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
```

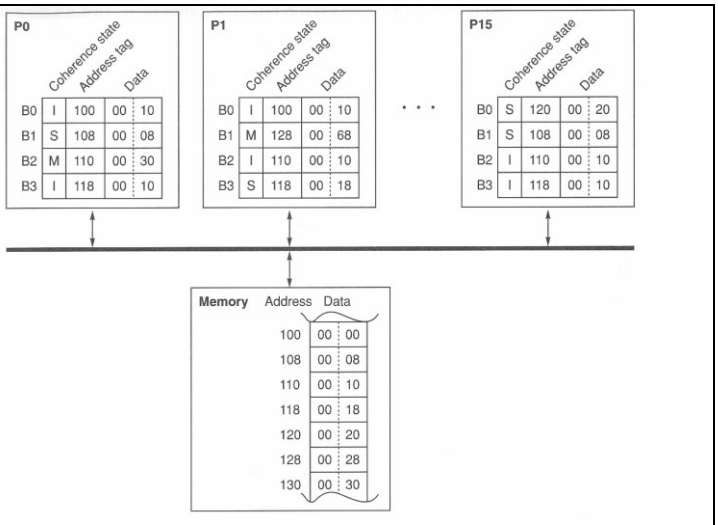
UNLOCK CODE:

```
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.



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;
```

HIGH PERFORMANCE COMPUTER ARCHITECTURE 20-01-2010
SOLUTION REVISED 23-10-2023

```
                                                    return x+y;
                                                }
```

EXERCIZE 1:

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	I waits F2 from 1/L.D
1	L.D	F6,400(R1)	2-5	3-3	4	5	15	
1	DIV.D	F6,F4,F6	2-29	14-28		29	30	I waits F4 from 1/MUL.D
1	S.D	F6,400(R1)	5-6	6-6	30		31	P waits LS-RS,M waits F6 from 1/DIV.D
1	ADDI	R1,R1,8	5-7	6-6		7	32	
1	SGTI	R3,R1,800	6-9	8-8		9	33	I waits R1 from 1/ADDI
1	BEQ	R3,R0,etic	6-10	10-10			34	I waits R3 from 1/SGTI
2	L.D	F2,0(R1)	7-10	8-8	9	10	35	
2	MUL.D	F4,F2,F0	7-21	13-20		21	36	I waits F2 from 2/L.D & MUL-FU avail.
2	L.D	F6,400(R1)	8-11	9-9	10	11	37	
2	DIV.D	F6,F4,F6	8-44	29-43		44	45	I waits F4 from 1/MUL.D & DIV-FU avail.
2	S.D	F6,400(R1)	11-12	12-12	45		46	P waits LS-RS,M waits F6 from 2/DIV.D
2	ADDI	R1,R1,8	11-14	12-12		14	47	CDB waits bus avail.
2	SGTI	R3,R1,800	12-17	15-15		16	48	I waits R1 from 2/ADDI
2	BEQ	R3,R0,etic	12-17	17-17			49	I waits R3 from 2/SGTI
3	L.D	F2,0(R1)	13-17	15-15	16	17	50	CDB waits bus avail.
3	MUL.D	F4,F2,F0	13-30	21-28		30	51	I waits F2 from 2/L.D & MUL-FU avail. CDB waits bus avail.
3	L.D	F6,400(R1)	14-18	16-16	17	18	52	LS-FU avail.
3	DIV.D	F6,F4,F6	30-59	44-58		59	60	P waits DIV-RS available, I waits DIV-FU avail.
3	S.D	F6,400(R1)	30-30	31-31	60		61	M waits F6 from 3/DIV.D
3	ADDI	R1,R1,8	31-33	32-32		33	62	
3	SGTI	R3,R1,800	31-35	34-34		35	63	I waits R1 from 3/ADDI
3	BEQ	R3,R0,etic	32-36	36-36			64	I waits R3 from 3/SGTI