# Trabajo Practico 3
# Data Path

66.20
Organización de Computadoras

Integrantes:
Franco Tomas
Marinaro Santiago

Fecha de entrega 28/06/2018

# 1 Enunciado

Se adjunta junto al informe

# 2 Introducción

El propósito de este trabajo practico es familiarizarse con la arquitectura de una CPU MIPS, especialmente el DATA PATH y la implementacion de las instrucciones. Para elle se utilizo el programa DrMips que simula simula el CPU.

El siguiente es un link al repositorio del Trabajo Practico en el que se encuentran las implementaciones pedidas:
https://github.com/Santiago39353535/TP-3-Orga

# 3 Desarrollo

En esta sección se desarrollara las distintas implementaciones pedidas para el trabajo.

## 3.1 Modificación del Data Path

### 3.1.1 Instrucción j en el pipeline

Para lograr que se realice el jump en el pipeline lo que se hizo fue agregar un multiplexor y conectarlo con lo que antes entraba al PC por un lado y por el otro entraba el calculo del jump realizado realizado en el EX/MEM. Para lograr el flush lo que se hizo fue usar un or en con las entradas del jump del pieline y el resultado de la and del branch, y con salida a los pipelines.

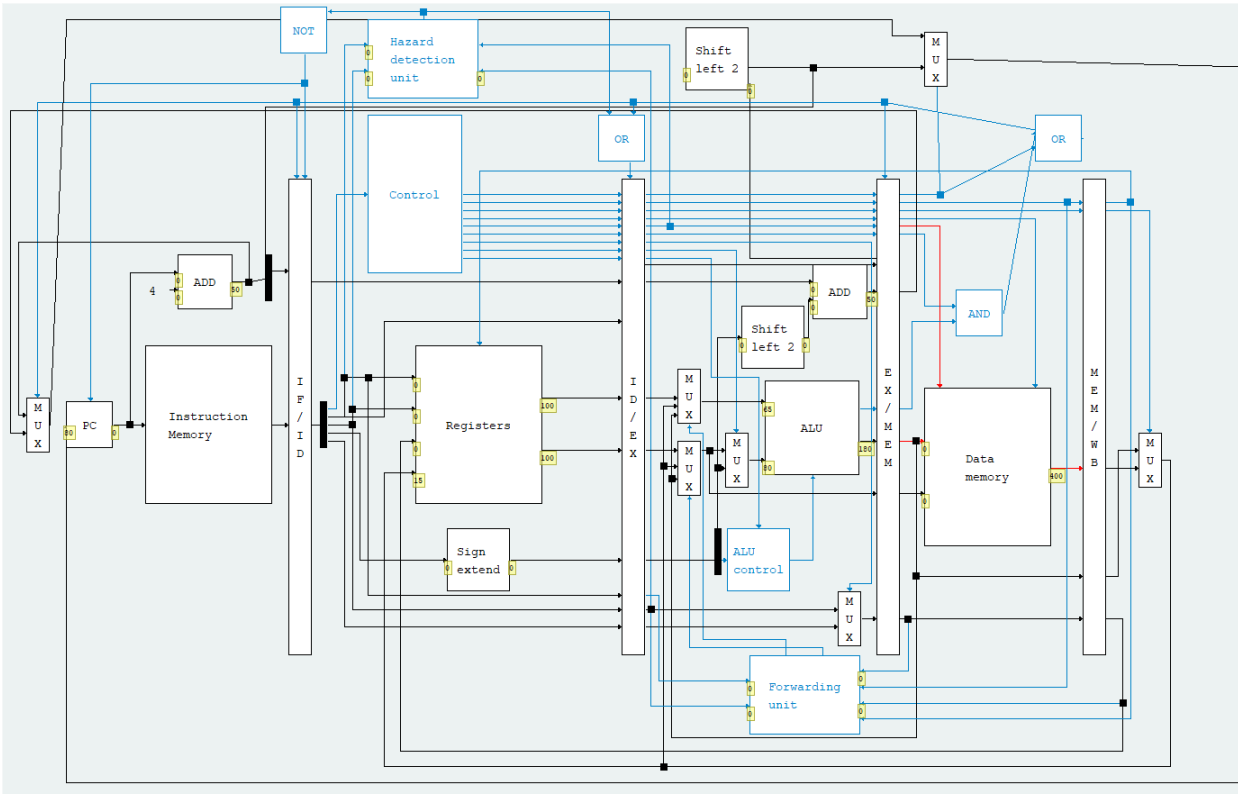


Figure 1: Modificación del data path para el jump

### 3.1.2 Instrucción jr en el uniciclo

Para lograr el jump register se conecto un cable que salia desde ReadData1 de RegBank hacia un multiplexor en la entrada 1 y en la entrada 0 se encuentra la entrada original al PC. Del multiplexor se conecto al Pc.

