HIGH PERFORMANCE COMPUTER ARCHITECTURE *final exam* 19-12-2018 **MATR.NO.**____________________

(REVISION 03-11-2025) **SURNAME**____________________

FIRST NAME____________________

1) (POINTS 25/40) Consider a four-processor bus-based multiprocessor using the DRAGON protocol. Each processor executes a TAS instruction to lock and gain access to an empty critical section. The initial condition is such that processor 1 has the lock and processor 2, 3, and 4 are spinning on their caches waiting for the lock to be released. Every processor gets the lock once and exits the program. These are the implementations of the lock and unlock:

```
Lock:   lw  R1, mylock     # R1 = &mylock
        bne R1, R0, Lock   # if (R1 != 0) jump to Lock
        TAS R1, mylock     # atomically_do {R1 = &mylock; mylock = 1;}
        bne R1, R0, Lock   # if (R1 != 0) jump to Lock
        ret

Unlock: sw 0, mylock       # write 0 into &mylock
        ret
```

Note1: the semantic of the TAS (Test And Set) instruction is the following: atomically reads the specified memory location (mylock) and writes a one into that memory location (mylock). Note2: this implementation of the Lock tries to minimize the probability to have the bus locked by the TAS (this implementation is also known as Test-and-Test-and-Set). Note3: the lock is closed when mylock==1 and it is open when mylock==0.

By using the following tables, show the operations and bus transactions (or comments): A) in the best case (least number of transactions) and B) in the worst case (highest number of transactions)

A) Best case:

Bus Trans. Number	Processor Operation	P1	P2	P3	P4	Bus Transactions/Comments
---	Init.state	SC	SC	SC	SC	Initially, P1 holds the lock
1	sw1	SM	SC	SC	SC	**BusUpd** – P1 releases the lock

B) Worst case:

Bus Trans. Number	Processor Operation	P1	P2	P3	P4	Bus Transactions/Comments
---	Init.state	SC	SC	SC	SC	Initially, P1 holds the lock
1	sw1	SM	SC	SC	SC	**BusUpd** – P1 releases the lock

2) (POINTS 15/40) Write a OPENMP function that reads a color array (int color[1024]) and writes an array "int histogram[256]" that contains the frequency of each of 256 possible colors (the 256 values are the value that each element of color[] can assume). A serial or serialized version **has to be avoided**. The program should be written in a way that it exploits Thread Level Parallelism as offered by. Template:

```
void histo_scalar(uint *histogram, uchar *color, uint size) {
    for(uint i=0; i<size; i++ ) histogram[ color[i] ]  += 1;
}
```

Hints: Use omp_get_max_threads() to get the number of threads, omp_get_thread_num() to get the current thread id, "#pragma omp parallel", "#pragma omp for" and "#pragma omp critical" as appropriate, try to perform operations in a hierarchical way.