FACULTAD DE INGENIERIA
Universidad de Buenos Aires

Trabajo Práctico 3: Data path y pipeline

6620 - Organización de Computadoras
Primer cuatrimestre de 2020

Alumno	Padrón	Email
Maximiliano Yacobucci	93321	myacobucci@fi.uba.ar
Federico del Mazo	100029	fdelmazo@fi.uba.ar
Ariel Vergara	97010	avergara@fi.uba.ar

Resumen

En este Trabajo Práctico se implementarán 3 instrucciones de MIPS en dos datapath diferentes, uno uniciclo y otro con pipeline, mediante el uso del simulador DrMIPS.

1. Introducción

En el siguiente Trabajo Práctico se implementarán diversas instrucciones utilizando dos diseños distintos de datapath: uniciclo y con pipeline.

Las instrucciones a desarrollar son las siguientes:

- `andi Rs, Rt, Imm`: Instrucción de tipo I que carga en Rs el resultado de hacer un AND entre el contenido del registro Rt y el valor de 16 bits Imm. La misma se debe implementar para los datapaths unicycle.cpu y pipeline.cpu.
- `j Rs, Rt`: Instrucción de tipo I que carga en el PC el resultado de sumar los contenidos de los registros Rs y Rt. Se debe implementar para los datapaths unicycle.cpu y pipeline.cpu.
- `lw Rs, Rd*Imm(Rt)`: Instrucción de tipo R que carga en el registro Rs la palabra de 32 bits cuya dirección es Rt + Rd * shamt. Se debe implementar para el datapath pipeline-extended.cpu.

2. Diseño e Implementación

A continuación se explicará cómo se implementó cada instrucción en los respectivos datapath.

2.1. `andi Rs, Rt, Imm`

2.1.1. DP unicycle

Para implementar esta instrucción fue necesario agregar una nuevo valor de ALUOp correspondiente al número decimal 3 (según se investigó, la señal ALUOp es de 2 bits) y que la misma realice una operación de tipo AND. Esto se debe a que sólo el valor de ALUOp 2 tiene asociado la operación AND y se debe utilizar el campo func de las instrucciones de tipo R para distinguirlo, el cual no se encuentra presente en las instrucciones de tipo I.

Dado que el DP utilizado ya implementa instrucciones de tipo I similares, como addi, no fue necesario modificarlo. La única diferencia recae en que en addi se suma el valor inmediato al del registro en lugar de realizar un AND lógico entre ambos valores.

