Departamento de Ingeniería de Sistemas
Arquitectura de Computadores y Laboratorio -2021-1
Laboratorio No. 4

UNIVERSIDAD DE ANTIOQUIA
Facultad de Ingeniería

Hecho por:

Valeria Granada Rodas CC. 1000763818
Juan Pablo Bedoya Sanchez CC. 1152691819

Diseño del procesador monociclo

1. **Objetivos:**
   - Comprender los requerimientos de una versión reducida de la arquitectura MIPS de 32 bits para realizar una implementación en forma de procesador monociclo (ruta de datos y unidad de control) .
   - Codificar, ensamblar y simular un programa de prueba para verificar el comportamiento correcto del procesador.
   - Emplear herramientas de software para el diseño y la simulación de computadores digitales.
2. **Descripción**

En esta práctica se realizó una implementación como procesador monociclo de una versión reducida de la arquitectura MIPS32 en la Herramienta *Logisim* Evolution. Una vez implementados la ruta de datos y la unidad de control, el procesador fue puesto a prueba para verificar el correcto funcionamiento de todas las instrucciones elegidas. Luego, el procesador fue empleado para ejecutar un programa, cargado en la memoria de instrucciones en código de máquina.

El procesador está en capacidad de ejecutar las siguientes instrucciones: **load word, store word, add, sub, and, or, nor, set-on-less-than, branch if equal, jump, jump-and-link** y **jump-register** , y tiene una organización como la que se muestra en la Figura 1.

El diseño de la ALU sigue un estilo estructural y jerárquico, en el que se diseñan bloques básicos que luego son instanciados para crear otros más complejos y de mayor nivel en la jerarquía de diseño. Para construir la ALU, de la biblioteca de componentes de Logisim FvoJution sólo se emplean los siguientes componentes: compuertas AND, OR, NOT y multiplexores. La estructura jerárquica básica a nivel de bloques de diseño que debe tener la ALU es la siguiente:

- ALU 32 bits
    - ALU 1 bit
        - Sumador completo de 1 bit

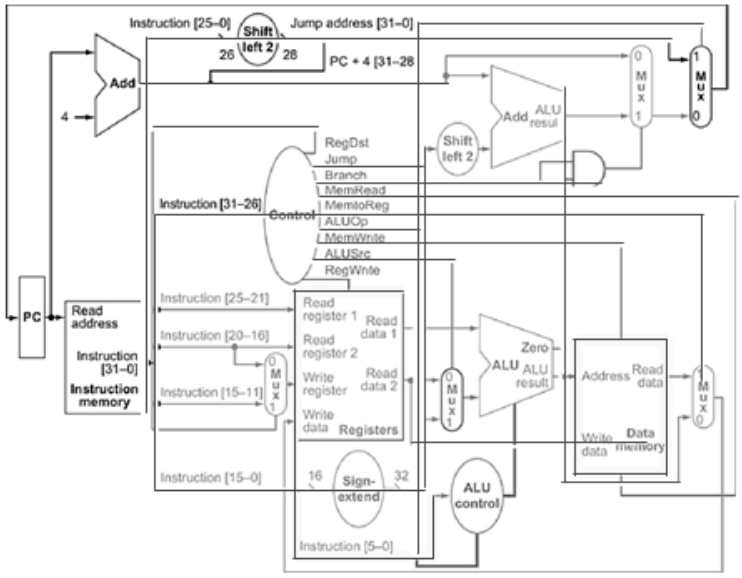


Figura 1.

La codificación de las señales de control asociadas a la ALU, ilustradas en la Figura 1, es la implementación mencionada en el tercer punto de este documento.

