Instituto Tecnológico y de Estudios

Superiores de Occidente – ITESO



**Materia: Arquitectura de Computadoras**

**Maestro: Rodrigo Aldana López**

**Práctica 3: Design and simulation of a multi-cycle processor based on the MIPS architecture using pipes.**

**Fecha: 19 de julio del 2019**

**Temas: MIPS**

**Autor(es):**

**Luis David Gallegos Godoy (709571)**

**Guillermo Hernández Landeros (713777)**

**Introduction**

The problem of the Towers of Hanoi is to move a column made from discs concentric of different diameters, stacked from the largest diameter to the smallest diameter, from its original position A to its destination position C, having point B available for movements intermediate This movement has the restriction that only one disc can be moved to the time, and that at no time can a disc of greater diameter be on a disk of lesser diameter.

But know we are making the processor that can handle the instruction of the program, that’s using Quartus to program in Verilog de MIPS.

**Development of the MIPS**

Implement in Verilog a processor based on the MIPS architecture, which can execute the instructions add, addi, sub, or, ori, and, andi, nor, sll, srl, lw, sw, beq, bne, j, jal, jr which must adhere to the MIPS Green Sheet specification. This implementation can use as a starting point the data-path seen in class, which was designed to support some of the previous instructions. This implementation must be able to execute the program that was used in practice 1, note that this implementation must support recursive functions.

**Development of HANOI**

This algorithm starts by filling up tower A with N discs, then proceeds to enter the Hanoi function by giving it the values of N, and the 3 towers (origin, auxiliary and destiny), if the N value is 1 it will directly move the disc at the top of the given tower directly to the destiny tower, if it's not one it will move the N-1 discs to the auxiliary tower by calling for Hanoi function with the values N-1, Origin and swapping auxiliary and destiny, then proceeds to move the Nth disc to destiny tower and move the N-1 discs to destiny tower, this time using origin tower as destiny.

**New modules:**

**OR Gate:**

A gate of type OR was added, this to be able to perform a branch, since in the case that BEQ or BNE is 1 then it becomes a branch.

**Forwarding Unit:**

This module serves as a way to send ahead data that hasn't been sent in a register and is required for a following instruction.

**Hazard Detection Unit:**

This module was made to stall and noop the next operation in cases of branches, jumps or if the value of a load word will be used right after it's been requested.

**Pipe**

This is a register module saves the needed data for the next step in a pipeline to be able to move on to the next step while loading the next instruction in the previous step.

**Equals**

This module serves as a way to put the branch conditions into a module, making it easier to release the wire that tells us if a branch is taken,

#Luis David Gallegos Godoy

#Guillermo Hernandez Landeros

.data

**.text**

**addi** $k0, $zero, 1

#numero de torres

addi $s0**,** $s0**,** 3

#direcciones de torres 1 a 3

addi $a1**,** $zero**,** 0x1001 #direction of tower 1 destination

sll $a1**,** $a1**,** 16

addi $a2**,** $zero**,** 0x1001

sll $a2**,** $a2**,** 16

addi $a2**,** $a2**,** 0x0020 #direction of tower 2 destination

addi $a3**,** $zero**,** 0x1001

sll $a3**,** $a3**,** 16

addi $a3**,** $a3**,** 0x0040 #direction of tower 3 destination

**add** $t1**,** $a1**,** $zero

#fill tower 1

**add** $t0**,** $s0**,** $zero #counter

**addi** $zero**,** $zero**,** 0

**addi** $zero**,** $zero**,** 0

**addi** $zero**,** $zero**,** 0

loop\_fill**:**

sw $t0**,** 0**(**$t1**)** #**add** disc

addi $t1**,** $t1**,** 4 #next disc **memory** value

addi $t0**,** $t0**,** **-**1 #next disc value

**addi** $zero**,** $zero**,** 0

**addi** $zero**,** $zero**,** 0

**addi** $zero**,** $zero**,** 0

bne $t0**,** $zero**,** loop\_fill

**add** $t0**,** $s0**,** $zero

**add** $t2**,** $a2**,** $zero

**add** $t3**,** $a3**,** $zero

jal Hanoi

j end

#algorithm to solve hanoi

#Hanoi**(**N**,** A**,** B**,** **C)**

#params**:** $t0**=** tower size**,** $t1 **=** origin**,** $t2 **=** auxiliary**,** $t3 **=** destiny

