

Guadalajara, Jalisco a 12 de mayo de 2023



**ITESO, Universidad
Jesuita de Guadalajara**

MATERIA: Arquitectura computacional

PROFESOR: Ibarra Esparza, Juanpablo

Práctica 2: RISC-V Single Cycle

Presentado por:

Sainz Ruvalcaba, Diego. Expediente: 727884

Tabla de contenidos

Cálculos	2
Diagrama propuesto	3
Modificaciones en los módulos.....	3
Pruebas.....	4
Torres de Hanoi	8

Cálculos

La frecuencia máxima que marca Quartus es de 39.32MHz. Otros datos que se obtienen son:

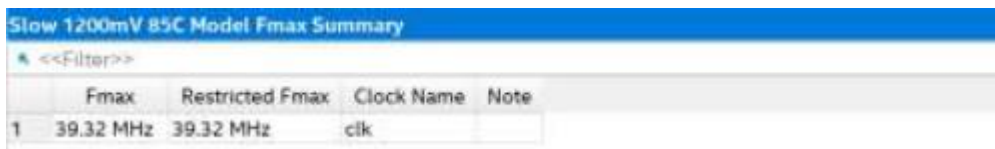
Instrucciones utilizadas (IC): 432

Ciclos por instrucción (CPI): 1

Velocidad de reloj (CLK): 39.32MHz

Tiempo de CPU = $IC \frac{CPI}{CLK} = 10.9867us$

$$CPU\ time: 432 * \frac{1}{39.32 \times 10^6} = 10.9867us$$



	Fmax	Restricted Fmax	Clock Name	Note
1	39.32 MHz	39.32 MHz	clk	

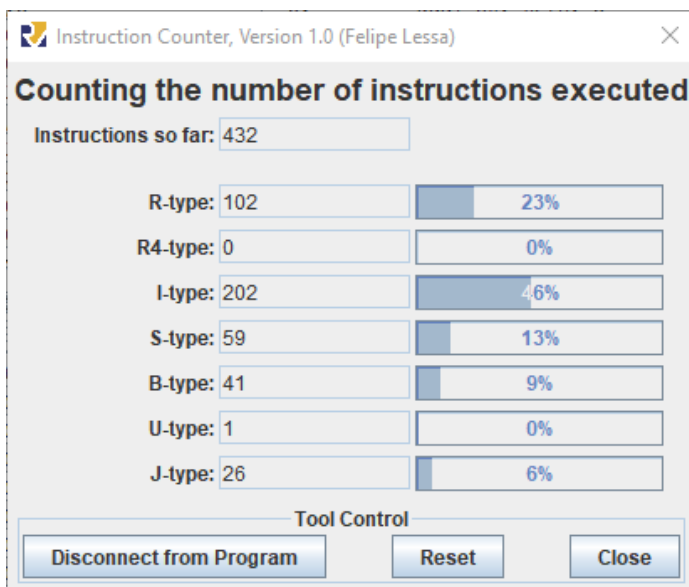
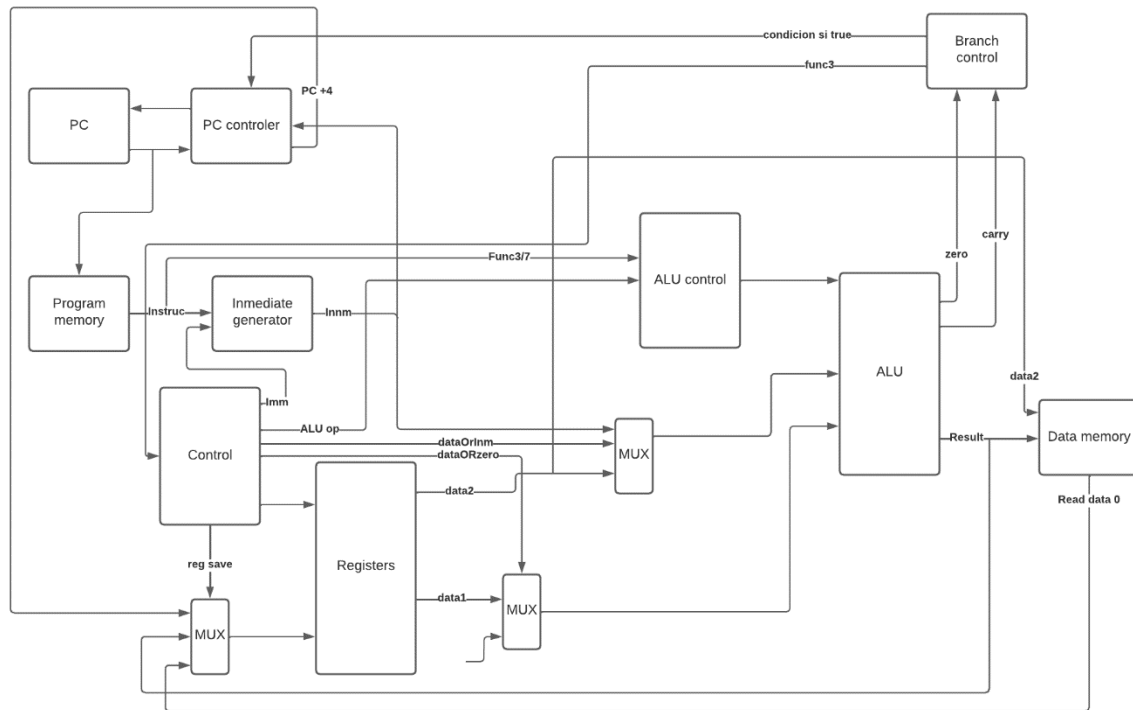


