Data Path

66:20 Organización de Computadoras

Trabajo práctico 3

Mauro Toscano (96890)
Axel Lijdens (95772)

Univesidad de Buenos Aires - FIUBA

Índice

Objetivos **2**

Alcance **2**

Requisitos **2**

Recursos **2**

Fecha de entrega **2**

Introducción **2**

Instrucciones a implementar **2**

Implementación **4**

Pruebas **9**

Jump 9

Jump Register 9

Jump and Link Register 10

Conclusiones **10**

Archivos .cpu **11**

Objetivos

El objetivo de este trabajo es familiarizarse con la arquitectura de una CPU MIPS, específicamente con el datapath y la implementación de instrucciones. Para ello, se deberían agregar instrucciones a diversas configuraciones de CPU provistas por el simulador DrMIPS

Alcance

Este trabajo práctico es de elaboración grupal, evaluación individual, y de carácter obligatorio para todos alumnos del curso.

Requisitos

El trabajo deberá ser entregado personalmente, en la fecha estipulada, con una carátula que contenga los datos completos de todos los integrantes.

Además, es necesario que el trabajo práctico incluya (entre otras cosas, ver sección 8), la presentación de los resultados obtenidos, explicando, cuando corresponda, con fundamentos reales, las causas o razones de cada resultado obtenido. Por este motivo, el día de la entrega deben concurrir todos los integrantes del grupo.

El informe deberá respetar el modelo de referencia que se encuentra en el grupo, y se valorarán aquellos escritos usando la herramienta TEX / LATEX.

Recursos

Usaremos el programa DrMIPS para configurar y simular el data path de un procesador MIPS, tanto uniciclo como multiciclo.

Fecha de entrega

La última fecha de entrega y presentación será el jueves 28 de junio de 2018.

Introducción

El programa DrMIPS nos permite evaluar distintos diseños de datapath para procesadores MIPS32, al darnos la posibilidad de organizarlo como queramos. Si bien sólo puede haber uno de algunos de los componentes del DP (como el registro de PC o la unidad de control), podemos poner sumadores, multiplexores, extensores de signo y conexiones arbitrariamente. También es posible modificar el conjunto de instrucciones. Además de la estructura lógica del DP, DrMips nos permite escribir programas simples y simular su ejecución en el DP, mostrando los valores que toman las diversas entradas y salidas de cada elemento. El programa se puede conseguir en https://bitbucket.org/brunonova/drmips/wiki/Home, o se puede descargar para Ubuntu, ya sea desde el repositorio de Ubuntu (aunque la versión está desactualizada) o autorizando un repositorio externo.

Instrucciones a implementar

1. Implementar la instrucción `j` en el DP `pipeline.cpu`.

2. Implementar la instrucción `jr` (Jump Register) en el DP `unicycle.cpu`.
3. Implementar la instrucción `jr` en el DP `pipeline.cpu`. Verificar que no se produzcan hazards.
4. Implementar la instrucción `jalr` (Jump and Link Register) en el DP `unicycle.cpu`.
5. Implementar la instrucción `jalr` en el DP `pipeline.cpu`. Verificar que no se produzcan hazards.

