Trabajo Práctico 2 — Data path y pipeline

[66.20] Organización de Computadoras
Curso 2
Primer cuatrimestre de 2021

Alumnos	Padrón	Email
ARRACHEA, Tomás	104393	tarrachea@fi.uba.ar

Índice

1. Introducción **2**

2. Diseño y detalles de implementación **2**
2.1. Instrucción andi Rs, Rt, Imm 2
2.1.1. DP uniciclo 2
2.1.2. DP pipeline 3
2.2. Intrucción j Rs, Rt 3
2.2.1. DP uniciclo 3
2.2.2. DP pipeline 4
2.3. Instrucción lw Rs, Rd, Rt 5
2.3.1. DP pipeline 5

3. Casos de prueba **6**
3.1. andi Rs, Rt, Imm 6
3.2. j Rs, Rt 6
3.3. lw Rs, Rt, Rd 7

4. Conclusiones **7**

1. Introducción

En este Trabajo Práctico se implementarán las siguientes instrucciones para la arquitectura MIPS utilizando dos diseños distintos de data-path: uniciclo y con pipeline.

- 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.
- j Rs, Rt: Instrucción de tipo I que carga en el PC el resultado de sumar los contenidos de los registros Rs y Rt.
- lw Rs, Rd, Rt: instrucción de tipo R que carga en el registro Rd la palabra de 32 bits cuya dirección es Rs + Rt.

2. Diseño y detalles de implementación

2.1. Instrucción andi Rs, Rt, Imm

2.1.1. DP uniciclo

Para implementar esta instrucción fue necesario agregar una nuevo valor de ALUOp correspondiente al número decimal 3 (ALUOp tiene asignados 2 bits así que no hay problema), que realice una operación de tipo AND. Esto se debe a que sólo el valor de ALUOp 2 tiene asociado la operación AND, pero usando el campo func de las instrucciones de tipo R para distinguirlo, que 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 realiza una suma en lugar de realizar un AND lógico entre ambos valores.

