**HIGH PERFORMANCE COMPUTER ARCHITECTURE midterm exam 31-10-2017** **MATR.NO.**_____

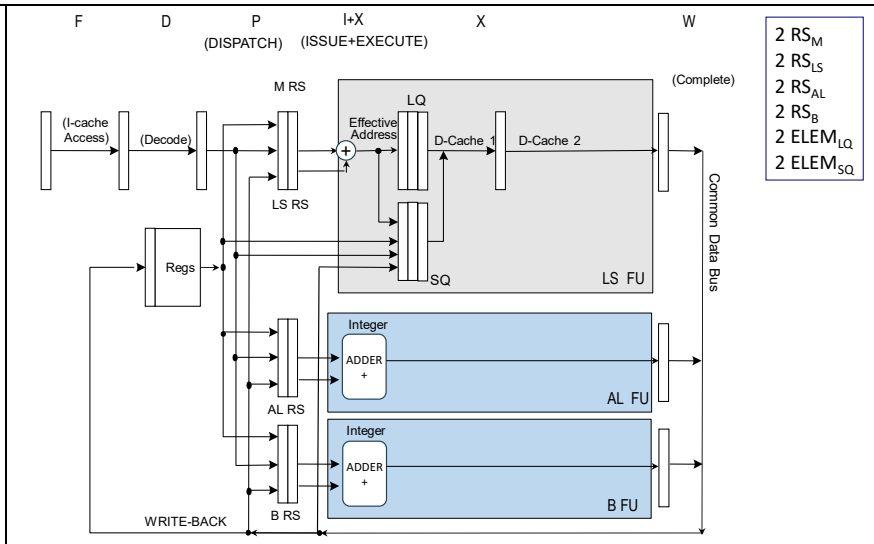
**(REVISED 21-10-2024)**

**SURNAME**_____

**FIRST NAME**_____

1) (POINTS 35/40) Consider a **triple-dispatch (3 instructions 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, 3     ; adds 3 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) and regarding the branch condition; high-performance fetch breaks after fetching a branch
- 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 5 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
- **ASSUME** that the reservation stations could be freed right before the start of issue phase (therefore extending the duration of P stage)
- 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
- the load queue has 3 slots; the store queue has 3 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 FOURTH iteration of the above code fragment in the case of dynamic scheduling with speculation. Also add the instruction that occupies a certain reservation station (one of the 6) as indicated:

| Instr. No.. | Instruction name | ALU RS1 | ALU RS2 | LS RS1 | LS RS2 | BU RS1 | BU RS2 | P: disPatch (clock) | I+X:Issue+Exec (clock) | M: MEM.ACCESS (start-stop) | W: CDB-write (clock) | C: Commit (clock) | Comments |
|---|---|---|---|---|---|---|---|---|---|---|---|---|---|
| I01 | **LW R2,0(R1)** | | | I01 1-1 | | | | 1 | 2 | 3-7 | 8 | 9 | |
| ... | ... | | | | | | | | | | | | |
| ... | ... | | | | | | | | | | | | |

1) (5/40) Explain what is (i) an anti-dependency and (ii) an output-dependency and give an assembly example for each one of the two cases in the code of the question (1).

**HIGH PERFORMANCE COMPUTER ARCHITECTURE midterm exam 31-10-2017 – SOLUTION (REVISED 22-10-2024)**

1)

| Count | Instruction | ALU RS1 | ALU RS2 | EAC RS1 | EAC RS2 | B RS1 | B RS2 | P: Dispatch (start-stop) | I+X: Issue+Exec (clock) | M: MEM (start-stop) | W: CDB-write (clock) | C: Commit (clock) | Comments |
|---|---|---|---|---|---|---|---|---|---|---|---|---|---|
| 01 | LW R2,0(R1) | | | I01 1-1 | | | | 1 | 2 | 3-7 | 8 | 9 | |
| 02 | ADDI R2,R2,1 | I02 1-8 | | | | | | 1 | 9 | -- | 10 | 11 | I waits R2 from 1/LW |
| 03 | SW R2,0(R1) | | | | I03 1-2 | | | 1 | 3 | 13 | -- | 14 | I waits issue logic; M waits R2 M waits mem (store alredy in queue and must be executed in order) |
| 04 | ADDI R1,R1,4 | | I04 2-3 | | | | | 2 | 4 | 4 | 5 | 15 | I waits issue logic; |
| 05 | BNE R2,R0,etic | | | | | I05 2-10 | | 2 | 11 | -- | -- | 16 | I waits R2 from 1/ADDI-R2 |
| 06 | LW R2,0(R1) | | | I06 3-5 | | | | 3 | 6 | 8-12 | 13 | 17 | I waits R1; M waits mem |
| 07 | ADDI R2,R2,1 | | I07 4-13 | | | | | 4 | 14 | -- | 15 | 18 | P waits A-RSs; I waits R2 from 2/LW; |
| 08 | SW R2,0(R1) | | | | I08 4-6 | | | 4 | 7 | 19 | -- | 20 | I waits R1; I waits issue logic; M waits R2; M waits mem |
| 09 | ADDI R1,R1,4 | I09 9-9 | | | | | | 9 | 10 | -- | 11 | 21 | P waits A-RSs; |
| 10 | BNE R2,R0,etic | | | | | | I10 9-15 | 9 | 16 | -- | -- | 22 | I waits R2 from 2/ADDI-R2; |
| 11 | LW R2,0(R1) | | | I11 10-11 | | | | 10 | 12 | 14-18 | 19 | 23 | I waits issue logic; I waits R1; M waits mem |
| 12 | ADDI R2,R2,1 | I12 10-19 | | | | | | 10 | 20 | -- | 21 | 24 | P waits A-RSs; I waits R2 from 3/LW |
| 13 | SW R2,0(R1) | | | | I13 10-12 | | | 10 | 13 | 25 | -- | 26 | I waits R1; I waits issue logic; M waits R2; M waits mem |
| 14 | ADDI R1,R1,4 | | I14 14-14 | | | | | 14 | 15 | -- | 16 | 27 | P waits A-RSs; |
| 15 | BNE R2,R0,etic | | | | | I15 14-21 | | 14 | 22 | -- | -- | 28 | I waits R2 from 3/ADDI-R2 |
| 16 | LW R2,0(R1) | | | I16 15-16 | | | | 15 | 17 | 20-24 | 25 | 29 | I waits R1; M waits mem; |
| 17 | ADDI R2,R2,1 | | I17 15-25 | | | | | 15 | 26 | -- | 27 | 30 | P waits A-RSs; I waits R2 from 4/LW |
| 18 | SW R2,0(R1) | | | | I18 15-17 | | | 15 | 18 | 28 | -- | 31 | I waits R1; I waits issue logic; M waits R2; M waits mem; |
| 19 | ADDI R1,R1,4 | I19 20-20 | | | | | | 20 | 21 | -- | 22 | 32 | P waits A-RSs |
| 20 | BNE R2,R0,etic | | | | | | I20 20-27 | 20 | 28 | | -- | 33 | I waits R2 from 4/ADDI-R2 |

2) Given two subsequent instructions I and J:
   - Antidependency: I reads a register that is written by J
   - Output-dependency: both I and J write in the same register

In the case of our code:

- Antidependency:

```
I05: bne R2, R0, etic
I06: lw R2, 0(R1)

I05: bne R2, R0, etic
I07: addi R2, R2, 1
```

- Output-dependency:

```
I01: lw R2, 0(R1)
I02: addi R2, R2, 1

I04: addi R1, R1, 4
I09: addi R1, R1, 4

I02: addi R2, R2, 1
I07: addi R2, R2, 1
```