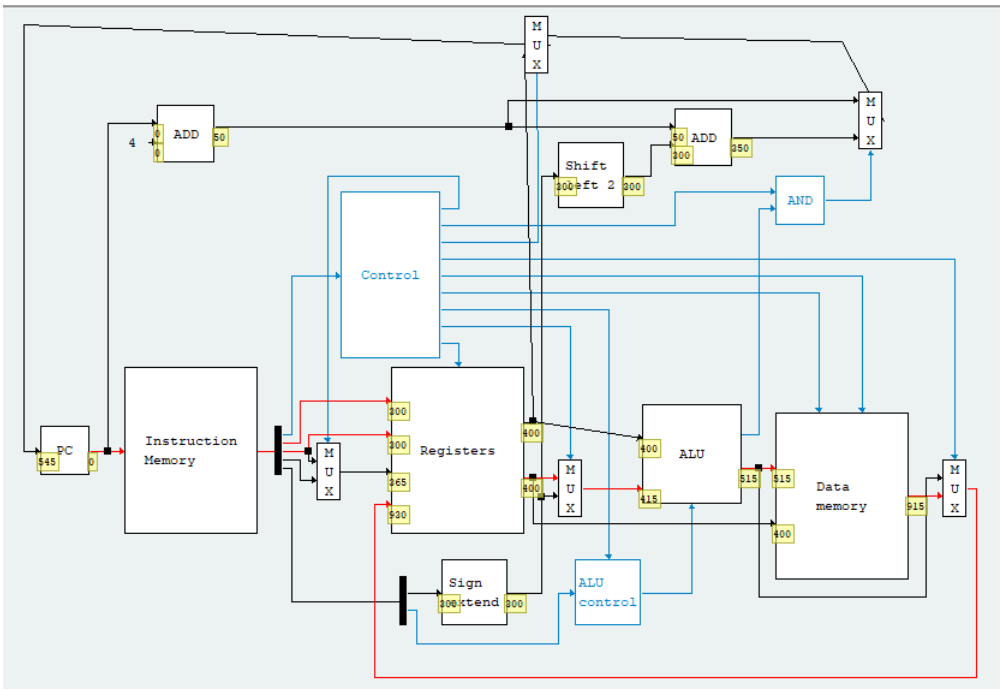


Figure 2: Modificación del data path para el jr

### 3.1.3 Instrucción jr en el pipeline

Muy similar al anterior. El registro se saca de ReadData1 del RegBank antes de que se entre al pipeline ID/EX. Para el flush se obtiene de un or entre la entrada anterior del flush de IF/ID con el jump del control.