3. Control de la ALU

Codificación #2

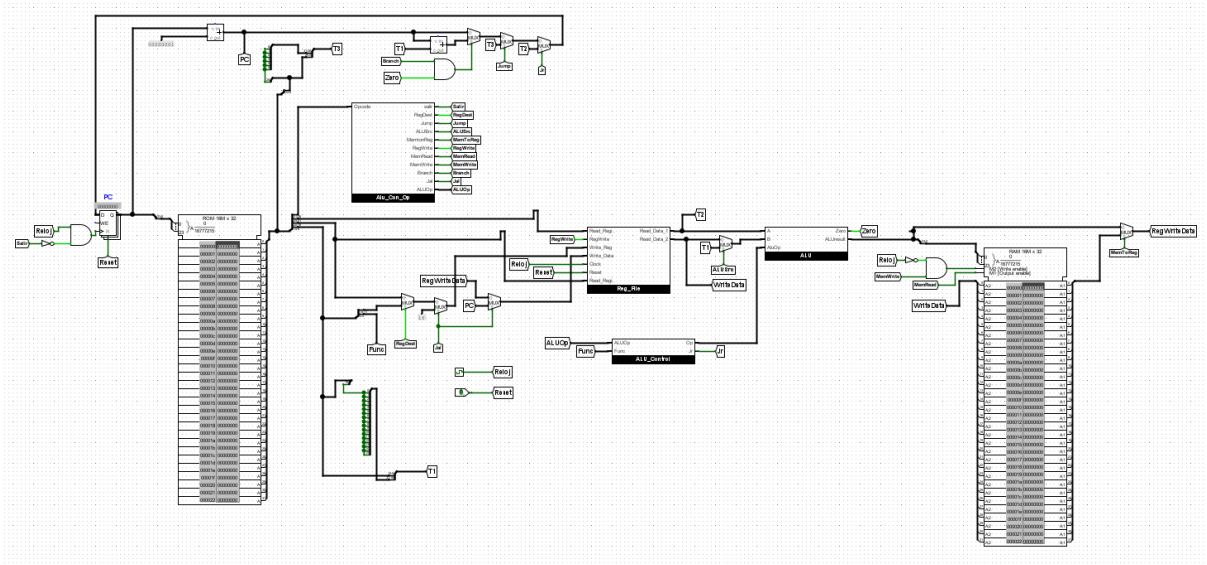
Opcode	ALUOp
lw, sw	11
beq	10
Tipo R	00

Función	ALU Operation
AND	010
OR	011
NOR	111
Suma	101
Resta	001
SLT (*Set on less than*)	100

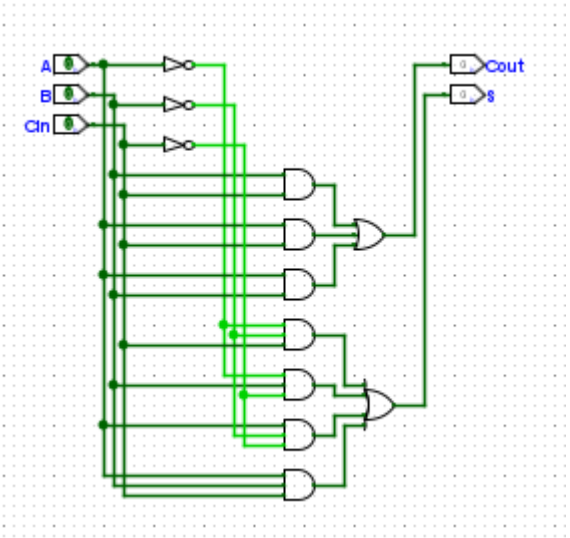
GITHUB CIRC:

