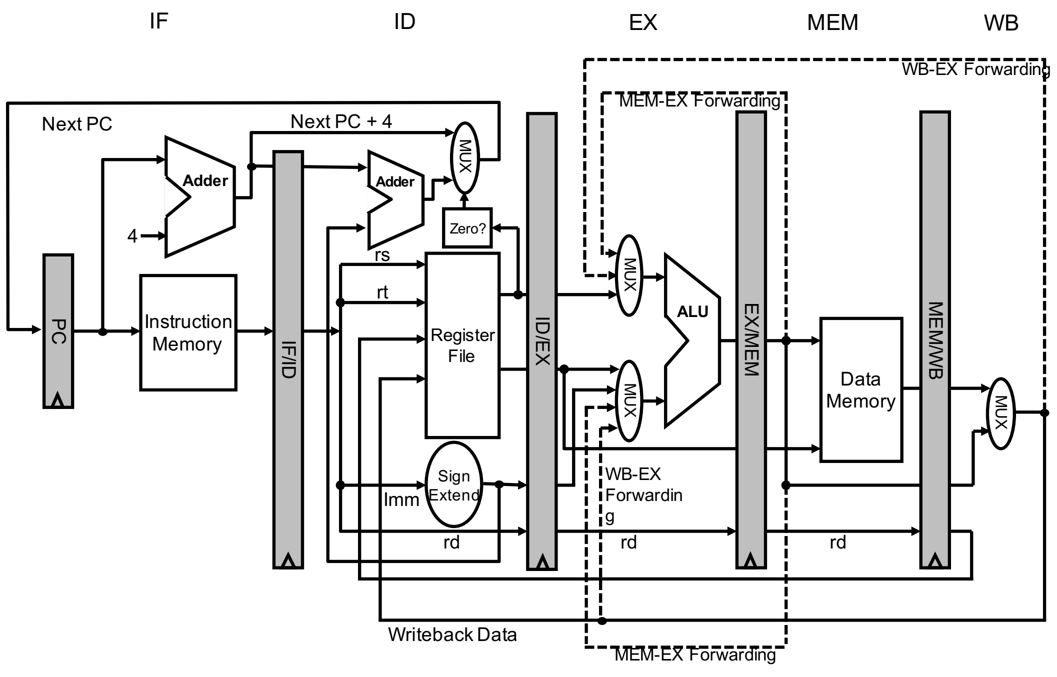
```
…
DADDI  R1, R1, #1
SW     R1, 0(R3)
…
```

<u>Code 2</u>

```
…
DADDI  R1, R1, #1
DSUB   R4, R3, R2
SW     R1, 0(R3)
…
```

(a) Show and explain why the datapath with the normal forwarding or bypassing shown below would not properly execute the above code sequences.

(b) Make the necessary modifications to the datapath shown below with normal forwarding or bypassing to allow the code above 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 assuming the pipeline latencies shown below and a 1-cycle delay branch that is resolved in the ID stage. In addition, the pipeline uses MEM-to-ID forwarding to forward the result of an ALU operation from the MEM stage to the ID stage.

```
; C code
for (i=100; i>0; i=i-1)
   x[i] = x[i] + 10
```

```
; MIPS code
loop:  L.D    F0, 0(R1)     ; F0 = array element
       ADD.D  F4,F0,F2      ; Add scalar constant
       S.D    0(R1), F4     ; Store result
       DADDIU R1, R1, #-8   ; Decrement array ptr.
       BNEZ   R1, R2, loop  ; Branch if R1!=R2
```

Latencies of FP operations

Instruction producing result	Instruction using result	Latency in clock cycles
FP ALU op	Another FP ALU op	3
FP ALU op	Store double	2
Load double	FP ALU op	1
Load double	Store double	0

(a) Show how this loop would execute without any scheduling. Maximize the performance of this code by applying both instruction reordering (also known 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 three times (i.e., make four copies) to schedule it without stalls and show the instruction schedule. Again, determine the number of cycles required to execute the code *before* and *after* unrolling with scheduling.

