FACULTAD DE INGENIERIA
Universidad de Buenos Aires

TP3 - Pipeline y Datapath

66.02 - Organización del Computador
Primer cuatrimestre de 2020

Integrante	Padrón	Mail
Torres Dalmas, Nicolás	98439	ntorresdalmas@gmail.com
Rodríguez Florencia	100033	florrr1997@gmail.com

Índice

1. Introducción **2**

2. Instrucciones implementadas **2**
2.1. ANDI . . . 2
2.1.1. Prueba de uso . . . 2
2.1.2. Datapath . . . 2
2.2. LW . . . 4
2.2.1. Prueba de uso . . . 4
2.2.2. Datapath . . . 4
2.3. JUMP . . . 5
2.3.1. Prueba de uso . . . 5
2.3.2. Datapath . . . 6

3. Conclusiones **7**

4. Código **7**

5. Material utilizado **7**

1. Introducción

En el siguiente informe se presentan las distintas instrucciones implementadas para las diversas configuraciones de CPU provistas por el simulador DrMIPS

2. Instrucciones implementadas

2.1. ANDI

andi Rs, Rt, Imm (And immediate).

Esta instrucción carga en Rs el resultado de hacer un AND entre el contenido del registro Rt y el valor de Imm.

Para que funcione tanto en pipeline como en unicycle, se agregó de forma nativa respectivamente en los archivos default-no-jump.set y defaul.set la siguiente declaración

```
"andi": {
        "type": "I",
        "args": ["reg", "reg", "int"],
                "fields": {"op": 7, "rs": "#2", "rt": "#1",
                "imm": "#3", "shamt": 0, "func": 36},
                "desc": "$t1 = $t2 & 23"
        },
```

Se puede ver que 'recibe' dos registros y un entero. La func 36 nos lleva la operación 0 de la ALU, que corresponde a and.

2.1.1. Prueba de uso

```
addi $t1, $t1, 1
addi $t2, $t2, 5
andi $t3, $t2, 5
andi $t4, $t1, 7
and $t5, $t3, $t4
```

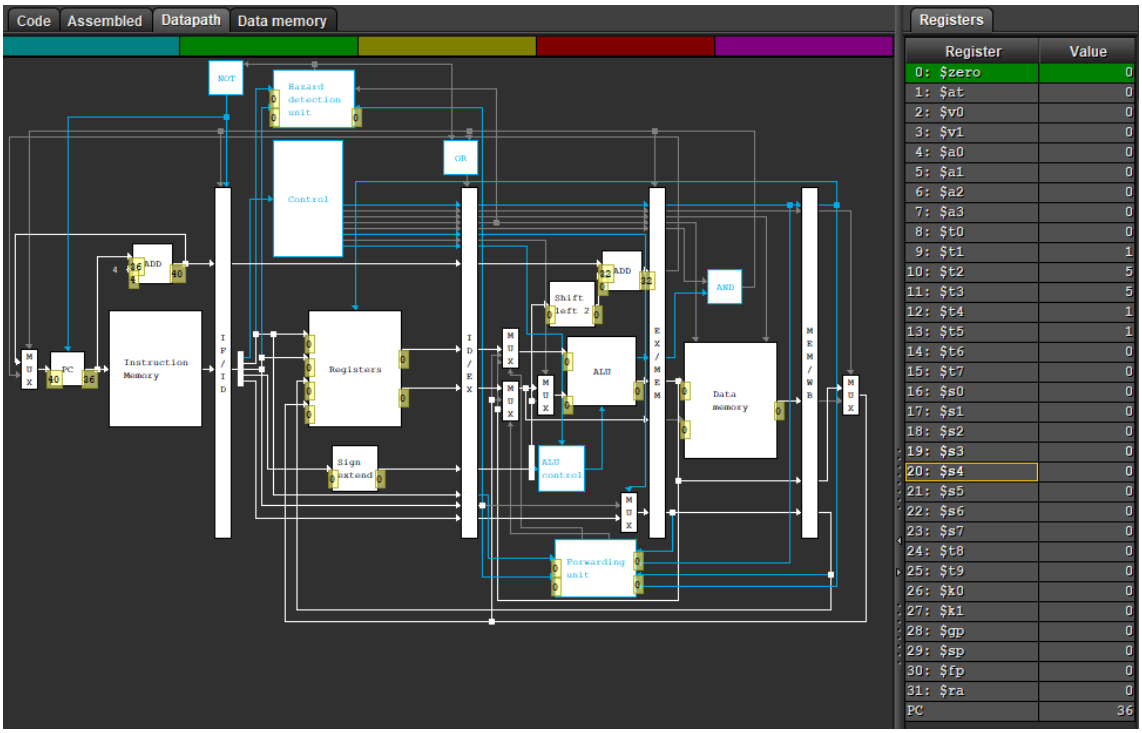
Para probar nuestra instrucción cargamos en $t1 el valor 1 y en $t2 el valor 5. Se puede observar qe hacer un andi entre t2 y el valor 5, por valer este ese mismo valor, resulta en t3 valiendo efectivamente 5. Por otra parte, t4 termina valiendo 1 ya que ese es el resultado de hacer 7&1.

