

Laboratorio 3

Microarquitectura MIPS

Arquitectura de Ordenadores

1. Objetivos de Aprendizaje

- Construir un procesador monociclo sencillo capaz de ejecutar un programa en lenguaje ensamblador
- Entender cómo funciona el proceso de decodificación de instrucciones
- Entender los principios básicos de funcionamiento de las rutas de datos y de control

2. Desarrollo y calificación

Debe realizar los problemas propuestos de forma autónoma durante la sesión de laboratorio ya que el profesor sólo proporcionará ayuda y contestará a preguntas antes del comienzo del desarrollo del trabajo práctico. Para realizar de forma provechosa este laboratorio, se recomienda haber realizado el tutorial sobre diseño de microarquitectura MIPS con LogiSim y estar al día con la teoría sobre arquitectura del procesador MIPS. El profesor analizará sus habilidades y destrezas durante el desarrollo de esta práctica. Esta práctica se compone de dos partes: una de diseño y otra de análisis y razonamiento cuyos resultados deben ser transcritos en esta hoja.

3. Entrega de los resultados

Al finalizar el laboratorio hay que entregar al profesor a través de Blackboard la memoria en PDF y el esquemático LogiSim con el diseño que ha realizado. Suba los dos documentos por separado, no en un ZIP.

4. Microarquitectura MIPS

Durante todos los desarrollos testee cuidadosamente todos los componentes asignando valores arbitrarios a las entradas y comprobando que los valores de salida son los esperados.

Utilice el fichero *processor.circ* que se proporciona con este enunciado como patrón para sus desarrollos.

4.1 La unidad de control

Ahora deberá implementar parte de una pequeña unidad de control. Para resolver este problema de diseño es indispensable que haya llevado al día la asignatura y que, en su tiempo, haya estudiado y comprendido el funcionamiento del procesador MIPS y de las rutas de datos y de control. Observe que la Unidad de Control que hallará en el fichero *processor.circ* es una versión muy simplificada de la presentada en clase y solo puede manejar un número muy reducido de instrucciones. El diseño ya está empezado y contiene una implementación (correcta pero no óptima) que implementa las señales de control para las instrucciones **addi** y **add**. Estas instrucciones precisan que la ALU realice una suma (**AluOp** = 010).

Complete el diseño añadiendo los componentes que implementan las señales de control para la instrucción **beq**. Esta instrucción precisa que la ALU realice una resta (**AluOp** = 110).

- * -

1. Explique brevemente cómo ha implementado las señales de control para la **beq** justificando sus elecciones de diseño:

Para ver si la instrucción es **beq**, la señal que sale del selector debería ser 000100, por lo que pongo un comparador de igualdad con la constante 4.

Para la señal branch envío la señal resultante de $\sim(\text{es add} + \text{es addi}) * \text{es branch}$, por lo que, si la operación no es add o addi, la primera mitad (el NOR) sale positivo y el and con el branch, que pasa el bit 1, da de resultado 1, por lo que hay branch. En cualquier otro caso el and no se cumple y no hay branch.

Para la alucontrol hay 2 opciones: 010 para add y addi y 110 para el beq. Para escoger en cada caso paso ambos valores por un multiplexor 2:1, donde el selector es 1 si la instrucción es add o addi, y deja pasar el valor 2, y si no lo es deja pasar el 6.

- * -

4.2 La ruta de datos (Datapath)

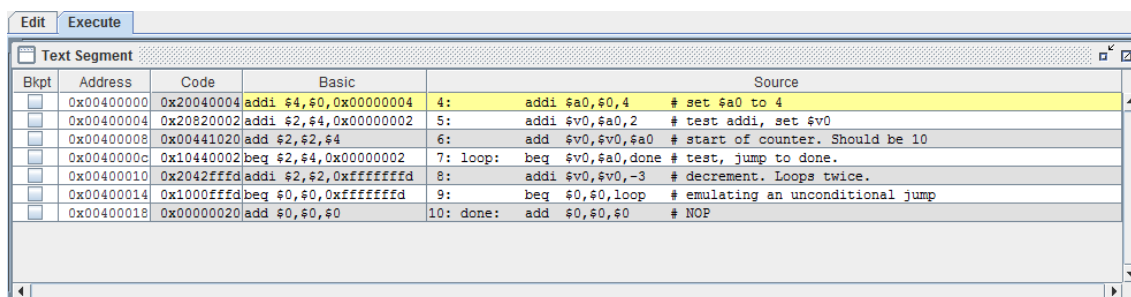
Ahora deberá implementar la ruta de datos capaz de ejecutar correctamente las tres instrucciones MIPS que ha implementado: **addi**, **add** y **beq**. Esta es la parte más complicada de esta práctica por lo que se aconseja seguir detenidamente las instrucciones y que verifique la correcta implementación de las instrucciones utilizando el programa de prueba (*test.s*) que se adjunta al enunciado.

