Imagen que contiene Icono

Descripción generada automáticamenteLogotipo

Descripción generada automáticamente con confianza media

Instituto Politécnico Nacional

Escuela Superior de Cómputo

Arquitectura de Computadoras

**“Implementación del Monociclo”**

Alumno:

Malagón Baeza Alan Adrian

Profesor:

Alemán Arce Miguel Ángel

Grupo: 5CV1

**Introducción**

**Objetivo:** Implementar el procesador Monociclo siguiendo los siguientes pasos:

1. Localizar el archivo monociclo.qar
2. Analizar el código.
3. Compilar.
4. Obtener el RTL
5. Descargar el Archivo código de burbuja.asm incluido en la misma carpeta de material de clase.
6. Ensamblar en RARS.
7. Correr el programa y analizar su funcionamiento.
8. Generar el archivo .HEX posteriormente guardar el archivo en la carpeta de la memoria de instrucciones, icahe, y simular.
9. Examinar el correcto funcionamiento del programa en el monociclo.
10. Cambiar los números que se ordenan en el programa que ordena mediante el código de la burbuja.
11. Repetir los pasos del 6 al 9.
12. Escribir un programa en ensamblador en RARS revisando las instrucciones que fueron implementadas.
13. Repetir los pasos del 6 al 9.

**Desarrollo**

* **Descripción RTL obtenida mediante Vivado 2022.2**

Escala de tiempo

Descripción generada automáticamente con confianza baja

Diagrama

Descripción generada automáticamente Diagrama

Descripción generada automáticamente Diagrama, Esquemático

Descripción generada automáticamente

Diagrama, Esquemático

Descripción generada automáticamente

Diagrama

Descripción generada automáticamente Diagrama, Esquemático

Descripción generada automáticamente Diagrama

Descripción generada automáticamente Diagrama

Descripción generada automáticamente

Diagrama

Descripción generada automáticamente

Podemos observar cómo utilizo FlipsFlops, multiplexores, operadores de suma, resta, no es igual, desplazamiento a la izquierda/derecha, menor que y mayor que, compuertas AND, OR, XOR y bloques de memoria ROM/RAM del FPGA seleccionado, y para el control del dato de salida, se implementó mediante un multiplexor.

* **Código de burbuja**

Tabla

Descripción generada automáticamente

* **Ensamblando el programa**

Interfaz de usuario gráfica, Tabla

Descripción generada automáticamente

* **Resultado de la simulación en RARS**

Interfaz de usuario gráfica, Tabla

Descripción generada automáticamente

Tabla

Descripción generada automáticamente

El programa es una implementación del algoritmo de ordenamiento de burbuja en lenguaje ensamblador RISC-V. A continuación, se analiza su funcionamiento paso a paso:

1. Cargar datos: Se cargan los valores iniciales en los registros `t0` a `t6`.

2. Cargar datos en memoria: Los valores se almacenan en la memoria a partir de la dirección `0(x0)` en incrementos de 4 bytes (ya que son enteros).

3. Bucle externo (`for\_i`): El registro `s1` se inicializa en 0 y representa el índice `i` del bucle externo. Este bucle se ejecutará `N` veces (en este caso, `N` es 6).

4. Bucle interno (`for\_j`): El registro `s0` se inicializa en 0 y representa el índice `j` del bucle interno. Este bucle se ejecutará `N` veces (`N` es 6).

5. Comparación y intercambio: Se cargan los valores de `Mem[j]` y `Mem[j+1]` de la memoria en los registros `t2` y `t3`, respectivamente. Luego, se compara si `t3` es menor que `t2` utilizando la instrucción `slt` (set less than). Si la comparación es verdadera, se intercambian los valores en la memoria.

6. Fin del bucle interno: Se incrementa `s0` en 1 (para pasar al siguiente valor de `j`) y se verifica si se ha alcanzado el límite (`s2`). Si no se ha alcanzado el límite, se repite el bucle interno (`for\_j`).

7. Fin del bucle externo: Se incrementa `s1` en 1 (para pasar al siguiente valor de `i`) y se verifica si se ha alcanzado el límite (`s2`). Si no se ha alcanzado el límite, se repite el bucle externo (`for\_i`).