Hanoi**:**

#STACK

addi $sp**,** $sp**,** **-**12

sw $ra**,** 0**(**$sp**)** #**push** return address

sw $t0**,** 4**(**$sp**)** #**push** return N value

sw $t5**,** 8**(**$sp**)** #**push** placeholder

**addi** $zero**,** $zero**,** 0

**addi** $zero**,** $zero**,** 0

beq $t0**,** $k0**,** move\_disc

#Hanoi**(**N**-**1**,** A**,** **C,** B**)**

**add** $t4**,** $zero**,** $t2

**add** $t2**,** $zero**,** $t3

**add** $t3**,** $zero**,** $t4

**sub** $t0**,** $t0**,** 1 #N**-**1

**add** $t5**,** $t0**,** $zero

jal Hanoi

#Hanoi**(**1**,** A**,** B**,** **C)**

**add** $t4**,** $zero**,** $t2

**add** $t2**,** $zero**,** $t3

**add** $t3**,** $zero**,** $t4

addi $t0**,** $zero**,** 1

jal Hanoi

#Hanoi**(**N**-**1**,** B**,** A**,** **C)**

**add** $t0**,** $zero**,** $t5

**add** $t4**,** $zero**,** $t2

**add** $t2**,** $zero**,** $t1

**add** $t1**,** $zero**,** $t4

jal Hanoi

#return each tower to its original position

**add** $t4**,** $zero**,** $t2

**add** $t2**,** $zero**,** $t1

**add** $t1**,** $zero**,** $t4

j finish

move\_disc**:**

addi $t1**,** $t1**,** **-**4 #reduce origin tower pointer

lw $t9**,** 0**(**$t1**)** #save value of origin tower **in** a placeholder

sw $zero**,** 0**(**$t1**)** #clean tower value

sw $t9**,** 0**(**$t3**)** #store value **in** destiny tower

addi $t3**,** $t3**,** 4

finish**:**

lw $ra**,** 0**(**$sp**)** #**pop** return address

lw $t0**,** 4**(**$sp**)** #**pop** return N value

lw $t5**,** 8**(**$sp**)** #**pop** placeholder

addi $sp**,** $sp**,** 12

jr $ra

end**:**

DIAGRAM (HANOI):