https://github.com/Juanpablo733/Lab4ArqYLAb

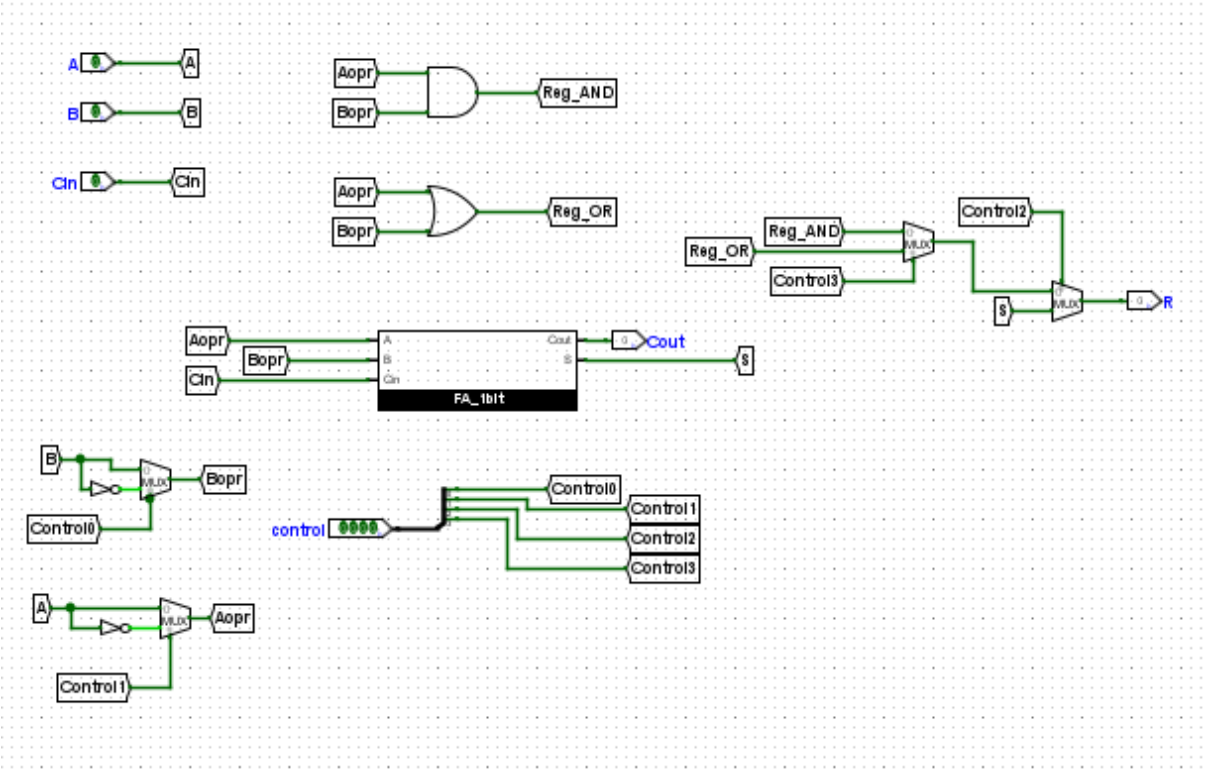
Queda implementado el procesador monociclo en Logisim, con sus diferentes componentes y estructura