Arquitectura de Computadoras

Ingeniería en Computación FCEFyN - UNC
Trabajo Práctico Final - MIPS

Aguerreberry Matthew. Mat.: 93739112
Maero Facundo. Mat.: 38479441

17 de enero de 2018

1. Introducción

Para el trabajo práctico final de la materia se realizó la implementación en Verilog de un procesador MIPS segmentado, que funciona con una unidad de control, la cual permite la comunicación con la PC mediante UART para programar la memoria del MIPS y ver el contenido de sus registros. Se desarrolló para la placa Arty, utilizando el software Vivado 2017.2.

2. Desarrollo

El módulo Top del proyecto contiene los siguientes elementos

- Procesador MIPS segmentado
- Máquina de Estados
- Recolector
- Módulo UART
- Módulo Clock Wizard

2.1. Procesador MIPS segmentado

Para desarrollar el procesador MIPS se partió del diseño obtenido en el libro **Computer Organization and Design**, por Patterson y Hennessy, el cual constituyó una excelente guía para comprender el funcionamiento de un procesador segmentado, las ventajas e inconvenientes, y los detalles de diseño a tener en cuenta.

Luego de terminar su descripción en Verilog, se procedió a agregar paulatinamente instrucciones a nuestro MIPS, dado que el desarrollado en el libro de texto tiene un instruction set reducido. A la par se fue desarrollando el llamado Çontrol Datapath", y las unidades de detección de riesgo, que permiten ejecutar ciertas combinaciones de instrucciones mediante mecanismos de çortocircuito".

El procesador de este trabajo cuenta con las 5 etapas clásicas, y es compatible con las siguientes instrucciones:

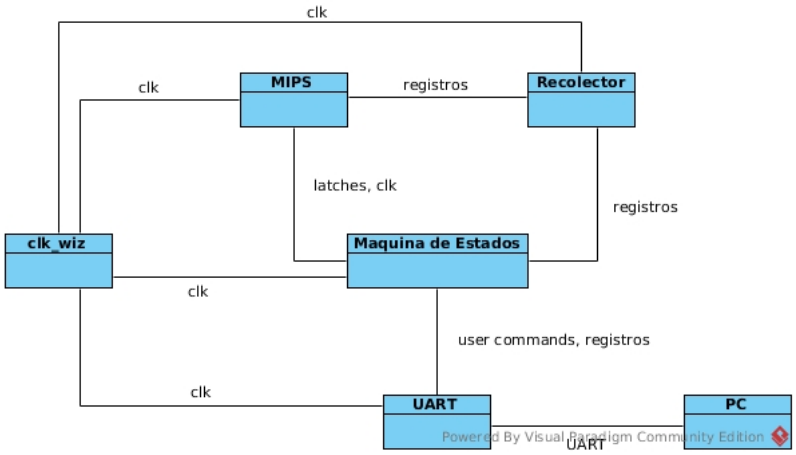

Figura 1: Diagrama en bloques del proyecto

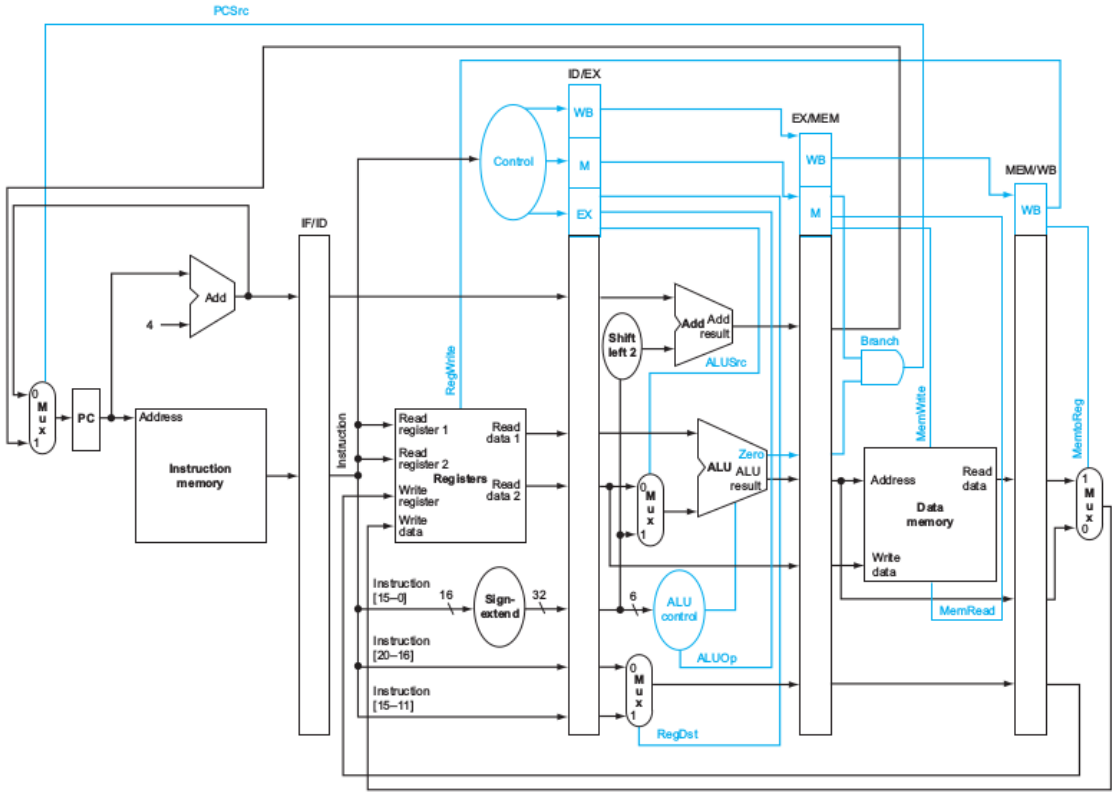

Figura 2: Diagrama de bloques básico del procesador MIPS

- R-type: SLL, SRL, SRA, SLLV, SRLV, SRAV, ADDU, SUBU, AND, OR, XOR, NOR, SLT
- I-Type: LB, LH, LW, LWU, LBU, LHU, SB, SH, SW, ADDI, ANDI, ORI, XORI, LUI, SLTI, BEQ, BNE, J, JAL
- J-Type: JR, JALR

Su funcionamiento y descripción puede encontrarse en el manual de instrucciones siguiente: `http://math-atlas.sourceforge.net/devel/assembly/mips-iv.pdf`

Se trata de un procesador de 32 bits, con 32 registros, una memoria de instrucciones de 2048 posiciones, y una memoria de datos también de 2048 posiciones, ambos valores parametrizables.

El bloque de forwarding implementado permite proveer a la etapa 3 (Execute) los argumentos necesarios que hayan sido calculados uno o dos ciclos antes. Los extrae de las etapas 4 (Memory) o 5 (Write-Back) respectivamente.