4.4.1 Implementar la unidad de fetch del procesador

1. Abra el módulo Datapath que ya se halla en el fichero *processor.circ*. Deberá añadir a este módulo todos los componentes del procesador que faltan.
2. El módulo ya contiene el registro contador de programa (PC) y la conexión con la memoria de instrucciones. Añada la lógica para calcular el valor futuro del contador de programa (incrementado la dirección actual en +4). Observe que el contador de programa es un registro de 8 bits de manera que la memoria de programa no sea demasiado grande. Testee el circuito utilizando el reloj (puede utilizar la opción de simulación **conmutar reloj** para realizar una simulación paso a paso).
3. La memoria de programa deberá ser implementada utilizando el módulo ROM de **LogiSim** (disponible en la librería de **Memorias**). Este elemento ya se encuentra implementado en el esquemático. Observe que he añadido también un desplazador antes del puerto de direcciones de la ROM. El desplazador realiza un desplazamiento lógico de dos posiciones a la derecha para direccionar correctamente la ROM. Añada un puerto de salida y testee el circuito para asegurarse que la memoria es direccionada correctamente.

4.4.2 Implementar la **addi**

1. Abra el programa de prueba (*test.s*) con MARS y abra la pestaña EXECUTE. Debería obtener el resultado que se muestra en la siguiente figura.



Bkpt	Address	Code	Basic	Source
<input type="checkbox"/>	0x00400000	0x20040004	addi \$4,\$0,0x00000004	4: addi \$a0,\$0,4 # set \$a0 to 4
<input type="checkbox"/>	0x00400004	0x20820002	addi \$2,\$4,0x00000002	5: addi \$v0,\$a0,2 # test addi, set \$v0
<input type="checkbox"/>	0x00400008	0x00441020	add \$2,\$2,\$4	6: add \$v0,\$v0,\$a0 # start of counter. Should be 10
<input type="checkbox"/>	0x0040000c	0x10440002	beq \$2,\$4,0x00000002	7: loop: beq \$v0,\$a0,done # test, jump to done.
<input type="checkbox"/>	0x00400010	0x2042ffff	addi \$2,\$2,0xffffffff	8: addi \$v0,\$v0,-3 # decrement. Loops twice.
<input type="checkbox"/>	0x00400014	0x1000ffff	beq \$0,\$0,0xffffffff	9: beq \$0,\$0,loop # emulating an unconditional jump
<input type="checkbox"/>	0x00400018	0x00000020	add \$0,\$0,\$0	10: done: add \$0,\$0,\$0 # NOP

En la columna **Code** aparecen las traducciones en lenguaje máquina de las varias instrucciones.

2. Vaya a **LogiSim** y copie los códigos máquina de las primeras dos instrucciones en la ROM. Para añadir las instrucciones anteriores, antes deberá borrar el contenido de la ROM (seleccione el módulo ROM, click botón derecho del ratón, y escoja la opción “**borrar contenidos**”). A continuación, seleccione la opción “**editar contenidos**” e programe la ROM con las instrucciones anteriores tal y como se muestra en la figura siguiente.