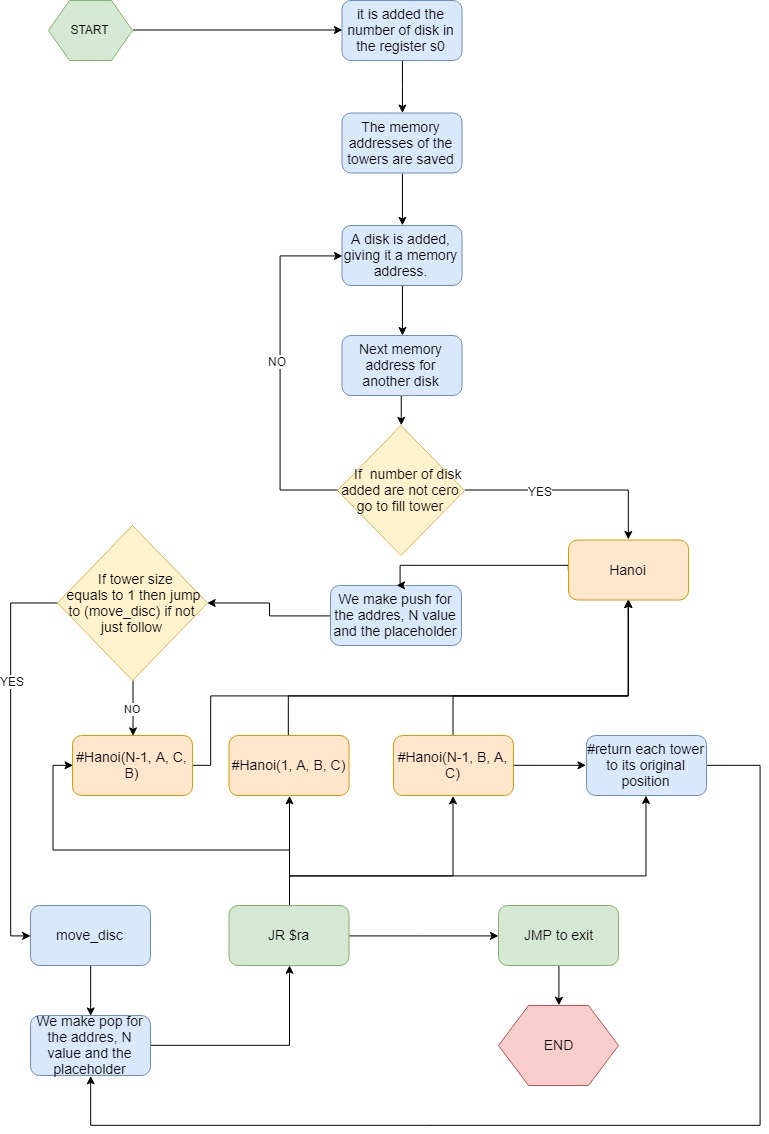
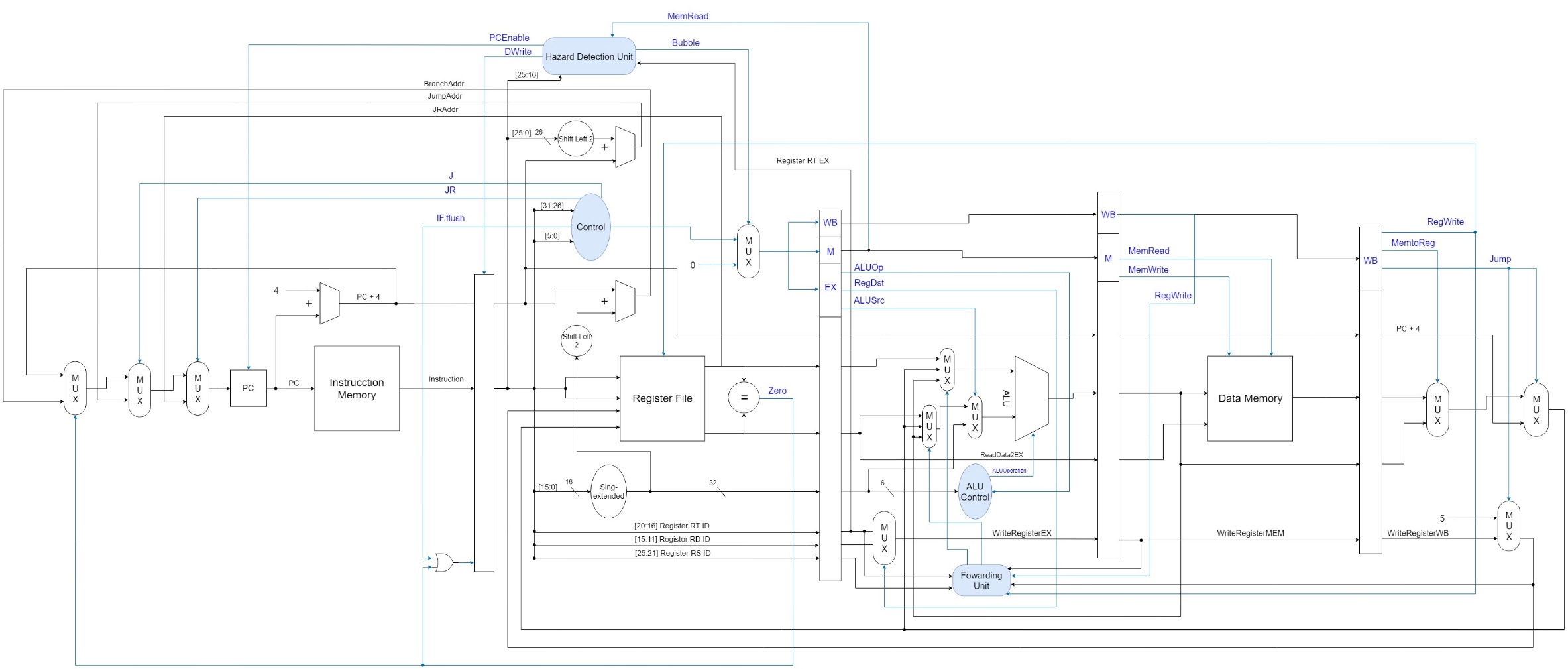