La última linea fue utilizada para ver que no sucedieran hazars en nuestra instrucción. Pese a que en unicycle no podian suceder

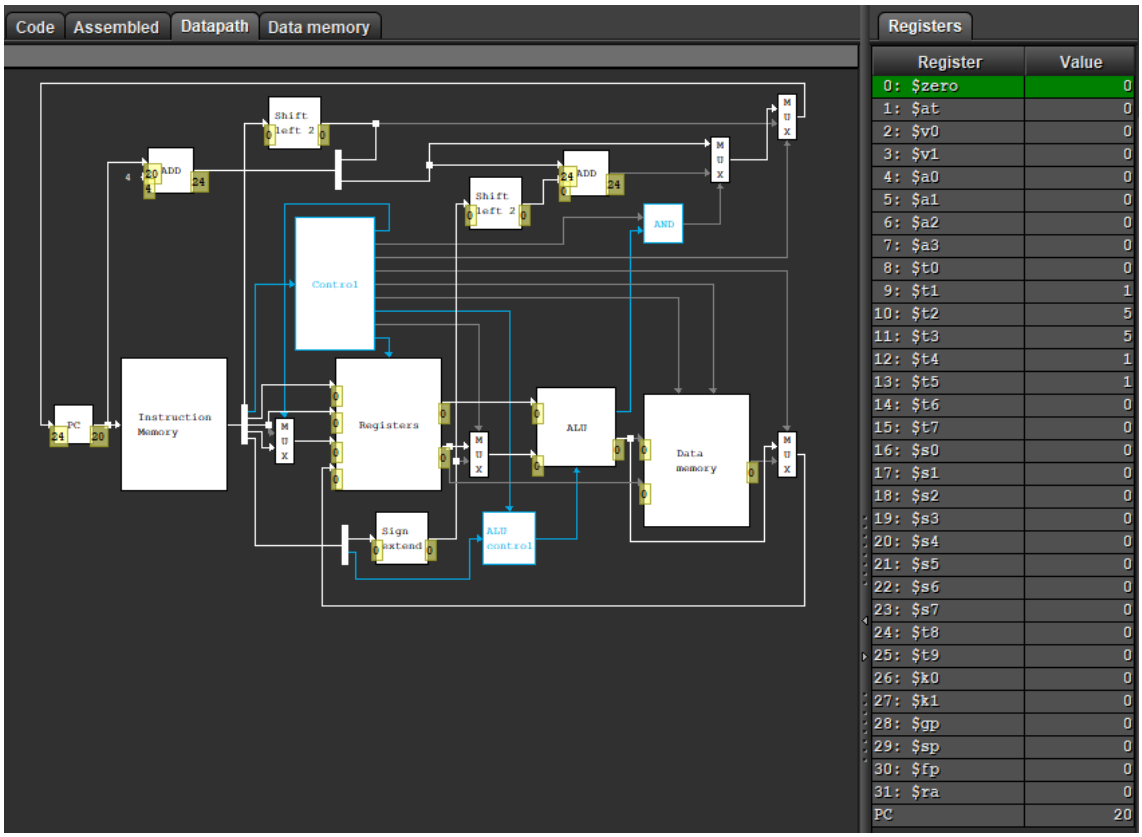
2.1.2. Datapath

Al implementar andi, no se realizó ningún cambio en el datapath, es decir, no se agregaron componentes extras para su correcto funcionamiento.

