Unit 2: Improving the pipelined processor

The objective of this task is to implement some improvements to the MIPS microprocessor (seen in theory class and developed in task 1). In this task, the hazards generated by pipelining the MIPS processor will be solved.

To carry out this practice, it is recommended to consult section 6.4 of the Patterson & Hennessy book and presentation of Unit 2.

Exercise 1: Data hazards (7 points)

In this exercise, the microprocessor datapath is modified to achieve correct results in situations in which the execution of an instruction depends on the result of a previous one still present in the pipeline, due to the use of the same register (Read After Write –RAW– data hazard). 3 mechanisms will be implemented:

1) Data forwarding to ALU.

The data forwarding shown in the following figure will be added to the previous microprocessor design, which will advance the result of the previous instruction (from MEM) or from two previous ones (from WR) for use in the stage "EX".

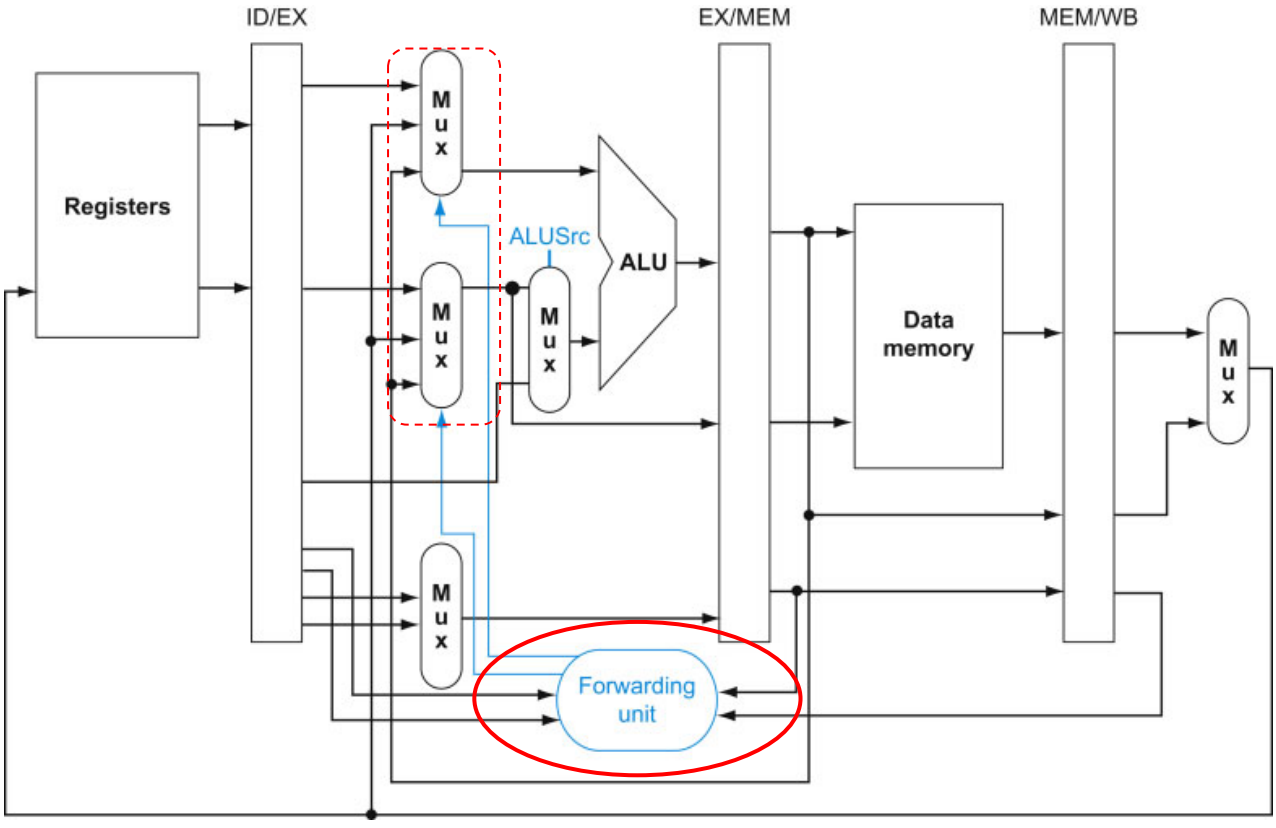

Fig 1. Datapath for data forwarding

In the presentation of Unit 2 and in the book, the conditions for activating the new routes in the forwarding multiplexers can be found in detail: it must happen if in any of the following two stages ("MEM" and "WB") the same register is being written as either of the two that were read from the record bank in the previous stage for use in this stage "EX", giving

precedence to a possible result present in "MEM" over another in "WB" (more recent) and considering the special case of register 0.

What is shown in the figure as “Forwarding unit” is simply the generation of the selection signals of the new multiplexers (those that are within the dashed red line in the previous figure), based on the conditions mentioned above. This logic must be implemented in the same architecture (that is, without creating any additional hierarchical block; both for this “Forwarding unit” and for the multiplexers themselves, as combinational processes or concurrent assignments).