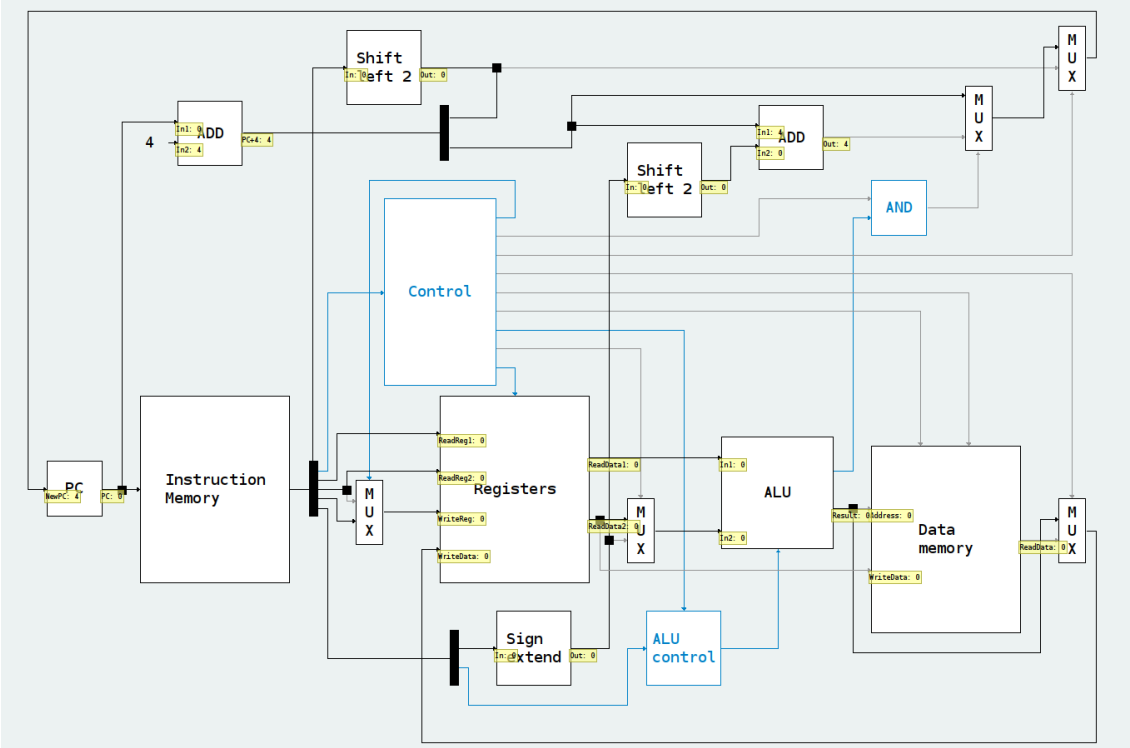

Figura 1: DP uniciclo para la instrucción andi Rs, Rt, Imm.

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.

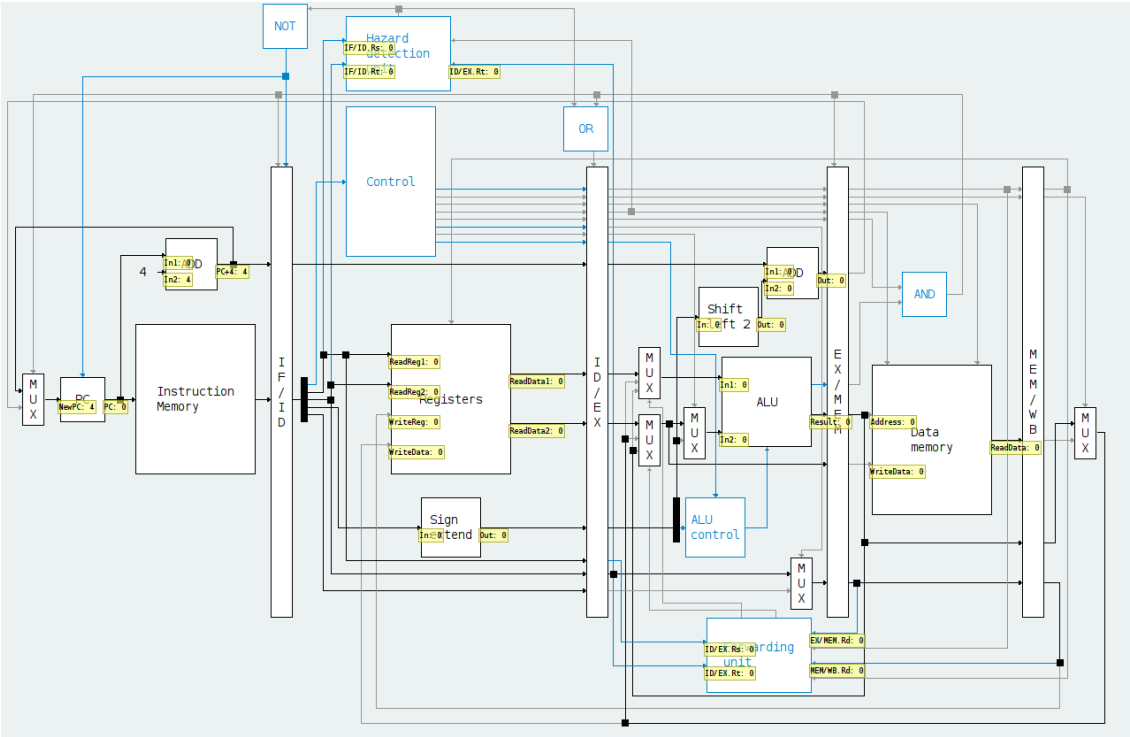

Figura 2: DP multiciclo para la instrucción andi Rs, Rt, Imm.

2.2. Intrucción j Rs, Rt

En el set de instrucciones del DP uniciclo ya está implementada la instrucción *j Imm* de MIPS, por lo que se renombró esta instrucción a *ji*, para que no se produzca un error por clave repetida.

También se estableció que el resultado de la suma de los valores de los registros indique el número de instrucción (comenzando en 0 porque se buscan posiciones de memoria) por lo que el mismo se debe multiplicar por 4, dado que las instrucciones ocupan 4 bytes.

