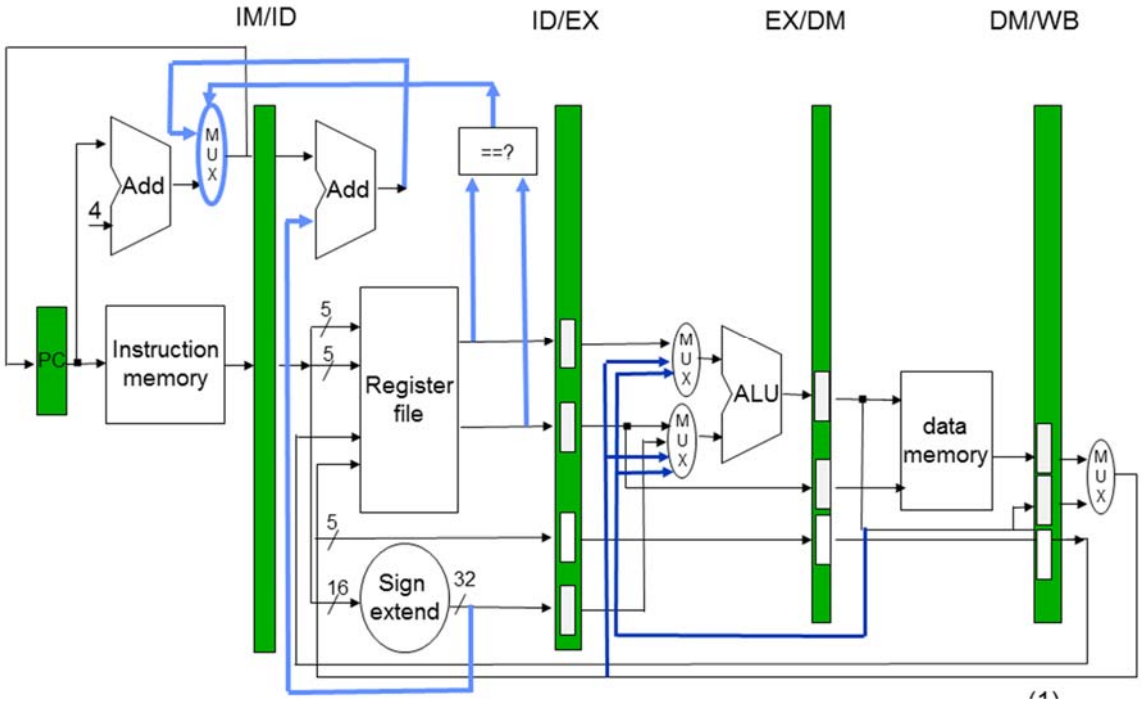
CS 2410 Mid term (fall 2019) **Name**:_____

Question 1 (2+12+4+2+5=25 points): In the five stage pipeline showed in the figure, a comparator is added in the ID stage to allow the branch to be resolved in the ID stage. Moreover, forwarding paths are shown from the WB and DM stages to the EX stage to prevent RAW hazards.



a) Argue that even with the forwarding paths shown, there is still a hazard when an instruction that writes into a register, R, is followed (not necessarily immediately) by a branch instruction that reads from R.

An instruction, I, which writes into R will do so when it is in WB – there will be a data hazard if the branch instruction reads R in ID and uses it to determine the branch condition before I is in WB.

Hence, the branch should be separated from I by at least two other instructions (assuming that the instruction in WB writes into the register file in the first half of a cycle and the instruction in ID reads from R in the second half of the cycle).

b) It is possible to resolve the data hazard indicated in part (a) by adding new forwarding paths to the architecture. Sketch on the figure the new forwarding path(s) that will prevent the hazard.

Add two muxes, one to each input of the comparator. Each mux will have three inputs;

1) the current input to the comparator,
2) a path from the ALU output (forwarding when the instruction I immediately follows the branch)
3) a path from the output of the data memory output (forwarding when I immediately is separated from the branch by one instruction)

c) An alternative solution is to stall the pipe when the dependence described in (a) is detected. How many cycles should the pipelined be stalled to prevent the hazard described in (a)?