[20 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)
```

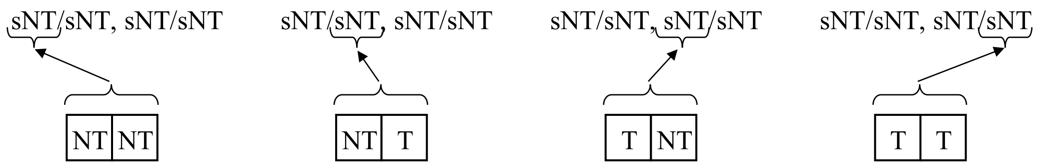
Assume that the following list of 8 values of `aa` and `bb` are to be processed:

	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 flip 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 all initialized to 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							
b1 actual								
b2 predicted	sNT							
b2 actual								
b3 predicted	sNT							
b3 actual								

(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 all initialized to sNT and g is initialized to (T, 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 for branches b1, b2, and b3 and updated g values in the table shown below. What are the prediction accuracies for b1, b2, and b3? What is the overall prediction accuracy?

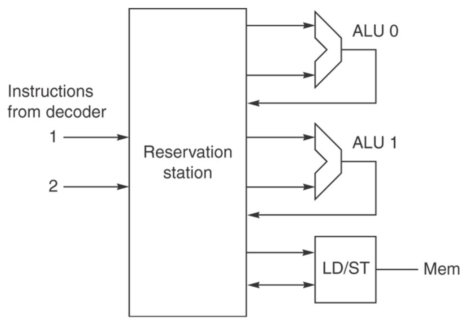
g=T, NT

aa,bb	**b1 prediction**	**g**	**b2 prediction**	**g**	**b3 prediction**	**g**
0, 1	sNT/sNT, sNT/sNT		sNT/sNT, sNT/sNT		sNT/sNT, sNT/sNT	
2, 2						
0, 1						
2, 2						
0, 1						
2, 2						
0, 1						
2, 2						

[20 pts.]

4- Consider the following code sequence executing on the microarchitecture shown below with the following assumptions:

```
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
```



- The ALUs can perform all arithmetic and branch operations, and that the centralized Reservation Station (RS) can dispatch at most one instruction to each functional unit (FU) per cycle (i.e., one instruction to each ALU plus one instruction to the LD/ST unit).
- (2) Dispatching instructions from RS to FUs requires one cycle and once instructions are in the FUs they have the latencies shown below:

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

- The front-end (decoder and register renaming logic) will continually supply two new instructions per clock cycle to the RS.
- At cycle 0, the first two register-renamed instructions of the code sequence are in the RS and dispatching is performed.
- There is no bypassing of results from FUs to dependent instructions, i.e., (1) results from FUs and/or LD/ST must be first written to the RS at the end of cycle *t*; (2) the RS can dispatch dependent instructions to FUs at cycle *t*+1, and (3) instructions can execute at cycle *t*+2.
- All FUs are fully pipelined.

(a) Suppose all 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 these instructions are dispatched and executed in the FUs using a timing table like 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			
…			

(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 resides in the RS at cycle 0. Show how these instructions are dispatched and executed out-of-order in the functional units to obtain the optimal performance using a timing table like the one in part (a).

(c) Part (b) assumes that the centralized RS contains all the instructions in the code sequence. But in practice, the entire code sequence of interest is not present in the RS, and thus the RS must 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 dispatch order of the RS using the format shown below. The contents of the RS for the Cycle 0 and Cycle 1 are shown below, and the instruction(s) highlighted in red indicate it (they) is (are) being dispatched in the current cycle and will start executing in the next cycle.

<u>Cycle 0</u>	<u>Cycle 1</u>	<u>Cycle 2</u>	<u>…</u>
`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)`		
…	…		
…	…		
…	…		

(d) Show how the RS should dispatch the instructions in part (c) using a timing table like the one shown part (a).

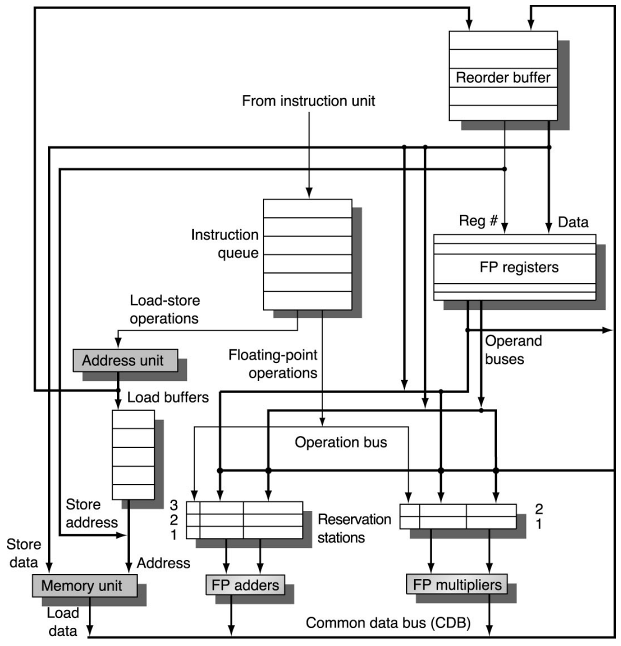
[20 pts.]

5- Consider the implementation of the Tomasulo's algorithm with Reorder Buffer (ROB) shown below, which 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 below).

- 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 latency is 1 cycle and multiply 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 7 cycles to complete!*

```
ADD.D  F4,F0,F8
MUL.D  F2,F0,F4
ADD.D  F4,F4,F8
MUL.D  F8,F4,F2
```



	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

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

Register File