COL-216 Assignment-4

Mohit Thakur 2019CS10373
Sidharth Agarwal 2019CS50661

April 10, 2021

Aim

Develop a MIPS simulator with DRAM timing model with memory request re-ordering.

Approach

1. We extended on the non-blocking feature from the submission.
2. We have used an array of queues to save the lw/sw instructions in respective rows and also maintained a doubly linked list to keep a track of instructions when row buffer changes and they are coordinated accordingly so that no instruction is executed twice.
3. DRAM instructions and rest of the instructions run in parallel, as DRAM instructions can be reordered we run all the instructions of same row and delete it from our doubly linked list. When their is no instruction of same row, we then activate instructions for other row.
4. If their is a normal instruction that can't be executed, PC stops their until it can be successfully executed.

Design Decisions

1. Command line parameters, if none is mentioned it then the default values are 10 and 2.
2. We take input from file input.txt and output the final result in file out.txt.
3. Instead of declaring memory as $1 << 10 \times 1 << 10$ bytes we declared it as $1 << 10 \times 1 << 8$ matrix of integers for ease of storing instructions.
4. Program starts from main instruction. Program terminates if it traverses all the instructions or it it is made to jump to instruction that is not a part of instruction set or encounters the exit label in between.
5. Memory is update once after all instruction are executed only when there has been a change in the present row buffer. So, that memory holds all the saved values at the end.
6. All instructions formats are very strict, i.e. code throw error when anything unexpected occurs in the file.
7. "main" in starting and "exit" at the end of code are necessary for the program to run.
8. addi instruction can only add numbers whose modulus is strictly less then 1<<15 ie. numbers that can be stored in 16 bits.

Testing

We tested the compiled code on manually written code(slightly modified from the one provided as test case)

Input :

```
main:
addi $s0, $zero, 1000
addi $s1, $zero, 2500
addi $s2, $zero, 5000
addi $s3, $zero, 6500
addi $t0, $zero, 1
addi $t1, $zero, 2
addi $t2, $zero, 3
addi $t3, $zero, 4
sw $t0, 0($s0)
sw $t1, 0($s1)
sw $t2, 0($s2)
sw $t3, 0($s3)
lw $t5, 0($s0)
lw $t6, 0($s1)
addi $t9,$zero,400
addi $t3, $t2, 40
lw $t7, 0($s2)
lw $t8, 0($s3)
exit:
```

Output :

```
line number 0: cycle 1: $s0 = 1000
line number 1: cycle 2: $s1 = 2500
line number 2: cycle 3: $s2 = 5000
line number 3: cycle 4: $s3 = 6500
line number 4: cycle 5: $t0 = 1
line number 5: cycle 6: $t1 = 2
line number 6: cycle 7: $t2 = 3
line number 7: cycle 8: $t3 = 4
line number 8 : cycle 9: DRAM request issued
line number 8 : cycle 10 - 21: memory address 1000-1003 = 1
line number 14: cycle 15: $t9 = 400
line number 12 : cycle 22 - 23: $t5 = 1
line number 9 : cycle 24: DRAM request issued
line number 9 : cycle 25 - 46: memory address 2500-2503 = 2
line number 13 : cycle 47 - 48: $t6 = 2
line number 10 : cycle 49: DRAM request issued
line number 10 : cycle 50 - 71: memory address 5000-5003 = 3
line number 11 : cycle 72: DRAM request issued
line number 11 : cycle 73 - 94: memory address 6500-6503 = 4
line number 15: cycle 95: $t3 = 43
line number 16 : cycle 96: DRAM request issued
line number 16 : cycle 97 - 118: $t7 = 0
line number 17 : cycle 119: DRAM request issued
line number 17 : cycle 119 - 131: $t8 = 0


Total number of cycles : 131
Total number of row buffer updates : 8
```

Instructions till line number 7 are only for setting up environment. Instructions at line (8 and 12), (9,13), (10,16) and (11,17) access same, and for optimization are to be executed simultaneously whenever possible. But since line 15 has dependency on line 11, line 11 needs to be executed before it. And thus output we got is expected.

Pros and Cons

1. **(Pro)** As implementation is an extension of Non-blocking part of minor, It can be treated as parallel for DRAM and processor instructions.
2. **(Pro)** row_buffer is copied to memory only when their is atleast one sw instruction executed.
3. **(Pro)** Our simulator is O(n) with respect to input size.
4. **(Con)** Simulator loads other row when their is no instruction of current row seen till now, but in cases their might be same row instruction available after and would simulator have saved 2* row_delay of time.
5. **(Con)** In the example shared in testing part, simulator could have further optimized code be executing only line 11 before 13 and not 10. This way, it could have saved some CPU cycles. But our current simulator, maintains linearity in case of different row accesses.