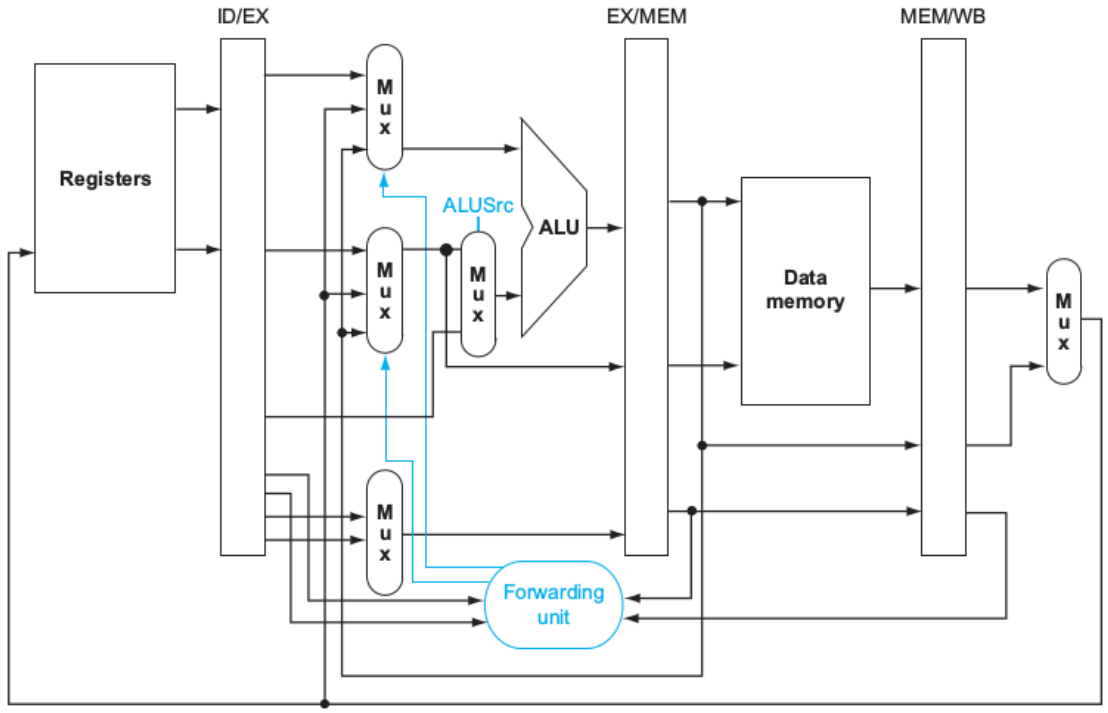

Figura 3: Forwarding unit en el procesador MIPS

El bloque de detección de riesgos identifica los casos en los que la ejecución no puede continuar, e introduce una burbuja en el pipeline, lo que corrige el problema. Esto sucede cuando luego de una instrucción load se quiere utilizar ese valor.

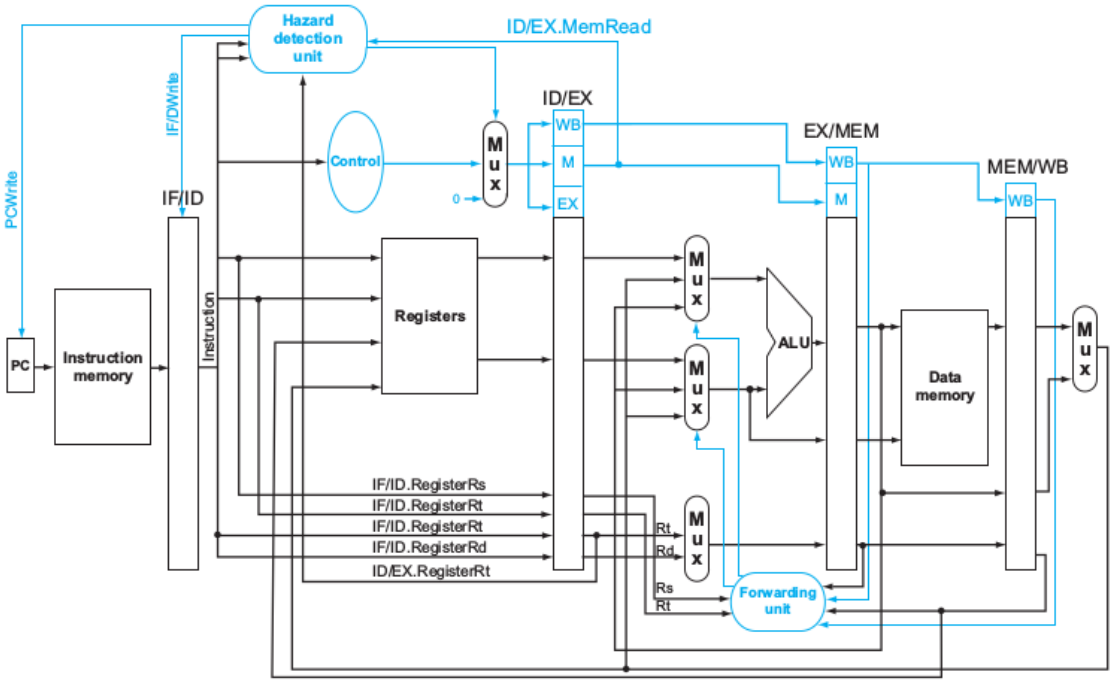

Figura 4: Diagrama de bloques básico del procesador MIPS con detección de riesgos