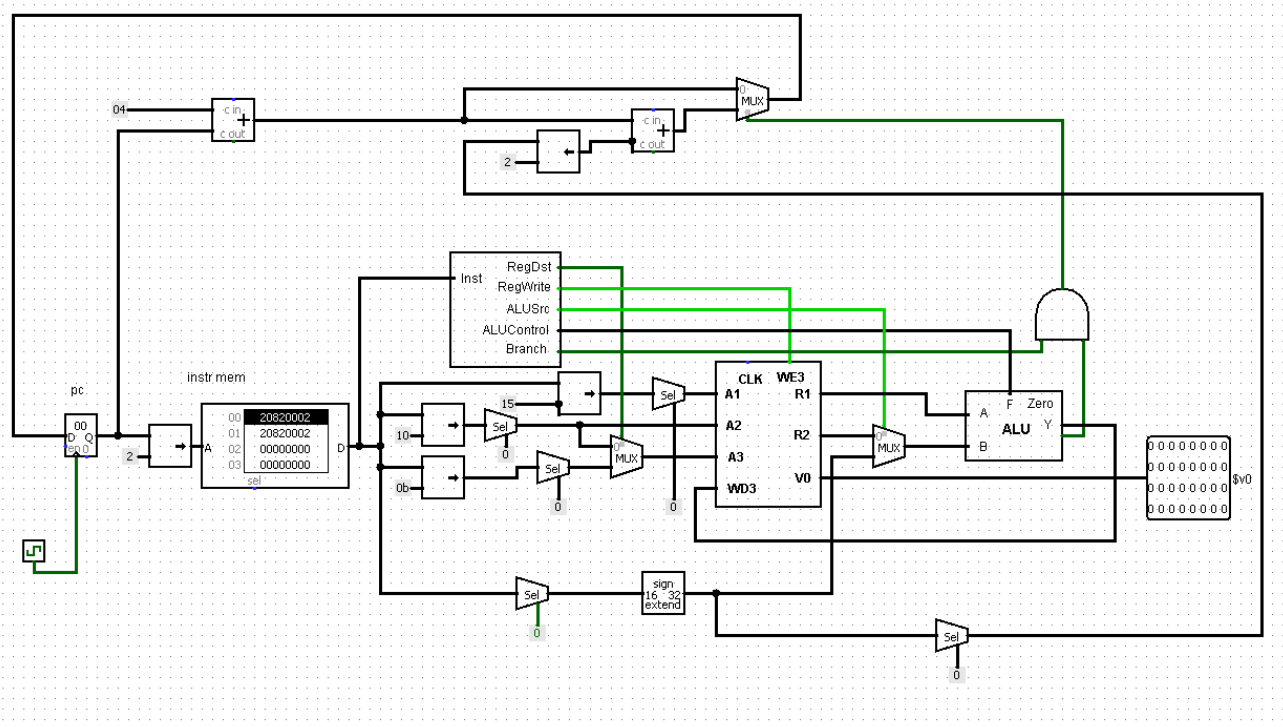
```

00 20040004 20820002 00000000 00000000 00000000
10 00000000 00000000 00000000 00000000 00000000
20 00000000 00000000 00000000 00000000 00000000
30 00000000 00000000 00000000 00000000 00000000
40 00000000 00000000 00000000 00000000 00000000
