Dept. of Computer Science and Engineering
IIT Delhi
COL216: Assignment 5
II Semester 2020-2021

Release date: 16 April 2021
Submission deadline: 1 May 2021 (Saturday), 11:59 PM

General Instructions

1. The assignment will be done individually or in groups of 2. Only one member of each group should submit the assignment on Moodle.
2. Each group member should understand the problem and contribute equally to the solution. Demos (online/phone) would be held for all the lab assignments.
3. **You will be awarded marks according to your design, implementation, and testing strategy.** Extensive testing is expected as part of the assignment.
4. Adopting any unfair means will lead to -MAX marks (**MAX=25** for this assignment).
5. **Late Penalty: same as in Assignment 2.**

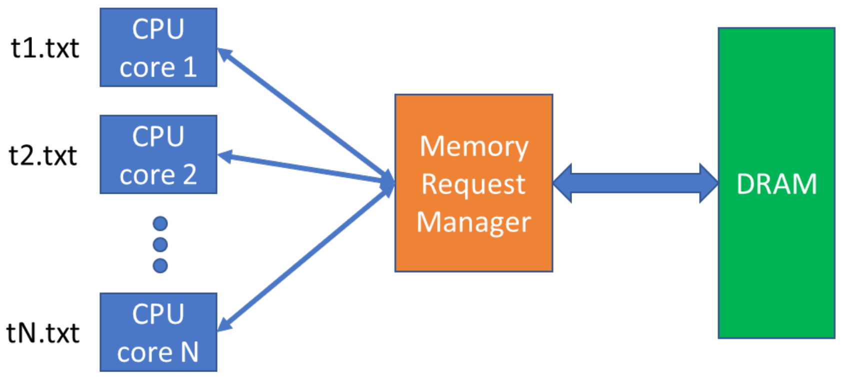
Submission instructions

- Prepare a small write-up (1-2 pages) on the approach taken to solve the problem along with test cases you have considered. The write-up can consist of handwritten notes.
- Explain the approach along with its strengths and weaknesses.
- Explain the testing strategy. **The testing strategy will be evaluated.**
- Zip the document along with the C++ file and test cases and submit at the Moodle submission link.

Problem Statement: DRAM Request Manager for Multicore Processors

In this assignment you will extend your earlier DRAM request manager to the *multicore* CPU case. Our architecture now consists of N CPU cores, each running a different MIPS program, and sending DRAM requests to a Memory Request Manager which interfaces with the DRAM (see figure). The DRAM has the same properties that you have already implemented in earlier assignments. See example below.

1. Extend your earlier MIPS simulator (with DRAM timings) to the multicore scenario. Your objective is to implement the Memory Request Manager in such a way that the instruction throughput (total number of instructions completed by the whole system in a given period, say from Cycle 0 to Cycle M) is maximised.
2. Estimate the delay (in clock cycles) of your own Memory Request Manager algorithm and incorporate it into your timing model. Justify the estimation. Remember this is an *estimate*. You don't have to design the entire manager hardware.