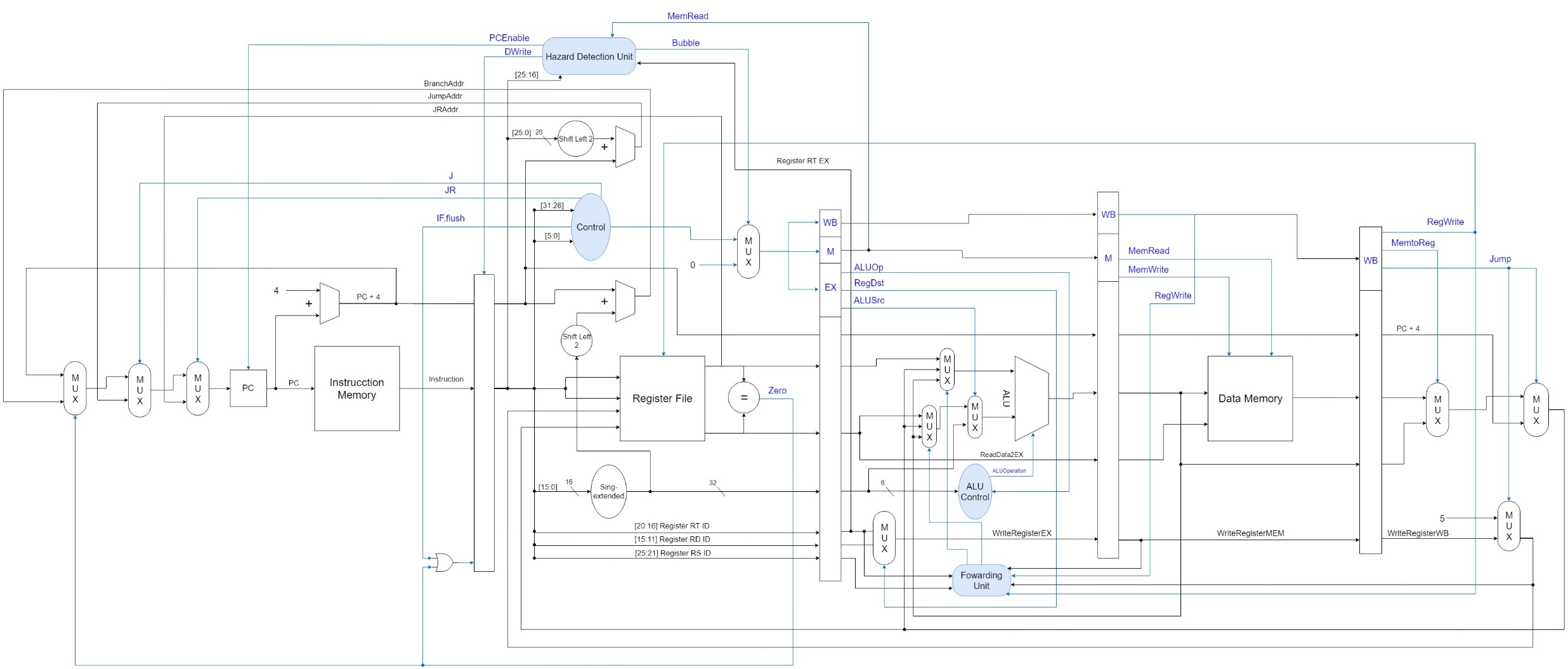