2.2.1. DP uniciclo

La única diferencia entre la instrucción de j ya implementada y la nueva instrucción consiste en cómo se obtiene la dirección. 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.

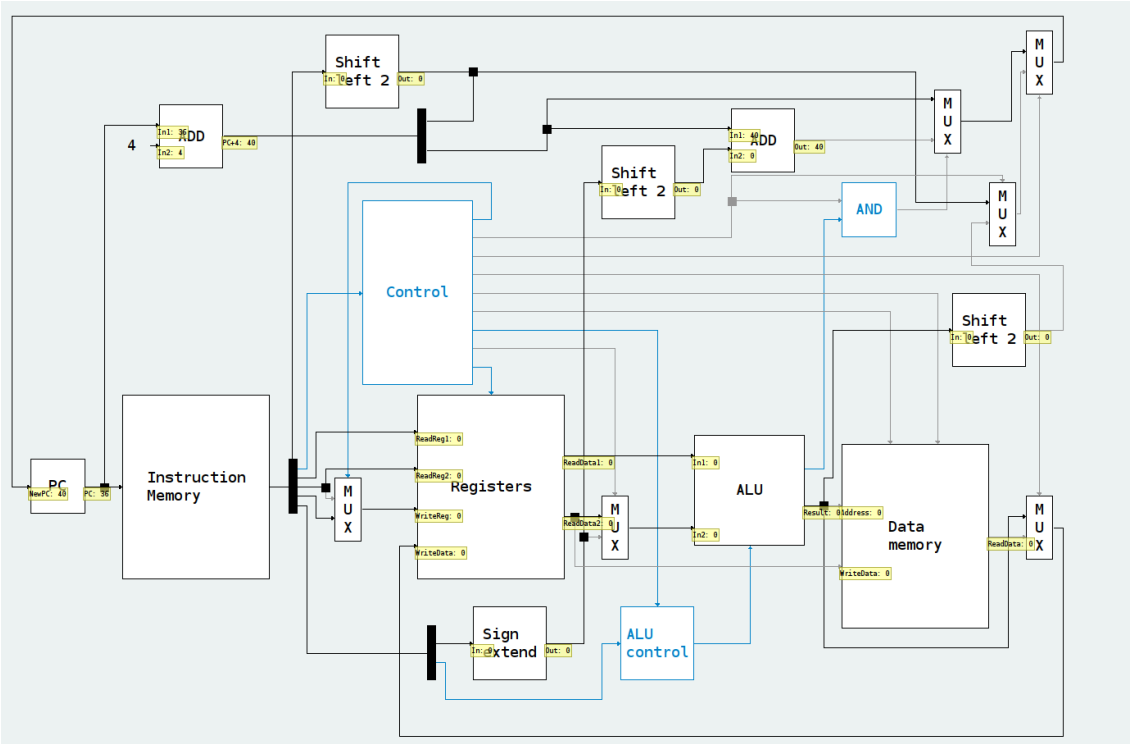

Figura 3: DP uniciclo para la instrucción j Rs, Rt.

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), que 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 uniciclo). 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.