2) *Internal data Forwarding* in register bank.

The data forwarding unit implemented with the previous figure contemplate the execution in the ALU of an operation with a new register value available in the pipeline registers EX / MEM and MEM / WB (actually in the latter case after an additional multiplexer), corresponding to one and two previous instructions respectively. An additional scenario occurs when the dependency is on the instruction that entered the pipeline 3 cycles earlier. In this case the solution is to create a data forwarding within the register bank itself (that is, in the ID stage instead of in the EX stage). The data relationships for the previous schema and for this additional case are shown in the following figure:

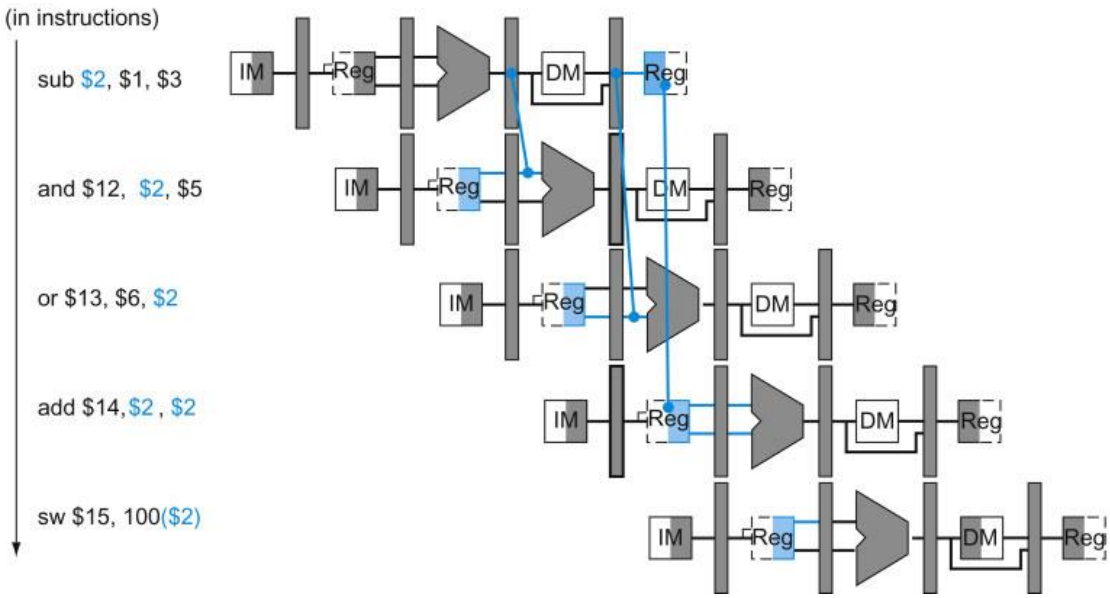

Fig 2. All possible data forwarding paths

The register bank provided in Task 1 must be modified in one of two possible ways:

- Adding a combinational path, so that, when the same register that is being written is read, the bank returns the value that is being written instead of the one in the accessed register. In addition, the special behavior of register 0 must be considered.

- By making the register bank run on a falling edge. In this alternative you should be able to explain why it should work

3) Detection of LW instruction data hazard.

In this case it is not possible to forward data. A “stall” cycle must be generated, repeating the current IF and ID stages and inserting a “bubble” (as a nop instruction) in the following stages. The “Hazard detection unit” shown in the following figure will detect the mentioned condition, preventing the update of the program counter (PC) and the IF / ID pipeline register, and setting the control signals to zero in the ID / EX register to create the "bubble":

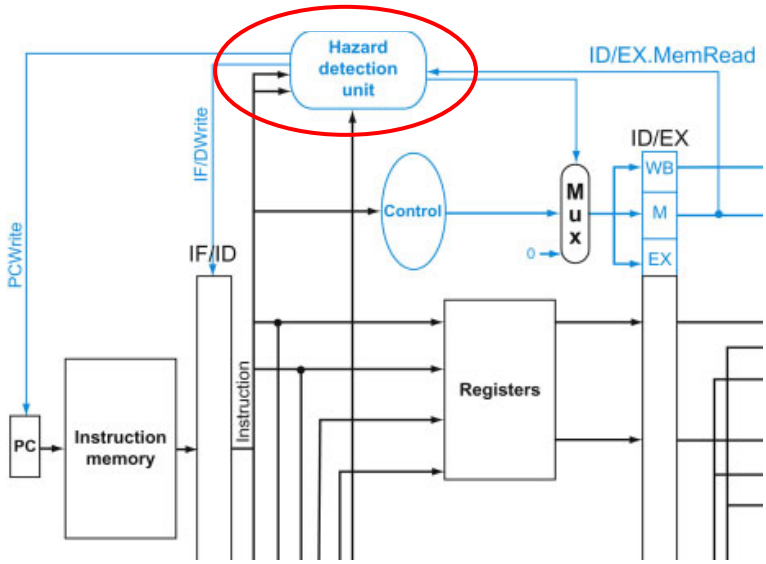

Fig 3. Datapath for LW data hazard

Note: to check the condition of load-use hazard, use what is seen in theory and exposed in the reference book, that is, compare the rs and rt fields of the new instruction with respect to the rt destination of a LW instruction in the previous cycle. This solution is not optimal as it generates unnecessary bubbles for the LUI and SW instructions (see optional proposed exercise).