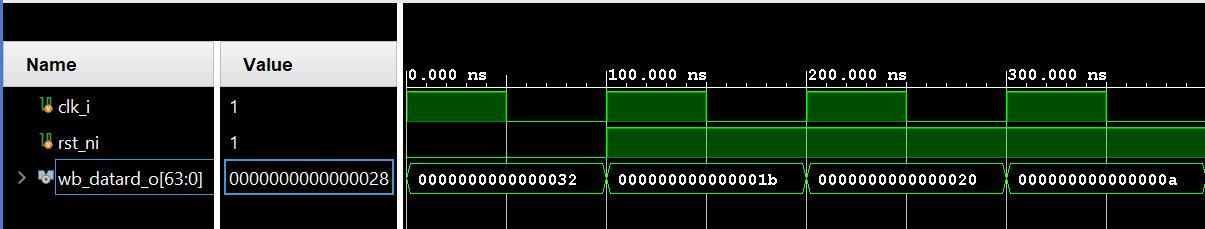
En resumen, el programa utiliza dos bucles anidados para comparar y ordenar los valores almacenados en memoria utilizando el algoritmo de ordenamiento de burbuja. El bucle externo controla el índice `i` y el bucle interno controla el índice `j`, realizando las comparaciones y los intercambios necesarios para ordenar los valores de manera ascendente.

