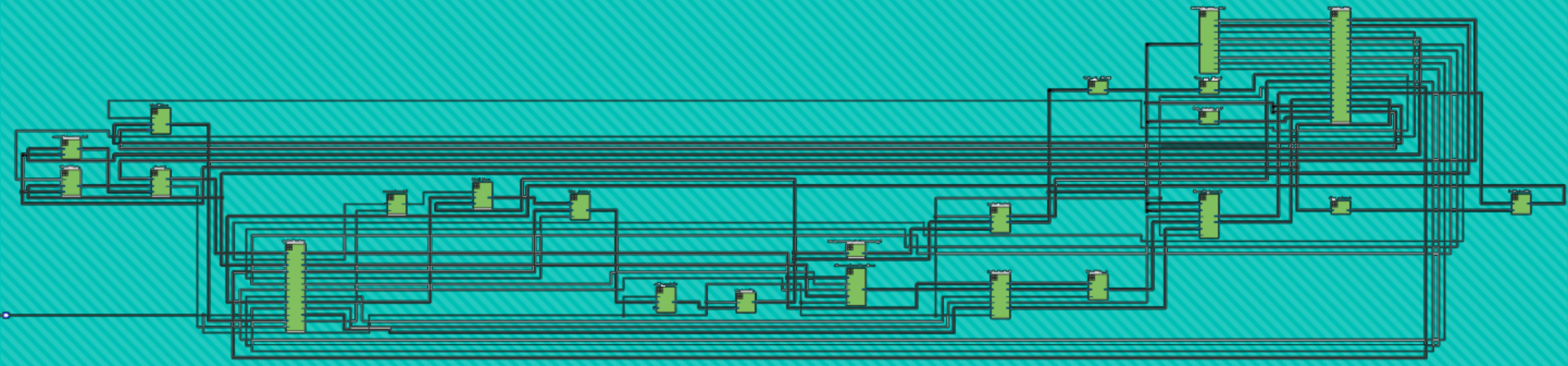


MIPS Proyecto Final

Jorge Humberto González Ramos

Ochoa Velasco Dana

Jiménez Vital Francisco

Introducción

MIPS por las siglas de Microprocessor without Interlocked Pipeline Stages, se conoce a toda una familia de microprocesadores de arquitectura RISC desarrollados por MIPS Technologies.

* El objetivo principal del proyecto es hacer una versión que tenga las funciones mínimas del procesador MIPS y correr programas simples en el.

Planeación

En nuestro caso el proyecto se dividió en 3 fases para crear el MIPS en Verilog, la documentación y decodificación fueron realizadas a la par que con el Verilog

