COL216 Minor

Aayush Goyal, 2019CS10452

March 2021

1 Aim

To enhance the MIPS interpreter by adding two new features, making it a MIPS simulator. The 2 features that need to be added are:

1. A model for the Main Memory using Dynamic Random Access Memory (DRAM) and integrate it into the basic interpreter.
2. Making the memory access **Non-Blocking** (i.e. subsequent instructions don't always wait for the previous instructions to complete).

2 I/O specifications

- **Input:**
  1. MIPS assembly language program (text file)
  2. DRAM timing values ROW_ACCESS_DELAY and COL_ACCESS_DELAY in cycles (as command line arguments).
- **Output:**
  1. **Clock cycle instructions:** The clock cycle number and any change during the clock cycle like in the value of register, or in the memory, or a request is issued to the DRAM.
  2. **After execution completes:**
     (a) The Memory addresses having non-zero values.
     (b) The registers having non-zero values.
     (c) No. of times different operations were executed.
     (d) Total number of clock cycles.
     (e) Total number of row buffer updates
- **To run the code:** Run the *make all* command. This will create two executables named "task_1" and "task_2". Now run the command *./task_x file_name ROW_ACCESS_DELAY COL_ACCESS_DELAY*, where task_x is task_1 or task_2, "file_name" is the name or relative path of the input file, ROW_ACCESS_DELAY and COL_ACCESS_DELAY are the DRAM timing values in cycles. task_2 must be used for checking Non-blocking access of memory.

Note: We assume that every instruction is executed in a single clock cycle other than Row access and Column access as mentioned in the problem statement.

3 Approach

1. For Timings characteristics of DRAM: We have made a 2D vector for implementing a 2D array and, it can store 2^{20} Bytes of data in it. Now for accessing memory from the DRAM, the corresponding row is first loaded into the Buffer row (if it's not already there) and then we have to reach the particular cell, this again requires some time. We have maintained a variable "clockCycles" which acts as a clock. Now whenever we need to update some time lag, we just add that value in the "clockCycles" variable. So this kind fo acts like a virtual clock for us. Using this clockCycle variable we can analyse the Timing characteristics of DRAM.

2. For Non-Blocking memory acces: From our implementation in the part 1 we can see that, once we issue a DRAM request, it waits for it to complete. But this is not required that it should wait at every DRAM request. There can be some operations which even after this DRAM request are safe to do without completing the DRAM task. And hence we have done them and stopped at the moment we encountered an unsafe operation or we need to issue a new DRAM request.

3.1 What is a safe instruction?

In MIPS Assembly we know that all the instructions are executed "sequentially". Suppose we have,

```
1    sw $t0, 1000
2    addi $s0, $zero, 1
3    addi $s1, $zero, 1
```

the sw issues a DRAM request and thus in the part 1 what we did is we wait for the value of $t0 to get sotred in Row 0, Col 250 of the DRAM array. This takes some time. This process is running in the DRAM and meanwhile our processor is idle. So what we can do to reduce this wastage if time is, we notice that the next instruction doesn't require anything from the what is being processed in the DRAM, so we can run it in the processor. This doesn't break the sequential nature because the sw command was executed before the addi command, just that the processing of that command is still going on in the DRAM. So all such instructions which are independent of what is being processed in the DRAM are considered as safe instructions. Now consider these instructions:

```
1    sw $t0, 1000
2    addi $s0, $zero, 1
3    addi $s1, $zero, 1
4    lw $t1, 1000
5    addi $s1, $s1, 1
6    add $t1, $s1, $s0
```

To execute 4, we issue need to issue a Request to DRAM and according to the current choice we have considered that it is not possible for DRAM to hold another request on hold, because this would require some memory to store this "waiting list". So 4th is an unsafe instruction. We stop the processor there and now wait for DRAM to complete the previous request so that we can issue another another request. Note that meanwhile we can't go to 5 because we have executed 4 as of now, and 4 must be executed before it because execution is sequential. After

1 is completed we issue a request for 4 and move on to 5. 5 is a safe instruction, so this is being processed in the processor, 4 is also running in the processor. Now when we come to 6 we see that we require the value of $t1 and thus this is not a safe instruction. here we will wait for the DRAM to complete the READ operation. After this is finished we will execute the 6^{th} command.

4 Strengths

1. The current Non-Blocking implementation is based on the principle that it will accept another READ or WRITE request only when the previous request gets finished. This is because with just a 2D array to store memory and row-buffer it won't be possible for it to store any information regarding the upcoming request(s). This can be visualized as, the data-bus which needs is transferring data from the processor is also being use to send address and register information to the DRAM. Thus as long as the data-bus is stuck at the DRAM station to pick up result of previous request we can't send another request to the DRAM. Thus we are saving on Storage. This would also lead to a simple hardware implementation. We can't create something out of nothing, if we want to increase the time efficiency of the program, we will have to sacrifice the Storage and Simplicity of hardware design.