lógica para el proceso de ejecución.



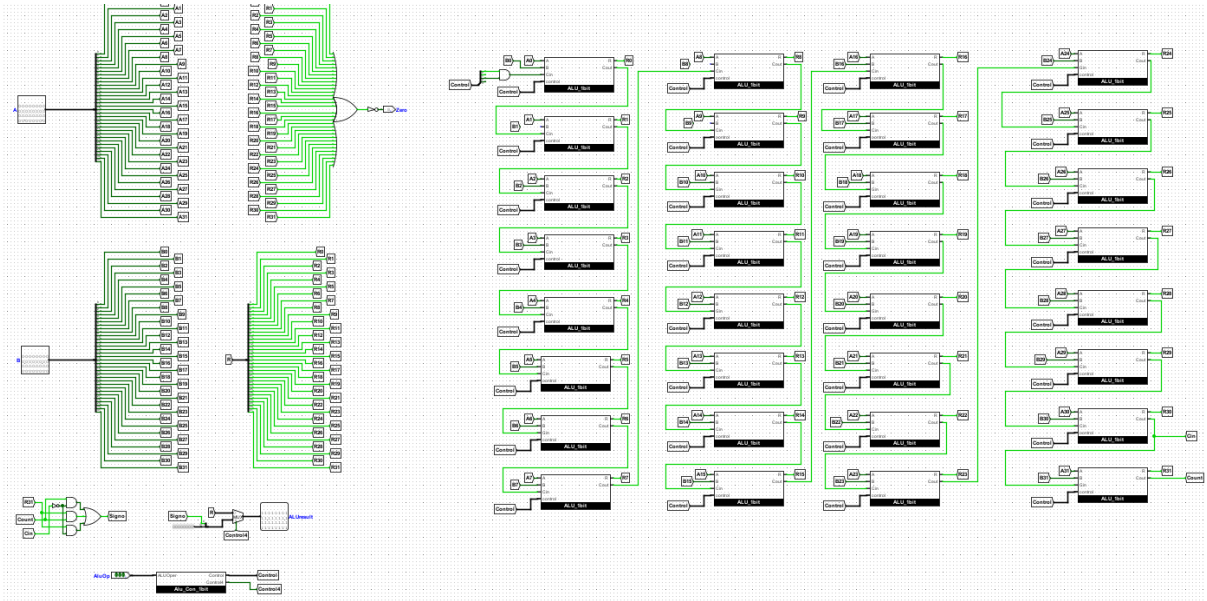
Para el proceso de construcción de la ALU, fue necesario crear el componente de Sumador completo de 1 bit