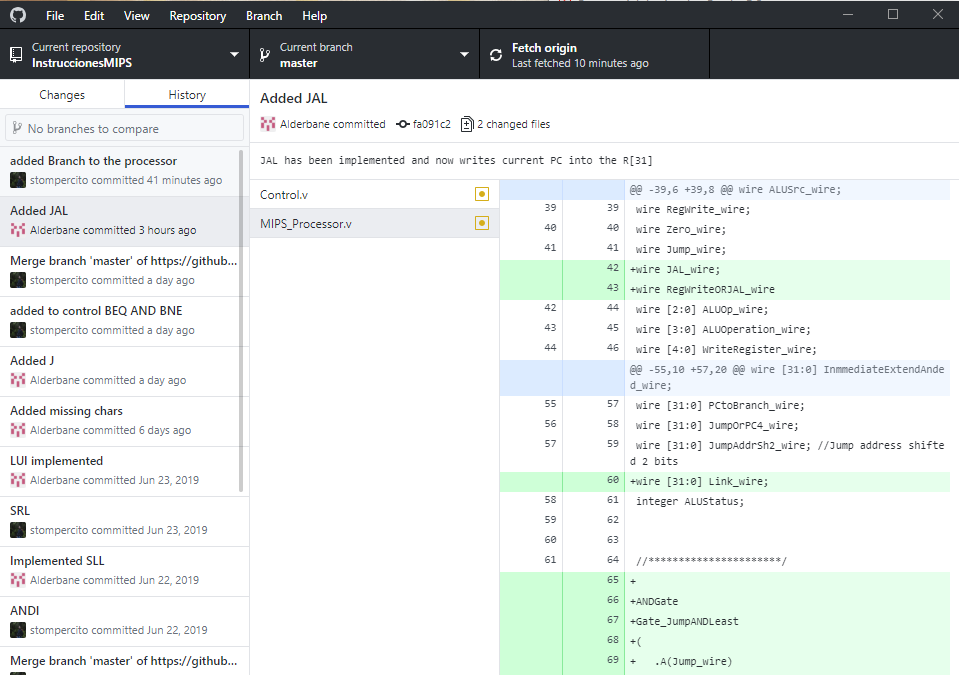
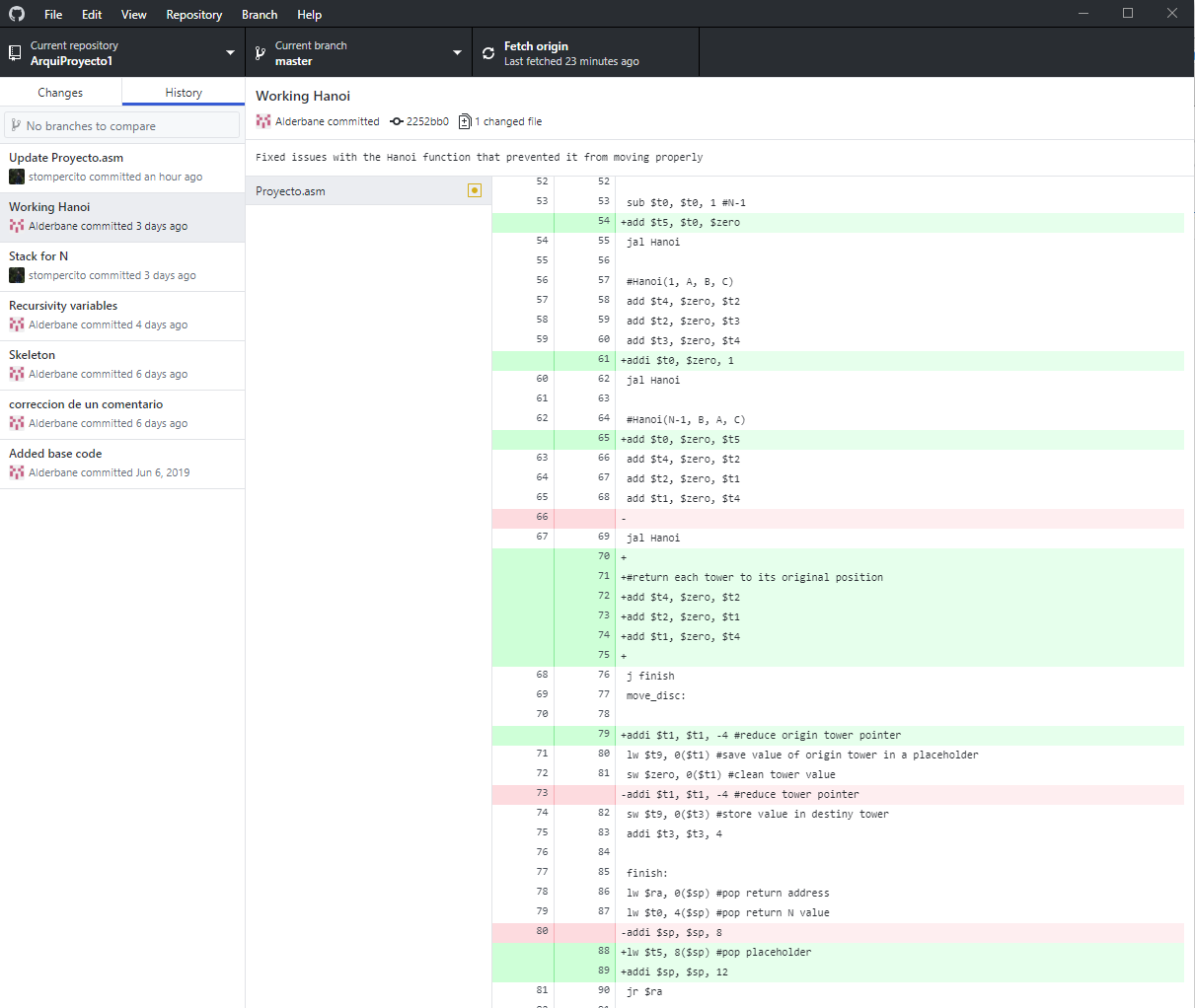