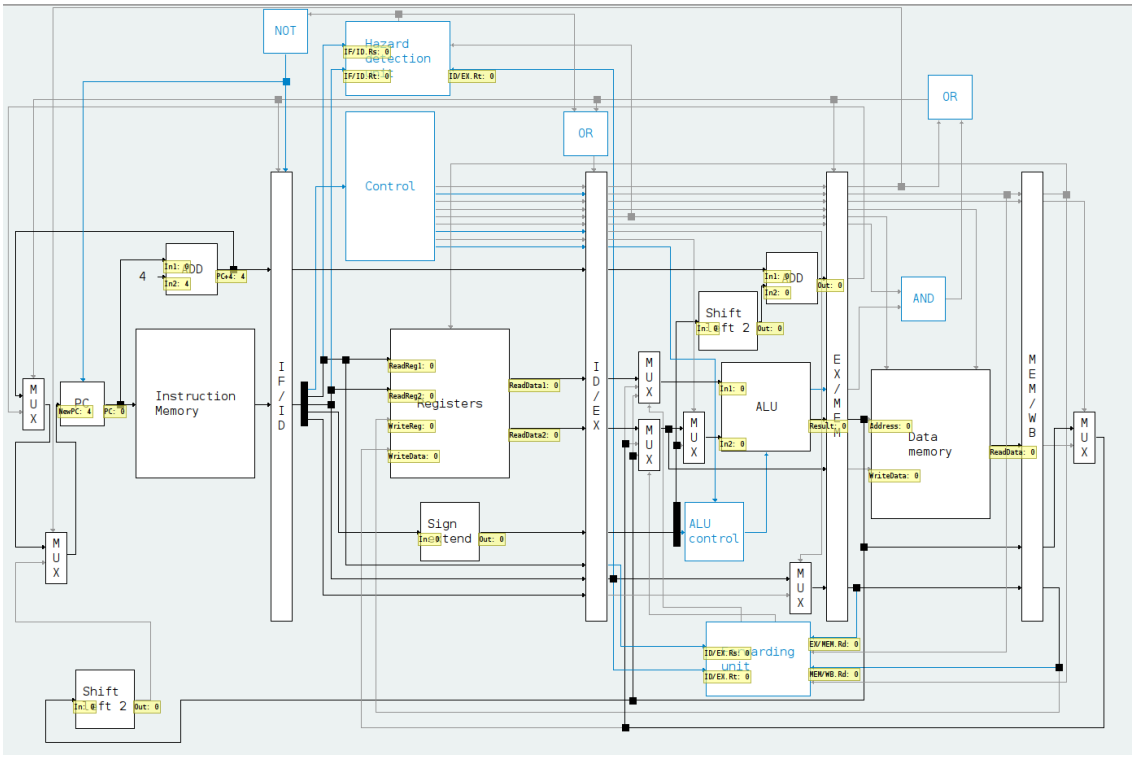

Figura 4: DP multiciclo para la instrucción j Rs, Rt.

2.3. Instrucción lw Rs, Rd, Rt

2.3.1. DP pipeline

El DP tiene implementada la instrucción *lw Rs, (Rd)*, así que se cambió su nombre a *lwi*.

Para implementar el load indexado se agregó la instrucción y un nuevo código para la unidad de control. La instrucción suma el contenido de los registros Rs y Rd, y ese resultado es buscado en memoria.

El DP tiene ya implementadas las instrucciones originales ld y sw, de forma que se envía el número de registro que se está modificando al Hazard Control Unit, para saber si hay que insertar un stall en el proceso. En este caso hubo que hacer una modificación, ya que el registro que ingresa al Hazard Control Unit puede ser el Rt, o en este caso también el Rd. Por lo tanto, se modificó el cable de entrada. En lugar de ser siempre el registro Rt, el número de registro será el de Rd o Rt, dependiendo de cuál es el registro de destino, indicado en el bit de RegDst. Esto permite que las instrucciones de load y store puedan manejarse con el resultado de la ALU, característica que no está contemplada en el DP original.