Con este sumador se procedió a crear una ALU de 1bit, con entradas A y B de un bit y un Cin adicional, para poder tener un acarreo de entrada; una salida Cout con el acarreo de salida y un registro R de salida.



Luego de creados estos componentes, se procedió a crear la ALU



El componente “ALU_CO” contiene el circuito con la entrada de función y la salida Jr y ALUOper de acuerdo a la especificación del taller

Función	ALU Operation
AND	010
OR	011
NOR	111
Suma	101
Resta	001
SLT (*Set on less than*)	100

El componente Alu_Con_Op contiene el circuito creado bajo la especificación pedida en el taller

Codificación #2

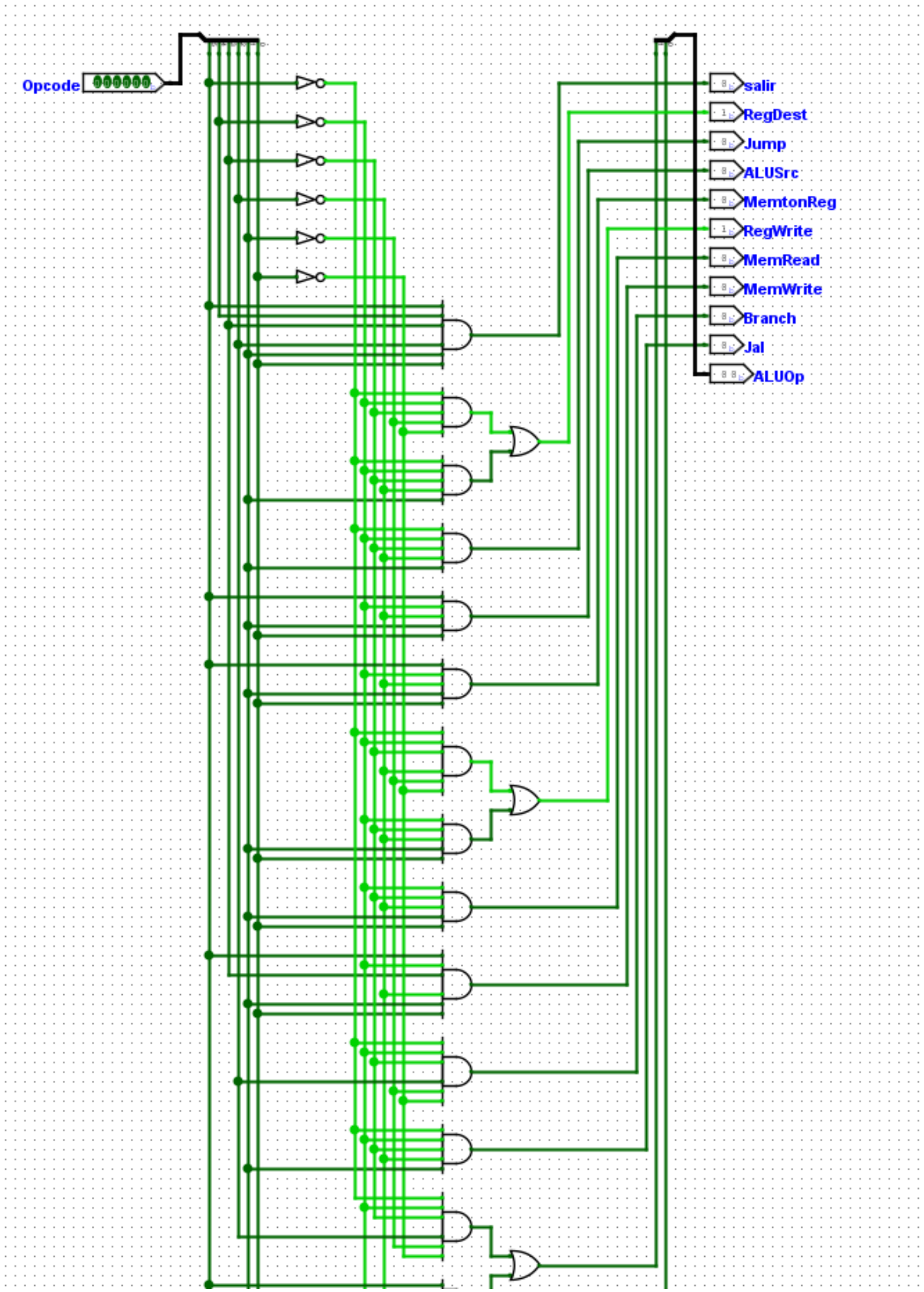
Opcode	ALUOp
lw, sw	11
beq	10
Tipo R	00

Haciendo la modificación de la Función de la unidad de control que se encuentran en la diapositiva de la clase 24. Con las modificaciones, se elaboró el circuito con las salidad apropiadas de ALUOp

Función de la unidad de control

Sus entradas son el campo *opcode* de la instrucción (`Op5..Op0`) y sus salidas son las señales de control

Input or output	Signal name	R-format	lw	sw	beq
Inputs	Op5	0	1	1	0
	Op4	0	0	0	0
	Op3	0	0	1	0
	Op2	0	0	0	1
	Op1	0	1	1	0
	Op0	0	1	1	0
Outputs	RegDst	1	0	X	X
	ALUSrc	0	1	1	0
	MemtoReg	0	1	X	X
	RegWrite	1	1	0	0
	MemRead	0	1	0	0
	MemWrite	0	0	1	0
	Branch	0	0	0	1
	ALUOp1	1	0	0	0
	ALUOp0	0	0	0	1


Opcode
000000
salir
RegDest
Jump
ALUSrc
MemtonReg
RegWrite
MemRead
MemWrite
Branch
Jal
ALUOp