Diagrama propuesto



Modificaciones en los módulos

Generador de inmediatos: Este módulo se modificó porque existen múltiples opcodes para un mismo tipo de instrucción. Al realizar el cambio en el generador, el opcode puede ser decodificado por el controller y esto hace que el módulo de inmediatos sea más preciso.

Cambiar "pc plus 4" a un pc controller: Este cambio fue creado específicamente para poder juntar los jumps y los jals. Proponiendo un pc controller se obtienen diversos cambios importantes. El primero es una salida que cumple con la suma del pc actual mas 4. El segundo cambio, que es el mas importante, es que tiene una entrada que recibe un valor de registro y un inmediato para poder ser enviados al pc (si se da el caso) y ser el siguiente valor del pc en lugar de únicamente sumar 4. Al recibir también un inmediato, este último se puede agregar al pc en caso de que exista alguna condición que active una bandera como verdadera y que el control active una señal para indicar que la instrucción actual es un Branch.

Valor inicial del PC: Ya que era un requerimiento, se modificó el valor que tenía el PC en el momento de inicio para que en este último tuviera un valor de 0x400000

Agregar módulo "mux zero": El mux solamente sirve para que la Alu agregue un zero a un dato inmediato. Su funcionamiento es como lo dice su nombre, un mux que tiene como entradas el dato, al igual que un 0, y una salida que llega a la ALU. La decisión de la entrada la toma la unidad de control

Agregar módulo para Branch: Se creó un módulo para poder recibir operaciones tipo Branch. Este funciona recibiendo el func3 desde el bus de instrucciones junto con zero de la ALU y el carry. El dato func3 determina el tipo de salto que se hará y el zero y el carry determinan si se cumple la condición de salto. Con estas entradas se decide si se cumplen los requisitos de saltar o no. El módulo tiene una salida que se manda directamente al control del pc

Modificar Data memory: El primer cambio es que es iniciado con ciertos parámetros. Se intentó crear

Crear mux write to regs: Este es un multiplexor que es activado desde el control para cambiar el valor de tres posibles entradas: la salida del data memory, la salida de la ALU o del pc+4 (anteriormente mencionado que ya no es pc+4)

