Trabajo Práctico 3
Data path y pipelines

Jonathan Medina, *Padrón 100052 Slack: U011C1EGWMD*
jcmedina@fi.uba.ar

Pablo Inoriza, *Padrón 94986 Slack: U011Q8GB197*
pinoriza@fi.uba.ar

Christian Flórez Del Carpio, *Padrón 91011 Slack: U011G8YJ18T*
chflorez@fi.uba.ar

1do. Cuatrimestre de 2020
86.37 / 66.20 Organización de Computadoras – Práctica Jueves
Facultad de Ingeniería, Universidad de Buenos Aires

1. Introducción

Este tp tenia como objetivo familiarizarse con la arquitectura MIPS, mas precisamente con el datapath y la implementacion de instrucciones. Se agrego diferentes instrucciones utilizando el simulador DrMIPS.

2. Diseño e Implementación

2.1. andi Rs, Rt, Imm en unicycle.cpu

Se definio en default.set la instruccion:

```
"andi": {
    "type": "I",
    "args": ["reg", "reg", "int"],
    "fields": {"op": 1, "rs": "#2", "rt": "#1", "imm": "#3", "func": 36},
    "desc": "$t1 = $t2 & imm"}
}
```

Y ademas en control se definio:

```
"1": {"RegDst": 0, "RegWrite": 1, "ALUOp": 2, "ALUSrc": 1, "MemToReg": 0}
```

Esto es porque dado que se quiere tomar el registro rt, se definio el RegDest en 0. Se quiere tambien tomar el Imm, por lo que se seteo ALUSrc en 1. No se tuvo que realizar ningun cambio en el datapath para agregar esta operacion. Se seteo el func 36, para poder obtener junto a al ALUOp la operacion 'and'.

2.2. andi Rs, Rt, Imm en pipeline.cpu

Se definio en default-no-jump.set la instruccion:

```
 "andi": {
     "type": "I",
     "args": ["reg", "reg", "int"],
     "fields": {"op": 1, "rs": "#2", "rt": "#1", "imm": "#3", "func": 36},
     "desc": "$t1 = $t2 & 23"
 }
```

Y en control:

```
"1": {"RegDst": 0, "RegWrite": 1, "ALUOp": 2, "ALUSrc": 1, "MemToReg": 0},
```

Esto es por lo dicho anteriormente. Ademas, no hubo necesidad de realizar modificaciones en el datapath.

2.3. j Rs, Rt en unicycle.cpu

Para implementar está instrucción se agregó un multiplexor(**MuxRegJump**) que mediante un entrada de control, decide si la siguiente dirección de PC es el resultado de la operación de la ALU, o es la que indica **MuxJump**. Y de está manera se puede sumar los registros indicados por parámetro, y que la siguiente dirección del PC sea dicha suma. Se tuvo que implementar todo esto ya que el datapath unicycle no provee estructuras suficientes para realizar dicha operación. La instrucción implementada se denominó **jalu**, porque ya existe la instrucción j en el set de instrucciones usado.

Entonces con los cambios realizados el grafico del datapath de unicycle.cpu quedo asi:

