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

Laboratory 4

1. Objectives of the next workshops

The goal of the following workshops of the Organization course is to design, implement, and verify through simulation an architecture that implements a subset of the MIPS instructions. Specifically, the labs will gradually implement a single-cycle MIPS processor and then expand it
multi-cycle architecture with 5 pipeline stages with new commands.

For this purpose you will use the hardware description language Verilog, and a library of units (components). The library will also include all the units necessary to implement the data part of the processor, namely Memory, Register File, and Arithmetic-Logic Unit (ALU).

2. Implementation of MIPS core units (6 units)

To practice hardware design with Verilog you will start building the ready-made module library by implementing 2 basic modules that will be used later in the MIPS implementation. You are given the file library.v which contains the
ALU and the Register File which are implemented except for some points which have been replaced by '...' and which you will have to fill in yourself.

a. Arithmetic Logic Unit - ALU

module ALU (out, zero, inA, inB, op); Arithmetic/
logical unit for use with MIPS instructions. Performs addition or subtraction of integers (using 2's complement), the Set On operation
Less (slt) and logical operations AND/OR/NOR.

Parameters - Ports:

N: Parameter: the width of the Arithmetic, Logical Unit in bits. **out:**

Output (N bits): data output. **zero:**

Output (1 bit): 1 when output out is 0.

inA: Input (N bits): first data input. **inB:** Input

(N bits): second data input

Op: Input (4 bits): operation selection input. The numbers below are expressed in the decimal system.

- 0 bitwise AND: out = inA & inB.
- 1 bitwise OR: out = inA | inB.
- 2 addition: out = inA + inB.
- 6 subtraction: out = inA – inB

- 7 Set On Less Than: out = ((inA < inB)?1:0)
- 12 NOR: out = ~(inA | inB).

Example of using a 32-bit ALU:

```
ALU #(parameter N=32) ALUInst (ALUOut, Zero, ALUinA,
ALUinB, ALUop);
```

Finally, in case the input **Op** takes some value not found in the table above, the output can be, for example, don't care X.

b. Register File

module RegFile (clock, reset, raA, raB, wa, wen, wd, rdA, rdB)

File of 32 Registers, each 32-bit in size, with two read ports and one write port.
The clock and reset inputs should appear in all sequential circuits such as
Flip-Flops, Registers, Finite State Machines (FSMs), coc. The clock in sequential circuits is the way of timing the circuit so that whatever changes occur in the circuit are only made at the time when the clock changes value. In this case the circuit

it is called edge-triggered, and when the change in the output of the circuit occurs during the transition of the clock from 0 to 1 it is called positive edge-triggered. When the change in the output of the circuit occurs when the clock changes from 1 to 0, the circuit is called negative edge triggered. In almost all cases that we will consider the sequence circuit is positively edge-triggered. Any exceptions will be noted on a case-by-case basis. Asynchronous reset causes the Flip-Flops of the sequencer circuit to go **immediately** to a known initial state (usually 0),

without waiting for the clock edge. As long as reset is 0 (active), all registers remain at 0, regardless of clock edges (reset is active low). The normal operation of the circuit begins when the reset becomes inactive, i.e. rises to the value 1.

Ports:

clock: Input. Synchronization clock. All writes to registers should take place on the negative edge of the clock.

reset: Login. Asynchronous reset. Active low.

raA: Input (5 bits): register address to be read from the first port. **raB:** Input (5 bits):

register address to be read from the second port. **wa:** Input (5 bits): write address. **wen:**

Input (1 bit): enable signal to write data.

wd: Input (32 bits): input data to write. **rdA:** Output (32 bits):

read data output of the first port.

rdB: Output (32 bits): read data output of the second port.