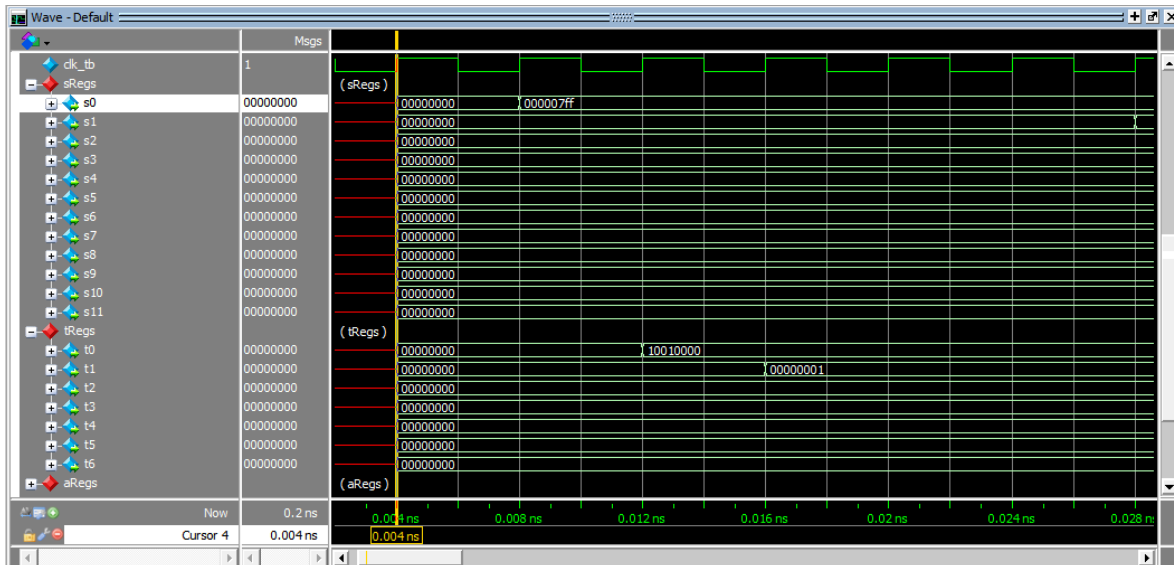
Agregar las instrucciones a control: Aquí se incluyeron las instrucciones que se solicitaron en la práctica junto con sus banderas. También se verificó las ya definidas para comprobar su correcto funcionamiento.

Pruebas

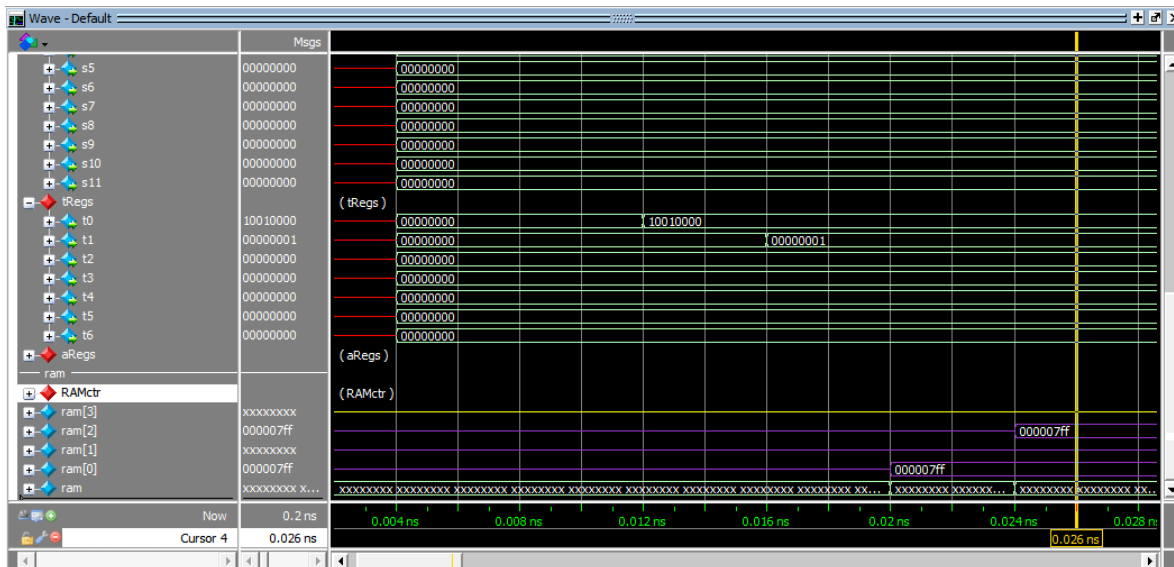
El código en ensamblador con el que se hicieron las pruebas es el siguiente:

```
prueba.asm  torresHanoi.asm
1  # Autor: Diego Sainz Ruvalcaba
2  # Fecha: 20/abr/23
3
4  .text
5  inicio:
6      addi s0, zero, 0x7ff
7      lui t0, 0x10010
8      addi t1, zero, 0x1
9
10     sw s0, 0(t0)
11     sw s0, 8(t0)
12
13     lw s1, 0(t0)
14     lw s2, 8(t0)
15     jal ra, funcion
16
17  pruebal:
18     beq zero, zero, prueba2
19     addi t1, zero, 2
20
21  prueba2:
22     bne zero, t0, pruebal
23     addi t1, zero, 1
24
25  funcion:
26     addi s0, s0, 2
27     jalr zero, ra, 0
```