DP Pipeline


Code
Assembled
Datapath
Data memory
Registers
Register
Value
0: $zero 0
1: $at 0
2: $v0 0
3: $v1 0
4: $a0 0
5: $a1 0
6: $a2 0
7: $a3 0
8: $t0 0
9: $t1 1
10: $t2 5
11: $t3 5
12: $t4 1
13: $t5 1
14: $t6 0
15: $t7 0
16: $s0 0
17: $s1 0
18: $s2 0
19: $s3 0
20: $s4 0
21: $s5 0
22: $s6 0
23: $s7 0
24: $t8 0
25: $t9 0
26: $k0 0
27: $k1 0
28: $gp 0
29: $sp 0
30: $fp 0
31: $ra 0
PC 36
NOT
Hazard detection unit
OR
Control
ADD
Instruction Memory
Registers
Shift left 2
ALU
AND
Data memory
Sign extend
ALU control
Forwarding unit

DP Unicycle


Code
Assembled
Datapath
Data memory
Registers
Register
Value
0: $zero 0
1: $at 0
2: $v0 0
3: $v1 0
4: $a0 0
5: $a1 0
6: $a2 0
7: $a3 0
8: $t0 0
9: $t1 1
10: $t2 5
11: $t3 5
12: $t4 1
13: $t5 1
14: $t6 0
15: $t7 0
16: $s0 0
17: $s1 0
18: $s2 0
19: $s3 0
20: $s4 0
21: $s5 0
22: $s6 0
23: $s7 0
24: $t8 0
25: $t9 0
26: $k0 0
27: $k1 0
28: $gp 0
29: $sp 0
30: $fp 0
31: $ra 0
PC 20
Shift left 2
ADD
AND
Control
Instruction Memory
Registers
ALU
Data memory
Sign extend
ALU control

2.2. LW

lw1 $t1$,t2, int

Instrucción:

```
"lw0":  {
                "type": "R",
                "args": ["reg", "reg"],
                "fields": {"op": 33, "rs": "#2", "rt": 0, "rd": "#1", "shamt": 0, "fu
                "desc": ""
},
```

Pseudo-Instrucción:

```
"lw1":  {
                "args": ["reg", "reg", "int"],
                "to": ["muli $1, #2, #3", "lw0 #1, $1"],
                "desc":"$t1 = MEM[$t2 * n]"
},
"muli":  {
                "args": ["reg", "reg", "int"],
                "to": ["li $1, #3", "mul #1, #2, $1"],
                "desc": "$t1 = $t2 * n"
},
```

No fue posible realizar la función lw1 con la funcionalidad especificada como un instrucción nativa, ya que requería la multiplicación previa de un registro y un entero para luego ser cargado en el registro indicado.

Para la siguiente instrucción creamos entonces la instrucción lw0, que difiere principalmente con lw por recibir dos registros (y no un registro y data)

Por otra parte, creamos la pseudo instruccion multi, que realiza la multiplicación de un registro con un valor numérico dado.

Con lw0 como muli ya implementadas, pudimos entonces añadir la pseudo-instrucción lw1. Esta carga en el primer registro el valor en memoria que se encuentra en la posición que resulta de multiplicar el valor del segundo registro con el entero dado.