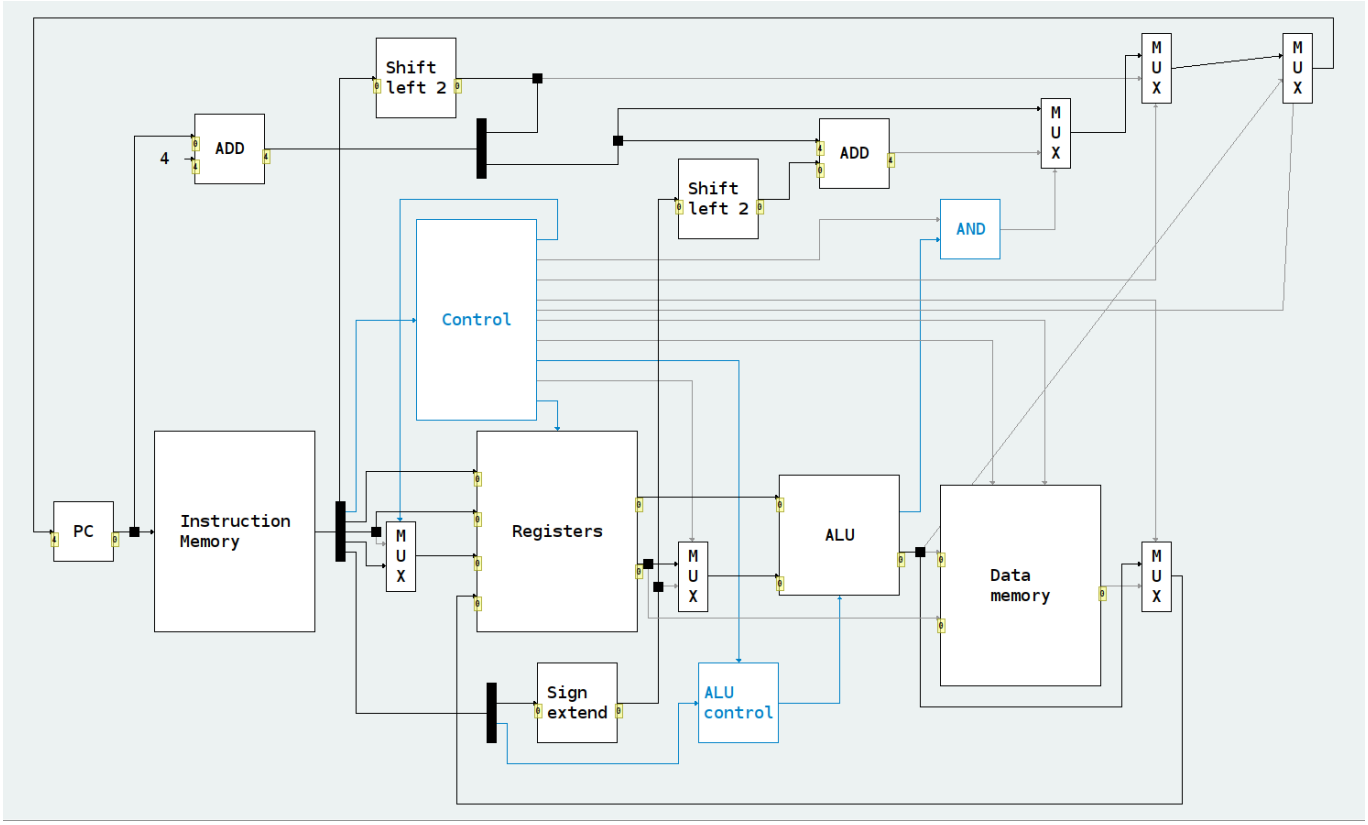

Figura 1: datapath unicycle.cpu

2.4. j Rs, Rt en pipeline.cpu

Se realizo de forma analoga que lo que se hizo en unicycle.cpu, ademas se aprovecho de como el beq evita los hazards, por lo que se utilizo una compuerta OR para seleccionar entre si es Branch o Jump, este realiza un flush, en los registros intermedios del pipeline. Añadiendo ciclos de Stall.

2.5. lw Rs, Rd*Imm(Rt) en pipeline.cpu

Se pedia que se cargue el valor en Rs de la direccion proporsionada por: Rt + Rd * shamt.

Esta instruccion presento diferentes problematicas:

- La instruccion lw ya esta definida, por lo que se definio como lwmod
- En MIPS no es posible definir una instruccion como fue pedida, por lo que se tomo que sea: lwmod rs, Rt, Rd, shamt.
- Se pedia realizar una suma, y una multiplicacion al mismo tiempo. Por lo que se decidio a utilizar pseudo instrucciones.
- La instruccion mul no era soportada por el pipeline provisto, por lo que se añadieron las instrucciones correspondientes para que pueda soportar esta instruccion, ademas se actualizo la ALU a una ExtendedALU dado que no soportaba la instruccion mult.
- Se utilizo el registro $1 (Assembler Temporary) para almcenar temporalmente el valor de las operaciones.

Entonces la pseudoinstruccion implementada fue:

```
"lwmod": {
    "args": ["reg", "reg", "reg", "int"],
    "to": ["li $1, #4","mul $1, $1, #3", "add $1, $1, #2", "lw #1, 0($1)"],
    "desc": "$t1 = $(Rt + Rd * shamt)"
}
```

Por lo tanto finalmente pipeline.cpu quedo:

