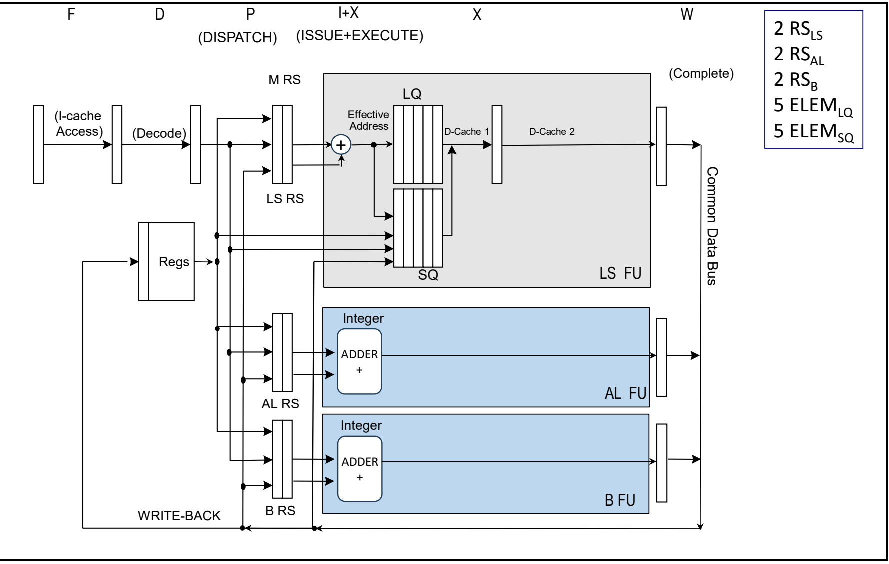
HIGH PERFORMANCE COMPUTER ARCHITECTURE 04-02-2013 **MATRICOLATION NO.**____________________

SURNAME____________________

REVISED 22-10-2024 **FIRST NAME**____________________

1) (POINTS 14/40) Consider a **single-dispatch (1 instruction per cycle)** processor using Tomasulo's algorithm to perform the dynamic scheduling of instructions on the pipeline shown in the following figure. This pipeline is executing the following program, which performs a search within a vector (initially, R1=0).

```
etic: LW   R2, 0(R1)    ; read Xi
      ADDI R2, R2, 1    ; adds 1 to Xi
      SW   R2, 0(R1)    ; write Xi
      ADDI R1, R1, 4    ; update  R1
      BNE  R2, R0, etic ; continue to loop if false
```



Working hypothesis:
- the loop executes speculatively in terms of direction (always taken) but not regarding the branch condition; high-performance fetch breaks after fetching a branch; case A) no-speculation on branch condition; case B) speculation on branch condition;
- the issue stage (I) calculates the address of the actual read/write and push it into load/store queues; **only 1 instruction is issued per cycle**
- **reads require 2 clock cycles**; writes take 1 cycle
- when accessing memory (M), **writes** have precedence over reads and must be executed in-order
- **there is a single CDB**
- dispatch stage (P) and complete stage (W) require 1 clock cycle
- only 1 instruction is committed (C stage) per cycle
- there are separated integer units: one for the calculation of the actual address, one for arithmetic and logical operations, one of the integer multiplication and one for the evaluation of the branch condition, as illustrated in this table:

Type of Functional Unit	No. of Functional Units	Cycles for stage I+X	No. of reservation stations
LS: Integer (effective addr.)	1	1	2
A: Integer (op. A-L)	1	1	2
B: Integer (branch calc.)	1	1	2

- the functional units do not take advantage of pipelining techniques internally
- **reservation stations are busy until the end of CDB-write, except for Stores)**
- the load queue has **5 slots**; the store queue has **5 slots** (writes wait for the operand in the store queue, i.e., in the execution stage)

Complete the following chart until the end of the **third** iteration of the code fragment above, both in the case of simple dynamic scheduling (case A) that in the case of dynamic scheduling with speculation (case B).

Instr. No..	Instruction name	P: disPatch (start-stop)	I+X:Issue+Exec (start-stop)	M: MEM.ACCESS (start-stop)	W: CDB-write (clock)	C: Commit (clock)	Comments
I01	**LW R2,0(R1)**	1-5	2	3-4	5	6	
...	...						
...	...						