50 00000000 00000000 00000000 00000000 00000000
60 00000000 00000000 00000000 00000000 00000000
70 00000000 00000000 00000000 00000000 00000000
80 00000000 00000000 00000000 00000000 00000000
90 ~~~~~~ ~~~~~~ ~~~~~~ ~~~~~~ ~~~~~~

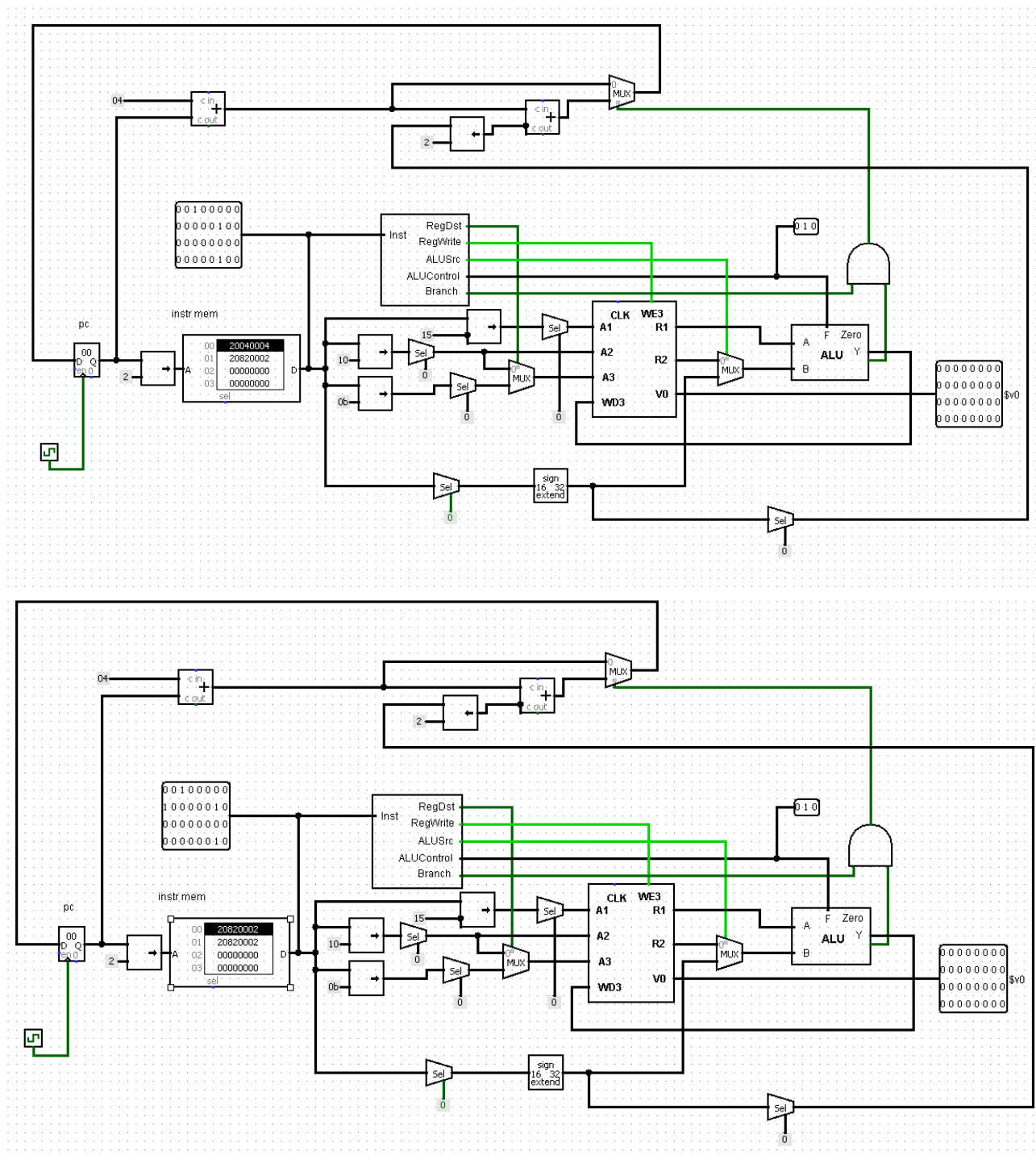
```