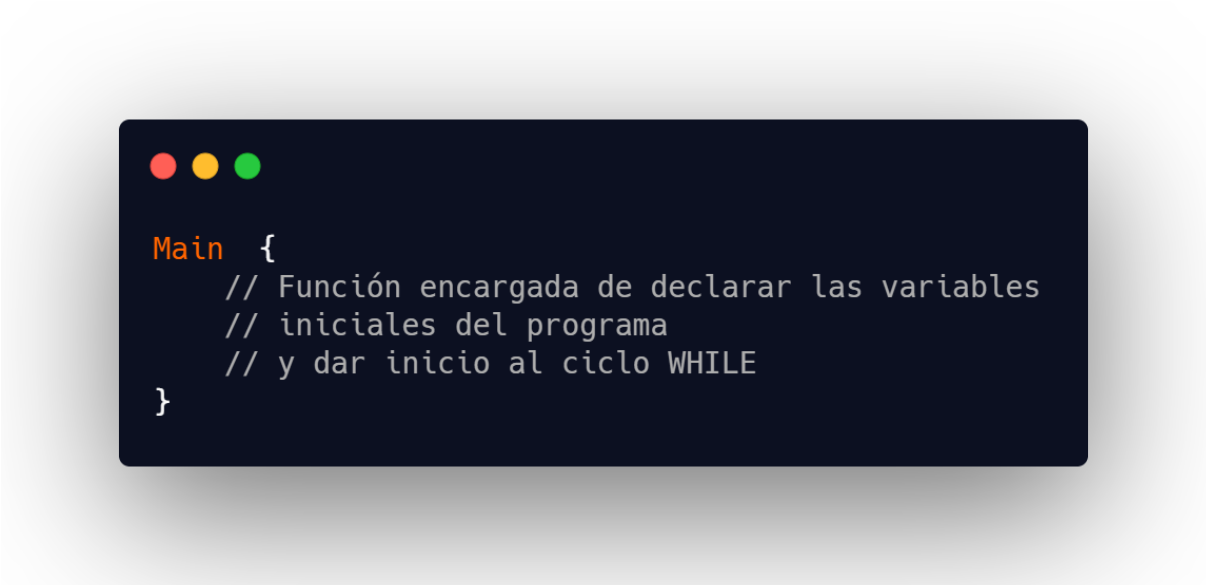
4. Programas de pruebas

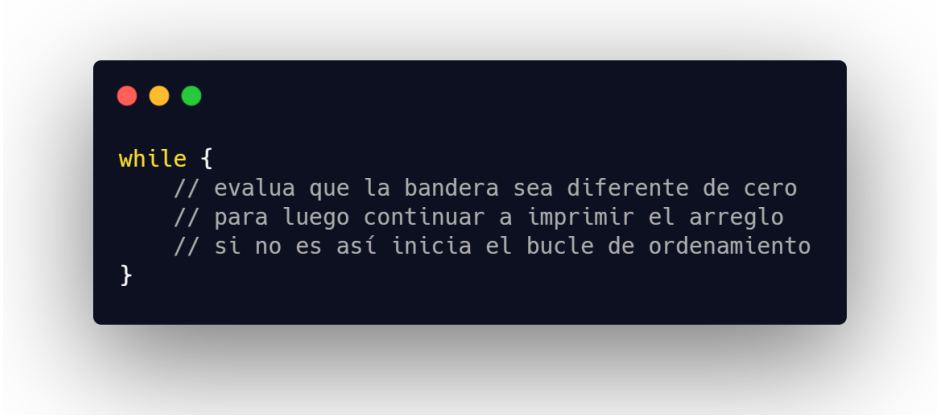
El programa de prueba cumple con los siguientes requisitos:

- Programación basada en el uso de procedimientos. El código que resuelve el problema se estructuró, se emplearon las instrucciones jal y jr
- Se sigue la convención para el uso de registros MIPS

"El programa debe tomar un vector de cien números enteros que contiene valores entre -100 y 100, y ordenarlo de manera ascendente, indicando cuántos elementos son negativos y cuántos son positivos."

Pseudocódigo:





```
for {
    // Iterador que se encarga de recorrer el arreglo
    // compara la posición actual con la siguiente
    // e intercambia las posiciones si la siguiente es menor
    // al valor de la posición actual.
}
```