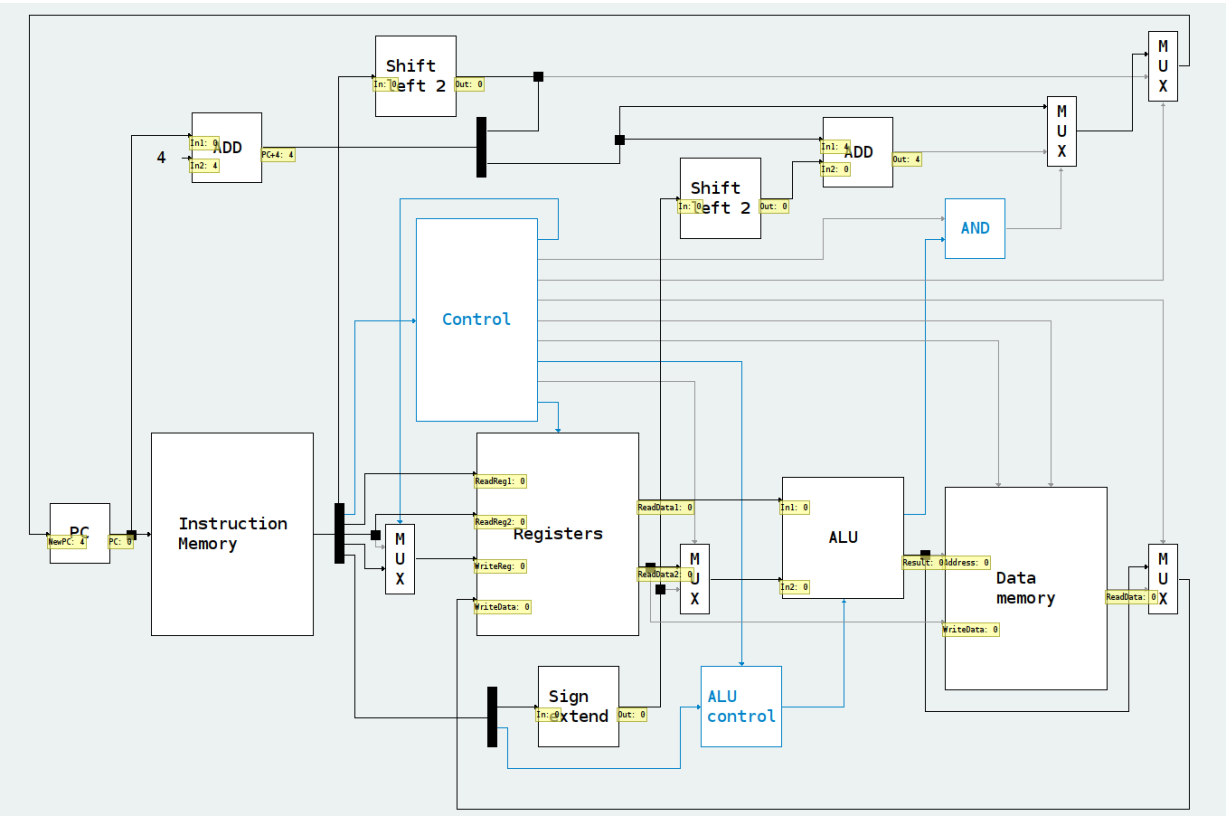

Figura 1: DP uniciclo para la instrucción `andi Rs, Rt, Imm`

2.1.2. DP pipeline

La instrucción se implementó de la misma manera que para el DP uniciclo y no fue necesario modificar el datapath por las mismas razones expresadas en el caso anterior.