3. Añada al diseño el banco de registros y la ALU. Observe que en nuestro diseño simplificado el banco de registros sólo tiene 8 registros por lo que deberá utilizar sólo los tres bits menos significativos de los campos de registros de la instrucción para direccionarlo. Para seleccionar un grupo de bits puede utilizar el componente **Bit Selector** (en la librería de multiplexores). Para entender cómo funciona el componente **Bit Selector** puede mirar los ejemplos en el módulo **Tests** o abrir la “referencias de las librerías” en el menú de ayuda de la herramienta. Haga referencia también a la Figura con la arquitectura del MIPS que se adjunta al final del enunciado para entender qué bloques hay que añadir y cómo conectarlos.
4. Conecte el datapath a las señales de control. No olvide conectar el campo de **opcode** de las instrucciones en la unidad de control.
5. Testee la secuencia de dos instrucciones para verificar que el procesador funciona correctamente. Es probable que tenga que añadir algún puerto de salida para observar las señales.

Diseño entero:



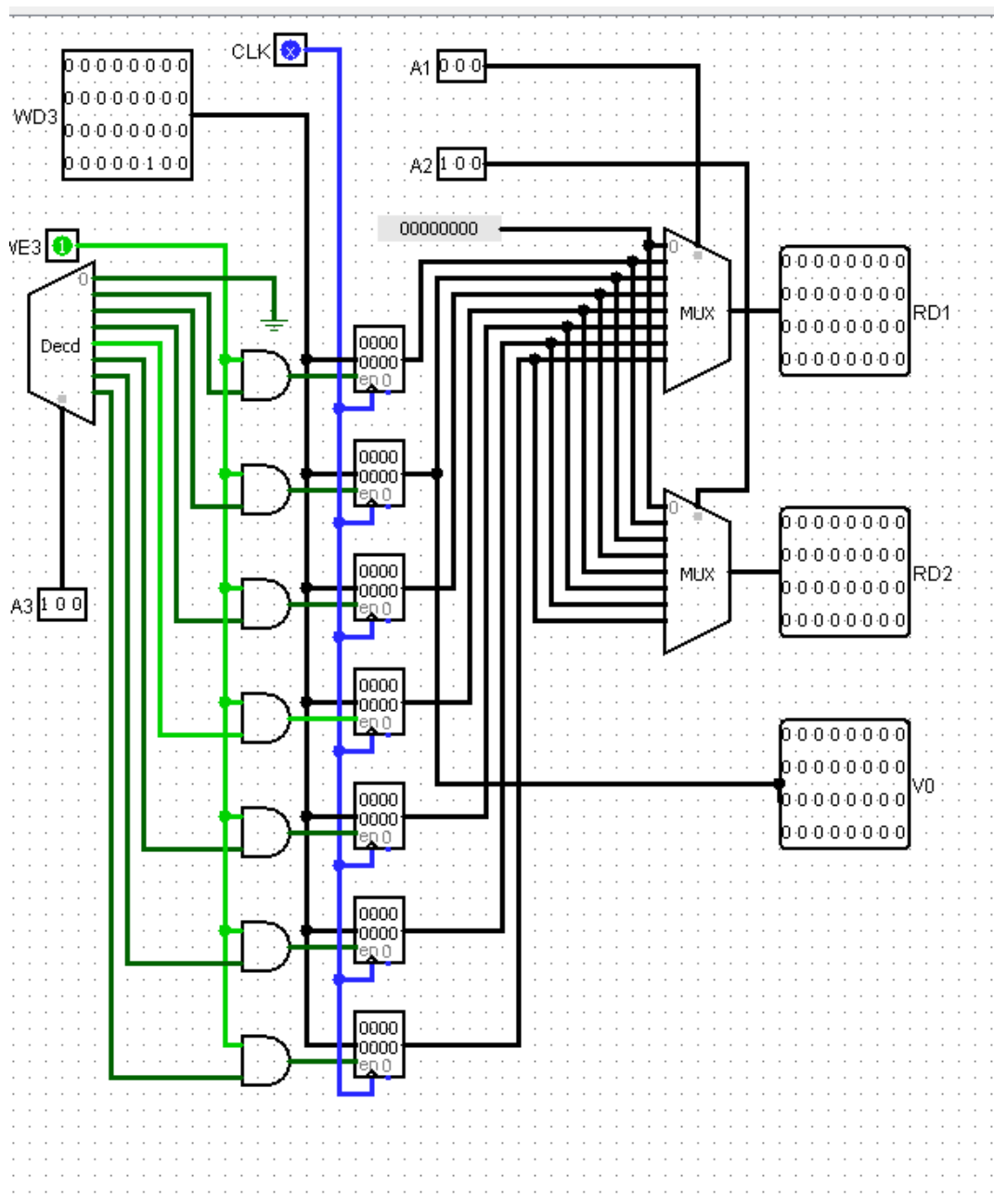
Comprobación de operaciones:



- * -

2. Explique qué ha añadido al diseño e incluya una captura de pantalla de la instrucción **addi** funcionando:

He añadido el sumador de instrucciones al pc, incluido con la opción de branch, que hace uso del and entre el flag zero y el bit de branch, ambos también implementados. He añadido las conexiones necesarias de los outputs de control unit al resto del programa, conectando cada uno a su lugar correspondiente.



