HIGH PERFORMANCE COMPUTER ARCHITECTURE 06-11-2015 **MATRICULATION NO.**____________________

(MIDTERM TEST) **SURNAME**____________________

FIRST NAME____________________

1) (POINTS 30/40)
Consider the following snippet of code running on 4-ways out-of-order superscalar processor.
Initially, all registers contain zero.

```
lab1:   LW   R2,0(R1)
        MUL  R2,R2,R2
        SW   R2,0(R1)
        ADDI R1,R1,4
        BNE  R2,R0,lab1
```

Working hypothesis:
* the fetch, decode and commit stages are 4 instructions wide
* the instruction window has 3 slots
* we have 18 physical registers in the free pool
* the reorder buffer has unlimited size
* the integer multiplier has 4 stages
* the load/store queues have 3 slots each and a common effective-address calculation unit
* there are 4 ALUs for arithmetic and logic operations and for branching
* an ALU performs its operation in the same cycle when the operation is issued
* reads require 1 clock cycle (after the addressing phase)
* the register file has 4 input- and 4 output-ports
* there are 9 logical registers (including R0 which is hardwired to 0)
* the store operation leaves the issue stage as it is inserted in the store queue

In order to calculate the total cycles needed to execute 3 iterations of the above loop on such machine, complete the following chart until the end of the third iteration of the code fragment above, including the renamed stream the precise evolution of the free pool of the physical registers (the register map), the Instruction Window, the Reorder Buffer (ROB) and the Load Queue (LQ) and Store Queue (SQ).

Iter.	Instruction	F- Fetch (clock)	D - Decode (clock)	P - disPatch (clock)	I - Issue (clock)	X – eXecute (clock)	W – Write-back (clock)	C - Commit (clock)	Renamed Stream	Instruction Window Pi Pj Pk I Qj Qk	Reorder Buffer Ri Pi,old Cplt	Load Queue	Store Queue
1	**LW R2,0(R1)**	0							P2,0(P1)	P2 P1 - 0 0 0	R2 - -		
…	...												
…	...												

2) (POINTS 10/40) Given the same code:

```
mylab:  LW      R2, 0(R1)       ; read Xi
        MUL     R2, R2, R2      ; R2=R2*R2
        SW      R2, 0(R1)       ; write Xi
        ADDI    R1, R1, 4       ; update R1
        BNE     R1, R0, mylab   ; continue to loop if false
```

Show how this code can take advantage of software pipelining by statically reordering and renaming the instruction (write the actual code).