

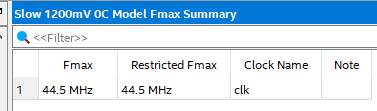
# ARQUITECTURA DE COMPUTADORAS

# Práctica 3

# Raúl Méndez Álvarez

# 3/junio/2019

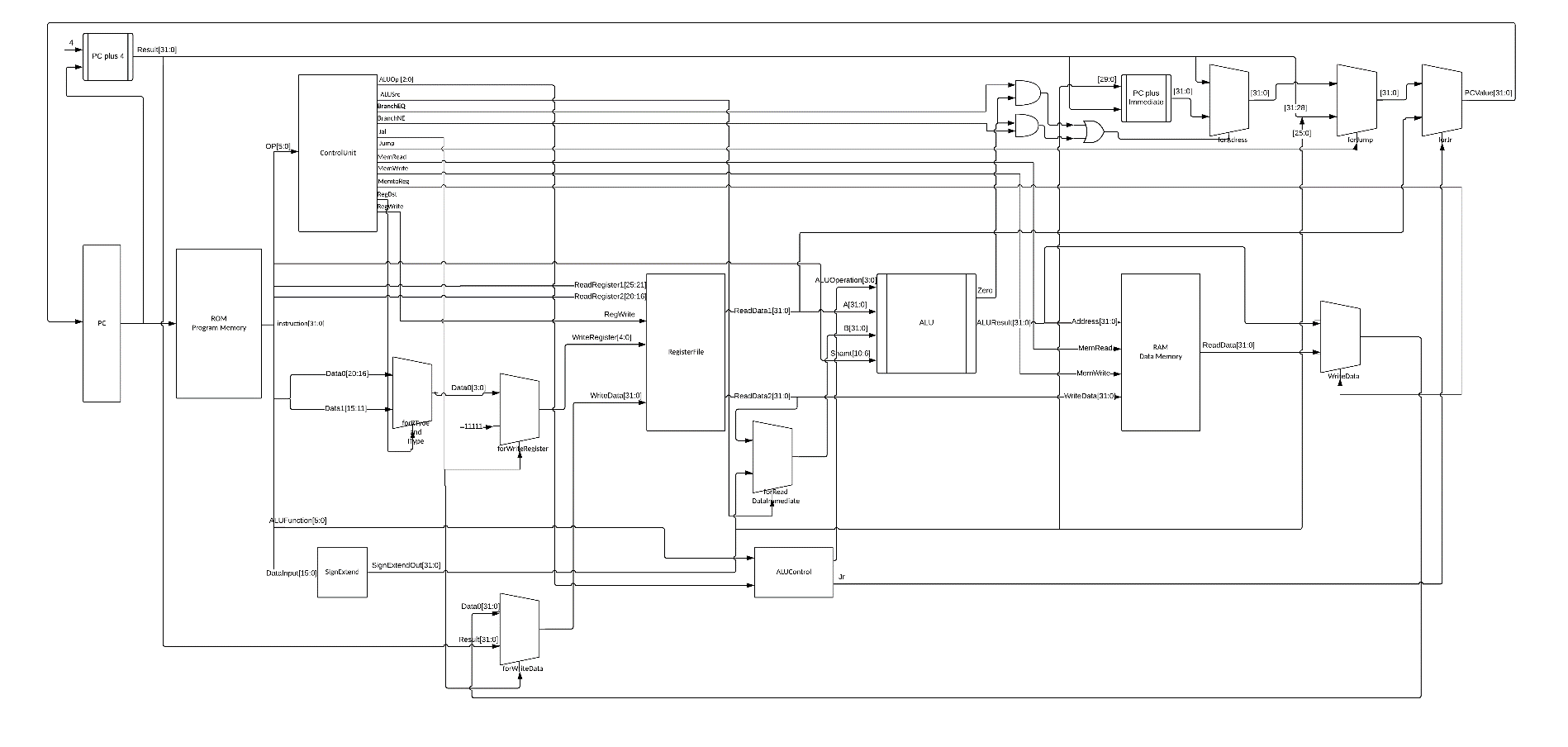
## IC, CPI, Clock Rate and CPU time for the MIPS Implementation and type R, I and J instructions percentage.