Para los riesgos de control, se optó por una política estática, en la que todos los saltos se consideran no tomados. Una vez conocido el resultado del branch, si se acertó, se continúa la ejecución. Caso contrario se limpia el pipe con las instrucciones leídas, y se comienza a ejecutar

desde el branch target.

El comportamiento de los saltos (Jump y sus variaciones) utiliza el concepto de branch delay slot, en el que se ejecuta la instrucción inmediatamente posterior al salto, para aprovechar ese ciclo de incertidumbre en el que todavía no se modificó el PC al valor de destino del salto.

2.2. Máquina de Estados

Este módulo es el encargado de manejar todo el diseño: controla el clock del MIPS, la UART, y va cambiando de estado en función del input que provea el usuario.

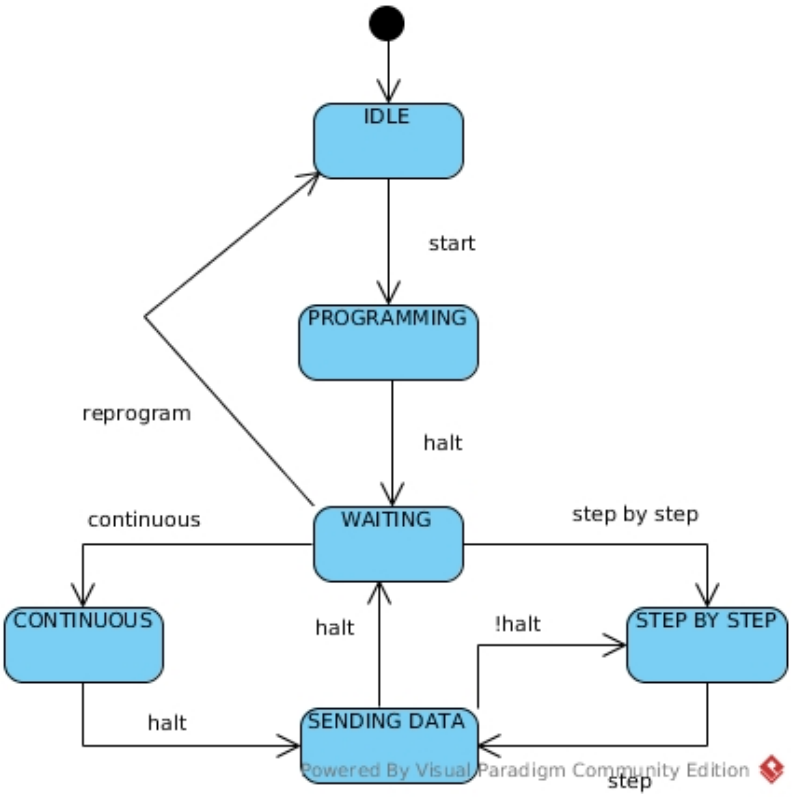

Figura 5: Máquina de estados

- Idle: Estado de espera, en el que el procesador no está funcionando, y se aguarda por una señal de **start** para comenzar la escritura de la memoria de programa.
- Programming: En este estado se está escribiendo la memoria de instrucciones del MIPS. Se reciben por UART los 4 bytes de la instrucción, y luego se la guarda en la memoria. Para ello cuenta con sub-estados, que permiten controlar este comportamiento. Una vez se escriba la instrucción de HALT (0xFFFFFFFF), se avanza al estado siguiente automáticamente.
- Waiting: Estado de espera. Aquí el procesador tampoco se encuentra funcionando, pero ya tiene las instrucciones que se quieren ejecutar en su memoria de programa. Se avanza de estado al recibir una señal del usuario, que puede ser avanzar al modo de ejecución continua, o paso a paso. También es posible volver al estado Idle, mediante una señal de reprogramación, para escribir en el MIPS otro programa ensamblador.
- Continuous: Al recibir la señal de ejecución continua, se habilita el clock del MIPS, y se ejecutan las instrucciones hasta llegar a un HALT, momento en el cual se avanza al estado de envío de registros.
- Step by Step: Al recibir la señal de ejecución paso a paso, la máquina de estados se encuentra a la espera de recibir la señal de step. Con ella, habilita el clock del MIPS solo por un ciclo, lo deshabilita y pasa al estado de envío de registros.