DIAGRAM (MIPS): 

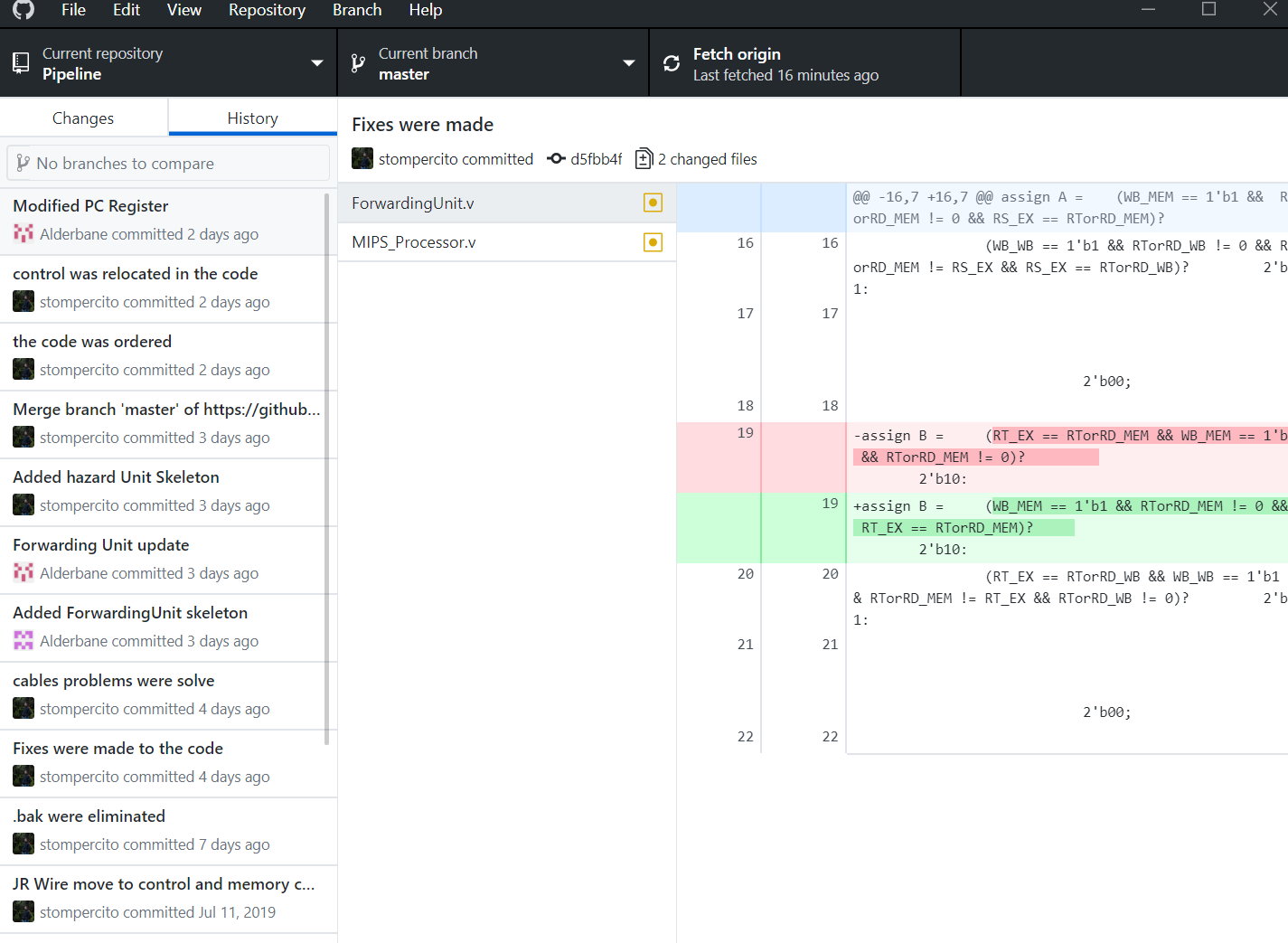
GITHUB (HANOI):