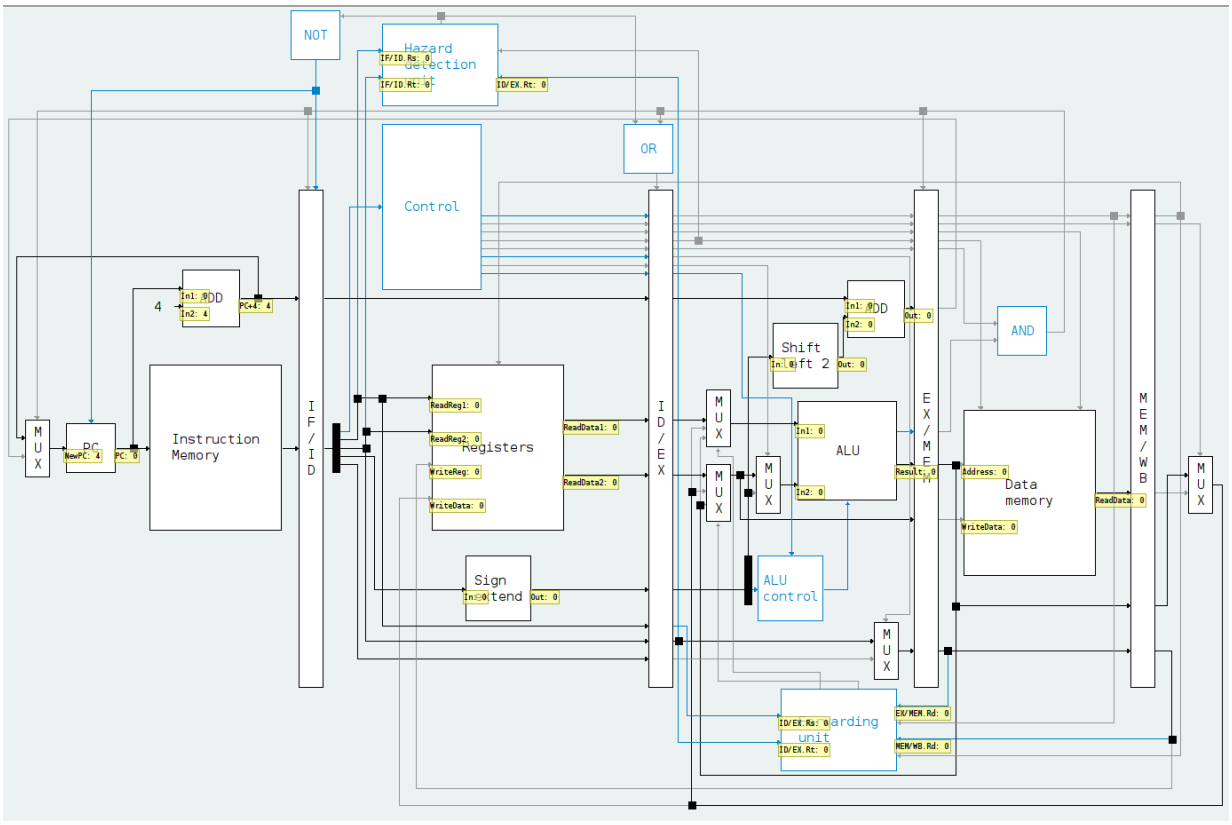

Figura 2: DP multiciclo para la instrucción `andi Rs, Rt, Imm`

2.2. `j Rs, Rt`

En el set de instrucciones del DP uniciclo ya se encuentra implementada la instrucción `j Imm` de MIPS, por lo que se renombró la instrucción a `jreg Rs, Rt` para que no se produzca un error por clave repetida. Esto se mantuvo para el set de instrucciones del DP multiciclo con el fin de poder utilizar el mismo archivo de pruebas.

A su vez, se decidió que el resultado de la suma de los valores de los registros indique el número de instrucción (comenzando desde 0 dado que buscamos posiciones de memoria) por lo que el mismo se debe multiplicar por 4. Esto se debe a que las instrucciones ocupan 4 bytes.

2.2.1. DP unicycle

Como se dijo anteriormente, la instrucción de jump ya se encuentra implementada. La única diferencia entre esta y la nueva instrucción consiste en cómo se obtiene la dirección. A su vez, en el DP uniciclo se tiene un multiplexor (llamado MuxJump en DrMIPS) que selecciona la siguiente dirección a cargar en el PC. Esta puede ser PC+4/dirección de branch (lo cual se decide en un multiplexor anterior a partir de la señal Branch de un bit) si la señal Jump de un bit es 0, o la direccion de salto si es 1. Por esta razón, se decidió agregar un nuevo multiplexor antes de este que determine la dirección de salto (llamado MuxJumpAddress). Sus entradas son el valor de la dirección de salto que antes ingresaba en el MuxJump y el resultado de la ALU luego de pasar por un Shift Left de 2 bits, dado que es una dirección de una instrucción y todas las instrucciones ocupan 4 bytes (de esta manera se realiza el mismo procedimiento que en `j Imm`). Para determinar cuál de las dos direcciones se debe elegir, se utilizó la señal Branch de la Unidad de Control dado que era la única señal que no tenía efectos colaterales. Si la señal es 0, se elige la dirección correspondiente a la instrucción j, si es 1 se elige la obtenida a partir de la ALU.

De esta manera, los bits Jump y Branch de la Unidad de Control valen 1 para la nueva instrucción, mientras que para `j` solamente el bit Jump es igual a 1.