****

**A screenshot of a cell phone

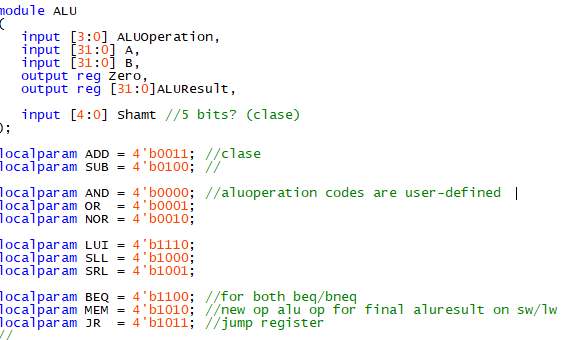
Description automatically generated**

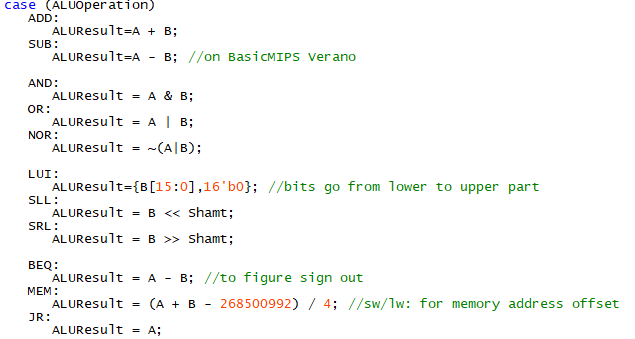
**MIPS Micro-Architecture**

****Feel free to look at the full-sized pdf as well as png files included in the archive/zip

**Modified Modules**

### ALU.V

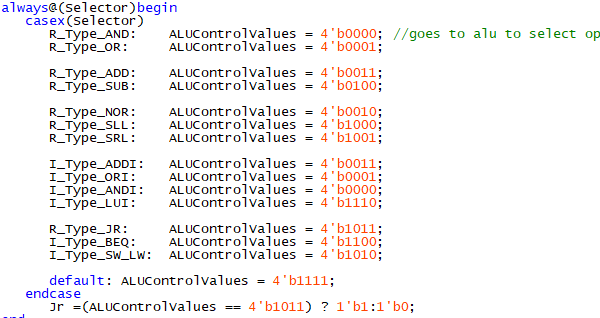
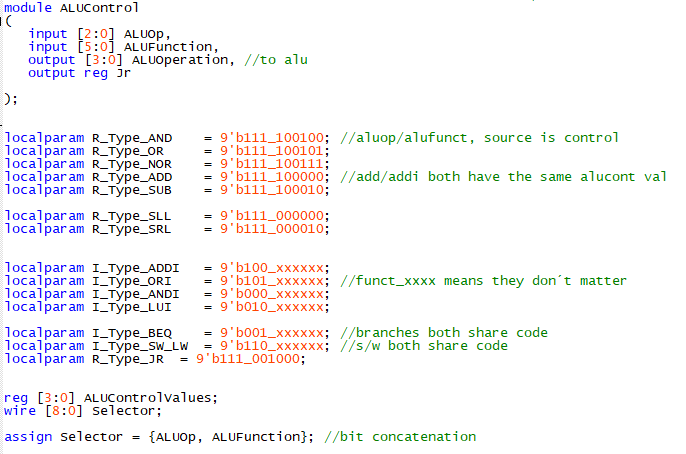
In this module I added the specific localparam for each instruction that was implemented and set cases for each instruction. In each case the ALUResult obtains a different result from the operation specified by the instruction. I also added a new input called “Shamt” (shift amount) for the SRL and SLL instructions.



*ALU Control.v*

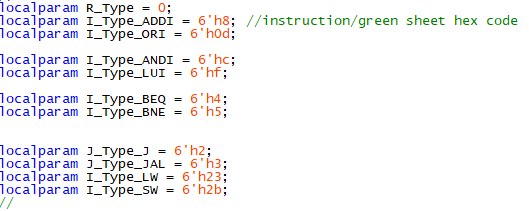
This module contains an specific localparam for each R-Type and I-Type instruction and each param contains an specific opcode which was taken from the MIPS´ Green Card.

I added new cases in the selector for the implemented instructions and each instruction has its own ALUControlValue that was arbitrarily specified in Alu.v module as a localparam.

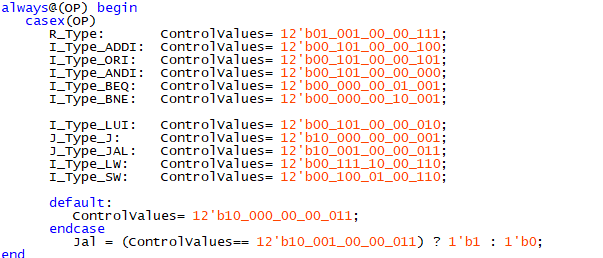


### Control.v

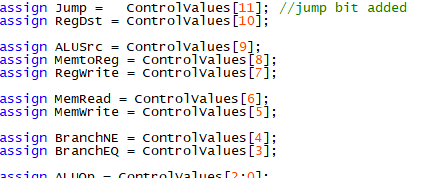
In this module I added new localparams for each implemented instruction and each localparam is the OPCODE of the instruction.