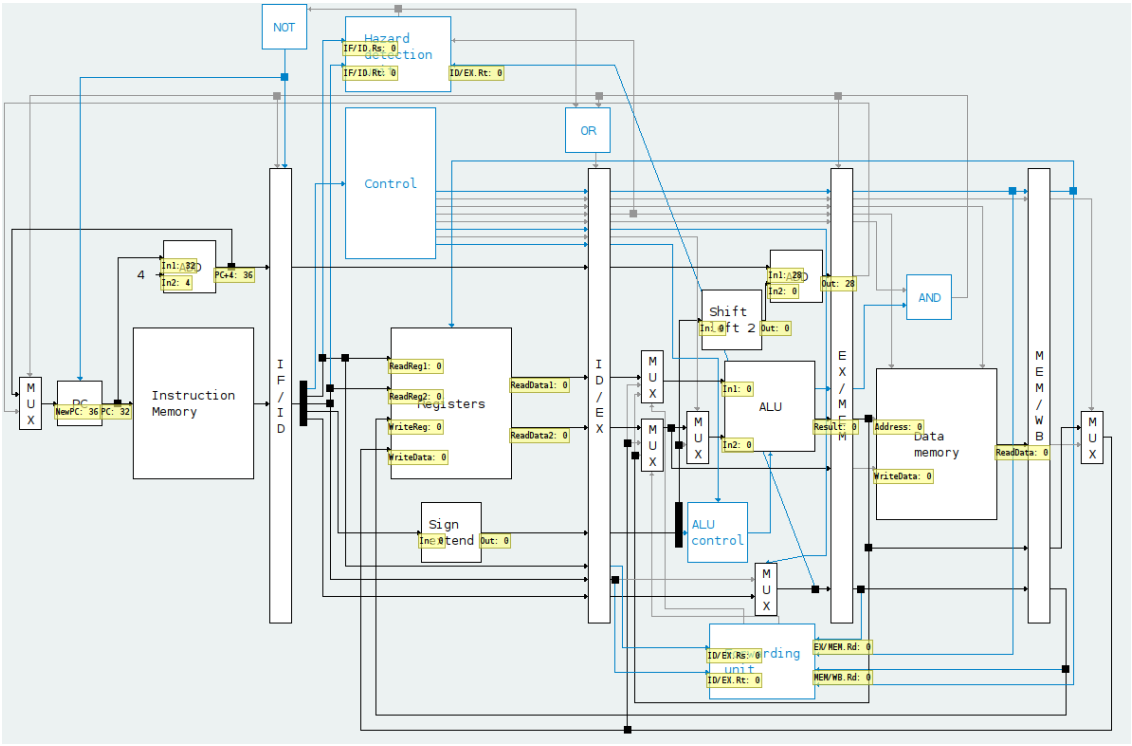

Figura 5: DP multiciclo para la instrucción lw Rs, Rt, Rd.

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

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

prueba_andi.S

El registro t0 contiene el valor 7 mientras que el valor inmediato utilizado es 10, siendo su representación en binario 0111 y 1010, respectivamente. Entonces, el resultado almacenado en el registro t1 debe ser 2, cuya representación en binario es 0010 y es el resultado un and lógico entre los dos números.

3.2. j Rs, Rt

```
1 li $t2, 1
2 li $t0, 1
3 li $t1, 7
4 j $t1, $t0
5 addi $t2, $t2, 1
6 addi $t2, $t2, 1
7 addi $t2, $t2, 1
8 addi $t2, $t2, 1
```

```
9 addi $t2, $t2, 1
```

prueba_j.S

En este caso, al ejecutar la instrucción j, los registros t0 y t1 contienen los valores decimales 1 y 7 respectivamente, por lo que se tiene que saltar a la instrucción 8. Es decir que se salta a la última instrucción del programa y el valor almacenado en el registro t2 es 2.

3.3. lw Rs, Rt, Rd

```
1  .data
2      cero: .word 0
3      uno: .word 1
4      dos: .word 2
5
6  .text
7      li $t0, 4
8      li $t1, 4
9      lw $t2, $t1, $t0
10     add $t3, $t1, $t2
```

prueba_lw.S

Al ejecutar la prueba, se esepra En los registros t0 y t1 se almacena un 4. Se hace un load con ellos, por lo que se debería leer la dirección 8 de memoria, que contiene el valor 2.Luego, para verificar que funcione bien el fowarding, se hace un cálculo con el registro t2, que debería resultar en guardar un 6 en el registro t3.

4. Conclusiones

Para implementar las instrucciones de j y lw se tuvo que modificar los data paths, mientras que la implementación del andi fue más sencilla, y similar a las ya existentes. Al modificar los data paths multiciclo se tuvo que prestar especial atención a los hazards.

Este trabajo permitió aprender el uso de DrMips y comprender, tanto en la teoría como en la práctica, las distintas maneras de implementar un Datapath y sus complejidades. Ver el funcionamiento de ambos tipos de caminos deja una buena idea de cómo se comparan, y sus ventajas y desventajas.