2.2.1. Prueba de uso

```
addi $t1, $t2, 4
addi $t2, $t2, 8
sw $t2, 8($t2)
lw1 $t1, $t2, 2
```

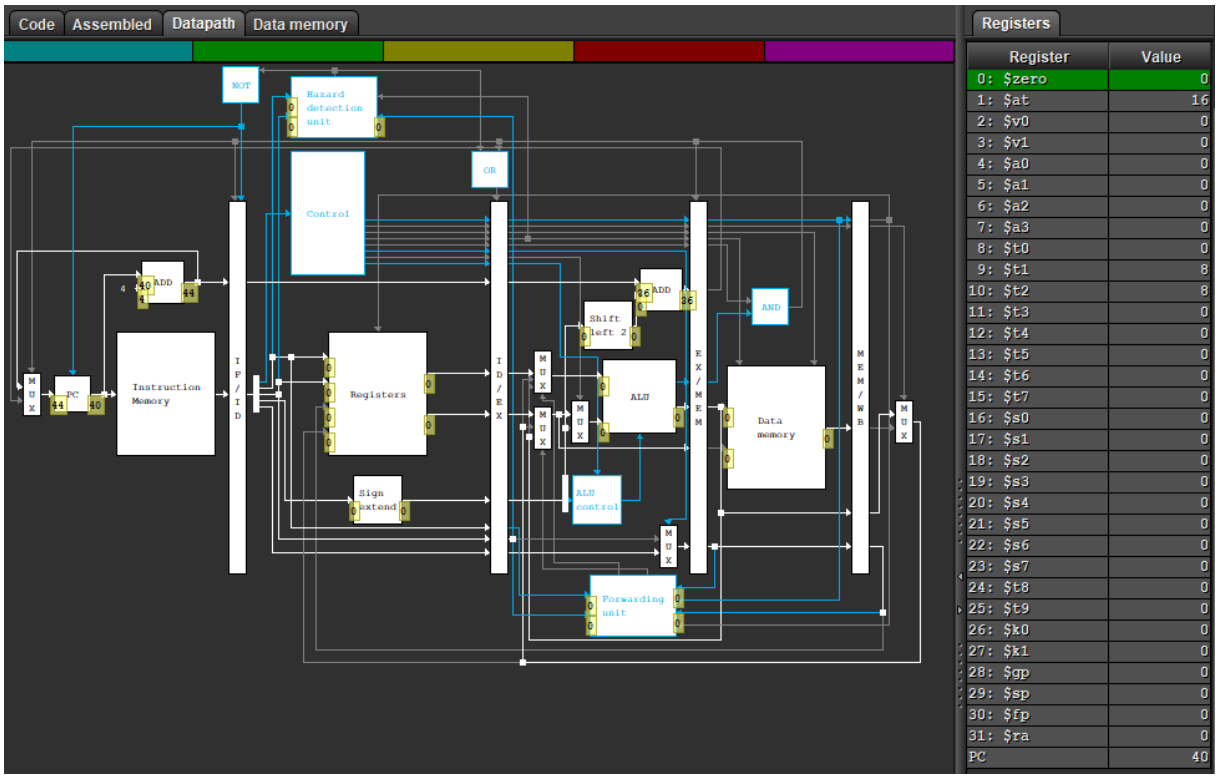
En la siguiente prueba se carga el valor 4 en el registro t1 y el valor 8 en el registro t2. Se carga en memoria, en la dirección 16, el valor 8.

Por otra parte, en la última linea, cargamos en t1 el valor que se encuentra en la dirección 16 de memoria (8 * 2) por lo que ahora vale este 8.

2.2.2. Datapath

Al implementar lw, no se realizó ningún cambio en el datapath, es decir, no se agregaron componentes extras para su correcto funcionamiento.

DP Pipeline Extended



2.3. JUMP

jadd $t1, $t2

Para evitar confusiones con instrucciones ya implementadas en DrMips, nuetra instrución se llama 'jadd' y hace uso de dos parámetros. Carga en PC el resultado de sumar los dos valores de los registros dados.