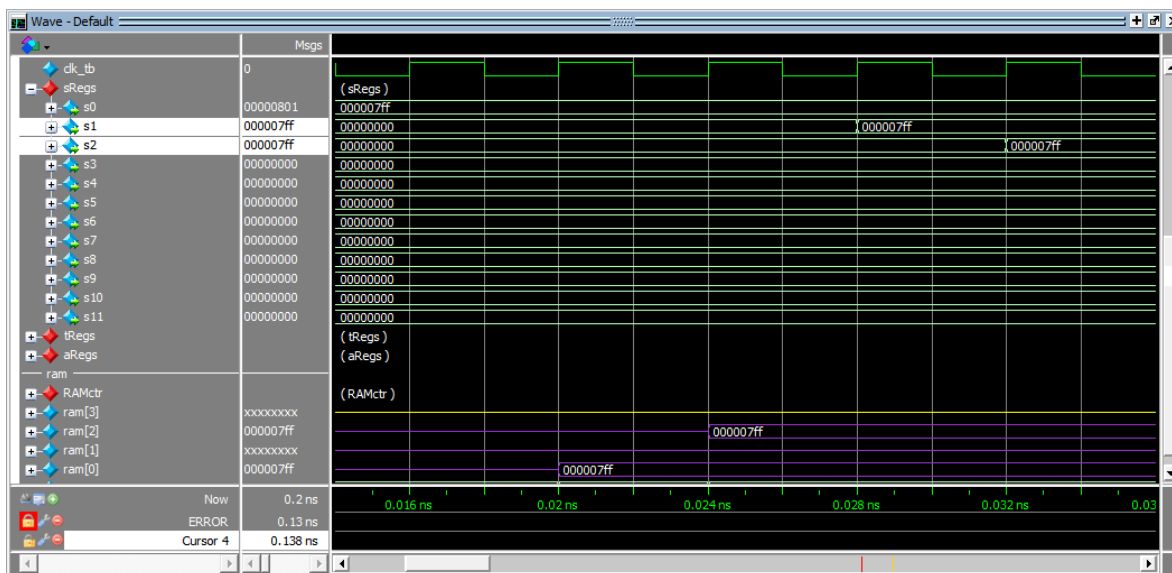
Primero verificamos que funcionen los registros. Se suman los valores correspondientes a cada registro como se marca en el código.