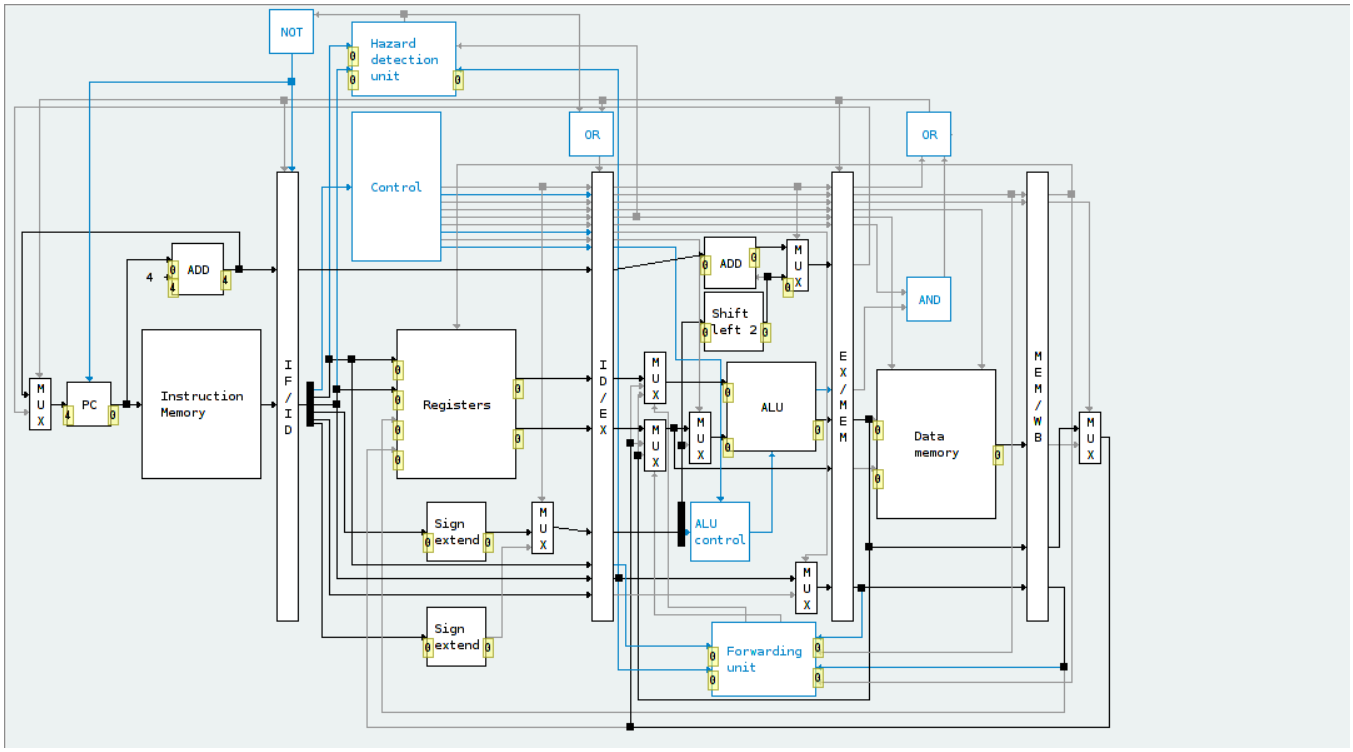

Figura 1: Pipeline con Jump

Implementación

Las instrucciones Jump deben poder modificar la actualización del PC, por lo tanto es necesario indicar de dónde se obtendrá la nueva dirección. En el caso de Jump, la dirección se obtiene desde la instrucción misma. Como las direcciones son múltiplo de 4, no es necesario indicar los 2 bits menos significativos (ya que son siempre 0). Al valor guardado en la instrucción se lo debe multiplicar por 4 (mediante un shift). Esto se controla mediante un mux que en base a la instrucción (Jump o Branch) elije si shiftear o sumar el offset.

Para las instrucciones que leen la dirección desde un registro, se debe poder asignar el valor del PC desde la salida de registros. Esto se logra uniéndolos con un cable y el PC recibe los datos desde un mutex controlado por la ALU.

Para evitar hazards, se optó por vaciar el pipeline cuando se ejecuta una instrucción de tipo Jump. De esta forma se evita ejecutar instrucciones que corresponden a la rama no tomada.