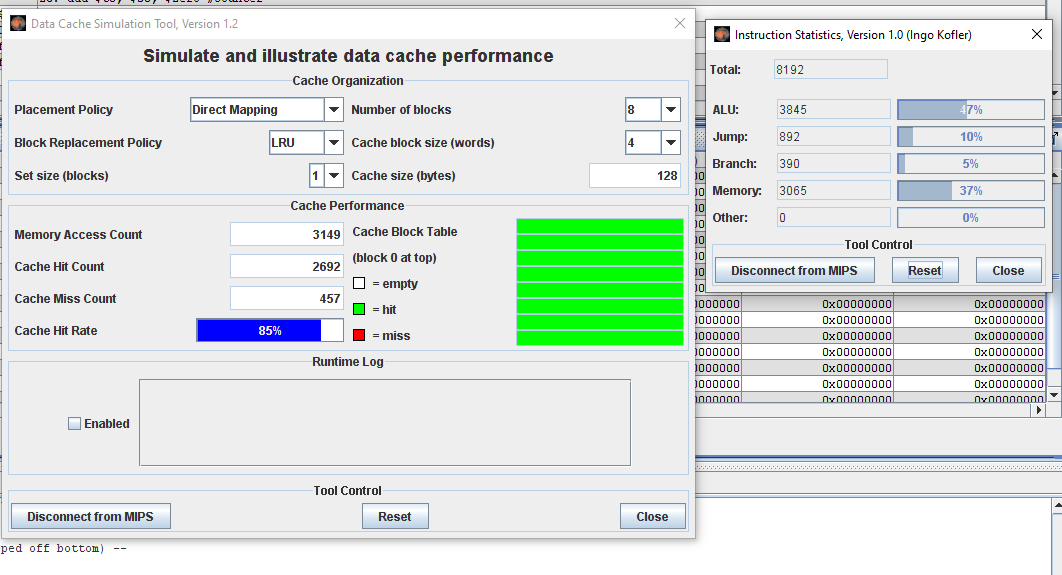
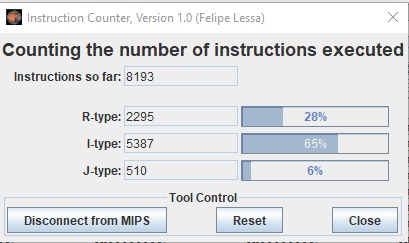
GITHUB (MIPS):



