COL216: ASSIGNMENT – 4

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_Ass4 <path to inputfile> <ROW_ACCESS_DELAY>
<COL_ACCESS_DELAY>
```

Approach:

Note: All the possible errors are handled in the Minor Assignment. Same code is used in this assignment. So, all possible errors are handled in this assignment. We modified our input format to standard MIPS format (Adding labels).

1) While executing the instructions sequentially, if we encounter lw/sw instruction for the first time we execute it using non-blocking part. If we encounter again we will add the lw/sw instruction to the queue. And whenever the instruction executing in DRAM is completed, we reorder the queue and execute the first instruction in the queue using non-blocking part.

Strengths:

How to reorder the queue?

2) Let's say the row in row buffer is 'r'. We'll iterate over the queue and find instructions whose memory address (offset + reg. value[r2]). If there is no such instruction, we'll execute the first instruction of the queue i.e., the instruction next to the current instruction. If there is such instruction, we'll reorder the queue based on the dependency of that instruction.

3) Suppose we find such an instruction. Then we compare that instruction with all the previous instructions. The instruction is moved to a place where it is dependent on all the previous instructions.

Let's say the queue is = [I1, I2, I3, I4, I5]. Suppose I4, I5 has same row as the row in row buffer. Assume that I4 is independent of I1, I2, I3 and I5 is dependent on I1 and is independent of I2, I3, I4. So, queue after reordering will be [I4, I1, I5, I2, I3]. Now I4 will be executed.

How to determine whether two instructions are dependent or Independent?

4)

Case – 1: Instructions are lw, lw

lw reg1, $offset_1$(reg2)

lw reg3, $offset_2$(reg4)

These are independent if reg1 != reg3 and reg1 != reg4 and reg3 != reg2.

Case – 2: Instructions are sw, sw

sw reg1, $offset_1$(reg2)

sw reg3, $offset_2$(reg4)

These are independent if $offset_1$ + reg_value[reg2] != $offset_2$ + reg_value[reg4]

Case – 3: Instructions are lw, sw

lw reg1, $offset_1$(reg2)

sw reg3, $offset_2$(reg4)

These are independent if reg1 != reg3 and reg1 != reg4 and $offset_1$ + reg_value[reg2] != $offset_2$ + reg_value[reg4]

Highlights:

1) We implemented reordering with non blocking part and queue.

2) All the instructions which are in the queue are considered while reordering and the instructions whose row is same as row buffer are reordered. These instructions are reordered efficiently i.e, there are reordered to a place where it depends on all the previous instructions.

3) The dependency check is optimal.

Weakness:

Only the lw/ sw instructions which are encountered while one of the lw/ sw is executing are added to queue. So, if there is a lw/sw instruction which has same row as row buffer but it is not in queue, it is not considered for reordering. But implementing this has it's own disadvantages like, there may not be an instruction which has same row as row buffer in the near future.

Output:

At every clock cycle period, the program prints the memory address of the instruction executed, modified registers, modified memory addresses and DRAM activity in the clock cycle period. At the end, the program prints total no of clock cycles, data stored in the memory and total row buffer updates.

TESTING

Syntax Errors: Syntax errors are tested on this code in Minor Assignment.

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

Test Cases:

Testing Strategy: I computed the output manually and verified it with the output

Testcases are provided in Testcases folder.