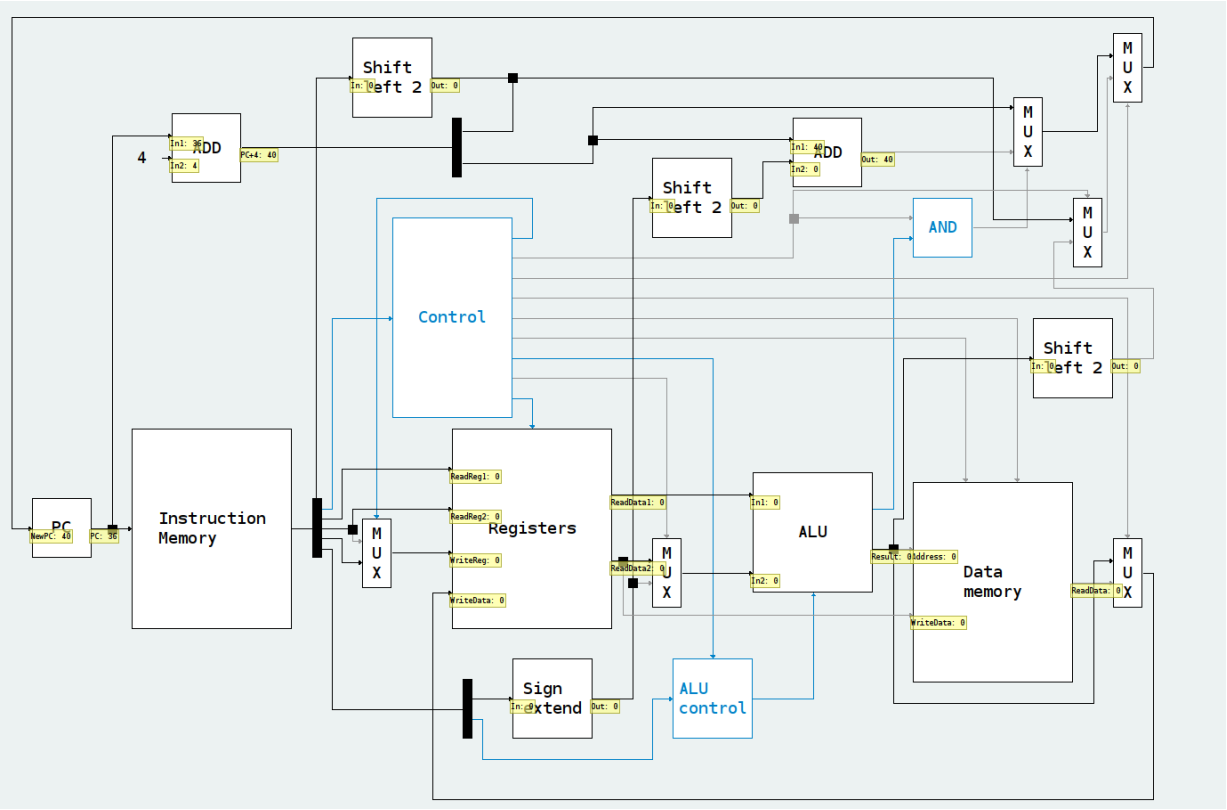

Figura 3: DP uniciclo para la instrucción `j Rs, Rt`

2.2.2. DP pipeline

Este DP no tiene implementada la instrucción `j Imm`. Por defecto, el nuevo valor que toma el PC proviene de un multiplexor (llamado MuxPC en DrMIPS) que tiene como entradas los valores de PC+4 y la dirección de branch.

Se redireccionó la salida de este multiplexor a uno nuevo (llamado MuxJump), el cual tiene como segunda entrada el resultado de la ALU luego de pasar por un Shift Left de 2 bits (al igual que en el DP unicycle). El resultado de la ALU proviene del registro de pipeline EX/MEM, al igual que dirección de branch que se ingresa en el MuxPC. Esto indica que los valores se obtienen en la etapa MEM del pipeline.

Para determinar la entrada a seleccionar, se utilizó una señal Jump de un bit, la cual se pasa desde la Unidad de Control a través de los distintos registros de pipeline hasta ser utilizada en la etapa MEM. Si la señal vale 1, se elije la entrada correspondiente al resultado de la ALU, sino se toma el valor proveniente del MuxPC.

A su vez, cuando la instrucción `jreg` llega a la etapa MEM, las instrucciones que le siguen se encuentran en las etapas anteriores. Es por esto que es necesario realizar un flush sobre dichas etapas para evitar un control hazard. Esta acción ya se encuentra implementada para el caso de las instrucciones de branch. Se tiene una compuerta AND (llamada AndBranch en DrMIPS) en la etapa MEM cuya salida se utiliza para indicarle a los registros de pipeline de las etapas anteriores (IF/ID y ID/EX) si se debe realizar un flush (esto ocurre en caso que se tome el branch). Por ello, se conectó la salida de esta compuerta a una nueva compuerta OR, que toma como segunda entrada la señal Jump del registro EX/MEM (comentada anteriormente) y cuya salida le indica a las etapas anteriores si hay que realizar el flush. Esto sucede cuando la señal Jump es 1 (es decir que se está ejecutando la nueva instrucción implementada) o cuando la salida de la compuerta AND mencionada es 1 (es decir si se da el caso de un branch taken).