* **Generando archivo .HEX**

Interfaz de usuario gráfica, Aplicación, Word

Descripción generada automáticamente

* **Resultado de la simulación en Vivado**

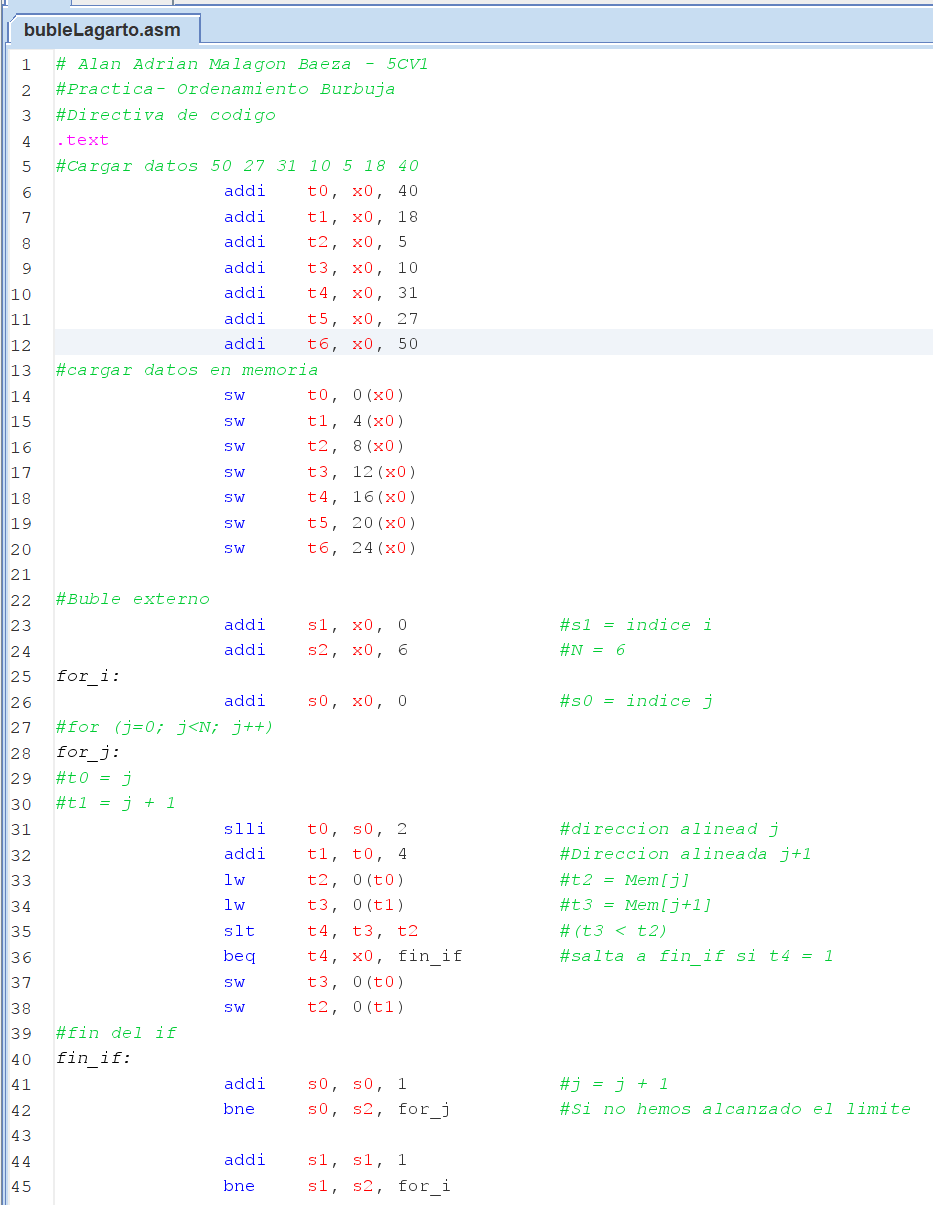


Gráfico, Diagrama

Descripción generada automáticamente

|  |  |  |
| --- | --- | --- |
| clk\_i | 1 |  |
| rst\_ni | 0 |  |
| FETCH - IF | |  |
| pcb\_pcsrc\_o | 0000000000000000000000000000000000000000000000000000000000000100 |  |
| pc\_next | 0000000000000000000000000000000000000000000000000000000000000000 |  |
| pc\_gen | 0000000000000000000000000000000000000000000000000000000000000100 |  |
|  |  |  |
| CACHE DE INSTRUCCIONES - IL1 | |  |
| icache\_addr\_i | pc\_next |  |
| we\_i | 0 |  |
| icache\_req\_i | 00000000000000000000000000000000 | hex mem |
| icache\_inst\_o | 0000000000000000000000000000000000000011001000000000001010010011 | 3200293 |
|  |  |  |
| DECODIFICACION/CONTROL - ID | |  |
| id\_opcode\_w | 0010011 | INSTRUCCIÓN TIPO I |
| id\_addrd\_w | 00101 |  |
| id\_aluop\_w | 0000 |  |
| id\_addrs1\_w | 00000 |  |
| id\_addrs2\_w | 10010 |  |
| id\_alusrc\_o | 1 |  |
| id\_regwrite\_o | 1 |  |
| id\_memread\_o | 0 |  |
| id\_memwrite\_o | 0 |  |
| id\_branch\_o | 0 |  |
| id\_memtoreg\_o | 0 |  |
| id\_aluop\_o | 010 |  |
| id\_pcregwrite\_o | 0 |  |
|  |  |  |
| BUSQUEDA DE OPERANDOS/LECTURAS DE REGISTROS - RR | |  |
| rers1\_i | 1 | HAB DE LECTURA |
| raddrrs1\_i | id\_addrs1\_w | DIR DE LECTURA |
| rdatars1\_o | 0000000000000000000000000000000000000000000000000000000000000000 | DATO DE LECTURA |
| rers2\_i | 1 | HAB DE LECTURA |
| raddrrs2\_i | id\_addrs2\_w | DIR DE LECTURA |
| rdatars2\_o | XXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXX | DATO DE LECTURA |
| we\_i | id\_regwrite\_o | HAB DE ESCRITURA |
| waddr\_i | id\_addrd\_w | DIR DE ESCRITURA |
| wb\_datard\_o | 0000000000000000000000000000000000000000000000000000000000110010 | DATO DE ESCRITURA |
|  |  |  |
| EXTENSION DE SIGNO - SE | |  |
| if\_inst\_i | icache\_inst\_o |  |
| se\_datars2\_o | 0000000000000000000000000000000000000000000000000000000000110010 | 32 |

