MIPS simulator with DRAM timing model

Aditya Goel — 2019CS10671

March 21, 2021

Program Design Specifications: Input & Output Format

- The C++ Program can be compiled using the following command line instruction
  g++ -o main main.cpp
- The executable can then be run by the following command line instruction
  ./main PART_NO ROW_ACCESS_DELAY COL_ACCESS_DELAY INPUT_FILE_NAME
- PART_NO as 1 implements the DRAM timing model with blocking memory access while PART_NO as 2 implements the DRAM timing model with non-blocking memory access.
  - An "Error: PART_NO not specified correctly" is printed if the PART_NO is not 1 or 2.
- ROW_ACCESS_DELAY and COL_ACCESS_DELAY are the delays in number of cycles for accessing row and column from the DRAM respectively.
  - An "Error: ROW_ACCESS_DELAY not specified correctly" or "Error: COL_ACCESS_DELAY not specified correctly" is printed if the ROW_ACCESS_DELAY or COL_ACCESS_DELAY are not integers respectively.
- INPUT_FILE_NAME is the name of the MIPS assembly language program (as text file, NOT machine instructions).
  - An "Error: File does not exist or could not be opened" is printed if the file doesn't exists or is not a text file.
- At every clock cycle the program prints the cycle number, the instruction and any modified registers, any modified memory locations or any activity on DRAM.
- After execution completes, the program prints relevant statistics such as the Total execution time in clock cycles and the Number of row buffer updates.

Strategy & Approach Used

- Each instruction in the input file is stored in a vector line by line. Any instance of a comment is ignored while any instance of a label is stored in a HashMap that maps that label as a key to the corresponding line number as its value.
- As we iterate through the instructions one by one in a sequential manner, each itertion of an instruction involves processes of parsing the instruction and performing relevant operations on concerned registers or memory locations.
- Appropriate errors are raised during parsing each line, if any instruction other than add, addi, sub, mul, slt, j, beq, bne, lw or sw is encountered or if these instructions are not found in the format accepted as MIPS assembly language.
- The DRAM is implemented as a 2-Dimensional array of size 1024 x 1024 bytes. A single Read/Write in a lw/sw instruction corresponds to accessing a 32-bit (4 byte) value from the DRAM. Within a row, a given piece of data is located at a column offset, so a memory address could be thought of as consisting of a ROW ADDRESS and COLUMN ADDRESS.

- ROW_BUFFER is implemented as a 1-Dimensional array of size 1024 bytes which helps in both accessing the value stored at a particular memory address or store and update any memory address by copying the necessary row from the DRAM and writing it back to DRAM whenever required.
- In case on Blocking Memory Access, each instruction is executed entirely before the execution of the next instruction begins.
- In case on Non-Blocking Memory Access, next few instructions can be executed even if a lw/sw instruction has not completed, when is it safe to do so. More about it is mentioned in Strengths & Weaknesses Section.

Strengths & Weaknesses

- Use of Arrays and HashMaps for DRAM, ROW_BUFFER and Labels helps in efficient access to the memory and labels during the execution of the program.
- Our implementation of Non-Blocking Memory Access, allows for execution of instructions even if a lw/sw instruction has not completed, only until we do not encounter any instruction that involves any register or memory location that was mentioned in the corresponding lw/sw instruction or until we find another lw/sw instruction or a bne/beq/j instruction, or until we execute instructions one less than the delay in number of cycles that the corresponding lw/sw instruction had to face. All such instructions which cannot be executed when the execution of corresponding lw/sw instruction has not completed are said to be unsafe for execution.
- The Non-Blocking Memory Access considerably reduces the total execution time in clock cycles. The effect increases in case we have a large value of ROW_ACCESS_DELAY and COL_ACCESS_DELAY.
- The Non-Blocking Memory Access has been implemented with use of O(1) space.
- This reduction in total execution time in clock cycles depends on encounter of instructions which are marked unsafe to be executed while a lw/sw instruction is not completely executed. Programs which involve such instructions just after the lw/sw statements will lead to no reduction in the total execution time in clock cycles.