En este caso, no fue necesario utilizar la señal Branch dado que no había que diferenciar nuestra instrucción de otra instrucción de salto como `j Imm`.

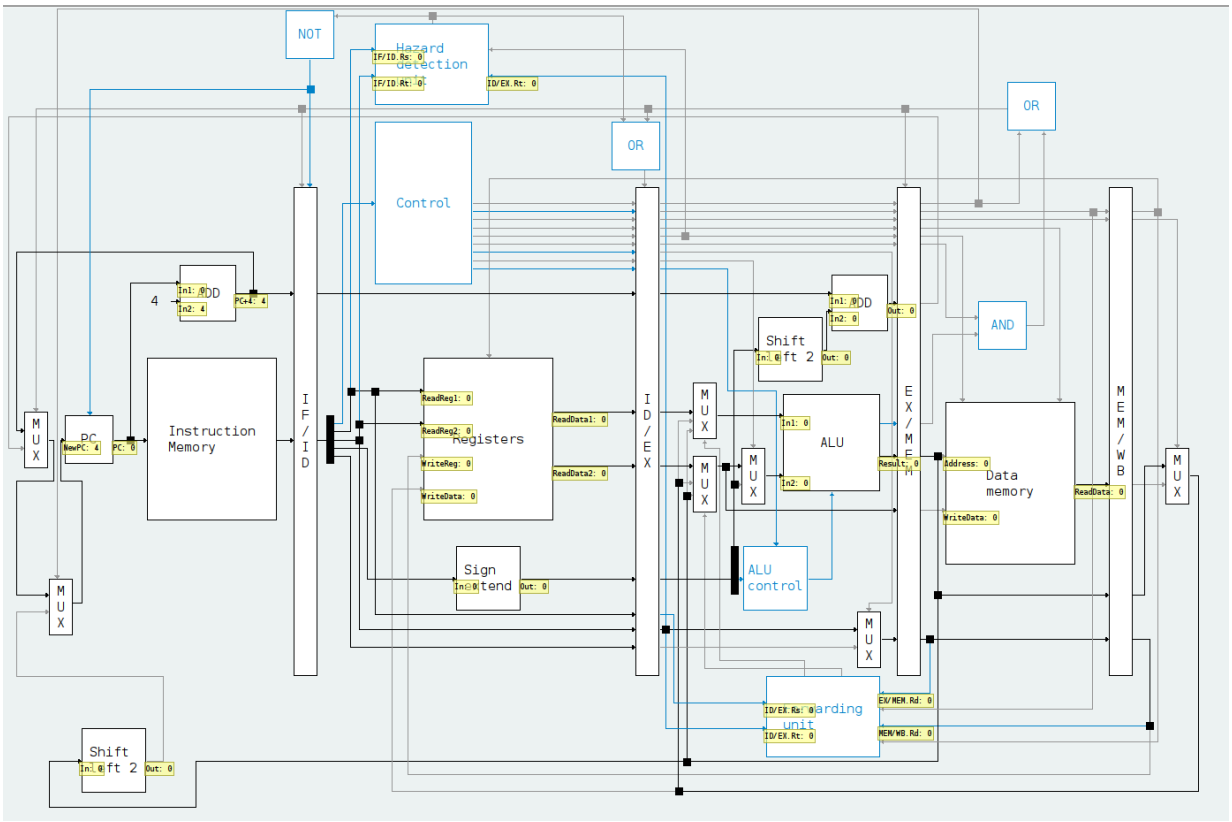

Figura 4: DP multiciclo para la instrucción `j Rs, Rt`

2.3. `lw Rs, Rd*Imm(Rt)`

2.3.1. DP pipeline

Se encontraron dos inconvenientes a la hora de implementar esta nueva instrucción. En primer lugar, la misma implica realizar una multiplicación entre el valor almacenado en el registro Rd y el valor Imm. Por ello, se deben realizar dos operaciones ya que el resultado de la misma es almacenado en un registro al cual solamente se puede acceder a traves de la operación mflo. Esto significa que se debe realizar más de una operación por lo que la instrucción no se puede implementar de forma "nativa" y debe ser implementada como una pseudo instrucción.