If the branch follows I immediately, then there should be a 1-cycle stall

If the branch is separated from I by one instruction, then there should be a 2-cycle stall

d) A third way to prevent the hazard described in (a) is to move the branch resolution (target address and condition) to the EX stage. What is the disadvantage of this solution compared to adding new forwarding paths (as in part c)?

A taken branch will stall the pipeline for two cycles rather than one.

e) Consider the two designs in (b) and (d). In the first design, the branch condition and target are resolved in the ID stage, while in the second design, the branch condition and target are resolved in the EX stage. Actual implementation shows that the clock cycle time is longer by 10% when the branch is resolved in ID. Assume that the instruction mix executing on the pipeline contains 20% branch instructions, 60% of which are taken.

- If the CPI of both designs is 1.5 if we do not account for stalling due to control hazards, what is the CPI for each of the two designs when accounting for the effect of control hazards?

  CPI_{ID} = the CPI if branch is resolved in ID = 1.5 + 0.2 * 0.6 * 1 = 1.62

  CPI_{EX} = the CPI if branch is resolved in EX = 1.5 + 0.2 * 0.6 * 2 = 1.74

- Specify which of the two designs is more efficient.

  Let x be the cycle time if branches are resolved in ID and compare the average instruction execution time for both designs (CPI * cycle time)

  1.62 * 1.1 * x = 1.782 * x is greater than 1.74 * x

  Hence, resolving the branch is EX is more efficient.

Question 2 (8+6+6=20 points)

(a) Consider the following code for a nested loop and assume that the inner loop executes 10 iterations and the outer loop executes 50 iterations.

```
L:   lw     $1, 100($5)
     lw     $2, 800($6)
     addi   $5, $5, -4
     add    $3, $1, $2
     sw     $3, 804($5)
B1:  bneq   $5, $10, L
     addi   $6, $6, -4
B2:  bneq   $6, $10, L
     …..
```

Complete the following sentences assuming that B1 and B2 do not collide in the Branch Target Buffer:

- A 1-bit branch predictor will correctly predict the branch at B1 400 times out of _500_ times
- A 1-bit branch predictor will correctly predict the branch at B2 48 times out of _50__ times
- A 2-bit branch predictor will correctly predict the branch at B1 449 times out of _500_ times
- A 2-bit branch predictor will correctly predict the branch at B2 48 times out of _50__ times

(b) Consider an 8-stage instruction pipeline (stages: IF1, IF2, ID, EX1, EX2, M1, M2, WB) where branch addresses are resolved at the end of the ID stage and branch conditions are resolved at the end of the EX2 stage. The typical workload has 20% conditional branches with 75% taken. Accounting only for control hazards (ignoring any other hazards), compute the CPI if the "predict-taken" scheme is used? Assume that no Branch Prediction Buffer/Table is used.

CPI = 1 + 0.2 (0.75 * 2 + 0.25 * 2) = 1 + 0.2 (2) = 1 + 0.4 = 1.4

The branch address is only known in ID, at this point we start to fetch from the target address. There are already two instructions in the pipe (will be removed if the branch is actually taken). When the branch condition is resolved in EX2, there are 4 instructions in the pipe, two from the normal flow and two from the target address. The two instructions in ID and EX1 are removed if the branch is taken and the two in IF1 and IF2 are removed if the branch is not taken.