2. The Non-blocking can also take care of the cases like continuous multiple READ operations into the same register from different addresses, in all of them only the last one is the one that matters, so this can be taken care of using Non-blocking and some storage. But again this is the fault of the user, they should realize that they have given multiple redundant statements and this will lead to unnecessary increase in runtime. So it is left for the user to make efficient program instead of expecting the compiler or synthesizer to do this.

3. Appropriated Error messages have been printed along with the line number. Telling the line number and the mistake will help the user debug his program quickly.

4. I have implemented other functions like mul, sub, j, bne, beq, li, slt. bne, beq, j hllp us in doing branching and thus we can make implement loops in our MIPS program.

5. The program outputs all the non-zero register values at the end of execution. Similarly it also outputs all the non-zero values stored in the DRAM. Since we have printed all the non-zero values and the other ones will be zero, it is a concise and cleaner way to tell any register or memory block value.

5 Weakness

We have not allowed the DRAM to store any extra space other than 2^{20} Bytes and and a Row-buffer. Thus it must complete a request before accepting another from the processor, because it has got no extra storage for storing the information of different requests. But if we allow it to store information for other requests we can increase the efficiency of non-blocking upto some extent. Like we can introduce the concept of waiting lists or queues to store the information from different request. Now let's say I have some consecutive WRITE operations. As long as we have the privilege to store the the value that needs to be written and to store where it is to be written, all operations between these WRITE operations are **safe** (because to execute them we need register values and nothing is being read from the Memory) and we can execute them without having to wait for one WRITE to complete before the other WRITE. Also implementing this will require complex Hard ware design

and more storage. Thus we could have Spent more Storage and a complex hardware design to increase the time efficiency.

6 Code Design and Implementation in C++

Elaborated in task1.cpp and task2.cpp files with appropriate comments. The data structures used, algorithms, time and space complexity analysis is given below.

6.1 Data Structures

1. DRAM: I have used a vector of vector named "DRAM" to implement a 2D array. Number of Rows in this 2D array are 1024 and no. of Columns are 256 (because each cell has a 4 Bytes word in it). Thus in-total we can store 2^{20} Bytes of data.

2. Row-Buffer: I have used a vector named "buffer" for keeping track of the row which is currently stored in the buffer.

   Note: I have specifically used vectors instead of arrays because to copy the content of one array into another, we have to manually change every entry in the array, whereas in case of vectors we can directly use assignment like v1=v2 and this copies the content of v2 into v1.

6.2 Algorithm

For Non-Blocking we have kept a variable Time req and register required which tells us whether a value is being loaded into some register or not. If some process is going on in the DRAM then the value of "time_req" would be non-negative and we will know that we have to wait for the result to come before executing a "unsafe" operation or issuing a new DRAM request.

6.3 Analysis

1. Time Complexity: The time complexity of the program is not actually related to the number of clock cycles required to run the MIPS program. If n is the no. of instructions that will be executed in the MIPS program (in our case we don't need to handle bne or beq or j so n = No. of non-empty lines in our input MIPS code), then the time complexity is just $O(n)$. The Row and Col access delays are just numbers that will get added to a variable "clockCycles", it will not be reflected during the actual runtime.

2. Space Complexity: Apart from storing instructions and other stuff, which was not the aim of this assignment, 2^{20} Bytes of Memory was used for making the 2D array for DRAM and 2^8 for maintaining the row-buffer. Apart from this DRAM was not allocated any extra memory and this is the reason why it waits for a previous running request to get complete before accepting any other request.

6.4 Testing & Exception Handling

1. Testing:- I manually created some test cases and provided them as input to the task1.cpp and task2.cpp . In case of correct input, I did a dry run of the MIPS code on paper and checked whether the output is correct or not. While for incorrect expressions, we terminated the

program with a appropriate error message which would be helpful to the user for debugging. We have basically tested out for different test cases like:

(a) We have converted the time memory into 2D array, and changed the clock cycle information. Irrespective of this the contents of registers and memory must not change from what was implemented in the assignment 3. So I have made sure that changes in the contents of registers and Memory are still the same for codes of task 1 and task 2.

(b) I have considered the those cases in which the next operations are always dependent on the current request in the DRAM. In these cases I have made sure that the output of Non-blocking and Simple implementation without non-Blocking are same.

(c) A test case in which I was doing only store operations helped me realize that I was missing one thing, the buffer row needs to be copied back to it's row in the end explicitly by us.

(d) It was tested that load to a register from the memory and some other safe operation in the memory can be done at the same time.

(e) I have implemented other functions like bne, beq, j and thus made a test case to check whether it was working fine for loops.

2. Error handling:- We have checked only allowed the use of 2^{20} Bytes of memory and in case the user tries to access more memory, I have thrown error. Since we are dealing with words only as of now and every word take 4 Bytes, the address given with lw and sw must be divisible by 4 else the program throws and error.

Hence all the test cases have helped us to ensure that our Algorithms and implementation are correct. Getting correct outputs on some test cases can never be the right criteria to judge if a program is correct or not, but extensive checking can still ensure us upto some good extent that we have done the things correctly.