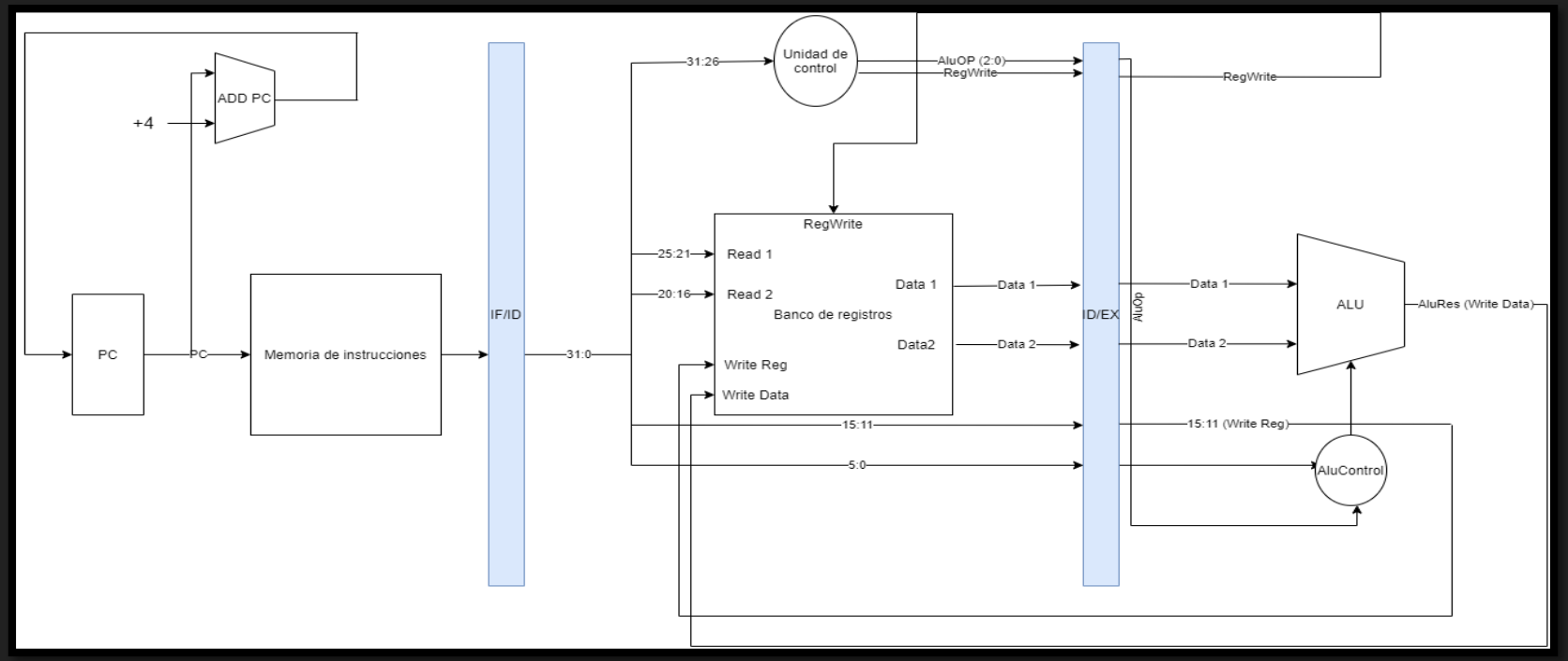
Verilog

Esta parte del proyecto fue creada principalmente por Dana, se encargó de crear las estructuras.

Se dividió en 3 fases que veremos a continuación

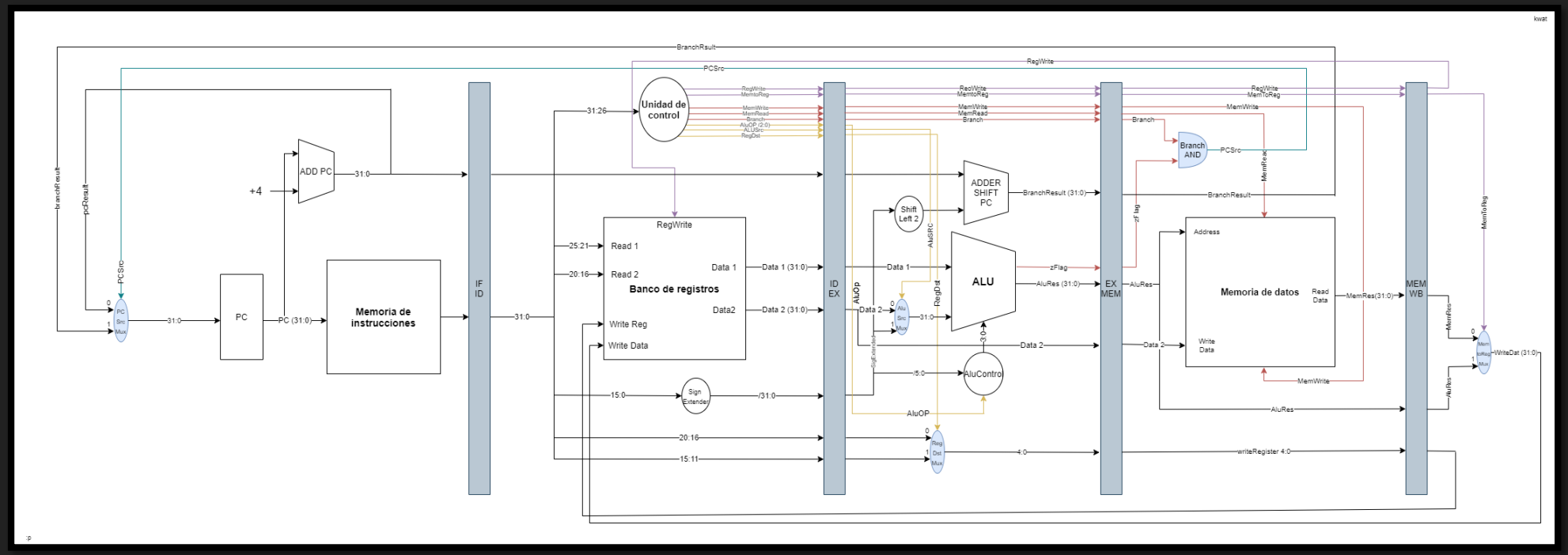
Fase 1

La primera fase consistió en crear los módulos para tener el soporte de las instrucciones de tipo R