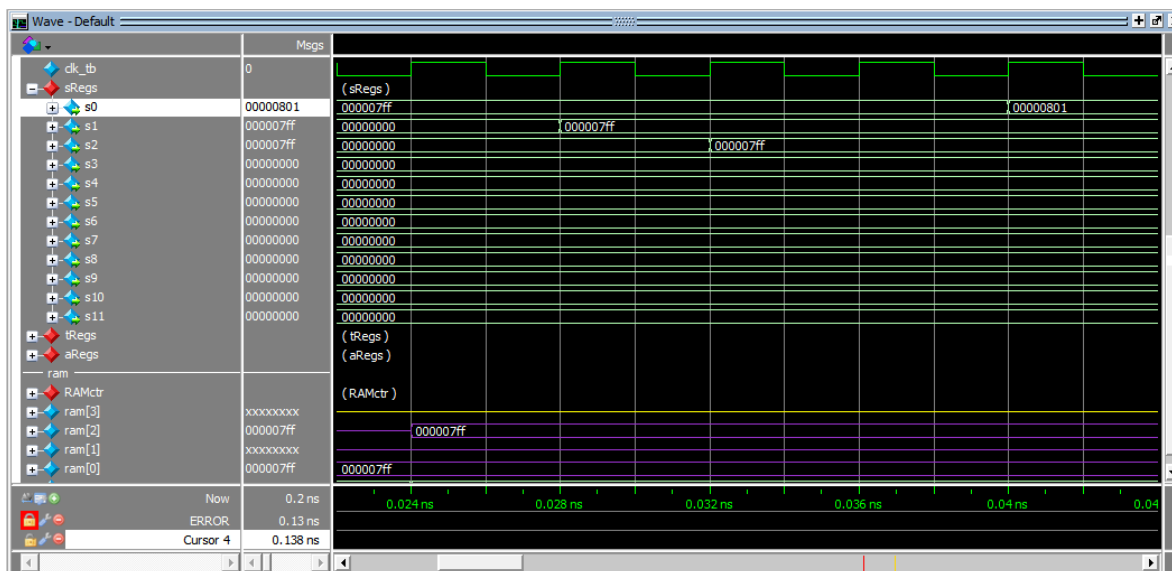
Vamos a memoria para verificar que funcione el “sw”. En este caso según el código se guardarán en ram 0 y ram 2.



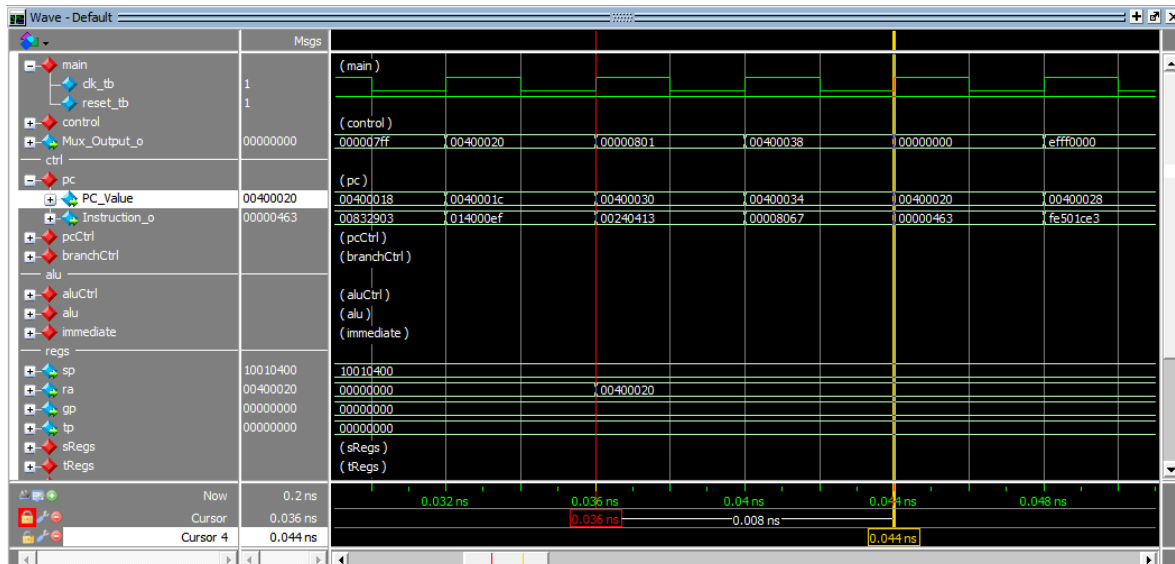
Ahora se asignan los valores que se acaban de almacenar a los registros s1 y s2