- Sending Data: En este estado se activa un modo de acceso sin clock en el MIPS, mediante el cual se extraen los valores guardados en los registros, la memoria de datos, los latches intermedios, el PC y el recuento de ciclos de clock empleados. Si la última instrucción ejecutada fue un HALT, se vuelve al estado Waiting. Caso contrario, se vuelve al estado Step by Step, a la espera de una nueva señal por parte del usuario.

2.3. Recolector

Este módulo posee parte de la lógica que permite leer el valor de los registros del procesador una vez iniciado el proceso de envío de datos por UART. Es controlado por la máquina de estados, y accede de manera combinacional a las memorias del MIPS.

2.4. Módulo UART

Este módulo es el desarrollado en el Trabajo Práctico 2 de la materia. Se conecta a la PC, donde se ejecuta un script en Python, que maneja la interfaz de usuario. La velocidad elegida es 9600 baudios.

2.5. Módulo Clock Wizard

La implementación del proyecto dió como resultado un clock de 40 MHz. Este módulo provee la frecuencia elegida a todos los demás bloques del proyecto.

2.6. Script en Python

En la PC se ejecuta un script que maneja la interfaz por línea de comandos. Mediante diferentes caracteres se envían por UART las distintas señales a la máquina de estados, para modificar su comportamiento. También permite observar los valores leídos del procesador.

Figura 6: Script en Python

3. Diseño, Validación y Testing

Para el diseño del procesador se confeccionó una hoja de cálculo donde se encuentra el estado de todas las señales de control frente a cada instrucción.

El testing se realizó primeramente mediante test benches, que permitieron corroborar que la simulación realizara lo esperado. Este proceso se expandió etapa a etapa, hasta completar la simulación del procesador en su totalidad.

Como paso siguiente se realizó un test bench que simula la conexión de la placa con la PC. Para ello se crearon dos instancias del módulo UART, y se enviaron mensajes entra una y otra.

Finalmente, una vez llegado a un grado de confianza suficiente, se realizó la implementación del diseño y se cargó el bitstream a la FPGA. Los resultados obtenidos se observaron mediante el script en Python, y se corroboraban con el test bench anteriormente descrito.

4. Clock Enable

Se modificó el comportamiento del clock en el procesador MIPS. La máquina de estados posee una salida que lo habilita, llamada **ctrl_clk_mips**. Además, por motivos de diseño fue necesario proveer una señal de clock desfasada 180 grados con respecto a la señal global para el módulo MIPS, que fue extraída del clock wizard.

En las siguientes imágenes puede verse que el procesador posee una entrada para el clock negado, y una entrada de clock enable. Observando más detenidamente, puede verse que sus registros poseen el clock antes mencionado como entrada, y la señal de enable como Chip Enable.