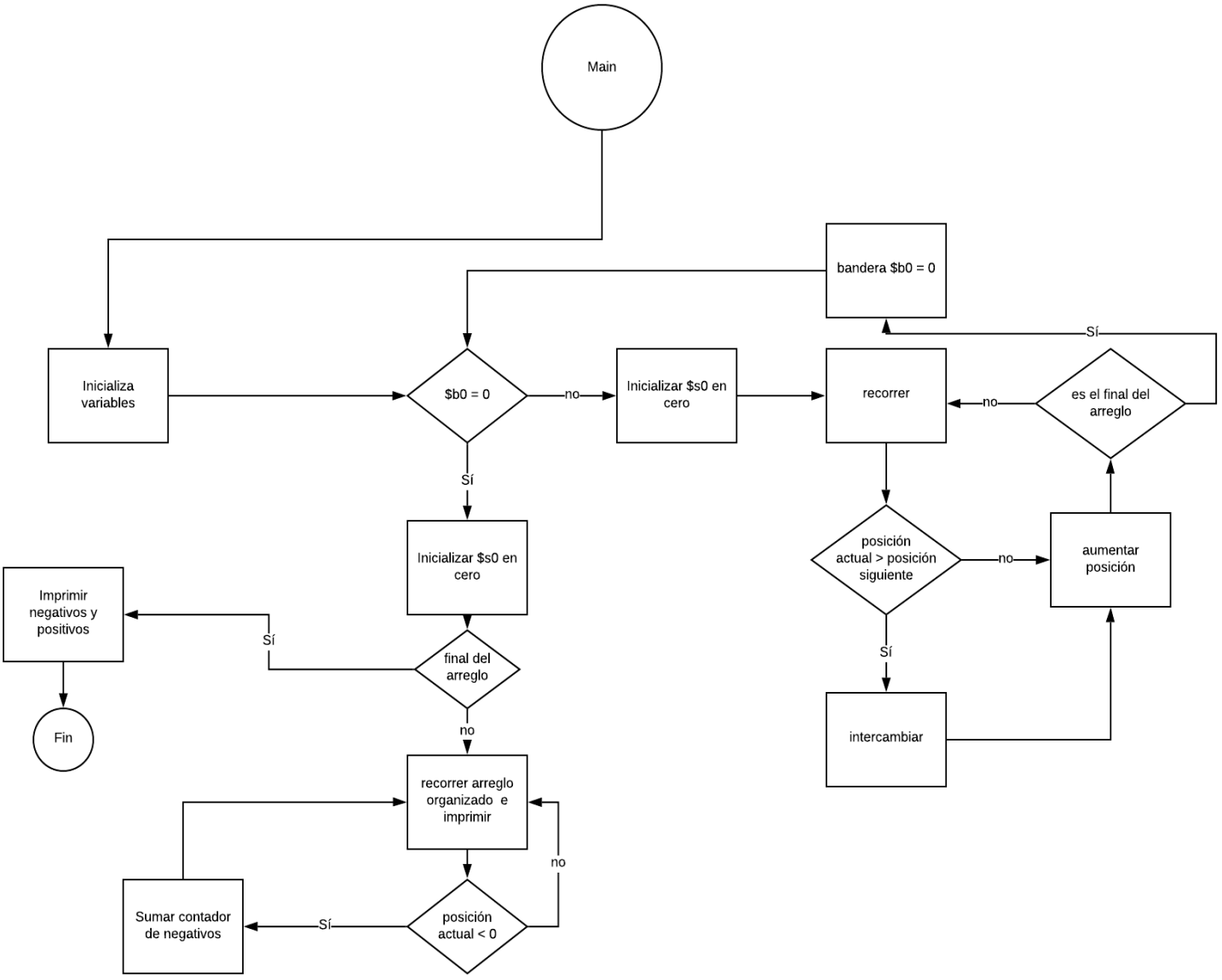
```
print {
   // su función es asignar a $s2 0
}

forPrint {
  // Iterador que se encarga de recorrer el vector organizado
  // mostrarlo en consola y contar los valores negativos
}

isNegative {
  // Cada que se notifique que existe un valor negativo
  // se aumenta 1 unidad a la variable contadora.
}

fin {
  // Se encarga de mostrar los valores negativos
  // mostrar los valores positivos
  // finalizar el proceso.
}
```