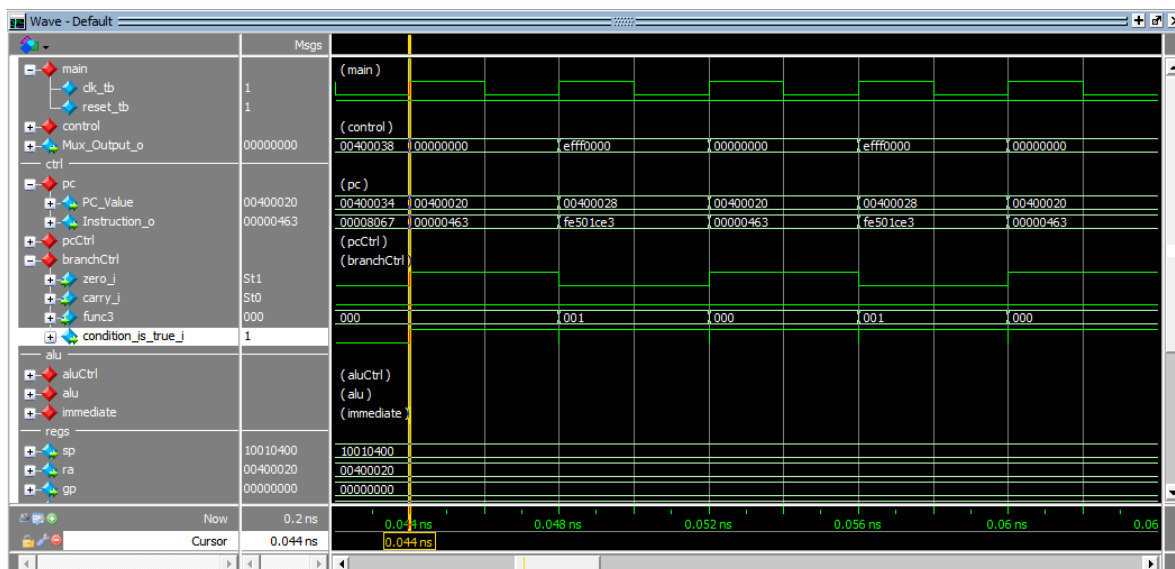
Se llama a un jal y va al *funcion* en el código donde se le agregará 2 a s0



Ahora se encuentra con un *jalr* que se verifica en la siguiente parte de la simulación por el comportamiento del pc. A los 0.034ns el pc salta a la dirección donde está *funcion* (0x400030), ejecuta el *addi* y el *jalr* donde regresa a donde estaba antes de saltar (0x40001c) pero sumado +4 (0x400020)



Lo siguiente es verificar que los branches funcionan debido a que entra en ciclo con un *bne* y un *bqe*. Esto se verifica mediante la unidad de control de Branch en su salida de si la condición para hacer el Branch es verdadera. Como en la simulación el valor nunca cambia, esto quiere decir que los saltos se están haciendo correctamente y la unidad funciona.



Torres de Hanoi

Conclusiones

El desarrollo de la práctica fue complicado, no en cuanto a la lógica o a entender lo que se tenía que hacer, sino a cómo implementarlo en Quartus. En un principio recibí consejo de unos amigos de dónde poner los multiplexores que faltaban, lo cual me llevó aún más problemas. Fueron tantos los problemas, sumados a la poca coordinación con mi equipo que me llevó a realizar la práctica casi desde cero pero ahora siguiendo únicamente las presentaciones y la lógica del diagrama de flujo. La implementación sigue teniendo detalles con la ejecución de las torres de Hanoi debido a que no hace un Branch correctamente, lo cual estoy intentando arreglar.

Respecto al RISCv puedo concluir que el que sea una implementación modular ahorra muchos problemas. Las señales de control y sus respectivos módulos son esenciales para el correcto funcionamiento de la arquitectura. Uno de sus puntos débiles es que hay ciertas instrucciones que tardan más ciclos de reloj que otras instrucciones.