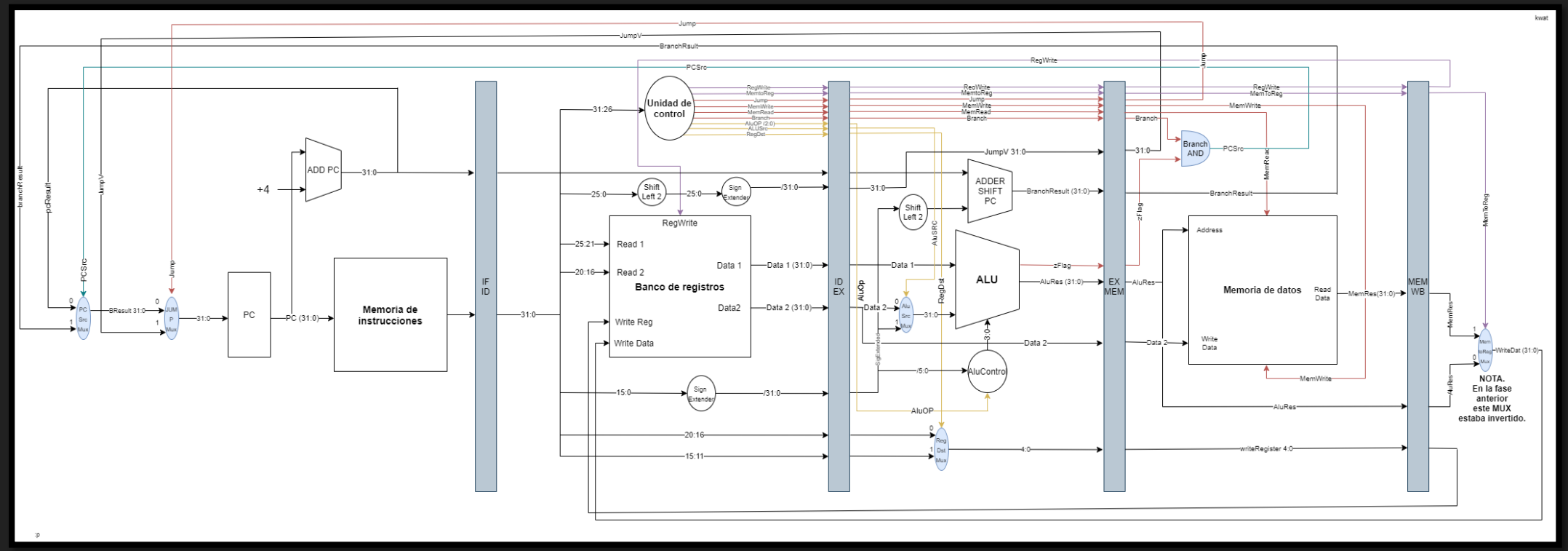
Fase 2

La segunda fase consistió en añadir el soporte de instrucciones que tengan un valor inmediato



Fase 3

En esta última fase se agregaron pocos módulos pero con ellos fue posible agregar la instrucción de tipo J



Resumen de Verilog

Al terminar la codificación resultaron los siguientes módulos

- Unidad de Control
- AdderPC
- AdderBranch
- ALU
- Alu Control
- Banco de registros
- Memoria de datos
- Shift Lefts
- Multiplexors y And
- Memoria de instrucciones

Los módulos están documentados en la wiki de github.

Decodificador

El decodificador fue creado por Jorge y todo el progreso de creación puede ser seguido en github.

Decidimos que el decodificador contaría con las siguientes características

- Soporte de etiquetas
- Generación de NOPS para errores
- Detección de errores de escritura
- Pseudoinstrucciones

Decodificador

- Posiblemente fue la parte más compleja del proyecto y se puede encontrar en el Github junto a su documentación.

Verilog y Decodificador

Ya con estas partes del proyecto terminadas ya podíamos poner en prueba el trabajo, creamos las 2 opciones que tenían el visto bueno en ensamblador y se probaron.