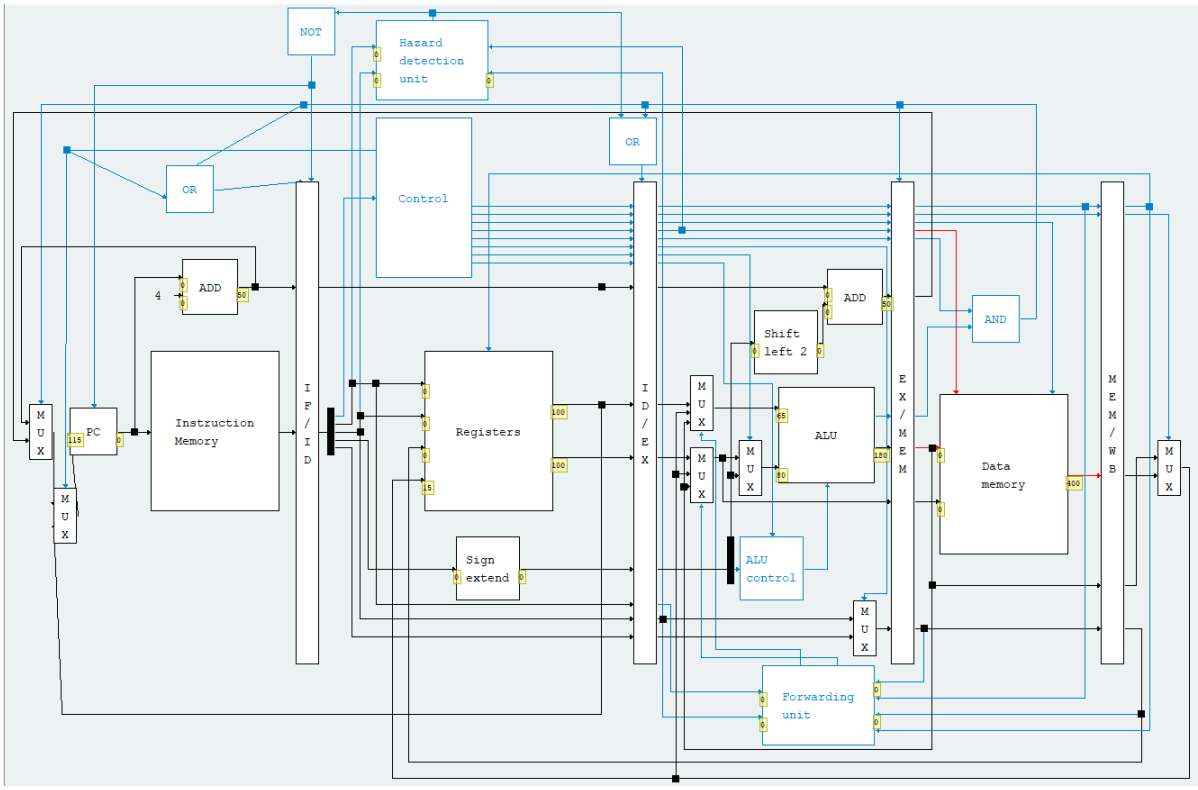


Figure 3: Modificación del data path para el jr

### 3.1.4 Instrucción jarl en el uniciclo y Instrucción jarl en el pipeline

Para estos dos no se realizo ninguna modificacion sobre el data path ya que fueron implementados como psudointruciones.

## 3.2 Implementacion de las instruciones

### 3.2.1 Instrucción j en el pipeline

Para la instrucción j se agrego al set de instrucciones la siguiente linea

```
"j":    {"type": "J", "args": ["target"], "fields": {"op": 2, "target": "#1"},
"desc": "PC = target"}
```

Y se agrego en control lo siguiente:

”2”: ”Jump”: 1

En la ejecución del jump en el pipeline genera un hazard, que fue corregido haciendo un flush de todos los pipeline menos el MEM/WB. Esto sucede ya

que el pipeline mientras se ejecuta una instrucción, ya se empieza a trabajar las siguientes.
Los ejemplos utilizados para testear fueron:

```
    j Target
addi $t1, $zero, 5
addi $t2, $zero, 5
addi $t3, $zero, 5
addi $t3, $zero, 5


 Target:
  addi $t0, $zero, 5
```

```
addi $t0, $zero, 5
j Target
nop
nop
nop
nop
addi $t1, $zero, 5
Target:
addi $t5, $zero, 5
```

### 3.2.2 Instrucción jr en el uniciclo

Para esta instruccion se agrego la siguiente linea en el set de intrcciones:

”jr”: ”type”: ”R”, ”args”: [”reg”], ”fields”: ”op”: 3, ”rs”: ”1”, ”rt”: ”0”, ”rd”: ”0”, ”shamt”: 0, ”func”: 8, ”desc”: ”The jr instruction loads the PC register with a value stored in a register.”

Y en el control lo siguiente:

”3”: ”Jump”: 1,
La instrucción jarl tiene un campo 0p igual a 3.
El ejemplo para testeo fue:

```
addi $t0, $zero, 16
```

```
jr $t0
addi $t1, $zero, 8
addi $t5, $zero, 8
addi $t4, $zero, 8
```

### 3.2.3 Instrucción jr en el pipeline

Se agrego lo mismo que en la sección anterior ya que lo que tienen que hacer es lo mismo. Pero en el pipeline se produjo un hazzard a diferencia del Unicycle, dado que a que a medida que esta ejecutando la instrucción ya esta trabajando en otra. Este hazzard es corregido haciendo un flush del pipeline IF/ID.

### 3.2.4 Instrucción jarl en el uniciclo y Instrucción jarl en el pipeline

Para estas instrucciones lo que se agrego fue:

"jalr": "args": ["reg", "reg"], "to": ["addi 1, 0,ra", "jr 2"], "jalr": "args": ["reg"], "to": ["addi 31, 0,ra", "jr 1"]

En la sección de pseudo del archivo .set.

El ejemplo para el testeo fue:

```
addi $t0, $zero, 24
jr $t0
nop
nop
addi $t1, $zero, 8
addi $t5, $zero, 8
addi $t4, $zero, 8
```

## 3.3 Conclusiones

Este trabajo cumplió el propósito de entender las distintas implementaciones del Data Path al igual que los inconvenientes que cada uno puede tener. Como los jumps y branches que generan hazzards en el pipeline.

# 66:20 Organización de computadoras
# Trabajo práctico 3: Data Path.

## 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 [4], 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://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 (ver [2]).

## 5.2. Datapaths

El programa 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

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.

## 6. Implementación.

Los archivos antes mencionados, así como los archivos `.set` que contienen los datos del conjunto de instrucciones, están en formato JSON [3], 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.

## 7. 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.

## 8. Informe.

Se debe entregar:

- Informe describiendo el desarrollo del trabajo práctico.
- Capturas de pantalla de los DP modificados.
- Programas de prueba.
- CD conteniendo los DP, los programas de prueba y los archivos generados para cada caso.
- Este enunciado.

## 9. Fecha de entrega.

La fecha de entrega del TP es el Jueves 28 de Junio.

## Referencias


[1] DrMIPS, `https://bitbucket.org/brunonova/drmips/wiki/Home`.

[2] PPA de Bruno Nova, `https://launchpad.net/~brunonova/+archive/ubuntu/ppa`.

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

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