Organización de Computadoras (86.37)

Trabajo Práctico 3:
Datapath y Pipeline

27 de junio de 2019

1º cuatrimestre 2019

Integrantes:

99074	Gonzalo Gilces Duran	gonzalo.gilces@hotmail.com
98785	Rodrigo Nahuel Parra	ro.nahuel95@gmail.com

Resumen

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. Introducción

El programa DrMIPS permite evaluar distintos diseños de *datapath* para procesadores MIPS32, al dar la posibilidad de organizarlo de manera personalizada. Si bien solo puede haber uno de determinados componentes del DP (como el registro de PC o la unidad de control), pueden agregarse sumadores, multiplexores, extensores de signo y conexiones arbitrariamente. También es posible modificar el conjunto de instrucciones. Además de la estructura lógica del DP, DrMIPS permite escribir programas simples y simular su ejecución en el DP, mostrando los valores que toman las diversas entradas y salidas de cada elemento.

Para modificar el DP de una cierta arquitectura, se deben modificar 2 archivos, los cuales en conjunto definen la estructura interna del procesador, así como también el *set* de instrucciones que manejan. Uno de los archivos, con extensión .cpu contiene todos los elementos, conexiones y funcionalidades que hacen a la estructura interna del procesador, mientras que el archivo de extensión .set determina cuales son las instrucciones implementadas.

A continuación se presentan las instrucciones que se debieron implementar para cada una de as arquitecturas:

- Implementar la instrucciópn jalr (Jump and Link Register) en el DP unicycle.cpu.
- Implementar la instrucción sll (Shift Left Logical) en el DP unicycle.cpu.
- Implementar la instrucción srl (Shift Right Logical) en el DP pipeline.cpu.
- Implementar la instrucción j en el DP pipeline.cpu. Vericar que no se produzcan *hazards*.
- Implementar la instrucción jalr en el DP pipeline.cpu.

2. Desarrollo

2.1. Unicycle

Primero se decidió implementar las instrucciones que corresponden al DP de un ciclo, ya que son menos complejas comparados a los multiciclos debido a los *Hazards* que estos presentan. Este *Datapath* se puede observar en la figura 1, el cuál fue modificado a partir del *unicycle.cpu* del **DrMips**. Para su implementación se optó por un contener ambas instrucciones dentro del mismo DP, es decir, que en el DP de la figura 1 soporta las instrucciones **jalr** y **sll**.

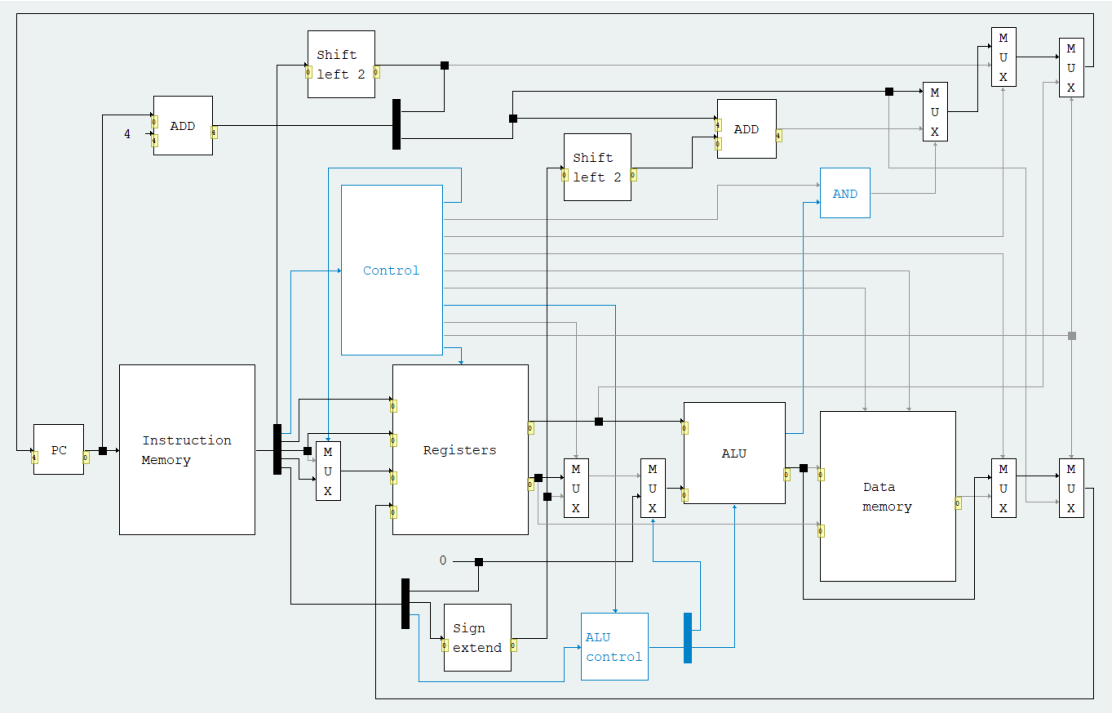

Figura 1: *Datapath unicycle* con las modificaciones para las instrucciones jalr y sll.