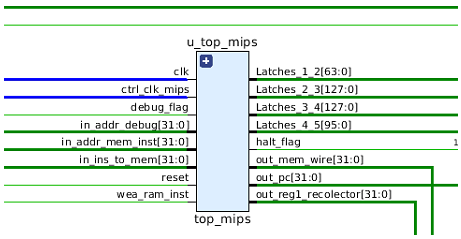

Figura 7: Entradas de clock al MIPS

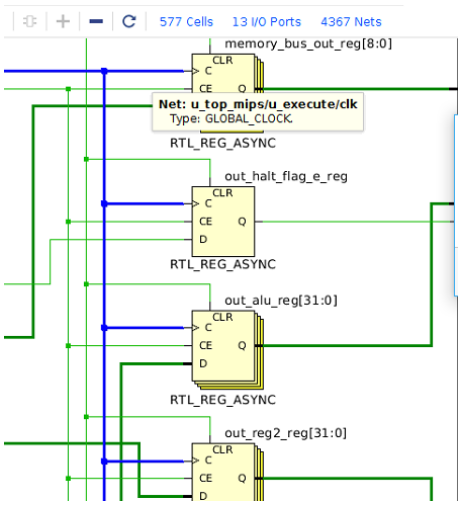

Figura 8: Señal de clock en registros del MIPS

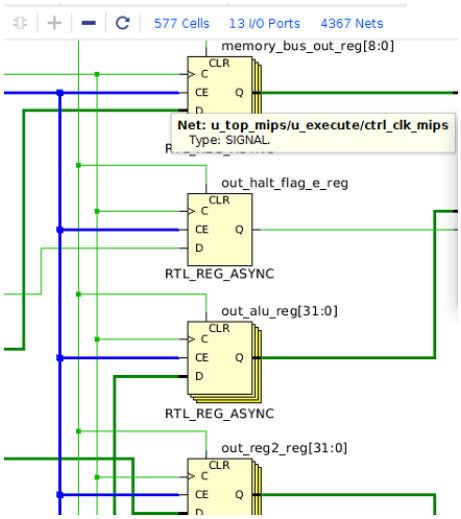

Figura 9: Señal de clock enable en registros del MIPS

Así efectivamente se utiliza el clock que proviene del clock wizard solamente, y no se lo controla con compuertas como funcionaba anteriormente.

A continuación se muestran capturas de pantalla de la ejecución de tests para comprobar que el sistema en su totalidad sigue funcionando acorde a lo esperado.

- **Test load y store**
  Resultado esperado:
  r2 = 1
  r3 = 0xFFFFFFFE
  r4 = 0x7FFFFFFE
  r5 = 0xFFFFFFFE
  r6 = 0xFFFE
  r7 = 0xFE
  r8 = 0
  r9 = 1
  mem[2] = 0xFFFFFFFE
  mem[3] = 0xFFFFFFFE

```
---Registros del procesador---
Registro 0:                                        0b0            0         0x0
Registro 1:                                        0b1            1         0x1
Registro 2:                                        0b1            1         0x1
Registro 3:  0b11111111111111111111111111111110           -2  0xfffffffe
Registro 4:   0b1111111111111111111111111111110    2147483646  0x7ffffffe
Registro 5:  0b11111111111111111111111111111110           -2  0xfffffffe
Registro 6:                  0b1111111111111110        65534      0xfffe
Registro 7:                           0b11111110          254        0xfe
Registro 8:                                  0b0            0         0x0
Registro 9:                                  0b1            1         0x1
Registro 10:                              0b1010           10         0xa
Registro 11:                              0b1011           11         0xb
Registro 12:                              0b1100           12         0xc
Registro 13:                              0b1101           13         0xd
Registro 14:                              0b1110           14         0xe
Registro 15:                              0b1111           15         0xf
Registro 16:                             0b10000           16        0x10
Registro 17:                             0b10001           17        0x11
Registro 18:                             0b10010           18        0x12
Registro 19:                             0b10011           19        0x13
Registro 20:                             0b10100           20        0x14
Registro 21:                             0b10101           21        0x15
Registro 22:                             0b10110           22        0x16
Registro 23:                             0b10111           23        0x17
Registro 24:                             0b11000           24        0x18
Registro 25:                             0b11001           25        0x19
Registro 26:                             0b11010           26        0x1a
Registro 27:                             0b11011           27        0x1b
Registro 28:                             0b11100           28        0x1c
Registro 29:                             0b11101           29        0x1d
Registro 30:                             0b11110           30        0x1e
Registro 31:                             0b11111           31        0x1f

---Posiciones de memoria---
Posicion 0:                                        0b0            0         0x0
Posicion 1:                                        0b1            1         0x1
Posicion 2:  0b11111111111111111111111111111110           -2  0xfffffffe
Posicion 3:  0b11111111111111111111111111111110           -2  0xfffffffe
Posicion 4:                                  0b100            4         0x4
```

