Computer Organization and Design (ECE 219)
Winter Semester 2021-2022

Laboratory 7

1. Extending the MIPS micro-architectural pipeline

In this lab you will make extensions to the MIPS architecture you completed in Lab 6.

Implementation of Program Flow Change Commands (8 credits)

To extend the previous micro-architecture as represented in figure 1) so that it can also execute the following MIPS branching instructions:

- **j Label**
- **beq rs, rt, Label**
- **bne rs, rt, Label**

In addition to the correct execution of these commands, you should also implement any extension of forwarding, stalling and flushing mechanisms that these additional commands may cause. For example, consider the effect branching commands have on commands already in the pipeline. The execution of the branch instruction should be done in the EX stage (and the result forwarded by the MEM stage, as we saw in the lesson). You should use and extend the *control.v* file and the *cpu.v* file that you developed in the previous lab. Your code should run correctly for each case like for example the assembly code given to you in *testbench.v* (which you should not change). In the comments of *testbench.v* you can also find the expected values of registers and memory after each executed instruction in both iterations of the loop. We assume that each register is initialized with the value reg[i] = i.

Your examination will focus on the correct functionality of branching commands.

Important Note: the correct functionality of the testbench requires that you name some specific signals in the CPU *(cpu.v)* as follows:
(a) the PC select multiplexer control signal between PC+4 and the branch jump address should be named *PCSrc.* (b) the control signal generated by the control unit *(control.v)* that is 1 when a bubble (NOP) needs to be generated in the EX stage due to stall or flush should be named *bubble_idex.*

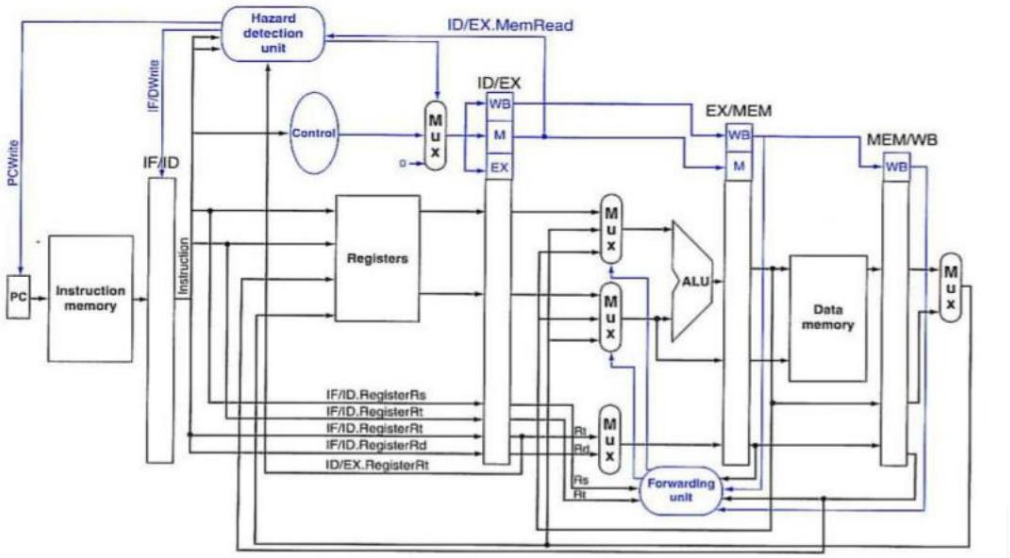

Figure 1. MIPS architecture with pipeline, risk detection unit and bypass unit. This architecture can execute format-R commands, as well as load/ store commands. On this architecture you will build the circuit of Project 7.

These restrictions are made to properly compile and simulate the testbench code:

string_manipulation pipe0(clock, reset, cpu0.PCSrc, cpu0.bubble_idex, cpu0.instr, cpu0.IFID_instr, stringvar);

Register File (3 units)

Until now, we have assumed that MIPS registers are written on the negative edge of the clock. This assumption helps us reduce the number of dependent instructions that read a register written by a previous instruction. But in real micro-architectures, registers are read and written on the positive edge of the clock like all other registers. This subquery asks you to make what changes are needed in the register

file so that they are both read and written on the positive edge of the clock.

Pay particular attention to the case where in the same machine cycle an instruction writes to a register from which another instruction reads. While writing to the negative edge of the clock addressed this problem in previous implementations, in this implementation you will have to work around this problem and ensure that the data being read is the correct one. Also the reading of the registers should be synchronous with the positive edge of the clock.

The new micro-architecture should correctly execute the code in the previous question.