- **Diagrama de flujo**



- **Código del problema**

```
        .data
vec:    .word 53, 78, -51, 28, -38, 96, 30, 80, 89, 86, -31, -32, 82, -70, 5, 70, -69, -75, 96, -65, -67,
-54, 40, 19, -87, -45, 35, -89, -78, 18, -53, 100, 10, 47, -8, -70, -82, 46, 97, -85, 43, 5, -74, -53, 95,
-30, 4, 100, -20, 62, 68, -9, 17, -27, 55, -98, 12, -70, 14, 4, 97, 100, -97, 63, 96, -1, -30, 78, 6, 38, -24,
-79, -47, -88, -57, 14, -44, 3, 70, -76, -68, 46, 95, -66, 83, -66, 90, -61, -2, 99, -61, -48, 39, -80, 27,
-16, -52, -23, -75, 40

guion:   .asciiz ", "                   # line tomara un guion separador
negatives: .asciiz                      "\n La cantidad de números negativos en Vector es: "
positives: .asciiz                      "\n La cantidad de números positivos en Vector es: "

.text
.globl main

main:
        li $s1 1                        # carga encendido de bandera
        li $s3 100                      # fin del ciclo for
        li $s0 0                        # funcionara como bandera
        la $t1 vec                      # dirección base vec
        addi $t8, $zero, 0              # contador de negativos
        addi $sp,$sp, -4
        sw $s7, 0($sp)
        j while

while:
        bne $s0 $zero print
        move $s0 $s1
        li $s2 0                        # inicializador controlador ciclo for
        j for

for:
        sll $t3 $s2 2                   # avance 4 bits en espacio de memoria
        add $t3 $t3 $t1
        lw $t6 0($t3)                   # vec[i]

        add $s4 $s2 $s1
        sll $t4 $s4 2                   # avance 4 bits en espacio de memoria
        add $t4 $t4 $t1
        lw $t7 0($t4)                   # vec[i+1]

        add $s2 $s2 $s1                 # incremento controlador del ciclo
        beq $s2 $s3 while

        sle $t0 $t6 $t7

        bne $t0 $zero for
        sw $t6 0($t4)
        sw $t7 0($t3)
        move $s0 $zero
        j for

print:
```

```
        li $s2 0

forPrint:
        beq $s2 $s3 fin
        sll $t5 $s2 2
        add $t5 $t5 $t1
        lw $t7 0($t5)

        move $a0 $t7
        li $v0 1                        # carga $v0 a 1: muestra un entero
        syscall

        la $a0 guion                    # carga separador
        li $v0 4                        # carga $v0 a 4: permite mostrar una cadena
        syscall

        addi $t9,$zero,0
        slt $t9,$t7,$zero               # comparo si el número almacenado en t3 es menor que cero
        bne $t9,$zero, isNegative

        add $s2 $s2 $s1
        j forPrint

isNegative:
        add $s2 $s2 $s1
        addi $t8,$t8,1                  # contador de negativos aumenta
        j  forPrint

fin:
        addi $s7, $v0, 0                # cantidad numeros negativos
        li $v0,4
        la $a0,  negatives
        syscall

        li $v0,1
        add $a0,$zero,$t8
        syscall

        addi $t5, $a0, 0
        addi $t2, $zero, 100
        sub $t6, $t2, $t5

        li $v0, 4
        la $a0,  positives
        syscall

        li $v0,1
        add $a0, $t6, $zero
        syscall

        lw $s7, 0($sp)
        addi $sp,$sp,4
        addi $v0,$t8,0

        li $v0 10
```

```
    syscall                     # salir
```

5. Conclusiones

Con lo aprendido en clases sobre "**Diseño del procesador monociclo"** se logró llevar a cabo satisfactoriamente la implementación de esta práctica, siendo conscientes de la importancia que en algún momento tuvo este lenguaje de bajo nivel para el uso de las tecnologías que existen actualmente. Se facilitó el desarrollo gracias al uso de la plataforma MARS y LOGISIM.

Se presentaron dificultades, debido a la poca documentación del lenguaje ensamblador MIPS, y confusiones acerca del enunciado del ejercicio, pero sin embargo, algunas de las dudas que surgieron fueron resultas en el transcurso de las clases.

Finalmente, para nosotros fue un reto interesante, ya que puso a prueba nuestra lógica. Agradecemos la respuesta oportuna a nuestras inquietudes y el acompañamiento en el transcurso de la práctica.

Con el desarrollo de esta práctica, se pudo comprender un poco más el funcionamiento de un procesador al momento de ejecutar una instrucción o un sin número de instrucciones con el fin ejecutar algún algoritmo o programa.