En segundo lugar, no se puede utilizar la firma de la instrucción tal cual se la indica en el enunciado. Esto se debe a que DrMIPS no acepta la descomposición de un argumento del tipo "data" en offset y base para ser pasados como argumentos de una instrucción dentro de una pseudo instrucción. Por ejemplo, si el argumento #3 es de tipo "data" y quiero realizar la suma del offset con el argumento #2, cuando se escribe `addi #2, #2, #3.offset`, DrMIPS lanza un error. Por lo tanto, se tuvo que separar el argumento de tipo "data" en dos argumentos: uno de tipo "int", que corresponde al valor inmediato Imm, y otro de tipo "reg", que corresponde al registro Rt. La firma de la función implementada es entonces `lw Rs, Rd, Imm, Rt`. Finalmente, como ya existe la instrucción lw en DrMIPS, se modificó el nombre de la misma a lws.

Como la pseudo instrucción solamente llama a instrucciones y pseudo instrucciones ya implementadas por DrMIPS, no fue necesario modificar el datapath. Se utilizó el registro Rs para guardar los resultados de las operaciones intermedias ya que en este se debe almacenar el dato final obtenido por la pseudo instrucción, por lo que no se debe preservar el valor que tenía anteriormente.

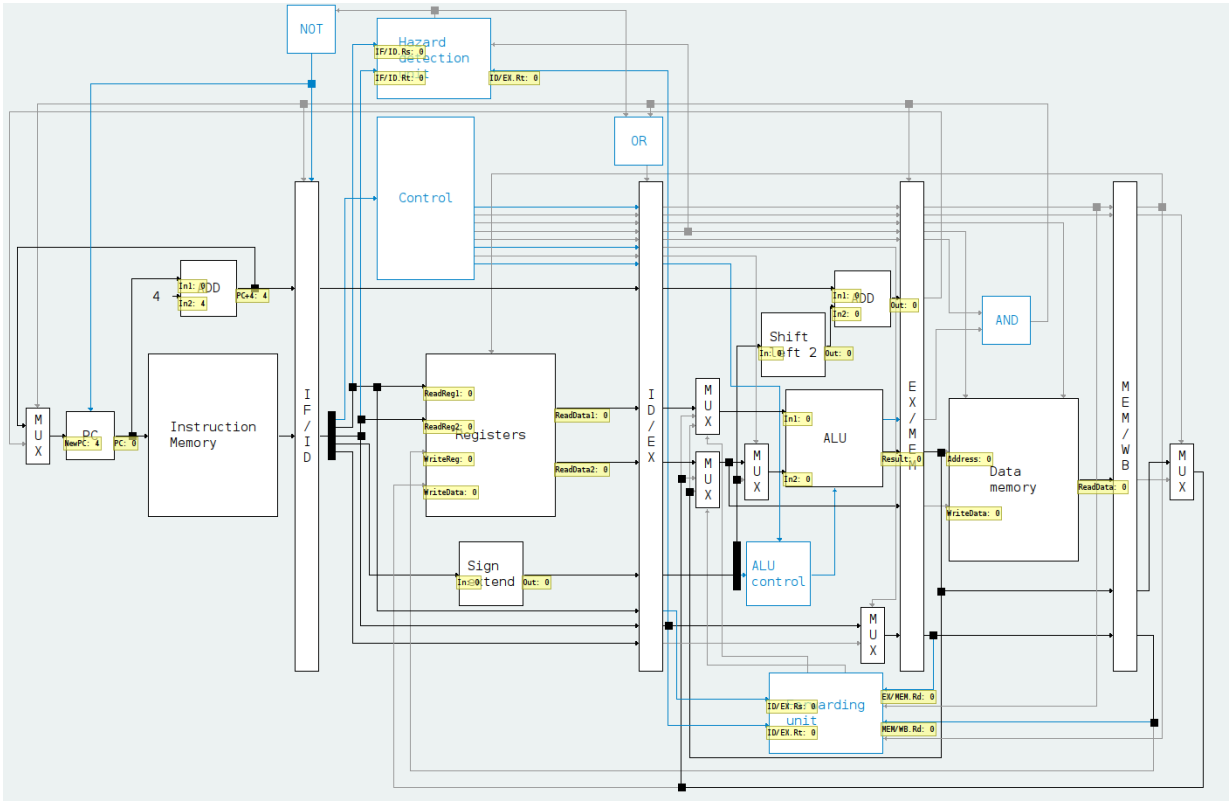

Figura 5: DP multiciclo para la instrucción `lw Rs, Rd, Imm, Rt`

3. Casos de prueba

Para las instrucciones que fueron implementadas en dos datapaths diferentes se utilizó el mismo caso de prueba.

3.1. `andi Rs, Rt, Imm`

```
li $t0, 7
andi $t1, $t0, 10
```

El registro t0 contiene el valor 7 mientras que el valor inmediato utilizado es 10, siendo la representación de estos en binario 0111 y 1010 respectivamente. Por lo tanto, el resultado almacenado en el registro t1 debe ser 2, cuya representación en binario es 0010 y es el resultado de realizar un and lógico entre los dos números mencionados anteriormente.

3.2. `j Rs, Rt`

```
li $t2, 1
li $t0, 1
li $t1, 7
jreg $t1, $t0
addi $t2, $t2, 1
addi $t2, $t2, 1
addi $t2, $t2, 1
addi $t2, $t2, 1
addi $t2, $t2, 1
```

