**COL216: ASSIGNMENT – 5**

**Student 1:** Bommakanti Venkata Naga Sai Aditya **Entry No. 1:** 2019CS50471

**Student 1:** Alladi Ajay **Entry No. 2** 2019CS10323

**WRITE-UP**

**Input:**

Input to our program is a text file with MIPS instructions. Input file is given through command line. Execute using the following instruction:

./2019CS50471\_2019CS10323\_Ass5 <no of cores> <simulation time> <path to inputfiles> <ROW\_ACCESS\_DELAY> <COL\_ACCESS\_DELAY>

**Approach:**

We used reordering in Assignment – 4 and same code is used in Assignment – 5. So, reordering is a part of Memory Request Manager in Assignment 5.

**Part – 1:**

1) Cores execute instructions sequentially. If the instruction is not lw/ sw, it is excecuted by the core in 1 clock cycle. If not, the lw/sw instruction is added to the queue. (MRM maintains a single queue for all the cores)

Memory Division between n-cores:

\* 220 memory addresses are divided for n – cores. Each core has seperate chunk of memory. For core i, memory address range between round\_to\_nearest\_4\_multiple( (220/n)\*(i – 1)) to round\_to\_nearest\_4\_multiple( (220/n)\*(i))

\* Wait buffer size: 128

2) **Memory Request Manager Implementation:**

a) After receiving DRAM requests, if wait buffer length is 1, then MRM sends that instruction to the DRAM.

b) If wait buffer length is > 1, then MRM will try to reorder the wait buffer based on the memory row of current executed instruction (MRM is remembering past history).

c) MRM will check if the current instruction is redundant or not. If redundant, it goes to step(b). \*\*\* We implemented this only in part-1.

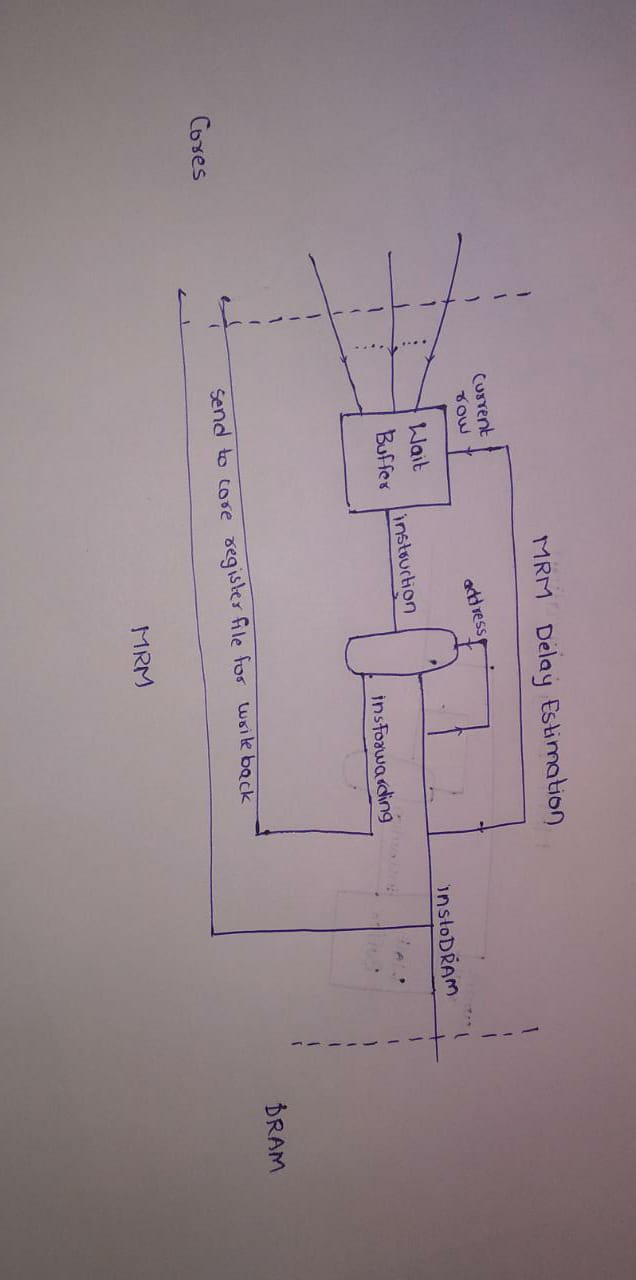
d) MRM will check if forwarding is possible or not to the current instruction. If possible, the instruction is executed in 1 clock cycle.

e) Forwarding Criteria:

\* Only sw can forward value to lw, if the memory address of both sw and lw are same.

f) If forwarding is not possible, then MRM will send instruction to DRAM.

**Part – 2:**



From the figure, the critical path has two components. So, MRM can give a delay of 2 clock cycles. We have incorporated this delay in assignment file.

MRM and DRAM will run parallelly i.e., while DRAM is executing an instruction, MRM will perform necessary steps and identifies the next instruction.

**Strengths:**

1) Performed reordering in MRM.

2) Used forwarding optimization from sw to lw.

3) DRAM and MRM executing parallelly.

4) Only one register is modified in one clock cycle.

5) We will stop program when cycle reaches simulation time. Even if there is any instruction running in DRAM/ pending in MRM we’ll discard them.

**Weakness:**

Redundant instructions are not removed by MRM, because it is said that removing redundant instructions is the job of compiler.

**Output:**

At every clock cycle period, the program prints the memory address of the instruction executed, modified registers, modified memory addresses for each core. At the end, the program prints total no of instructions executed in each core, data stored in the memory.

**TESTING**

**Syntax Errors:** Syntax errors are tested on this code in Assignment - 4.

**General Case:** All the instructions follow the given syntax as mentioned above.

**Test Cases:**

**Testcase-1:** Forwarding of instructions from sw to lw

**Testcase-2:** Only one register modifying in one clock cycle.

**Testcase-3:** Wait buffer reaching max value. (Adjust the simulation time accordingly)

**Testcase-4:** Reordering of lw, sw

**Testcase-5:** DRAM and MRM running parallelly.

**Testcase-6:** Some lw/sw instructions remaining in wait buffer after simulation time. (Adjust the simulation time accordingly)

**Testcase-7:** Remove redundant instructions. \*\*\*Only for part-1

We can send any no of these files to our program as different cores.