Para que funcione tanto en pipeline como en unicycle, se agregó de forma nativa respectivamente en los archivos default-no-jump.set y defaul.set la siguiente declaración

```
"jadd": {
                "type": "I",
                "args": ["reg", "reg"],
                "fields": {"op": 1, "rs": "#1", "rt":"#2", "imm": 0},
                "desc": "PC = $t1 + $t2"
},
```

2.3.1. Prueba de uso

```
addi $t1, $t1, 1
addi $t2, $t2, 3
jadd $t1, $t2
addi $t4, $t4, 1
```

Para probar nuestra implementación utilizamos el código presentado. En este caso se cargan en t1 y t2 los valores 1 y 3 respectivamente. A continuación 'jadd' suma los dos valores (resultado 4) y carga este valor en PC.

Valiento PC 4, se vuelve a sumar a t2 el valor 3, y nuevamente sucede un salto con PC valiendo 7, que resulta nuevamente en la segunda linea.

Los consecuentes saltos llevan a un loop, ya que cuando a 7 se le suma 3 (correspondiente al segundo registro), PC vale 10 y salta a la misma instrucción que tiene el salto, por lo que nuestro programa no se mueve de ahí y nunca correrá la última línea.

2.3.2. Datapath

Para ambas implementaciones, se creo una nueva señal, llamada JumpA, la cual pasa por los distintos componentes del dataPath. Como el valor de esta operación (recordando que antes de cargar un valor en PC debemos sumar el valor de los registros) debe ser tomado de la AlU, se mira si JumpA está activo o no. Si está activo (vale 1) se carga en MuxPC el valor de la ALU, en caso contrario, se mantiene el que posea en el momento.

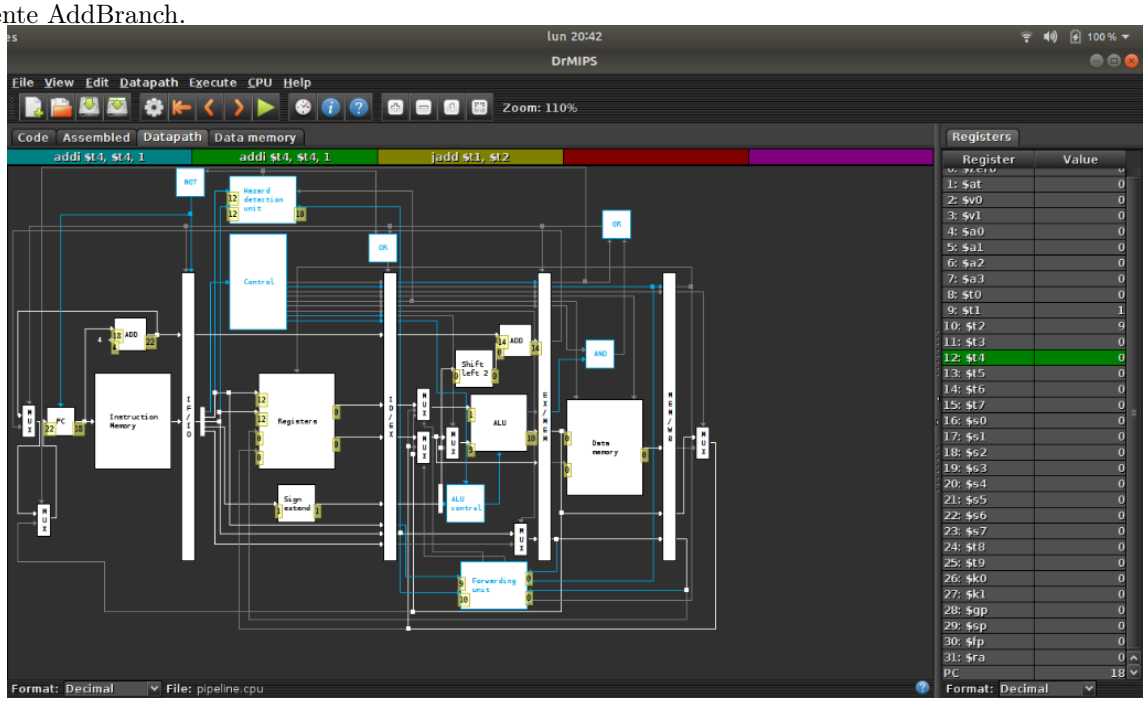
DP Pipeline

Se agregó en pipeline los siguientes componentes:

- MuxJumpA
- ForkRes
- OrJumpBranchFlush
- ForkJumpA

Para evitar hazards, se realizó (en pipeline, no en unicycle ya que no son en este los hazards posibles) un flush, para que las operaciones 'pendientes' sean realizadas antes de cargar cualquier valor en MuxPC.