Figura 10: Test load y store

- **Test jump, jump register**
  Resultado esperado:
  r0 = 11
  r1 = -4
  r2 = 4
  r4 = -8
  r5 = 0
  r6 = 3
  r7 = 10
  r31 = 3

```
---Registros del procesador---
Registro 0:                                0b1011          11         0xb
Registro 1:  0b11111111111111111111111111111100          -4  0xfffffffc
Registro 2:                                 0b100           4         0x4
Registro 3:                                  0b11           3         0x3
Registro 4:  0b11111111111111111111111111111000          -8  0xfffffff8
Registro 5:                                   0b0           0         0x0
Registro 6:                                  0b11           3         0x3
Registro 7:                                0b1010          10         0xa
Registro 8:                                0b1000           8         0x8
Registro 9:                                0b1001           9         0x9
Registro 10:                               0b1010          10         0xa
Registro 11:                               0b1011          11         0xb
Registro 12:                               0b1100          12         0xc
Registro 13:                               0b1101          13         0xd
Registro 14:                               0b1110          14         0xe
Registro 15:                               0b1111          15         0xf
Registro 16:                              0b10000          16        0x10
Registro 17:                              0b10001          17        0x11
Registro 18:                              0b10010          18        0x12
Registro 19:                              0b10011          19        0x13
Registro 20:                              0b10100          20        0x14
Registro 21:                              0b10101          21        0x15
Registro 22:                              0b10110          22        0x16
Registro 23:                              0b10111          23        0x17
Registro 24:                              0b11000          24        0x18
Registro 25:                              0b11001          25        0x19
Registro 26:                              0b11010          26        0x1a
Registro 27:                              0b11011          27        0x1b
Registro 28:                              0b11100          28        0x1c
Registro 29:                              0b11101          29        0x1d
Registro 30:                              0b11110          30        0x1e
Registro 31:                                 0b11           3         0x3
```

Figura 11: Test jump, jump register

- **Test stall caso del load**
  Resultado esperado:
  r1 = 8
  r2 = 1
  r3 = 7
  r4 = 1

```
---Ciclos de clock empleados---
Ciclos:                                    0b1001           9         0x9

---Registros del procesador---
Registro 0:                                   0b0           0         0x0
Registro 1:                                0b1000           8         0x8
Registro 2:                                   0b1           1         0x1
Registro 3:                                 0b111           7         0x7
Registro 4:                                   0b1           1         0x1
Registro 5:                                0b101           5         0x5
Registro 6:                                0b110           6         0x6
Registro 7:                                0b111           7         0x7
Registro 8:                               0b1000           8         0x8
Registro 9:                               0b1001           9         0x9
Registro 10:                              0b1010          10         0xa
Registro 11:                              0b1011          11         0xb
Registro 12:                              0b1100          12         0xc
Registro 13:                              0b1101          13         0xd
Registro 14:                              0b1110          14         0xe
Registro 15:                              0b1111          15         0xf
Registro 16:                             0b10000          16        0x10
Registro 17:                             0b10001          17        0x11
Registro 18:                             0b10010          18        0x12
Registro 19:                             0b10011          19        0x13
Registro 20:                             0b10100          20        0x14
Registro 21:                             0b10101          21        0x15
Registro 22:                             0b10110          22        0x16
Registro 23:                             0b10111          23        0x17
Registro 24:                             0b11000          24        0x18
Registro 25:                             0b11001          25        0x19
Registro 26:                             0b11010          26        0x1a
Registro 27:                             0b11011          27        0x1b
Registro 28:                             0b11100          28        0x1c
Registro 29:                             0b11101          29        0x1d
Registro 30:                             0b11110          30        0x1e
Registro 31:                             0b11111          31        0x1f
```

Figura 12: Test stall caso del load