* **Código de burbuja modificado**

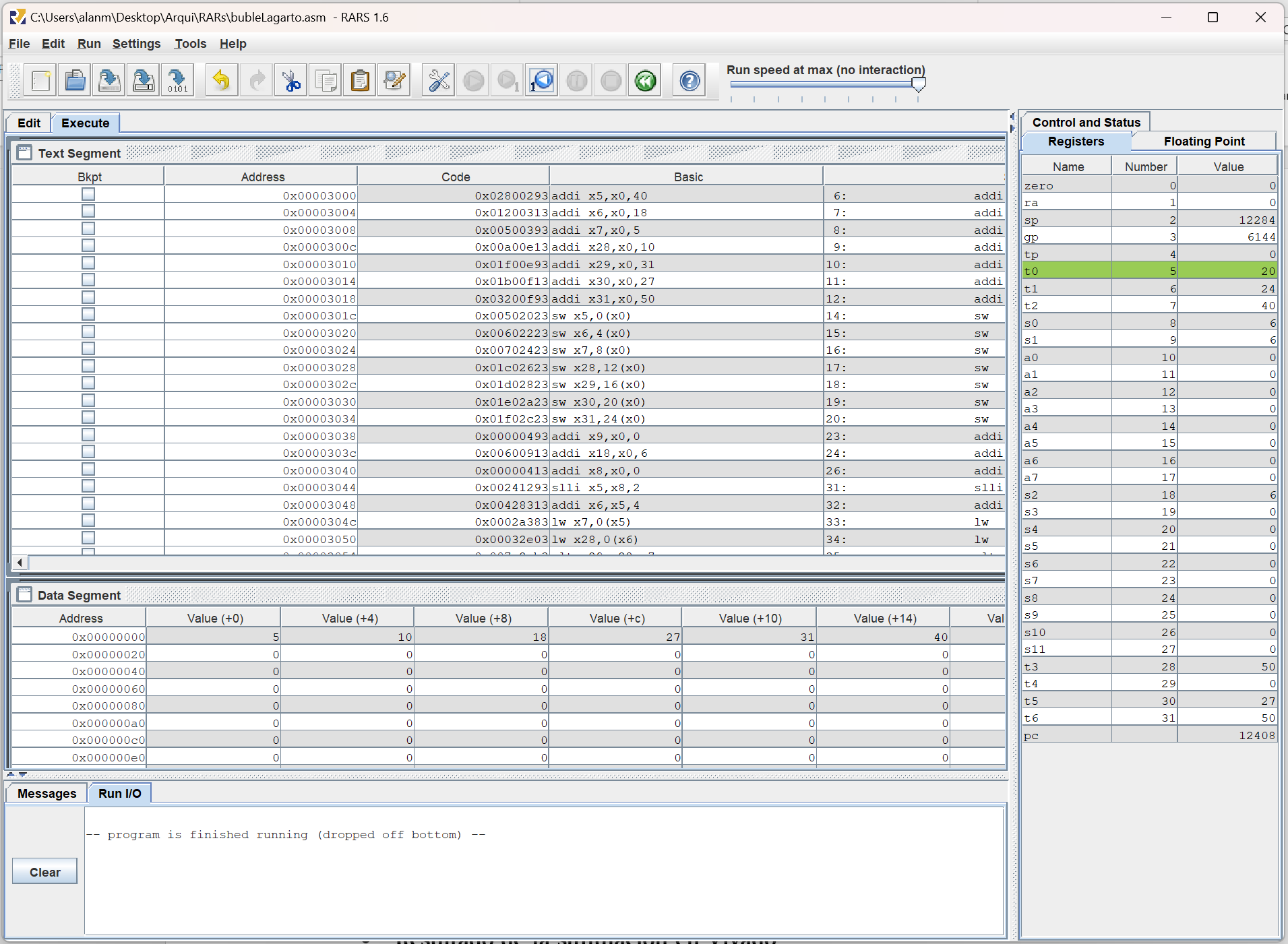


* **Ensamblando el programa**

Interfaz de usuario gráfica, Aplicación, Tabla, Excel

Descripción generada automáticamente

* **Resultado de la simulación en RARS**

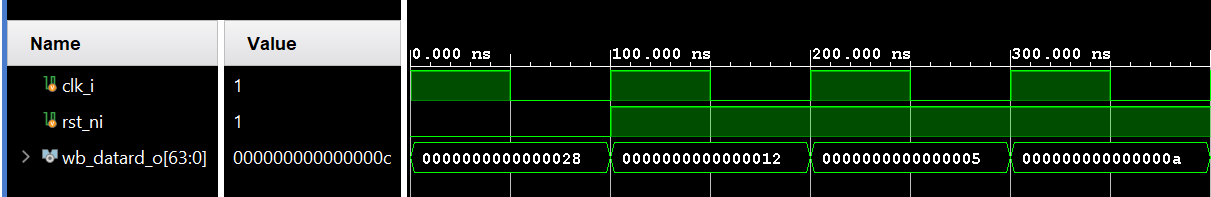


* **Generando archivo .HEX**

Interfaz de usuario gráfica

Descripción generada automáticamente

* **Resultado de la simulación en Vivado**



Interfaz de usuario gráfica, Sitio web

Descripción generada automáticamenteDiagrama

Descripción generada automáticamente

* **Código propuesto: Ordenamiento por Inserción**

Texto

Descripción generada automáticamente

Texto

Descripción generada automáticamente

Se encuentran las instrucciones para cargar los datos en registros temporales (t0-t6). En este caso, se cargan los valores 50, 27, 31, 10, 5, 18 y 40. Estos valores pueden modificarse según sea necesario.

Después de cargar los datos en los registros temporales, se utilizan las instrucciones "sw" para almacenar los valores en la memoria. Cada valor se almacena en una ubicación de memoria específica utilizando un desplazamiento relativo a la dirección base (x0).

Una vez que los datos se han cargado y almacenado en memoria, comienza el algoritmo de ordenamiento por inserción.