I added new ControlValues for each instruction, this control values are the ones that specify the actions that the processor will be doing for each one of the instructions.

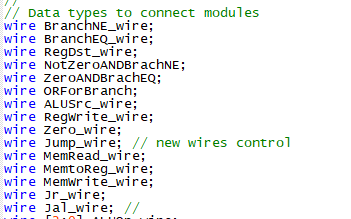


Each bit of the control value has its own meaning and it is specified in this part of the module.



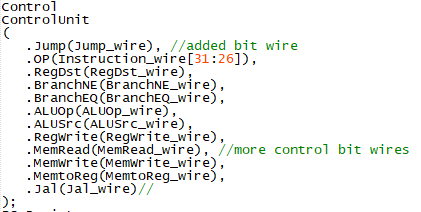
### MIPS\_Processor.v

This module contains all the connections in the processor, in the top of this module I added all the new wires that I needed to connect the each module correctly with each other and the signals for each multiplexer.

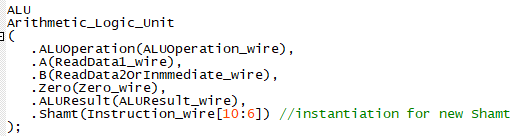


I modified some of the modules that were already instantiated with new connections with the new wires and I also instantiated some more that I needed in order to connect all the modules together.

In the instantiation of the control unit, I added the signal cables for the multiplexers and for the decoding and execution part of the processor.



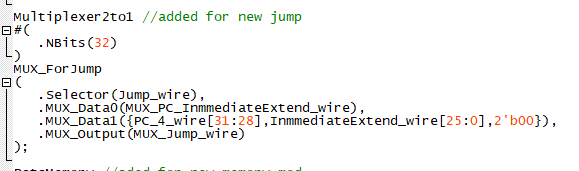
In the AritmethicLogicalUnit I connected shamt to the new input was added in Alu.v module for the SLL and SRL instructions.



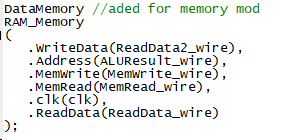
In the bottom part of this module I assigned two new cables for the Program Counter in case of a BEQ or a BNE instruction.



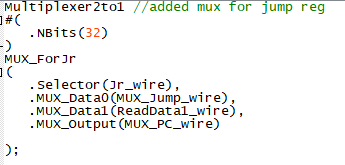
I instantiated a new MUX for the jump instruction.



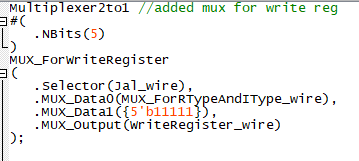
I also added here an instantiation for the RAM memory and its signals.



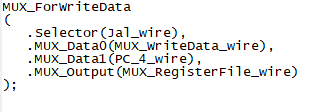
This new MUX in the module was instantiated for the JR instruction



I also added this MUX called “MUX\_ForWriteRegister when the fetching instruction is a JAL.

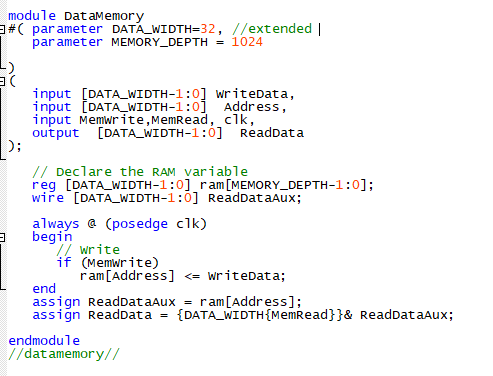


New MUX for the WriteData signal



### DataMemory.v

We changed the parameter DATA\_WIDTH from 8 to 32.



