HIGH PERFORMANCE COMPUTER ARCHITECTURE 20-01-2010 MATRICULATION NO.____________________

(FORMER "CALCOLATORI ELETTRONICI 2") SURNAME____________________

REVISED 23-10-2023 FIRST NAME____________________

1) (POINTS 13/40) Consider the following snippet of code running on a **dual-dispatch** processor using Tomasulo's algorithm to perform the dynamic scheduling of instructions. The program performs the operation Y=aX/Y on a vector of 100 elements. Initially, R1 = 0 and F0 contains the value of the constant 'a'.

```
lab:    L.D     F2, 0(R1)       ; read Xi
        MUL.D   F4, F2, F0      ; multiply a*Xi
        L.D     F6, 400(R1)     ; load Yi
        DIV.D   F6, F4, F6      ; a*Xi/Yi
        S.D     F6, 400(R1)     ; store Yi
        ADDI    R1, R1, 8       ; update R1
        SGTI    R3, R1, 800     ; R1 >=? 800, result in R3
        BEQ     R3, R0, lab     ; continue to loop if false
```

Working hypothesis:
- speculative execution is permitted and branches are predicted taken; high-performance fetch breaks after a branch
- the issue stage (I) calculates the address of the actual reads and writes;
- reads require 1 clock cycle; writes take 0 cycles (they are written in a write-buffer + split-cache)
- when accessing memory (M), writes have precedence over reads and must be executed in-order
- there's only one CDB
- dispatch stage (P) and complete stage (W) require 1 clock cycle
- there are separated integer units for the calculation of the actual address, for arithmetic and logical operations, for the evaluation of the branch condition
- 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 queue buffer, i.e., in the execution stage)
- the rest of the processor and has the following characteristics

Type of Functional Unit	No. of Functional Units	Cycles for stage I	No. of reservation stations
Integer (effective addr.)	1	1	2
Integer (op. A-L)	1	1	2
Integer (branch calc.)	1	1	2
FP Adder	1	4	3
FP Multiplier	1	8	3
FP Divider	1	15	2

Complete the following chart until the end of the third iteration of the code fragment above.

Iter.	Instruction	P: Dispatch (start-stop)	I+X: Issue+Exec (start-stop)	M: MEM ACCESS (clock)	W: CDB-write (Complete) (clock)	C: Commit (clock)	Comments
1	`L.D F2,0(R1)`	1-4	2-2	3	4	5	
1	`MUL.D F4,F2,F0`	1-13	5-12		13	14	
1	`L.D F6,400(R1)`						

2) (POINTS 17/40) The test-and-set method is the simplest synchronization mechanism and it is available in the large majority of commercial shared-memory machines. Such mechanism is based on the atomic exchange operation **EXCH** that consists in loading the old value at a given address and store into the same address a new value. The "lock" mechanism is in turn implemented upon such atomic operation by spinning on a specific memory address until the lock is open (the returned value is a zero, meaning "unlocked", instead of a one meaning "locked"). The following code represent a possible implementation:

LOCK CODE:

```
tas:    ADDI R2, R0, 1
lockit: EXCH R2, 0(R1)
        BNE  R2, R0, lockit
```

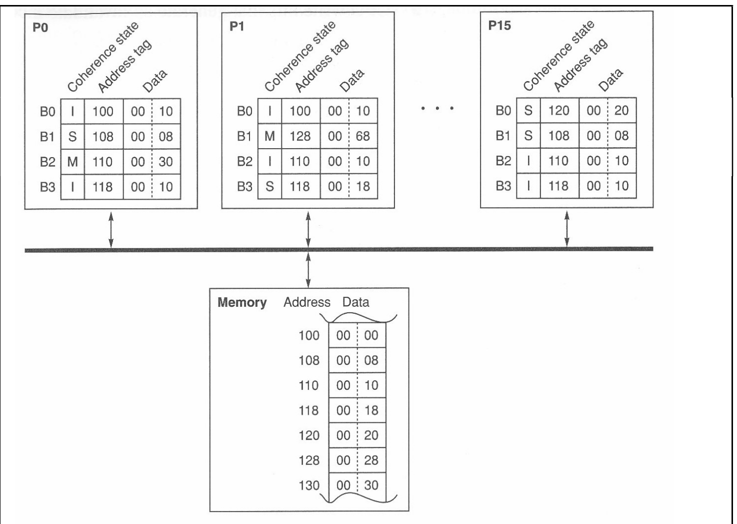
UNLOCK CODE:

```
unlock: SW    R0,0(R1)
```

Let's consider a situation in which three processors (P0, P1, P15) that try to lock the address 0x100 in a machine having 16 processors. Assume an MSI coherence protocol and the cache contents represented in figure. The bus-transaction costs are:

- Creadblk =100
- Ccache-to-cache=70
- Cinvalidate =15
- Cwrite-back =10.

For the sake of simplicity, assume also that the critical sections last 1000 cycles.



P0

	Coherence state	Address tag	Data	
B0	I	100	00	10
B1	S	108	00	08
B2	M	110	00	30
B3	I	118	00	10

P1

	Coherence state	Address tag	Data	
B0	I	100	00	10
B1	M	128	00	68
B2	I	110	00	10
B3	S	118	00	18

P15

	Coherence state	Address tag	Data	
B0	S	120	00	20
B1	S	108	00	08
B2	I	110	00	10
B3	I	118	00	10

Memory

Address	Data	
100	00	00
108	00	08
110	00	10
118	00	18
120	00	20
128	00	28
130	00	30

Assuming that the processors acquire the lock in the order P0→ P1→P15 and given the initial situation of caches and memory represented in the above figure, , calculate: a) how many bus transactions are there; b) how many memory stall cycles for each of the processors are necessary before acquiring the lock.

3) (POINTS 10/40) Calculate the PARALLELISM, by using WORK e SPAN, for the following Cilk implementation of the recursive Fibonacci code in case of n=5.

```
int fib(int n)
{
    if (n < 2) return;
    int x, y;
    x = cilk_spawn fib(n-1);
    y = fib(n-2);
    cilk_sync;
```