Interior de RegisterFile tras hacer el primer addi (0+4), guardado en WD3

- * -

4.4.3 Implementar la **add**

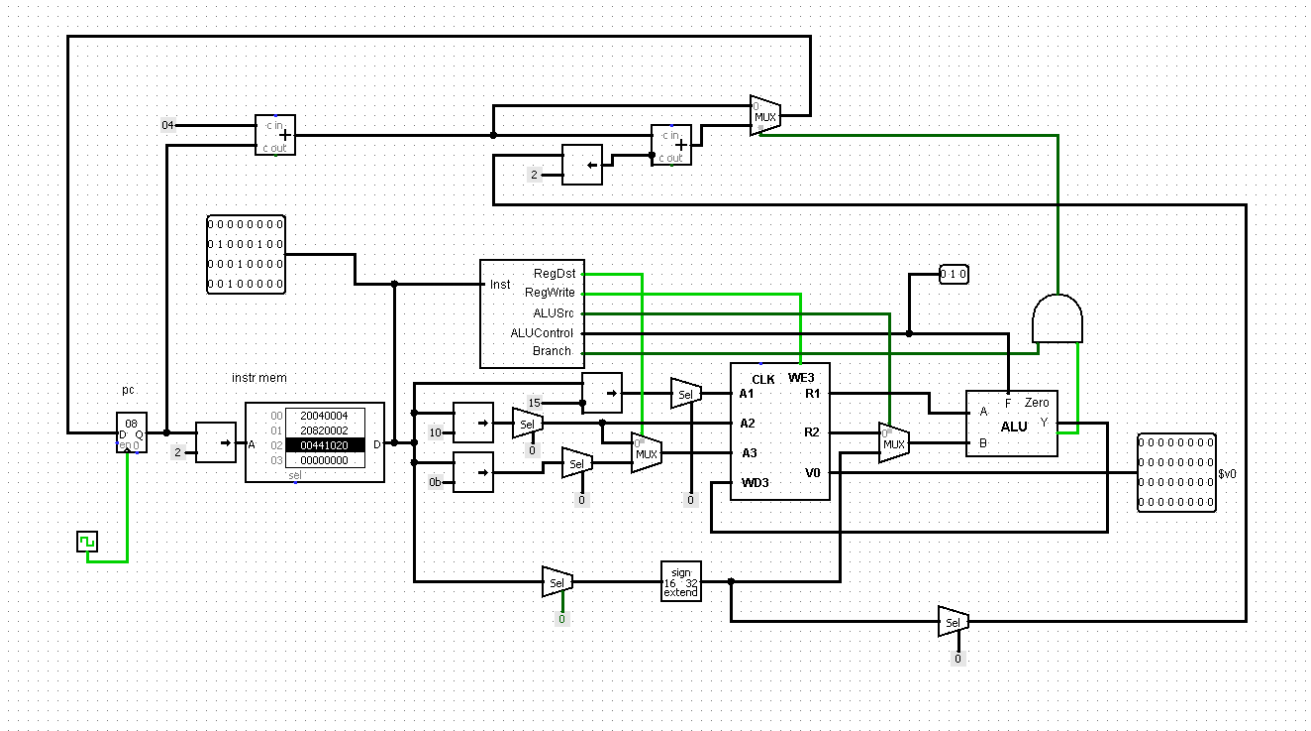
1. Abra el fichero **test.s** en MARS, identifique el código máquina de la tercera instrucción (es decir, la **add**) y a continuación añádalo a la ROM de su procesador.
2. Implemente la lógica necesaria para ejecutar la instrucción. Haga referencia también a la Figura con la arquitectura del MIPS que se adjunta al final del enunciado para entender qué bloques hay que añadir y cómo conectarlos.
3. Testee la secuencia de tres instrucciones para verificar que el procesador funciona correctamente.

- * -

3. Explique qué ha añadido al diseño e incluya una captura de pantalla de la instrucción **add** funcionando:

La lógica de todas las instrucciones será mostradas en el apartado 4.4.4.

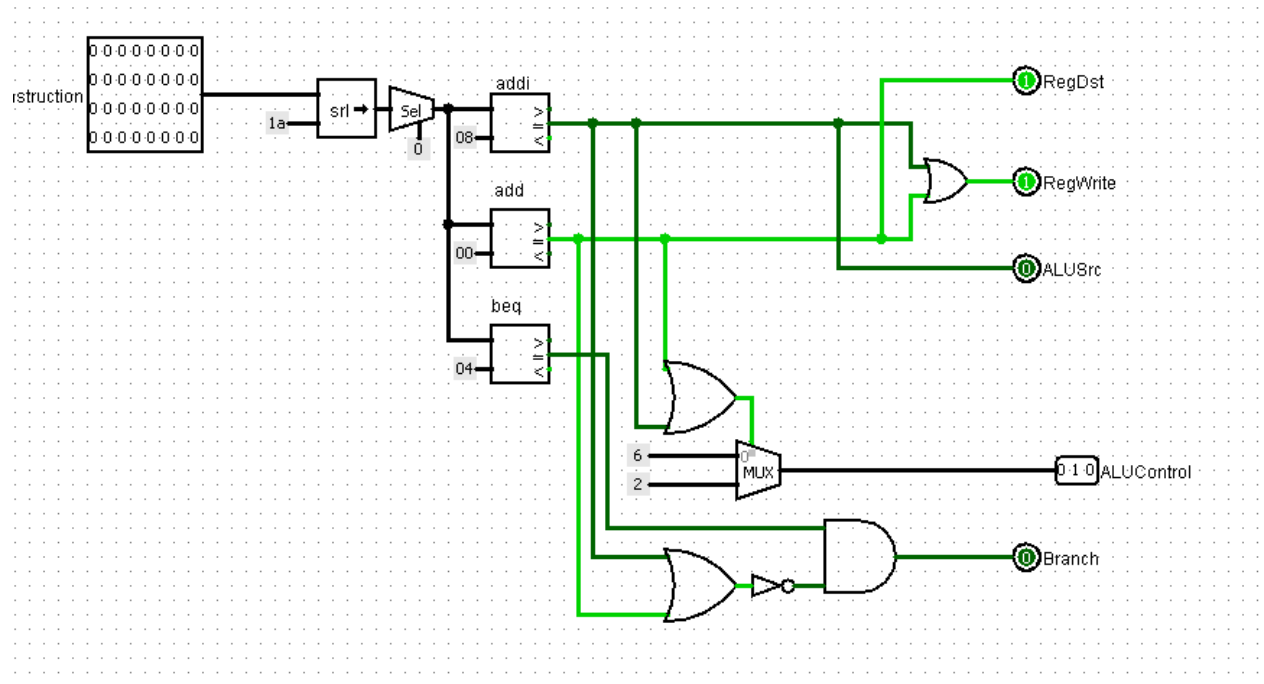
Al avanzar el clock el pc avanza, y compruebo que la instrucción es correcta:



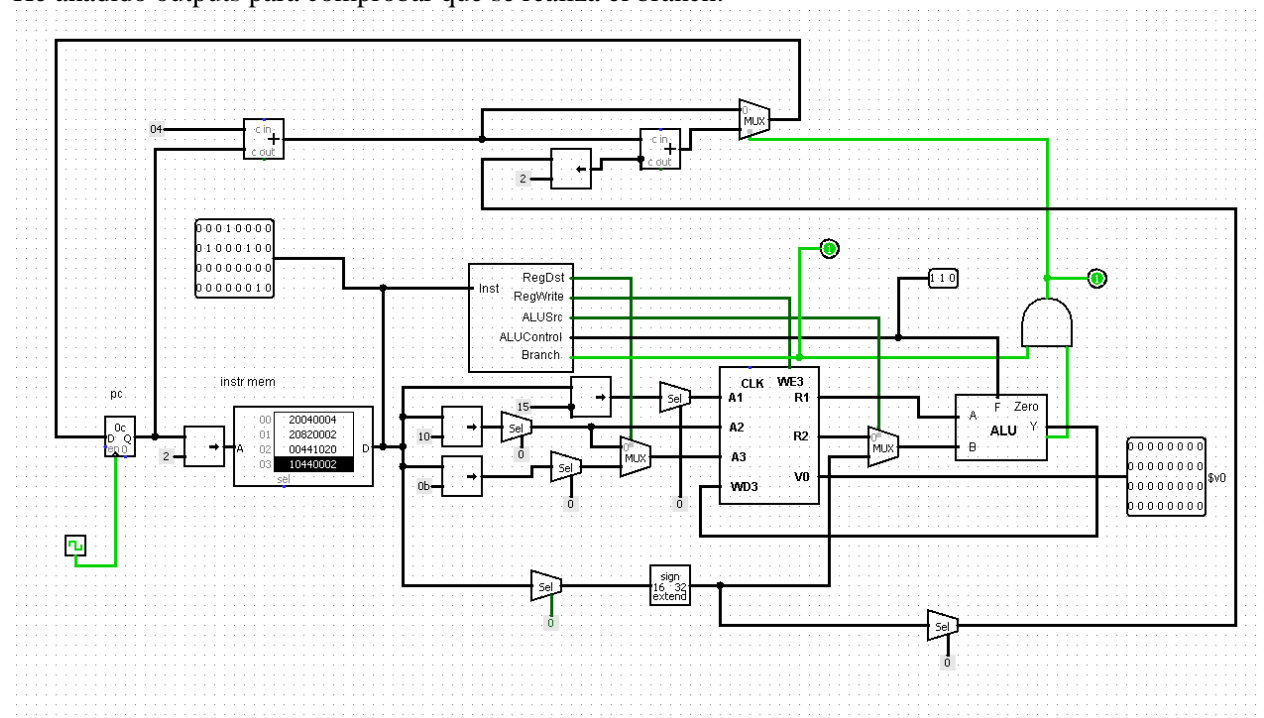
- * -

4.4.4 Implementar la **beq**

1. Implemente la lógica necesaria para ejecutar la **beq**. Haga referencia también a la Figura con la arquitectura del MIPS que se adjunta al final del enunciado para entender qué bloques hay que añadir y cómo conectarlos.