El algoritmo utiliza dos bucles "for" anidados para iterar a través de los elementos del arreglo y realizar las comparaciones necesarias para ordenarlos.

El bucle externo "for\_i" se encarga de iterar sobre todos los elementos del arreglo. Se utiliza una variable de índice "s1" para rastrear la posición actual del elemento que se está comparando.

Dentro del bucle externo, se inicializa la variable de índice "s0" en cero para el bucle interno "for\_j". El bucle "for\_j" se encarga de buscar la posición correcta para el elemento actual dentro de la porción ya ordenada del arreglo.

Dentro del bucle "for\_j", se cargan los valores del elemento actual (t2) y el elemento anterior (t3) en registros temporales. Se realiza una comparación (t3 < t2) para determinar si el elemento anterior es menor al elemento actual.

Si la comparación es verdadera, se ejecuta un bloque de código donde se intercambian los valores de los elementos. Primero, se almacena el valor del elemento anterior (t3) en la posición del elemento actual (Mem[i]). Luego, se decrementan las direcciones de memoria para comparar el siguiente par de elementos en el siguiente ciclo del bucle "for\_j".

Si la comparación es falsa, el bucle "for\_j" se salta el bloque de código de intercambio y pasa al siguiente par de elementos.

Una vez que el bucle "for\_j" ha terminado de iterar sobre todos los elementos en la porción ya ordenada del arreglo, se almacena el valor del elemento actual (t2) en la posición correcta dentro del arreglo.

Después de eso, se incrementa el índice "s1" en el bucle externo "for\_i" para pasar al siguiente elemento del arreglo y repetir el proceso.

El resultado final será un arreglo ordenado almacenado en memoria.

* **Ensamblando el programa**

Interfaz de usuario gráfica, Aplicación, Tabla, Excel

Descripción generada automáticamente

* **Resultado de la simulación en RARS**

Interfaz de usuario gráfica, Aplicación, Tabla

Descripción generada automáticamente

* **Generando archivo .HEX**

Interfaz de usuario gráfica, Aplicación

Descripción generada automáticamente

* **Resultado de la simulación en Vivado**

Interfaz de usuario gráfica, Aplicación

Descripción generada automáticamenteDiagrama

Descripción generada automáticamenteDiagrama

Descripción generada automáticamente

**Conclusión**

En esta actividad, se implementó y probó un procesador monociclo siguiendo una serie de pasos específicos. Durante el desarrollo del proyecto, se analizó el código fuente, se compiló y se obtuvo el RTL del procesador. Además, se ensambló un programa utilizando RARS, se ejecutó y se examinó su funcionamiento.

El proceso de implementación y prueba del procesador monociclo proporcionó una valiosa experiencia práctica en el diseño y funcionamiento de los procesadores. Se adquirieron conocimientos sobre el proceso de compilación, ensamblaje y simulación, así como sobre la interacción del procesador con la memoria de instrucciones.

Al modificar el programa para ordenar diferentes números utilizando el algoritmo de ordenamiento de burbuja, se pudo observar cómo las instrucciones del procesador afectan el resultado final y el rendimiento del programa. Esta experiencia también permitió comprender la importancia de la optimización de algoritmos y cómo las instrucciones del procesador pueden influir en ello.

En resumen, la implementación y prueba del procesador monociclo a través de los pasos propuestos proporcionó una comprensión más profunda de los conceptos teóricos relacionados con los procesadores y su funcionamiento. Esta actividad práctica permitió aplicar los conocimientos adquiridos, adquirir habilidades en el uso de herramientas de desarrollo y simulación, y obtener una visión más completa del diseño y rendimiento de los procesadores.

En conclusión, este proyecto fue una oportunidad enriquecedora para aprender y experimentar con el diseño y funcionamiento de un procesador monociclo. Los conocimientos adquiridos y las habilidades desarrolladas durante este proceso son valiosos para comprender y trabajar en el campo de la arquitectura de procesadores y la programación de bajo nivel.

**Referencia**

1. Brock J. LaMeres, Introduction to Logic Circuits & Logic Design with Verilog, Springer, 1st Edition, USA, 2017.
2. TheThirdOne. (n.d.). GitHub - TheThirdOne/rars: RARS -- RISC-V Assembler and Runtime Simulator. GitHub. <https://github.com/TheThirdOne/rars>
3. GeeksforGeeks. (2023). Insertion Sort  Data Structure and Algorithm Tutorials. *GeeksforGeeks*. https://www.geeksforgeeks.org/insertion-sort/