**Part1: DRAM Implementation**

Basic data structure (DRAM\_DATA) is made to store the data for DRAM implementation which stores the current row number in row buffer (-1 if none), row access delay, col access delay and the memory location in case of “sw” instruction and the register in which value is to be loaded in case of “lw” instruction and also the instruction cycle in which the process will end (to simulate DRAM implementation). In this part after execution of any instruction, it is checked if DRAM have any request issued that is to be completed, if so, it is done and the corresponding output is shown.

The variable exec, write, read, mem, memValue, reg, regValue, ROW\_BUFFER\_UPDATE\_COUNT, START\_CYCLE in DRAM\_DATA are declared for the purpose of output. ROW\_ACCESS\_DELAY, COL\_ACCESS\_DELAY and CLOCK\_CYCLES\_INSTRUCTIONS are for the purpose of stimulating concept of parallel execution of instructions. A variable data of type DATA\_BUS is made to simulate the purpose of data bus, it stores a integer (can be thought as byte/bit) ins(0 for lw and 1 for sw) and an integer value (the value to be stored in memory in case of sw instruction and register in case of lw instruction). DRAM\_DATA also have a rowBuffer of 1024\*8 bits. A proper simulation of loading a row in row buffer, write back of row from row buffer to memory and reading or writing to a address using row buffer is implemented.

**Part2: Non-Blocking Memory**

The DRAM is implemented as if it is connected to a single processor and ones it is busy any other DRAM request cannot be issued (a single row buffer and a data bus storing request information of single instructions is considered).

The following cases are considered for checking which instructions are blocked:

1. If any DRAM request is issued for “sw” instruction than it is executed parallel to any instruction other than “lw” or “sw”, i.e, it only blocks another “lw” or “sw” instruction.
2. If any DRAM request is issued for “lw” instruction than it is executed in parallel to other instructions until a “lw” or “sw” instruction is encountered or an instruction which contains any of its argument register same as the one used as destination register in corresponding “lw” instruction which initiated the DRAM request. In case the DRAM execution blocks any instruction than the DRAM execution is completed first while the processor waits for the completion and then the instructions are executed normally.

**STRENGTHS:**

1. The execution of the overall program remains linear in terms the total number of instructions executed as the check we need to do if an instruction is blocking or not is with respect to a single instructions DRAM request. (Contrary to this, in case of an approach with multiple row buffers/data bus having enough memory to queue up multiple “lw”/ “sw” instruction would cause the time/clock cycles with respect to checking for a statement to be safe or not to be proportional to number of instructions queue up in DRAM at that time).
2. The memory requirement for my DRAM implementation is low as a copy of single row (1024 bytes) and a space for storing information of single instruction is required.
3. Any instruction other than “lw” or “sw” which does not depend on a current execution in DRAM memory is not needed to wait, reducing cycles per instruction count.

**WEAKNESS:**

1. The data bus is assumed to be of 32+1 bits only in my implementation, 32 for storing the value needed to be stored in memory or 1 for indicating weather the request is for “lw” or “sw”. So, in case there are multiple “sw” instruction at different column number and same row number, the later instruction still has to wait for the prior one to complete before its own DRAM request is issued even if the two statements are completely independent of each other.
2. In case there are multiple “sw” instruction at same memory location which on parallel execution will not result in an overlap of col access delay cycles, still the later has to raise its DRAM request after the completion of the prior due to the limitation of data bus.
3. Similarly, any combination of multiple “lw” and “sw” are restricted from executing parallel to each other by the limitation to data bus. For example, “lw” followed by “lw” are safe to be executed sequentially but are restricted, or instructions such as “lw” followed by “sw” (or vice-versa) with distinct col access delay cycles are safe to be executed sequentially but are restricted (the above example are only for cases where the row number is same for the instructions).
4. Also, as there is only one row buffer, so, even after increasing capacity of data bus, a combination of “lw” and “sw” instruction in different rows would not be executed parallelly.
5. In case there is an instruction like add $t0,$t1,$t2 after an instruction lw $t0, 1000, in this case the later instruction waits for the prior to complete even though there is no more importance of the prior instruction as the final value of $t0 register depends on the later instruction only. The execution in DRAM of prior execution can be killed midway to avoid this in the c++ code but I am not able to think of proper feasible method in terms of the hardware working which will do so as there would be cases when the later is instruction is executed in after the col access delay cycles of prior instructions execution begins and killing such process midway may cause some buffer values to be loaded in the register which may corrupt the final value.