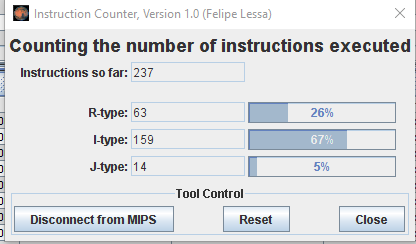
GITHUB (MIPS\_WITH\_PIPES:

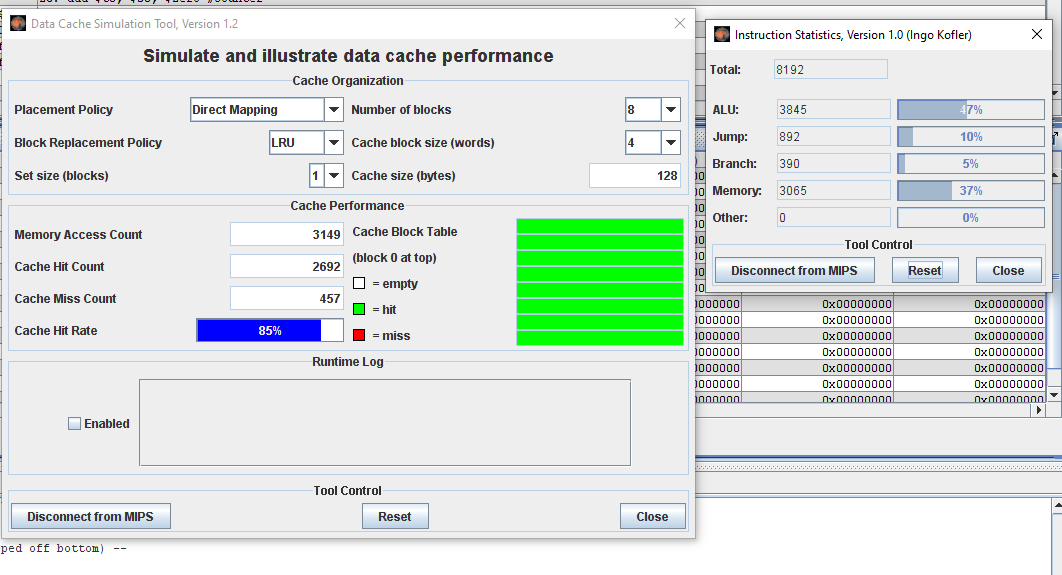


**8 Discs percentages:**

****

**3 Discs percentages:**

****

****

**Conclusions**

**Guillermo Hernandez Landeros:** This practice was complicated and posed a real challenge to show us how a processor can be further sped up by pipelining the steps needed to do an instruction, not just by overlapping the instructions but also by increasing the frequency and the CPI. This proved to be an appropriate final practice to learn how to optimize and speed up a processor.

This practice turned out to be quite interesting since we understood how MIPS works and how to come to create the design of a processor, at the same time we understood how to demonstrate the operation of the various processor instructions, and how the processor grows as instructions are added.

**Luis David Gallegos Godoy:** Understanding how the processor works, which executes certain lists of instructions, made me understand better how computers communicate with each other and how a processor works.

A complicated part was the implementation of new instructions to the processor, because sometimes it was a bit confusing. The level of practice was quite suitable for current knowledge and at the same time give us a better understanding through practice.