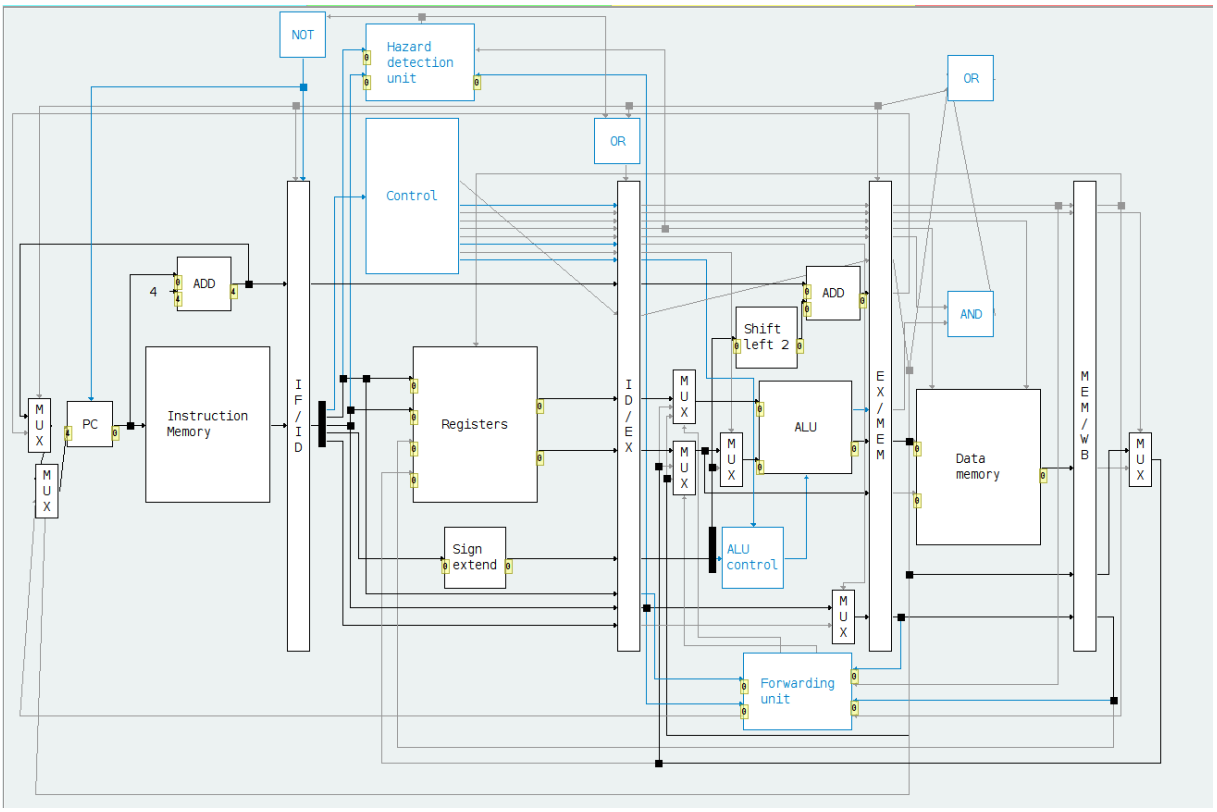

Figura 2: datapath pipeline.cpu

3. Casos de Prueba

3.1. Prueba andi Rs, Rt, Imm en unicycle.cpu y pipeline.cpu

```
addi $t1, $zero, 12
andi $t2, $t1, 3
```

Realizando la operacion manualmente se obtuvo:

```
 12 00001100 0xc
& 3 00000011 0x3
------------------------
  0 00000000 0x0
```

Efectivamente el valor en t2 fue 0.

Otra prueba:

```
addi $t1, $zero, 21
andi $t2, $t1, 19

  21 00010101 0x15
& 19 00010011 0x13
-------------------------
  17 00010001 0x11
```

Efectivamente el valor de t2 fue 17.

3.2. Prueba j Rs, Rt en unicycle.cpu y pipeline.cpu

En unicycle:

```
li $t1, 8
li $t2, 4

jalu $t1, $t2
```

Esto realiza un jump a la dirección t1 + t2, en este caso a 8 + 4 = 12, cuyo valor se almacena en el program counter (PC).

En pipeline:

3.3. Prueba lw Rs, Rd*Imm(Rt) en pipeline.cpu

```
li $t1, 20
li $t2, 30
sw $t2, 44($zero)
li $t3, 4

lwmod $t4, $t3, $t1, 2
```

Esto realiza un load word de la direccion 20*2+4 Efectivamente se escribe en el registro t4, el valor 30.

```
li $t3, 30
li $t1, 99
li $t2, 20
sw $t2, 30($zero)
lwmod $t4, $t3, $t1, 0
```

Efectivamente en el registro t4, se escribio el valor 20

4. Conclusión

El trabajo nos ayudó a entender como funciona el data path de un procesador MIPS, tanto de un unicycle como de un pipeline. Modificar el datapath, las estructuras y el cableado permitio entender mucho mejor el funcionamiento del procesador y de cada estructura en particular, ademas nos ayudo a entender los conceptos de Hazard, y refrescar conocimientos de Assembly MIPS, proporsionandonos una muy util herramienta para poder entender los conceptos del pipeline y data path.

Referencias

[1] D. Patterson, J. Hennessy, *Computer Organization and Design.*

[2] Dr MIPS configuration tutorial

A. Código fuente

El proyecto se encuentra en el siguiente repo de github: https://github.com/jomedina96/orgaTP1