En este caso, al ejecutar la instrucción jreg, los registros t0 y t1 tienen contienen los valores decimales 1 y 7 respectivamente, por lo que se tiene que saltar a la instrucción 8 (numerándolas desde 0 dado que las posiciones de memoria comienzan en 0). Es decir que se salta a la última instrucción del programa y el valor almacenado en el registro t2 es 2 dado que se llama a la instrucción `addi $t2, $t2, 1` una única vez y al comienzo del programa se carga el valor decimal 1 en dicho registro.

3.3. `lw Rs, Rd*Imm(Rt)`

```
.data
uno: .word 1
dos: .word 2
tres: .word 3

.text
li $t0, uno
li $t1, 4
lws $t2, $t1, 2, $t0
```

En el registro t1 se almacena un 4 dado que cada uno de los números es una palabra, es decir que ocupa 4 bytes. Como el valor Imm es 2, se deberá acceder a la segunda palabra (offset de 2 * 4 bytes = 8 bytes) después de la que se encuentra almecenada en la dirección de memoria correspondiente al label "uno". Esto significa que se carga el valor 3 en t2.

4. Conclusiones

Este trabajo permitió aprender el uso de DrMips y comprender, tanto en la teoría como en la práctica, las distintas maneras de implementar el Datapath.

Se notó una gran facilidad para entender los conceptos al utilizar DrMips. Esta herramienta es de un gran valor a la hora de aprender temas así de complejos.

Ver el funcionamiento de ambos tipos de caminos deja una gran idea de como estos se comparan y las ventajas y desventajas de cada uno.

66:20 Organización de computadoras
Trabajo práctico 3: Data path y pipeline.

1. 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 deberán agregar instrucciones a diversas configuraciones de CPU provistas por el simulador DrMIPS [1]

2. Alcance

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

3. 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.

El informe deberá respetar el modelo de referencia que se encuentra en el grupo[1], y se valorarán aquellos escritos usando la herramienta TEX / LATEX.

4. Recursos

Usaremos el programa DrMIPS [1] para configurar y simular el data path de un procesador MIPS [3], tanto uniciclo como multiciclo.

