COL216

ASSIGNMENT 4

Anjali Sharma Hardeep Kaur

2019CS50422 2019CS10354

Approach:

AIM: We designed a strategy for efficient ordering of DRAM requests at runtime in the interpreter we had implemented previously with non blocking access.

The execution is done by maintaining internal data structures, as Register file, Instruction memory, data memory and a queue.

Since we need to reorder the instructions to minimise the no of cycles needed,queue structure helps in storing the independent load or store instructions.Thus, we are able to reorder independent instructions in an optimised manner,which in turn reduces the number of total clock cycles needed.

Design:

1. Memory is byte addressable and has 2^{20} bytes, out of which first 1000 bytes store the addresses of instructions, each one being one word wide. It is assumed that all the data transfer instructions involve memory addresses which are multiple of 4 and greater than or equal to 1000.

2. For a load word instruction,the register to which the value is loaded should not be same as the lw/sw register whose dram request is currently being executed.

The values of row and column access delays are given by user as input.
./main <inputfilename> <row access delay> <column access delay>

Strengths:

- When a DRAM request is in process, the independent data tranfer instructions lying ahead are stored in a queue. While these are waiting for completion of previous request, we can execute the independent R format and branch instructions lying ahead.They won't need to wait for the DRAM requests to be complete.
- When the DRAM request is complete and there are more than one requests lying in queue, we can choose the instruction which involves the row already in buffer. Thus, we need not perform redundant writebacks and activations.
- Reordering is done to minimise the number of clock cycles needed.The updates related to same rows are executed first,thus substantially decreasing the number of row access delays.

Weakness:

- If the load/store instructions to be executed are dependant,the queue would be empty.In such a case there is no reordering done.All the instructions execute sequentially(as in Non blocking memory)

Testing Strategy:

- If the command line is wrong, an error is printed saying the input is to be checked
- if the input file is not available, unable to open file is printed.
- Since we can initialise registers and data memory .it would give a warning to the user,that these files were not initialised.
- The following input is executed in 51 cycles and 5 buffer row updates.
- addi $s0, $zero, 1000
- addi $s1, $zero, 2000
- addi $t0, $zero, 1
- addi $t1, $zero, 2
- addi $t3, $zero, 4
- sw $t0, 0($s0)
- sw $t3, 4($s0)
- sw $t1,4($s1)
- lw $t2, 0($s0)
- lw $t5, 0($s1)
- lw $t7, 4($s0)
- lw $t4, 4($s1)

- OUTPUT:
  The updated values in data memory are written to “dataMem_final.txt”.
  The states of registers after each instruction are written to “RF_final.txt”.
  Statements related to processing are printed on console. Statistics involving the number of clock cycles and the number of times different instructions are executed, number of row buffer updates , are also printed to console.
  Also,at every clock cycle, number of clock cycle along with all executions happening in the cycle are printed on the console.
  .