Computer Architecture COL216

MIPS Simulator with DRAM timing model

Arpit Chauhan, 2019CS10332
21/03/2021

Description

In this assignment, we have written a C++ program that reads a MIPS assembly language program as input and executes it as per MIPS ISA by maintaining internal data structures representing processor components such as Register File and Memory. We have developed a model for the Main Memory which can be thought of as a 2-dimensional array of ROWS and COLUMNS with DRAM timing values ROW_ACCESS_DELAY and COL_ACCESS_DELAY in cycles. The memory access has been made non-blocking so that subsequent instructions don't always wait for the previous instructions to complete .

Input / Output

- The input to the program will be a text file containing MIPS instructions , along with 2 arguments , ROW_ACCESS_DELAY and COL_ACCESS_DELAY.
- Each instruction will be in a new line.
- The allowed instructions are: **add, sub, mul, beq, bne, slt, j, lw, sw, and addi.**
- If any instruction does not follow MIPS convention, then a corresponding error is thrown along with the line number.
- If all the instructions are syntactically correct, then the program starts executing the operations starting from the first instruction.
- The labels must also follow MIPS convention i.e. must start with a letter, maybe followed by letters, numbers, or “_”, must be terminated by ”:” and should be unique.
- At every clock cycle , we print the executed instruction , Integer registers , Data Memory , and activity on the DRAM such as row buffer updates .
- After the execution completes , relevant statistics such as total execution time in cycles , total number of row buffer updates , Data memory used , Instruction memory used, etc are consoled out.

Approach / Algorithm

- The Register File contains 32 integers registers , namely $zero, $r0 - $r9 ,$t0 - $t9, $s0 - $s9 and $sp . All of them are initialized to 0. The $zero is a reference register and is non-mutable.
- We have allotted 2^{19} Bytes of memory each for **Instructions** and **Data,** summing up to a total of 2^{20} Bytes. We have taken the memory to be **word addressable**.
- The Data Memory is stored in a 2D array(vector of vectors) with 512 ROWS and 1024 Bytes of memory in each row.

- The C++ program reads the input text file line by line and checks the syntactic correctness of the instructions using RegEx and various other validators. If the instruction is valid, then the instruction type and its parameters are stored in a map(params) indexed by the line number. After all the instructions are loaded into the program, the execution phase starts.
- Each instruction occupies 4 bytes and is executed in one clock cycle except for lw and sw .
- For lw(READ) and sw(WRITE) operations , activating the corresponding row takes zero, one or two times ROW_ACCESS_DELAY cycles depending upon the current row in the row buffer . Then copying/updating the data in the row buffer takes COL_ACCESS_DELAY cycles.
- But the program does not wait for the lw/sw instruction to finish , it continues executing the following instructions unless there is some dependency on the ongoing lw/sw instruction.
- To incorporate this non-blocking feature , I have maintained 2 vectors , r and w which contain the registers that will be read/written in the ongoing lw/sw instruction, which are then used to check the safety of any instruction executed while the lw/sw is being processed in the background . If any non-safe instruction is found , it is executed only after the ongoing lw/sw final response has been received .
- We maintain an iterator named **Program Counter**: pc, that always contains the address of the next instruction to be executed. This counter is updated after each instruction execution.
- The program ends when pc exceeds the address of the last instruction. Relevant statistics of the entire MIPS program are then printed.

--- Strengths

- This design of the MIPS simulator functions well with a large number of instructions such as **add, sub, mul, beq, bne, slt, j, lw, sw, and addi.**
- While the DRAM is processing a lw/sw request , the program executes next few **safe** instructions to reduce the execution time .
- The simulator ensures that the instructions are executed in Sequential order and only one instruction request is processed at a time in DRAM.
- The simulator outputs a very detailed log of the execution of the MIPS program which is very useful for the user.

--- Weaknesses

- In this design , I have assumed any lw/sw operation to be unsafe while another lw/sw is in process . This weakness can be overcome by using Queue data structure to store the waiting DRAM requests, but will make the simulator more complex and expensive(trade-off between Cost and Efficiency).
- The design can be made even cheaper and more organised by making the code modular so that it is easier to test and refactor.

ERROR HANDLING

All errors are reported with the appropriate line number and reason for the user to debug.

1. **Syntax Error**

   A syntax error is thrown in the following cases:
   - If an invalid instruction is provided to the code.
   - If the operands of any instruction do not follow the convention specified by MIPS ISA.
   - If the integer value of an I-type immediate value is larger than 2^16 bits.
   - If a label is not in the specified format or is repeated twice.
   - If the total number of instructions exceeds the allotted space for Instruction memory.

   Note: The extra whitespaces or indentation are ignored by the program
2. **Execution Error**

   An Execution error occurs in the following cases:
   - The instructions beq, bne, or j asks for a label that has not been initialized.
   - If the data memory exceeds the allotted space for data. Or asks for an invalid memory address. (Like not a multiple of four or negative)
   - If an instruction tries to modify the value of the zero register.

TESTING

- To test the code, save the MIPS code in a text file in the same directory as the main file and then run the command {{ ./a.out (Name of Test File) (ROW_ACCESS_DELAY) (COL_ACCESS_DELAY) }}. The output of the function will be saved in a file and any error if any will be reported on the console.
- I have done extensive testing as a part of the assignment to check the validity of the MIPS simulator. The test cases have been added to the Zip folder.
- Each test case was first run on Part1(DRAM with blocking) and then the complete program(Non-blocking) to test the efficiency of the Non-blocking DRAM timing model. The final outputs were matched against the outputs of the simple MIPS interpreter built in Assignment 3.
- All the allowed instructions with labels are included in the test cases to check the validity of each of them.
- Details of execution performed in each cycle along with the state of Registers and Data Memory are stored in the output which help to check the validity of the MIPS simulator.
- I have attempted to handle syntax errors and execution errors of almost every type and tested them rigorously.

THE END