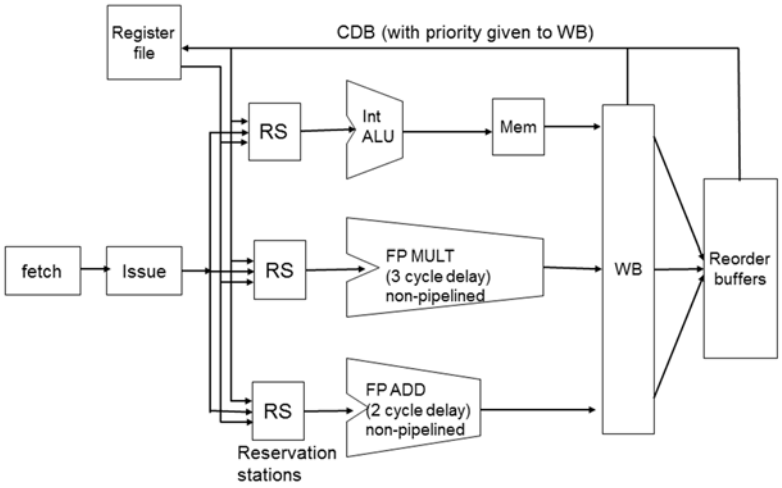
(c) Indicate which of the following statement is true and which is false:

	T	F
In the open wow policy, a row is closed only if another row is to be accessed	x	
VLIW architectures rely on the compiler to avoid data hazards.	x	
The hit rate for an 8-way set associative cache is smaller than the hit rate for a 4-way associative cache of the same size		x
A TLB miss causes a page fault		x
DRAM cells have to be periodically refreshed	x	
Variable length op-code is used to simplify the decoding stage in a pipeline		x

Question 3 (17+8=25 points): In this question, you will trace the execution of the shown eight instructions on the dynamically scheduled (with speculation) out of order processor shown in the figure assuming that:

- The cache is perfect (no misses),
- The number of reservation stations and reorder buffers is very large, with the ROBs used to rename the registers
- Up to two instructions can be issued in the same cycle as long as they are not issued to the same execution unit,
- The architecture has only one CDB with priority given to the WB stage. In case of structural hazard on the WB stage, the instruction that issued first has higher priority.
- If an instruction is in WB while occupying the first entry in the ROB then it is committed in the same cycle (see the first two entries of the table where C indicates that the instruction has been committed and the ROB entry has been released).
- The "Int ALU" is used for integer operations as well as for memory address computation of load and store instructions.

I0:	fld	F0, 0(R1)
I1:	fld	F2, 8(R1)
I2:	fmul.d	F0, F0, F2
I3:	fadd.d	F1, F0, F1
I4:	addi	R1, R1, 16
I5:	fld	F2, 8(R1)
I6:	fadd.d	F0, F0, F2
I7:	bne	R1, R2, L



(a) Complete the table in the last page of the exam booklet (can detach it) to indicate the state of each instruction in consecutive cycles (I=issued, EX=executing, M=accessed memory, WB=write back, ROB= in ROB, C=committed).

(b) Complete the following table which specifies the relevant entries of the register status table at the end of cycles 1, 2, 3, 4 and 5.

	F0		F1	F2	…..	R1	…..
Cycle 1	ROB0		--	--		--	
Cycle 2	ROB 2		--	ROB1		--	
Cycle 3	ROB 2		ROB3	ROB1		ROB4	
Cycle 4	ROB6		ROB3	ROB5		ROB4	
Cycle 5							

Question 4 (5+2.5+5+2.5=15 points)
Assume a system with a 4-way set associative L1 cache and a 8-way set associative L2 cache, both with block size of 8 words. The hit times for the L1 cache and the L2 cache are 1 and 5 cycles, respectively. Assume also that the L1 miss rate is 10% and that 80% of the misses in L1 are found in L2. On a L2 miss, a block is fetched from memory and it takes 92 cycles for the first word of the block to reach the L2 and 4 cycles for each of the 7 following words to reach the cache. That is, the L2 miss penalty is 120 cycles. Ignore any time needed to transfer blocks (or words) from L2 to L1 and to the CPU.

a) Compute the average memory access time (in cycles)

$AMAT_a =$ $1 + 0.1 * (5 + 0.2 * 120) =$
$1 + 0.1 * (5 + 24) =$
$1 + 2.9 = 3.9$

b) What would be the L2 miss penalty and the average memory access time if early restart and critical word first are used when a miss occurs in L2 and a block is fetched from memory.