The Register File has asynchronous access for reading (read), where after applying the register number **(raA** and **raB)** the data is displayed. In order to write to the Registrars File, you need, in addition to the number of the registrar
wa and the data input **wd,** the enable signal **wen** and the negative edge of the clock clock. Also the reset must be 1 (inactive).

i. Registration in the Register of Registrars

Registration is based on the wen mark, with the following procedure:

o Wen becomes 1. o

The write address, **wa** constant. , and the data, **wd ,** must already be

o Clock transitions from 1 to 0 (negative spike firing).

ii. Reading from the Register File

Reading is done only based on the address of the register, i.e. without a read activation signal.

c. Interconnection of the two elements

The last part of the lab asks you to connect the ALU and the register file as follows:

- the two inputs of the ALU (inA and inB) should be connected to the two outputs register file read data,
- the ALU output (out) should be connected to the write data input of the registrar file,
- the remaining signals (op, wen, ...) will be set by the testbench.

In other words, the register file feeds the ALU with inputs (operands) and receives back the result of the ALU operations.

3. Simulation to Verify Correct Operation (4 credits)

To verify the data part of the processor you provide a testbench skeleton template, which you should modify (file testbench.v). also
there are some points that have been replaced with '…' and which you will have to fill in yourself. This template file contains instructions for displaying the processor unit, initializing the register, setting the clock, and applying the control signals per cycle. The register file initialization

it will be done by directly assigning the value to each register.

To verify your circuit, it is recommended that you use the waveform window, or both Verilog's **$display** and **$monitor** commands , which display or actively monitor the value of the signals respectively. Thus, you can monitor all the outputs of the data part of the processor, while after each instruction you are suggested to print the values of the registers. For example if we want to print register 2, through the command $display:

```
$display("Register %d : %x", i, regs.data[i]);
```

Where in this example, regs is the name of the display of the data section of the processor, registerfile is the display of the register file inside the testbench, and data[i] is the i word in the internal table of registers in the library (0<= *i <=31).*

To check the correct operation of your circuit there is a for-loop in the testbench which traverses all the registers and performs the ADD operation in the ALU. Your we recommend that you also write your own tests, as the ALU and the register file you will create will be the basis for the next workshops. We're not looking for anything specific in the testbench, just a way to test as many register file and ALU functions as possible.

4. Hardware development and simulation software (optional for Modelsim)

Mentor Graphics ' *Modelsim* is a complete circuit simulation and synthesis package that supports Hardware Description Language
Language) Verilog. Modelsim will be useful for exercises related to the implementation, simulation and verification of operation of a MIPS-type processor in hardware in Verilog language . The software supports Linux and Windows platforms and is installed on the department's Linux computers as part of a larger one

CAD tools package called *questa.* You can also download *Modelsim student* for free from related website *http:// www.mentor.com/ company/ higher_ed/ modelsim- student-edition* to work on your desktop/laptop.

On the Linux platform, starting the program and displaying the windows is done with the *vsim command.*
%vsim

After placing the Verilog files you have developed in a directory, say *$HOME/ ece219/ labs/ components,* you can go to this directory from the console using familiar *Linux commands.*

% cd $HOME/ ece219/ labs/ components

In this directory you should first create the name of the library that will contain the binary files that *Modelsim* uses to represent the circuit. In *Modelsim* the name of the current library is **work** (by default).

%vlib work

You only need to do the act of creating the **work** library once. H
vlib command creates the work directory under the current directory and initializes it with some files. Do not edit these files by hand, since any changes to them should only be made through the individual tools of *Modelsim.* To compile the Verilog files use Modelsim 's *vlog* command , or

better yet, select *Compile Compile* from the pop-down menu at the top of the window and select the files you want to compile. Compilation simply checks the lexical and syntactic correctness of your program, not the assembly of individual sub-modules. It would be good to compile your program often during development and not just at the end.

As long as everything goes well with the compilation and all the Verilog files are syntactically correct, you will be able to load your circuit into *Modelsim* for simulation using the *vsim command.* In the pop-down menu: *Simulate Start Simulation and*
select the *work/ testbench file.* At this

point, the so-called *elaboration takes place,* the phase in which the individual modules are interconnected and their hierarchy is created. A common error at this stage is a size mismatch between interconnected signals at different levels of the hierarchy. As long as the *elaboration* is correct and the whole process terminates without errors or

warnings, you can start the simulation using commands like:

run 100 ns // run design for 100 ns
restart // restart simulation from time=0

The best way to check the correct operation of your circuit is by observing the waveforms.