Tipo de triángulo

El primer programa en ensamblador fue el de detectar el tipo de triángulo

```
.main:
    li $t0 5 # lado 1
    li $t1 5 # lado 2
    li $t2 5 # lado 3
    sub $t4 $t0 $t1
    sub $t5 $t0 $t2
    beq $t4 $zero .f1

    .iso:

    sub $t6 $t1 $t2

    beq $t4 $zero .isoceles  # a == b
    beq $t5 $zero .isoceles  # a == c
    beq $t6 $zero .isoceles  # b == c

    # si no entonces es escaleno
    j .escaleno

    .f1:
    beq $t5 $zero .equilatero

    j .iso

.equilatero:
    li $t3 1
    j .end

.isoceles:
    li $t3 2
    j .end

.escaleno:
    li $t3 3

.end:
    sw $t3 0($zero) # guardar tipo de triangulo
```

Una prueba

En el código de la imagen anterior se ingresaron los datos 5,5,5 que es un triángulo equilatero, se decodificó y se cargó el archivos de instrucciones en el testbench.

El resultado obtenido fue el esperado.

El resto de testbenchs están en el github.

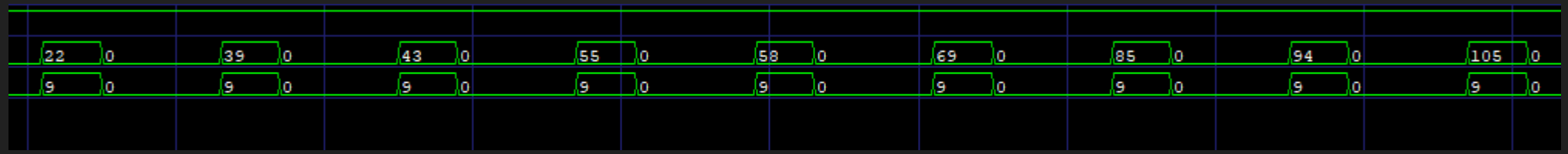
Selective Sort

El Segundo programa en ensamblador fue el selective sort, un tipo de ordenamiento.

```
main:
    li $t5, 85
    li $t4, 22
    li $t3, 94
    li $t2, 55
    li $t0, 105
    li $t6, 58
    li $t7, 43
    li $t8, 39
    li $t1, 69
    sw $t5, 0($zero)
    sw $t4, 1($zero)
    sw $t3, 2($zero)
    sw $t2, 3($zero)
    sw $t1, 4($zero)
    sw $t6, 5($zero)
    sw $t7, 6($zero)
    sw $t8, 7($zero)
    sw $t0, 8($zero)
    li $t9, 8 # total de nums
.sort:
    add $t0, $zero, $zero # i = 0
    .loop1:
        addi $t1, $t9, -1 # n - 1
        add $t2, $t0, $zero # index = i
        addi $t3, $t0, 1 # j = i+1
        .loop2:
            nop
            lw $t4, 0($t3) # arr[j]
            lw $t5, 0($t2) # arr[index]
            # if arr[j] < arr[index]
            blt $t4, $t5 .change
            j .addj
            .change:
                add $t2, $zero, $t3 # index = j
            .addj:
                addi $t3, $t3, 1 # j = j +1
                blt $t3, $t9, .loop2 # mientras j < n
        .swap:
            lw $t4, 0($t2) # arr[index]
            lw $t5, 0($t0) # arr[i]
            add $a1, $zero, $t4 # temp = arr[index]
            sw $t5, 0($t2) # arr[index] = arr[i]
            sw $a1, 0($t0) # arr[i] = temp
        .add_i:
            addi $t0, $t0 1 # i = i + 1
            blt $t0, $t1 .loop1 # mientras i < n - 1
.end:
    lw $t1, 0($zero)
    lw $t1, 1($zero)
    lw $t1, 2($zero)
    lw $t1, 3($zero)
    lw $t1, 4($zero)
    lw $t1, 5($zero)
    lw $t1, 6($zero)
    lw $t1, 7($zero)
    lw $t1, 8($zero)
```

Otra prueba

Se eligieron los datos **85, 22, 94, 55, 105, 58, 43, 39, 69.** Se cargó el código en el testbench y obtuvimos los siguientes resultados:



Los resultados fueron los esperados.

Documentación

La documentación estuvo principalmente hecha por Francisco quien se encargó de crear el documento y hacer mayoría de la wiki.

https://github.com/kwattt/MIPS-teams/wiki

Referencias

The MIPS32 MIPS Architecture for Programmers Volume II-A
http://www.nec.co.jp/press/en/9801/2002.html Computer Organization and Design

5thEdition David A. Patterson/Jhon L. Hennessy

https://web.archive.org/web/20150719075343/http://www.sumagamer.com/noticias/newhorizons-mismo-procesador-playstation/