Miss penalty = 92

$AMAT_b =$ $1 + 0.1 * (5 + 0.2 * 92) =$
$1 + 0.1 * (5+18.4) =$
$1 + 2.34 = 3.34$

c) Assume that **early restart and critical word first are not used**, but that way prediction is used for L1. With correct prediction, the L1 hit time is still 1 cycle, but a pseudo hit (way mis-prediction) takes 2 cycles (one cycle after the discovery that the predicted way resulted in a miss). What is the AMAT if the way prediction is correct 60% of the time?

$AMAT_c =$ $(0.6 * 1 + 0.4 * 2) + 0.1 * (5 + 0.2 * 120) =$

$1.4 + 2.9 = 4.3$

d) If your answers to (a) and (c) are correct, then $AMAT_c$ should be larger than $AMAT_a$. Why would anyone suggest the use of way prediction??

Way prediction results in a faster access to the cache since it hit time is that of a direct mapped cache rather than an associative cache. Hence, if the clock cycle time is determined by the cache access time, then way prediction can lead to a faster clock.

Question 5 (4+3+3+5=15 points): Consider the execution of the following code for the multiplication of a matrix A with two matrices B1 and B2 on a CPU with a fully associative L1 cache (with LRU replacement) having block size of 4 words.

```
for (i=0 ; i < 100 ; i++)
  for(j=0;  j < 100;  j++) {
     T1 = T2 = 0 ;
     for (k=0 ; k < 100  ; k++) {
                    T1 = T1 + A[i,k] * B1[k,j] ;
                    T2 = T2 + A[i,k] * B2[k,j] ;
                    }
     C1[i,j] = T1 ;
     C2[i,j] = T2 ;
  }
```

Assume that T1, T2, i, j and k are allocated to registers and that matrices are allocated row-wise in memory (for example, A is allocated in the order A[0,0], A[0,1], … , A[0,n-1], A[1,0],….). Also assume that the cache is a write through with "no write allocate" on a write miss – that is elements of C1 and C2 are not allocated in the cache.

(a) If the cache is very large (enough to store all elements of A, B1 and B2 simultaneously), then
the average miss rate while accessing the elements of A is __0.125%__,
the average miss rate while accessing the elements of B1 is __0.25%__, and
the average miss rate while accessing the elements of B2 is __0.25%__.

(b) If the cache can only store 12000 elements of A, B1 or B2 simultaneously (note that one matrix has 10000 elements), then
the average miss rate while accessing the elements of B1 is __25%__, and
the average miss rate while accessing the elements of B2 is __25%__.

(c) If the cache can only store 200 elements of A or B simultaneously, then
the average miss rate while accessing the elements of B1 is __100%__, and
the average miss rate while accessing the elements of B2 is __100%__.

(d) If your architecture is reflected by case b (cache can hold 12000 elements), describe a way to rewrite the code such that you can improve the hit rate to a level similar to that of a very large cache (case a).

Can split the code into two separate parts, one for A*B1 and the other for A*B2. Each part will still have three nested loops, but now the entire matrix (B1 or B2) will fit in the cache (in addition to one row of A).

	Instruction status in cycle															
	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16
fld F0, 0(R1)	I	EX	M	WB	C											
fld F2, 8(R1)		I	EX	M	WB	C										
fmul.d F0, F0, F2		I	I	I	I	EX	EX	EX	WB	C						
fadd.d F1, F0, F1			I	I	I	I	I	I	I	EX	EX	WB	C			
addi R1, R1, 16			I	EX		WB	ROB	ROB	ROB	ROB	ROB	ROB	ROB	C		
fld F2, 8(R1)				I	I	I	EX	M	M	WB	ROB	ROB	ROB	ROB	C	
fadd.d F0, F0, F2				I	I	I	I	I	I	I	I	EX	EX	ROB	ROB	C
bne R1, R2, L					I	EX		WB	ROB	ROB	ROB	ROB	ROB	ROB	ROB	ROB