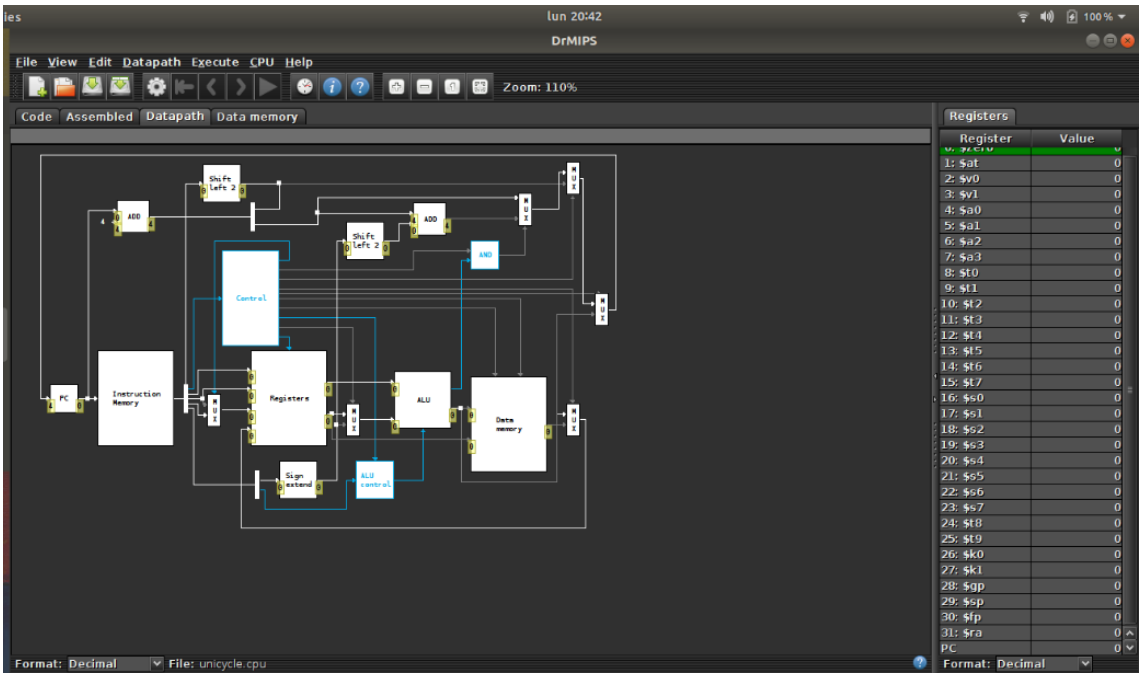
Para realizar el flush, utilizamos OrJumpBranchFlush, conectando a este componente previamente AddBranch.



DP Unicycle

Se agregó en unicycle los siguientes componentes:

- MuxJumpA



3. Conclusiones

Durante el trabajo nos fue de gran ayuda DrMIPS tanto para visualizar el DataPath, como para hacer el seguimiento de los registros y la memoria corriendo nuestras pruebas de uso.

Al implementar la instrucción 'j' tuvimos inconvenientes para que no se produzcan 'hazards', lo pudimos resolver notanto que la instrucción 'beq' ya utilizaba Branch y Flush posteriormente, por lo que utilizamos esta de guía.

4. Código

Todas las implementaciones se pueden encontrar en el siguiente link: https://github.com/Florencia-97/tp3_orga

5. Material utilizado

Las implementaciones de este trabajo práctico se realizaron sobre el programa DrMips, el cuál puede ser encontrado en https://brunonova.github.io/drmips/