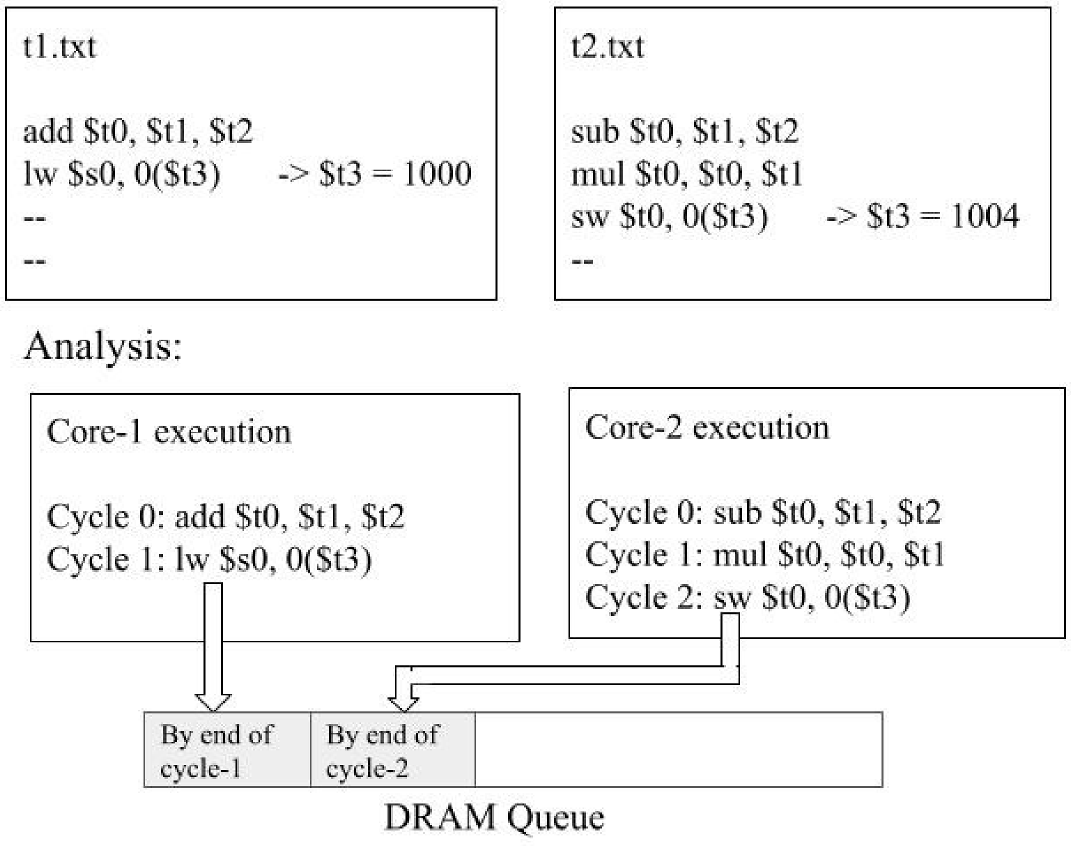
Input:

1. Parameter N (number of CPU cores).
2. Parameter M (simulation time: number of cycles). Execution stops after the simulation time, even though all the instructions may not have completed.
3. MIPS assembly language files t1.txt, t2.txt,…,tN.txt
4. DRAM timing values, as in earlier assignments.

Output:

1. Include an option to print out the activity in the system at every clock cycle.
2. After execution completes, print the relevant statistics individually for each CPU.

Example:



```
t1.txt

add $t0, $t1, $t2
lw $s0, 0($t3)      -> $t3 = 1000
--
--
```

```
t2.txt

sub $t0, $t1, $t2
mul $t0, $t0, $t1
sw $t0, 0($t3)      -> $t3 = 1004
--
```

Analysis:

```
Core-1 execution

Cycle 0: add $t0, $t1, $t2
Cycle 1: lw $s0, 0($t3)
```

```
Core-2 execution

Cycle 0: sub $t0, $t1, $t2
Cycle 1: mul $t0, $t0, $t1
Cycle 2: sw $t0, 0($t3)
```

Assume the following:

1. Programs running in the different CPU cores are independent of each other.
2. Instructions themselves are not accessed from the DRAM. Only lw/sw instructions result in DRAM accesses.
3. Use the same architectural and ISA assumptions as in Assignment 3.

Test cases [IMPORTANT]: Carefully design and document the scenarios you will use to evaluate your implementation. This will be evaluated.

Marks Distribution:

1. Multicore functionality [7 Marks]
2. Throughput efficiency [5 Marks]
3. Delay estimation and its incorporation [5 Marks]
4. Comprehensiveness of testing [5 Marks]
5. Documentation [3 Marks]