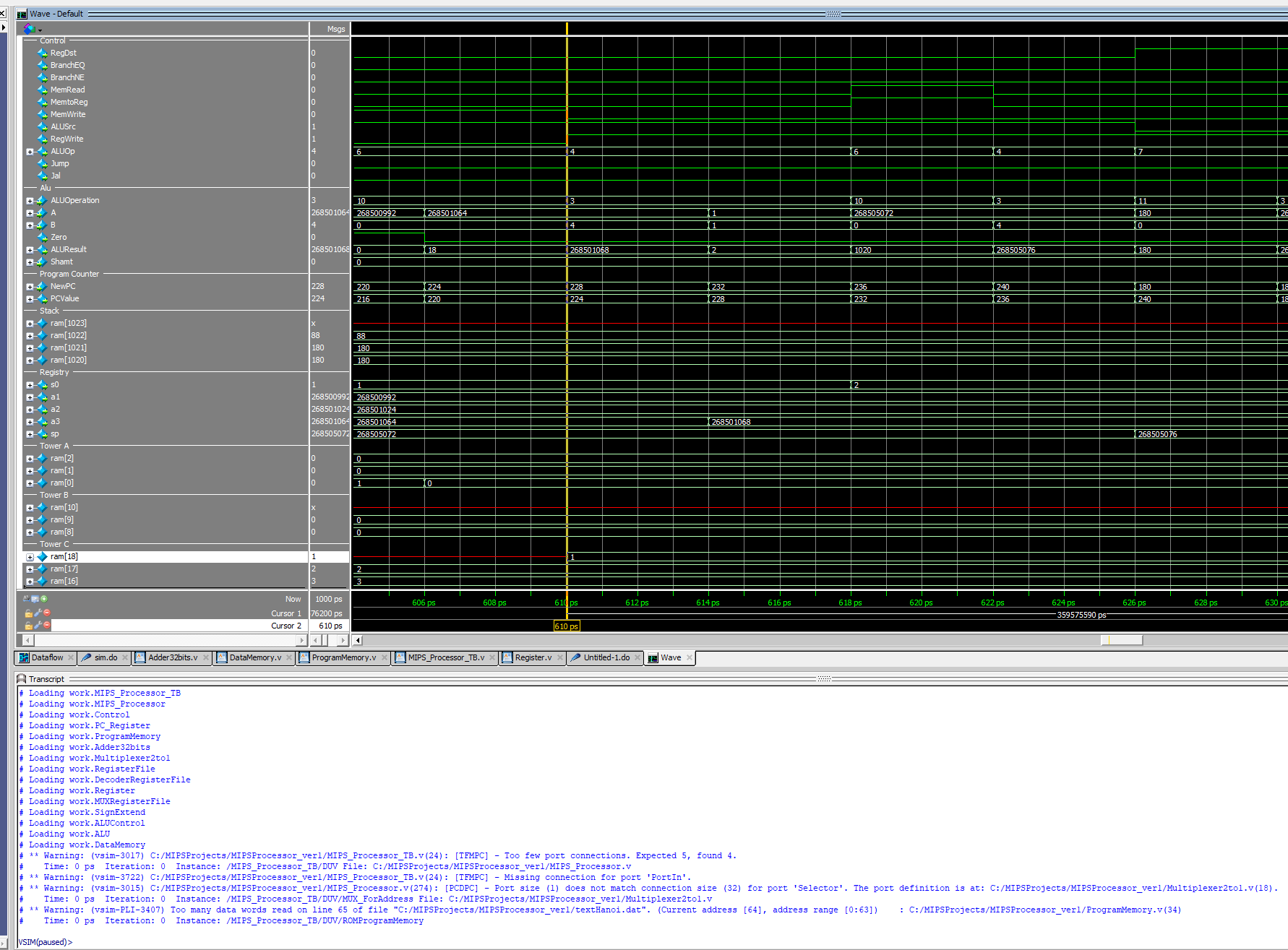
## ModelSim MIPS Simulation

## 

The ps pointer/line is marked at precisely the moment/cycle where the disk with disk #3 would be placed at its initial tower A position.

## 

The ps pointer/line is marked at precisely the moment/cycle where the disk with disk #2 would be placed at its mid-process tower B position.

****

The ps pointer/line is marked at precisely the moment/cycle where the disk with disk #1 would be placed at its final tower C position.

Feel free to enlarge the image to view all of the registers´ finals status (assuming the document format allows you to), as well as to run the simulation yourself on ModelSim (files included).

**Flow**

**A screenshot of a cell phone

Description automatically generated**

Flow description:

My algorithm would begin by associating each of the addresses to each of its respective tower as well as register. We´d also always have n stored in s0, even though such variable is inputed by the user beforehand.

The loadDisks function or subroutine would then begin to load each and every disk onto its respective space or place on tower A/origin tower until the amount of disks are complete. And then it would proceed to call the HanoiTower function for the first time, which would initially allocate the memory necessary to store the ra registry onto the top stack position.

The function would first try to validate whether the amount of disks is or isn´t 1 in order to jump to the base case (which would move the respective tower to its according position) or to the internal function steps (which would swap the pointers or addresses on each of the towers accordingly in order to follow the algorithm described on the assignment).

The process would loop in and out of the steps, unwinding and then rewinding when its ready to swap the disks to their respective next tower, step by step, all by switching the auxiliary tower accordingly. Once the algorithms takes us to the last step or to the top of the rewinding process, it would put the last disk onto its final place and finally it would restore the size of the sp as well as s0 to finally be done with the process.