1) (POINTS 6/40) Given the sequence P1: R, P2: W, P3: R, P1: W, P2: R, P3: W (Px:R = read by the processor Px, Px:W write by the processor Px), with respect to a certain variable 'a ', show for each processor the sequence of states, and transactions on the bus that occur in a multiprocessor UMA with write-back cache of each processor and coherence protocol DRAGON.

2) (POINTS 8/40) Explain the operation and draw a diagram of a PAp branch 2-level predictor with a 12-bit BSHR and size 2^{12} x 2 bit for the PHT.

3) (POINTS 6/40) Explain the operation of the Reorder Buffer by making reference to a superscalar processor with dynamic scheduling as described in the question no.1 but with dual dispatch.

4) (POINTS 6/40) Write a CUDA program to perform in parallel the matrix multiplication of two square matrices of size 1024x1024.

HIGH PERFORMANCE COMPUTER ARCHITECTURE 04-02-2013
SOLUTION
REVISED 22-10-2024

EXERCISE 1

CASE A (no speculation on branch condition: dispatch WAITS for branch condition verification):

Iter.	Instruction	P: Dispatch (start-stop)	I+X: Issue+Exec (start-stop)	M: MEM ACCESS (start-stop)	W: CDB-write (clock)	C: Commit (clock)	Comments
1	`LD R2,0(R1)`	1-5	2	3-4	5	6	
1	`ADDI R2, R2, 1`	2-7	6		7	8	I waits R2 from 1/LW
1	`SD R2, 0(R1)`	3-3	4-7	8		9	M waits R2 from 1/ADDI_R2
1	`ADDI R1, R1, 4`	4-6	5		6	10	
1	`BNE R2, R0, etic`	5-8	8			11	I waits R2 from 1/ADDI_R2
2	`LD R2,0(R1)`	9-13	10	11-12	13	14	P waits decision su 1/BNE
2	`ADDI R2, R2, 1`	10-15	14		15	16	I waits R2 from 2/LW
2	`SD R2, 0(R1)`	11-16	12-15	16		17	M waits R2 from 2/ADDI_R2
2	`ADDI R1, R1, 4`	12-14	13		14	18	
2	`BNE R2, R0, etic`	13-16	16			19	I waits R2 from 2/ADDI_R2
3	`LD R2,0(R1)`	17-21	18	19-20	21	22	P waits decision su 2/BNE
3	`ADDI R2, R2, 1`	18-23	22		23	24	I waits R2 from 3/LW
3	`SD R2, 0(R1)`	19-24	20-23	24		25	M waits R2 from 3/ADDI_R2
3	`ADDI R1, R1, 4`	20-22	21		22	26	
3	`BNE R2, R0, etic`	21-24	24			27	I waits R2 from 3/ADDI_R2

CASE B (speculation: dispatch DOES NOT WAIT for branch condition verification):

Iter.	Instruction	P: Dispatch (start-stop)	I+X: Issue+Exec (start-stop)	M: MEM ACCESS (start-stop)	W: CDB-write (clock)	C: Commit (clock)	Comments
1	`LD R2,0(R1)`	1-5	2	3-4	5	6	
1	`ADDI R2, R2, 1`	2-7	6		7	8	I waits R2 from 1/LW
1	`SD R2, 0(R1)`	3-3	4	8		9	M waits R2 from 1/ADDI_R2
1	`ADDI R1, R1, 4`	4-6	5		6	10	
1	`BNE R2, R0, etic`	5-8	8			11	I waits R2 from 1/ADDI_R2
2	`LD R2,0(R1)`	6-10	7	9-10	11	12	I waits R1 from 1/ADD_R1; conflict on M
2	`ADDI R2, R2, 1`	7-12	12		13	13	I waits R2 from 2/LW
2	`SD R2, 0(R1)`	8-8	9	14		14	M waits R2 from 2/ADDI_R2
2	`ADDI R1, R1, 4`	9-11	10			15	
2	`BNE R2, R0, etic`	10-13	14			16	I waits R2 from 2/ADDI_R2
3	`LD R2,0(R1)`	11-15	13	15-16	17	18	conflict on I; conflict on M
3	`ADDI R2, R2, 1`	12-17	18		19	20	I waits R2 from 3/LW; conflict on CDB
3	`SD R2, 0(R1)`	13-13	15	20		21	conflict on I; I waits R2 from 3/ADDI_R2
3	`ADDI R1, R1, 4`	14-16	16		18	22	conflict on I; conflict on CDB
3	`BNE R2, R0, etic`	15-18	20			23	I waits R2 from 3/ADDI_R2