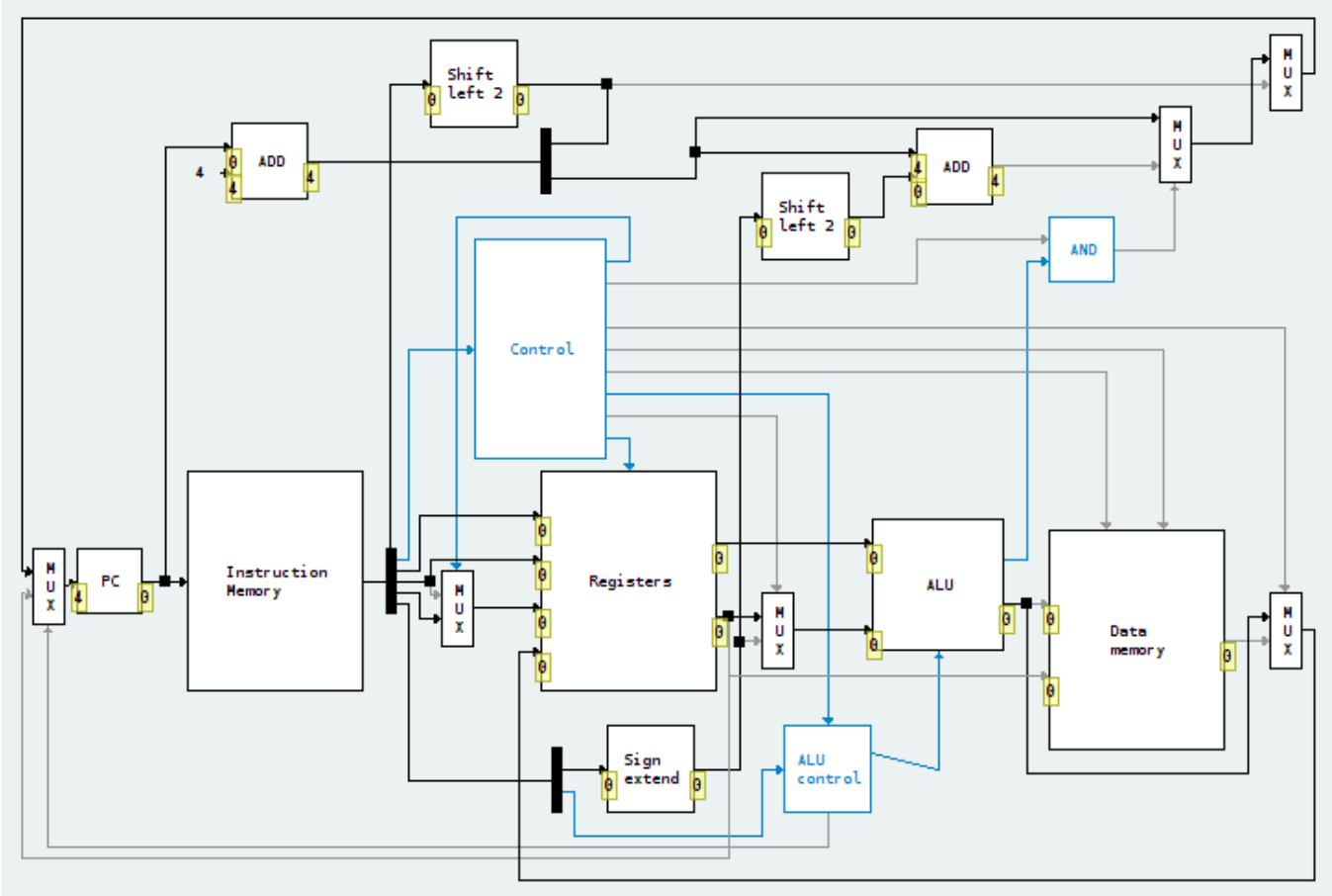

Figura 2: Unicycle con Jump Register

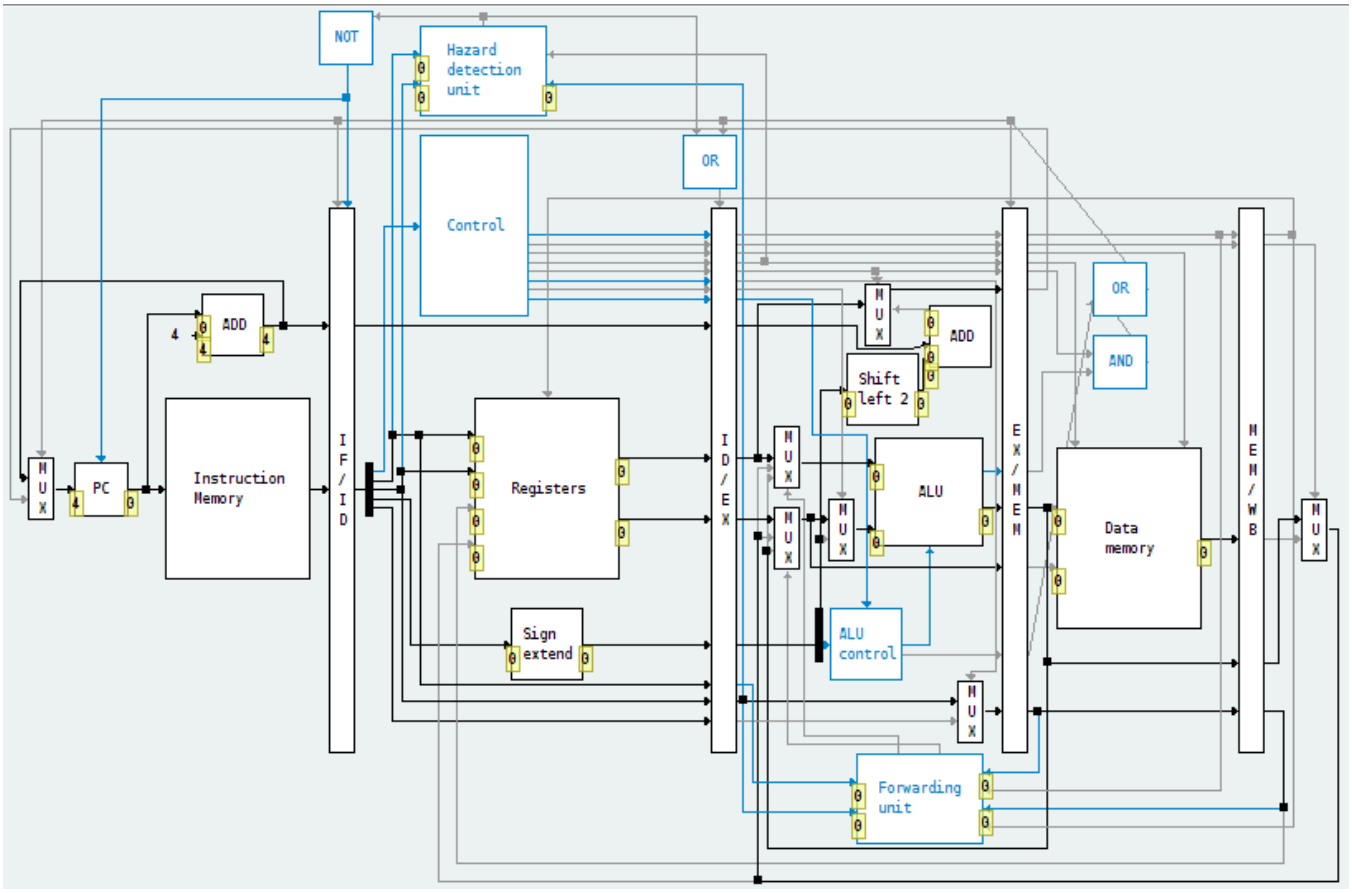

Figura 3: Pipeline con Jump Register

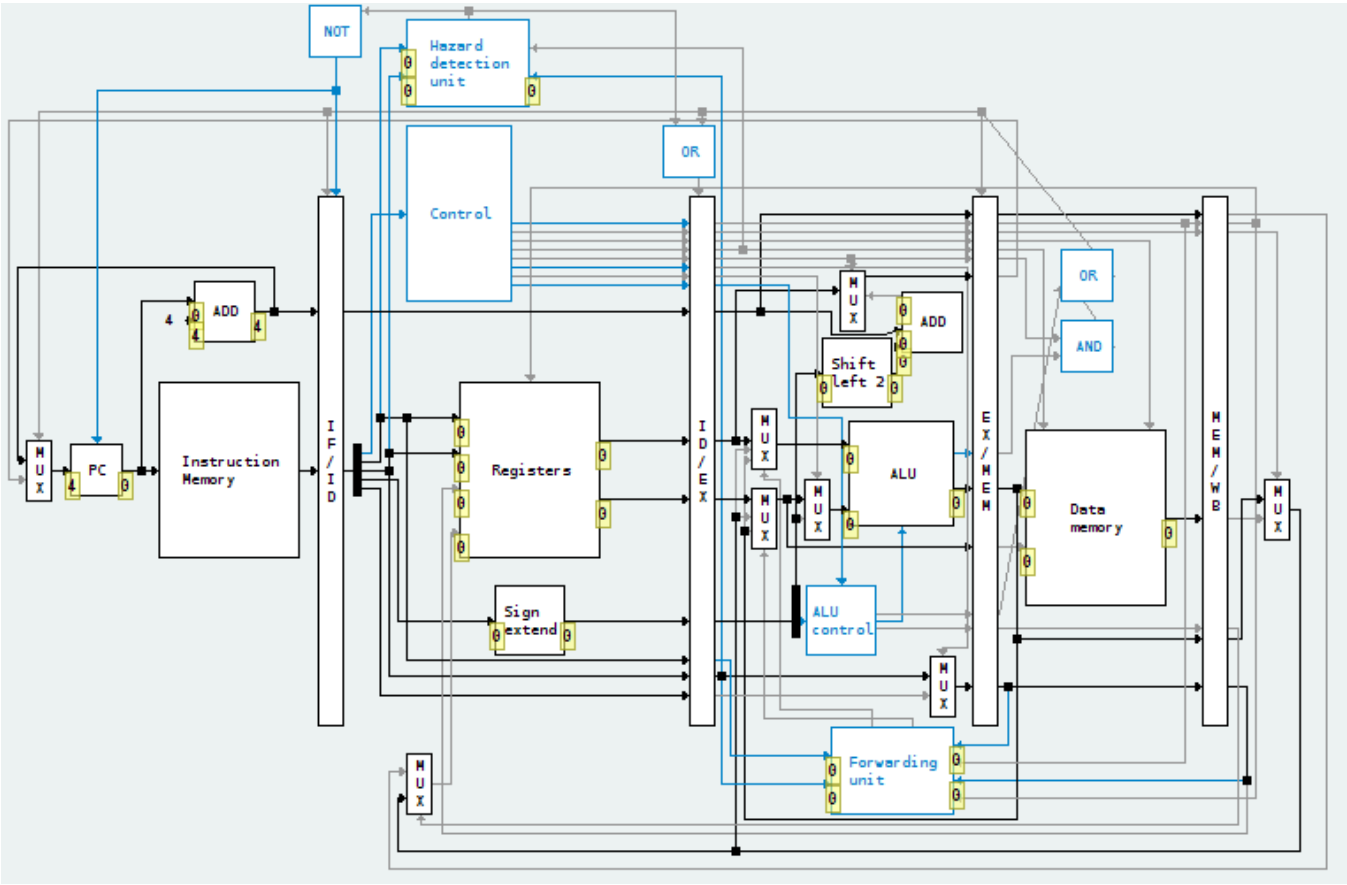

Figura 4: Pipeline con Jump and Link Register

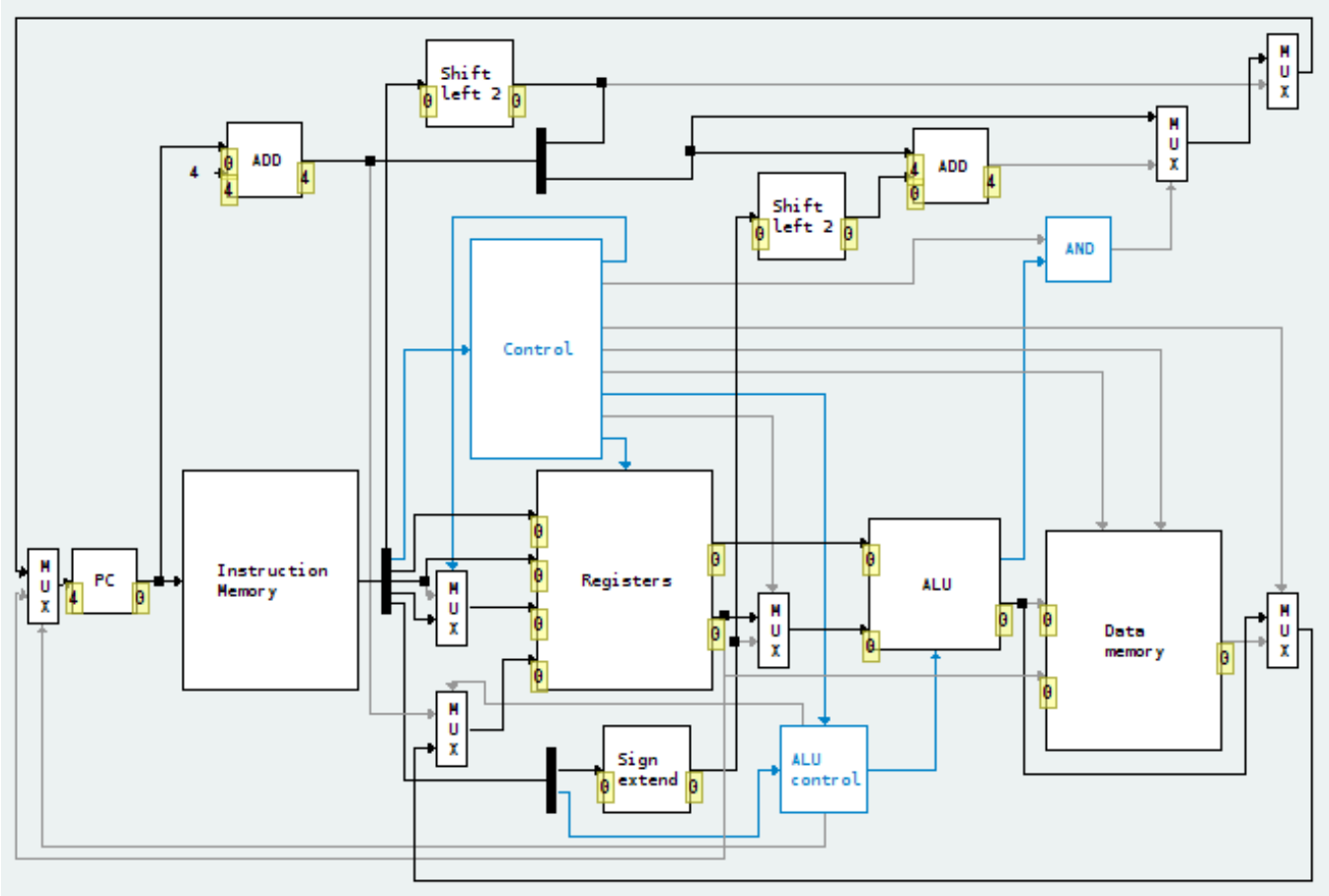

Figura 5: Unicycle con Jump and Link Register

Pruebas

En todos los casos debe verificarse que la instrucción se ejecute correctamente. Esto implica que el PC tome el valor deseado, y además que en el caso del DP multiciclo no se produzcan hazards, como ser la ejecución de la instrucción siguiente al salto, o en el caso de utilizar el valor de un registro, que éste tenga el valor correcto.

Jump

Para probar la instrucción `J` se utilizó el siguiente programa:

```
1   li  $t0, 111
2   li  $t1, 222
3   j test
4