Test Cases & Corner Cases Handled

1. In case of a comment both, as a separate line or along with those Instructions are completely ignored by the program.
   - A test on input
     ```
     #My New Increment Program
     addi $t0, $t0, 1 #Incrementing $t0
     ```
     resulted in a successful execution of the program and incremented the value of $t0 by 1 , ignoring any comments that came in between.
   - A test on input
     ```
     #My New Increment Program
     #addi $t0, $t0, 1 Incrementing $t0
     ```
     resulted in a successful execution of the program with no Instructions to be evaluated.
2. Entering any invalid instruction, other than those specified in the program design specifications prints "Error: Invalid operation" error.
   - A test on input
     ```
     move $t0, $t1
     ```
     resulted in an "Error: Invalid operation" error.
3. If results of instructions such as add, sub or addi results in an arithmetic overflow, corresponding errors are raised.
   - A test on input
     ```
     addi $t0, $t0, 1
     addi $t0, $t0, 2147483647
     ```

resulted in an "Addition Overflow" error.

4. If at any stage of program, the number of instructions executed is found to exceed 131,072, the execution stops, and print "Number of Instructions exceeded the maximum allowed values" error.

5. If the input to commands such as j, bne, beq is a label which is not present in the Source Code, a "Label Not Found" error is raised.

   – A test on input
   ```
   label1:
   addi $t0, $t0, 1
   bne $t0, $zero, label2
   ```
   resulted in an "Label Not Found" error.

6. Entering a memory address beyond the allowed bounds of (0-1048576) or a memory address that is not a multiple of 4, prints "Invalid Memory address" error.

   – A test on input
   ```
   addi $s0, $zero, 1048580
   sw $t0, 0($s0)
   ```
   resulted in an "Invalid Memory address" error.

7. Entering an input with register names other than the 32 registers specified, prints an "Error: Invalid register" error. Also entering non-32 bit integer values in instructions such as addi, prints an "Error: Given value is not a valid number" or "Error: Out of Range Integer" error respectively.

   – A test on input
   ```
   addi $m0, $m0, 1
   ```
   resulted in an "Error: Invalid register" error.

   – A test on input
   ```
   addi $t0, $t0, abc
   ```
   resulted in an "Error: Given value is not a valid number" error.

   – A test on input
   ```
   addi $t0, $t0, 2147483648
   ```
   resulted in an "Error: Out of Range Integer" error.

8. In case of Non-Blocking Memory Access, execution of instructions that were taking place even if a lw/sw instruction was not completed, halted on encounter of any instruction that involves any register or memory location that was mentioned in the corresponding lw/sw instruction. The program proceeds only after complete execution of the lw/sw instruction.

   – A test on input
   ```
   addi $s0, $zero, 1000
   addi $t0, $zero, 10
   sw $t0, 0($s0)
   lw $t0, 0($s0)
   addi $t1, $t1, 1
   add $t1, $t1, $t0
   add $t0, $t0, $t1
   ```

   The Program with ROW_ACCESS_DELAY = 4 and COL_ACCESS_DELAY =2 halts execution at the 7th line, and proceeds execution of line 7 only after line 4 is executed completely.

9. In case of Non-Blocking Memory Access, execution of instructions that were taking place even if a lw/sw instruction was not completed, halted on encounter of another lw/sw instruction or a bne/beq/j instruction. The program proceeds only after complete execution of the lw/sw instruction.

   – A test on input
   ```
   addi $t0, $zero, 10
   addi $s0, $zero, 1000
   sw $t0, 0($s0)
   ```

lw $t0, 0($s0)
addi $t1, $t1, 1

The Program with ROW_ACCESS_DELAY = 4 and COL_ACCESS_DELAY =2 halts execution at the 4th line, and proceeds execution of line 4 only after line 3 is executed completely.

10. In case of Non-Blocking Memory Access, execution of instructions that were taking place even if a lw/sw instruction was not completed, halted after execution of instructions one less than the delay in number of cycles that the corresponding lw/sw instruction faced. The program proceeds only after complete execution of the lw/sw instruction.

    – A test on input

```
addi $t0, $zero, 10
addi $s0, $zero, 1000
addi $s0, $zero, 1024
sw $t0, 0($s0)
sw $t2, 0($s1)
lw $t0, 0($s0)
addi $t1, $t1, 1
addi $t1, $t1, 1
addi $t1, $t1, 1
addi $t1, $t1, 1
addi $t1, $t1, 1
addi $t1, $t1, 1
addi $t1, $t1, 1
addi $t1, $t1, 1
addi $t1, $t1, 1
addi $t1, $t1, 1
addi $t1, $t1, 1
addi $t1, $t1, 1
```

The Program with ROW_ACCESS_DELAY = 4 and COL_ACCESS_DELAY =2 halts execution at the 16th line, and proceeds execution of line 16 only after line 6 is executed completely.