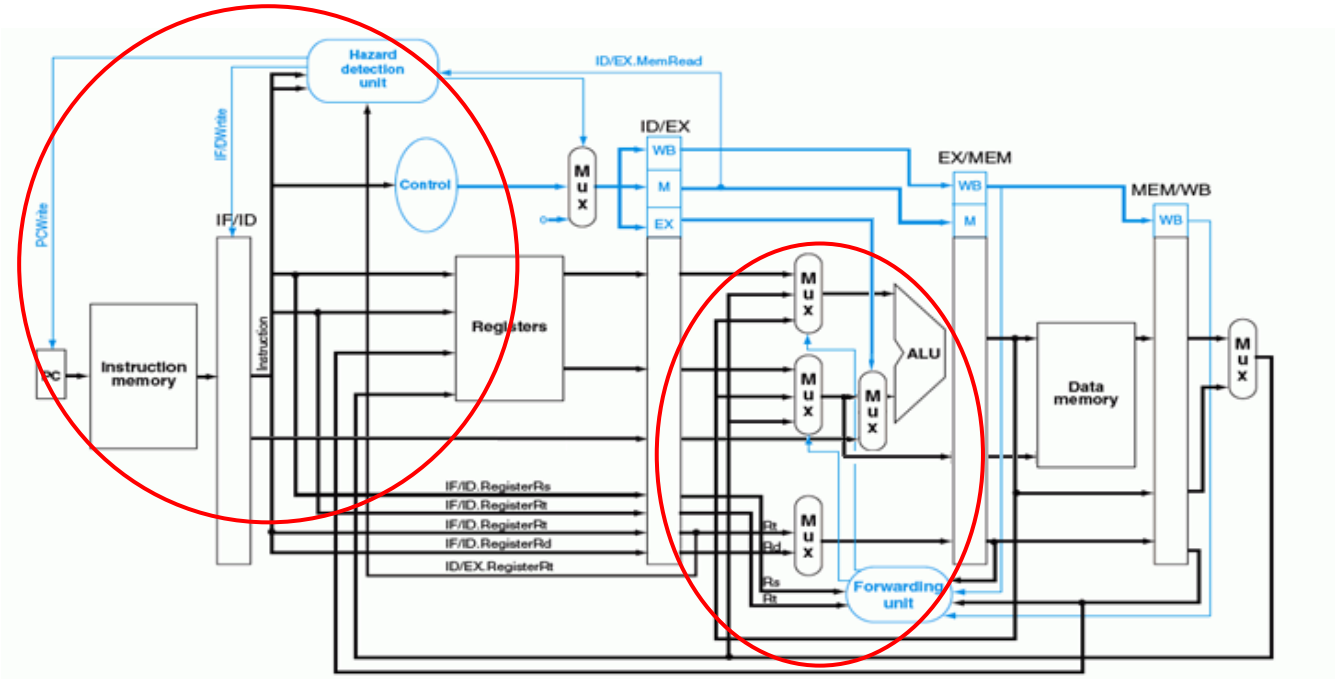

Fig 4. Complete datapath for RAW data hazards

Note 1: The processor must execute all instructions correctly, except *branch* when a data hazard is involved.

Note 2: You can use the web based compiler provided in order to compile the code.

Exercise 2: Control Hazards (branch) (3 points)

In exercise 1 the hazards due to branch instructions have not been considered. In this exercise, the processor must be modified to correctly execute the branch instruction under any previous condition of the program. In particular, processor modifications must be made so that the branch instruction is executed correctly after an ALU type operation as well as after a memory load of data.

NOTE: In this exercise we will consider as an objective that the instruction following the beq is not executed when the branch is effective (unlike in the real MIPS architecture where this first instruction after the branch is always executed). Because the assembler generates a "nop" automatically after the beq, we will not be able to verify well in simulation that this is true, but the code has to consider it. In particular, it must be considered that if the instruction that follows the beq is an lw, the effective jump condition must prevail over a possible stall caused by the lw.

This exercise also requests the source code of a program (file in assembler) to verify the correct behaviors of the branch in these scenarios:

- A previous R-type instruction writes a register and this register is used in a branch. Test this scenario when the branch is **effective**, and when it is **not effective**.
- A memory instruction reads a value and saves it to a register and this register is used in a branch. Test this scenario when the branch is **effective**, and when it is **not effective.**

MATERIAL TO SUBMIT

- VHDL files of exercises 1 and 2.
- The assembler program used to test the two different scenarios in exercise 2.

Notes:

- Submit a zip file through Moodle.
- Deadline is theon Friday 29th October, before 23:59. Check the schedule in Moodle.
- The lab instructor can request the presentation / defense of the task in any laboratory class after the submission. This defense is part of the task grade.