5   # las siguientes instrucciones no deberian ejecutarse si el salto se
        realiza
6   li  $t0, 333
7   li  $t1, 444
8
9 test:
10  li $t2, 555
```

Listing 1: Test j

De funcionar correctamente la instrucción, el resultado debe ser `$t0=111 , $t1=222 y $t2 = 555`, ya que las otras 2 instrucciones no se ejecutarían.

Se comprobo el resultado es el esperado con la implementación que lo utiliza.

Jump Register

Esta instrucción debe saltar a la dirección guardada en el registro indicado. El ejemplo es similar al anterior con la diferencia que se espeficica una dirección particular.

```
1   li  $t0, 111
2   li  $t1, 222
3   li  $ra, 24
4   jr  $ra
5
6   # las siguientes instrucciones no deberian ejecutarse si el salto se
        realiza
7   li  $t0, 333
8   li  $t1, 444
9
10 test:
11  li $t2, 555  # si las instrucciones comienzan desde la dirección 0, esta
        tiene la dirección 24
```

Listing 2: Test jr

De funcionar correctamente, la ejecución debe continuar en la dirección indicada en `$ra`, que será la de la operación `nop` (salteando las instrucciones indicadas).

Se comprobo el resultado es el esperado con las implementaciones que lo utilizan.

Jump and Link Register

Eesta instrucción guarda la dirección siguiente en `$ra` y luego realiza el salto al registro indicado. Se utiliza el siguiente programa de prueba:

```
1   li  $t0, 111
2   li  $t1, 222
3   li  $t2, 24
4   jalr  $t3, $t2
5
6   # las siguientes 2 instrucciones no deberian ejecutarse si el salto se
        realiza
7   addi  $t0, $t0, 333
8   addi  $t1, $t1, 444
9
10  test:
11          li $t2, 555
```

Listing 3: Test jr

Al igual que el caso anterior, se debe verificar que las operaciones indicadas no se hayan ejecutado. Además, se debe verificar que el registro `$t3` contenga el valor correspondiente a la instrucción siguiente del `jalr`, es decir, 16. Se comprobo el resultado es el esperado con las implementaciones que lo utilizan.

Conclusiones

DrMIPS es una herramienta útil para poder entender el funcionamiento del DP en MIPS, así como el mecanismo de pipeline, ya que se puede ejecutar un programa paso a paso y observar los cambios que cada instrucción realiza. Al permitir modificar libremente el DP se puede lograr un mejor entendimiento de su funcionamiento. También es útil para simular instrucciones en desarrollo, ya que permite encontrar errores de la misma forma que un debugger o hace en un lenguage de programación.

Archivos .cpu

Repositorio: https://github.com/MauroFab/orga-6620-tp3