5. Descripción.

5.1. 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

[1] `http://groups.yahoo.com/group/orga6620`

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://brunonova.github.io/drmips/` o descargar directamente desde los repositorios de Ubuntu con `sudo apt install drmips`.

5.2. Datapaths

DrMIPS viene con algunos DP ya implementados, a saber:
Uniciclo:

- `unycicle.cpu`: El DP uniciclo por defecto.
- `unycicle-no-jump.cpu`: Variante más simple del DP uniciclo que no soporta la instrucción `j`.
- `unycicle-no-jump-branch.cpu`: Una variante aún más simple que no soporta `jump` ni `branch`.
- `unycicle-extended.cpu`: Una variante que soporta instrucciones adicionales, como multiplicación y división.

Multiciclo:

- `pipeline.cpu`: El DP de pipeline por defecto, implementa detección de hazards. Los DP de pipeline no soportan la instrucción `j` (salto).
- `pipeline-only-forwarding.cpu`: Variante del DP de pipeline que implementa forwarding pero no genera stalls (genera resultados incorrectos).
- `pipeline-no-hazard-detection.cpu`: Otra variante que no hace hazard detection de ninguna manera (genera resultados incorrectos).
- `pipeline-extended.cpu`: Una variante que soporta instrucciones adicionales, como multiplicación y división, como `unycicle-extended.cpu`.

5.3. Instrucciones a implementar

A continuación se describen las instrucciones a implementar en el data path. Algunas de estas existen como instrucciones de MIPS pero no están implementadas; otras no.

- `andi Rs, Rt, Imm` (And immediate). Esta instrucción de tipo `I` carga en `Rs` el resultado de hacer un `AND` entre el contenido del registro `Rt` y el valor de 16 bits `Imm`.
- `j Rs, Rt` (Jump Rs+Rt). Esta instrucción de tipo `I` carga en el PC el resultado de sumar los contenidos de los registros `Rs` y `Rt`.
- `lw Rs, Rd*Imm(Rt)`. Esta instrucción de tipo `R` carga en el registro `Rs` la palabra de 32 bits cuya dirección es `Rt` + `Rd` * `shamt`. Esto resulta en el agregado del modo de direccionamiento escalado, que MIPS no tiene nativamente.

5.4. Tareas a realizar

1. Implementar la instrucción `andi Rs, Rt, Imm` en el DP `unicycle.cpu`.
2. Implementar la instrucción `andi Rs, Rt, Imm` en el DP `pipeline.cpu`.
3. Implementar la instrucción `j Rs, Rt` en el DP `unicycle.cpu`.
4. Implementar la instrucción `j Rs, Rt` en el DP `pipeline.cpu`. Verificar que no se produzcan hazards.
5. Implementar la instrucción `lw Rs, Rd*Imm(Rt)` en el DP `pipeline.cpu`.

6. Implementación.

Los archivos antes mencionados, así como los archivos `.set` que contienen los datos del conjunto de instrucciones, están en formato JSON [2], y se pueden modificar con un editor de texto. Se sugiere uno que pueda hacer *color syntax highlighting*, como el `gedit` que viene con el Ubuntu. La explicación de los formatos se encuentra en el archivo `configuration-en.pdf` que se distribuye con el programa. Para cada instrucción, implementarla si es posible como instrucción "nativa" y en caso de no ser posible, justificarlo e implementarla como pseudoinstrucción.

7. Pruebas

En todos los casos debe verificarse que la instrucción se ejecute correctamente. Esto implica que el registro de destino 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.

8. Informe.

Se debe entregar:

- Informe describiendo el desarrollo del trabajo práctico.
- Capturas de pantalla de los DP modificados.
- Programas de prueba.
- Archivos de los DP, los programas de prueba y los conjuntos de instrucciones usados en cada caso.
- Este enunciado.

9. Fecha de entrega.

La entrega de este trabajo práctico debe realizarse el Jueves 30 de Julio de 2020.

Referencias

[1] DrMIPS, `https://brunonova.github.io/drmips/`.

[2] ECMA-404 The JSON Data Interchange Standard, `http://www.json.org/`.

[3] "Computer organization and design: the hardware-software interface", John Hennessy, David Patterson. Capítulo 5.