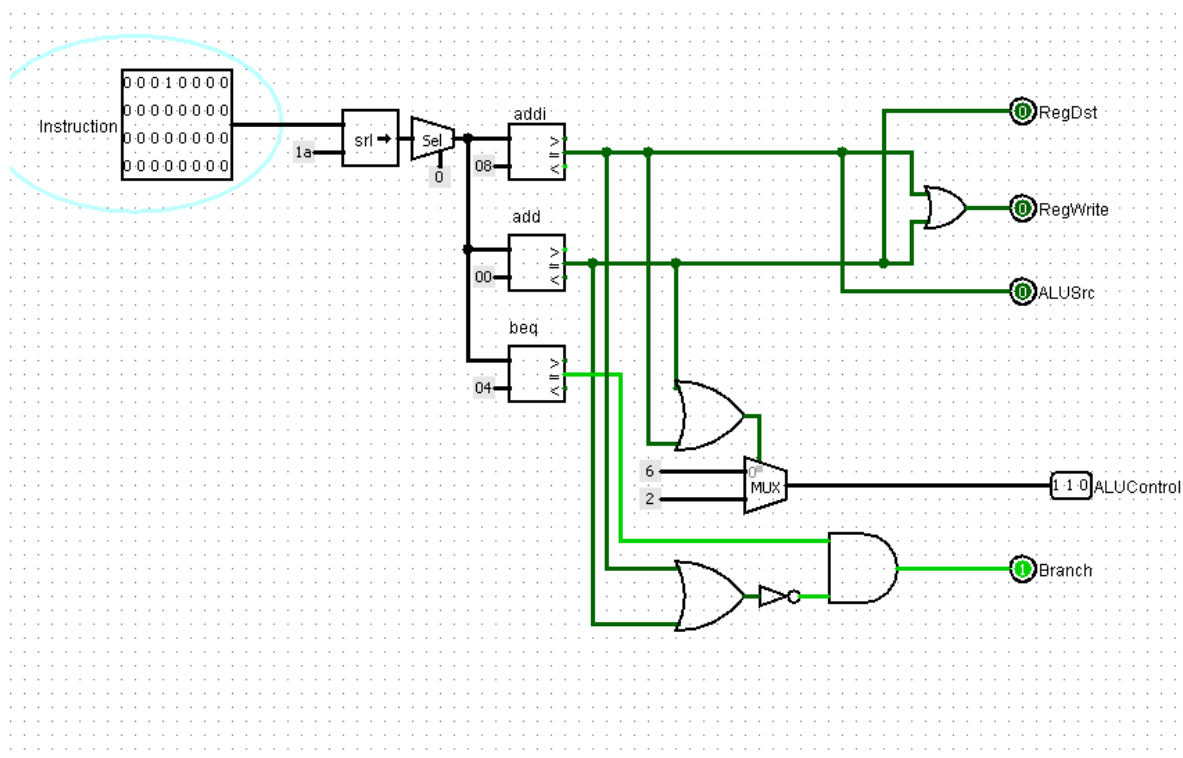
- Abra el fichero **test.s** en MARS, identifique los códigos máquina de las otras instrucciones que forman el programa y a continuación añádalos a la ROM de su procesador.
- Inicialice el contador de programa a la dirección 0x14 y compruebe que el salto se toma.
He añadido outputs para comprobar que se realiza el branch:



- * -

- Explique qué ha añadido al diseño e incluya una captura de pantalla de la instrucción **beq** funcionando:

En el control unit, al introducir la instrucción con opcode 4 (beq), se activan los siguientes valores:



El branch se activa porque comprueba que la instrucción no sea add y a su vez sea branch mediante un and y un nor gate. El alucontrol es elegido mediante un multiplexor que escoge las salidas del add o addi si la instrucción es una de las 2, comprobación realizada por una or gate. Los otros valores no son de importancia para la instrucción beq.

- * -

5. Extender el procesador (1 punto extra)

Durante la realización de la práctica se han tratado las unidades de control como cajas negras. Explique cómo funcionan y cómo las extendería para implementar una instrucción tipo **sub** (restar).

La Control Unit coge la instrucción e, ignorando los valores que no sean opcode, determina los valores que enviar a los outputs(regdst,alusrc,branch,etc.). Esta determinación se hace mediante puertas lógicas, plexación y otros métodos posibles, y en algunos casos no es necesario siquiera calcular los valores de una instrucción si no importan, como ocurre en el branch. Para añadir una operación sub, añadiría verticalmente otro comparador de igualdad para el opcode, añadiría su entrada para seleccionar 6 en el mux del alucontrol, añadiría la entrada al nor del branch y copiaría los otros valores del add, ya que son iguales.