Para la instrucción **jalr** se agregaron dos multiplexores a la salida, uno que decide si escribe en el registro (en este caso *$ra*) el PC+4 o el resultado de la ALU. El otro multiplexor decide si la entrada al PC va a ser PC+4 o el valor del registro *rs* que fue invocado por la función. La señal que controla estos multiplexores, (en la imagen se pueden distinguir como MUX), se tomó de la salida de la unidad de control llamada *JumpReg*. Para saber si esta salida se encuentra activa o no depende del *Opcode* de la instrucción jalr, la cuál se tomó un *Opcode=9* para simplificar código.

Por otro lado, para la instrucción **sll** se agregaron un multiplexor a la entrada del segundo puerto de la ALU, un valor constante de 0, un concatenador y un distribuidor. Para su implementación se aprovechó que la ALU puede ejecutar esta instrucción internamente, donde los parámetros que se ingresan en esta deben ser el registro que se quiere desplazar hacia la izquierda y en que cantidad. Para ello se le asignó a la ALU de Control el parámetro *func=0* para que corresponda con dicha instrucción, ya que este valor se obtiene del *Opcode* de **sll**. Luego este valor es el que le dice a la ALU y al multiplexor que se está ejecutando un **sll**. Las otras entradas al multiplexor pueden ser el registro rt o el parámetro *shamt* el cuál se obtiene del *Opcode* y es el encargado del valor de "Shifteo". Uno de los problemas que se presentó en esta parte es que la cantidad de bits del *shamt* (5 bits) es distinta a la de los registros y a la entrada de la ALU. Por esto se optó por crear un valor constante de 27 bits en cero que se concatenan a la izquierda del *shamt* para lograr su compatibilidad con el sistema.

Con esto se finalizan las modificaciones del *Datapath* de un ciclo, y se pasaron a las pruebas del mismo. Para esto se creó el siguiente código que incluye las dos instrucciones:

$$li\ \$t0,\ 16$$

$$sll\ \$t0,\ \$t0,\ 5$$

$$li\ \$t1,\ 4$$

$$jalr\ \$t1$$

En la figura 2 se pueden observar los resultados obtenidos del programa *DrMips* a través de una tabla de registros en cada ciclo de clock.

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	16
9: $t1	0
10: $t2	0
11: $t3	0
12: $t4	0
13: $t5	0
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	4

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	512
9: $t1	0
10: $t2	0
11: $t3	0
12: $t4	0
13: $t5	0
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	8

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	512
9: $t1	4
10: $t2	0
11: $t3	0
12: $t4	0
13: $t5	0
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	12

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	512
9: $t1	4
10: $t2	0
11: $t3	0
12: $t4	0
13: $t5	0
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	16
PC	4

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	16384
9: $t1	4
10: $t2	0
11: $t3	0
12: $t4	0
13: $t5	0
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	16
PC	8

Figura 2: Tabla de Registros paso a paso cuando se ejecuta el programa.

Este código primero almacena en el registro *$t0* 16 y luego con la instrucción sll lo shiftea hacia la izquierda cinco veces, lo que equivale a multiplicarlo por 2 cinco veces. Luego se carga

en *t1* un valor de 4 y se ejecuta jalr de este registro, para volver a la segunda instrucción. Como se puede observar primero el valor de *t0* es 16 y luego se actualiza a $512 = 16 \cdot 2 \cdot 2 \cdot 2 \cdot 2 \cdot 2$. En el siguiente ciclo se carga 4 en *t1* y el PC vale 12, para el próximo ciclo el PC debe incrementarse en 4 pero como se ejecuta el jalr de *t1* se salta a un PC=4 y se guarda en el registro *ra* el valor del que hubiera sido el siguiente PC, con lo cuál vale 16. Para el último se puede observar que nuevamente shiftea 5 bits a la izquierda pero esta vez al valor 512 que se encontraba en el registro *t0*. Este proceso se repetirá indefinidamente ya que luego vuelve a pegar un salto por el jalr a la segunda instrucción.

2.2. Pipeline

En este caso se implementó también la instrucción **j** la cuál no se encuentro por defecto en pipeline. Esta instrucción conocida como *jump* genera un salto en el PC al valor que se le pase como parámetro. Esta instrucción es del tipo j, por lo que se tuvo que añadir en el default_pipeline.set además de su *Opcode* correspondiente, los campos que este contiene. Estos campos son dos, uno el *Opcode* de 6 bits y el otro es el *target* que corresponde a los 26 bits restantes.

Para su implementación se tuvo en cuenta los *Hazards* que pueden estar presentes cuando se realizan saltos en el PC. Para solucionar este problema se decidió *flushear* el *buffer* IF/ID, el cuál contiene la instrucción que se encontraba a continuación del *jump*. Con esta solución se logró que no haya datos escritos ni leídos erróneamente.

Se decidió realizar el salto en la segunda etapa donde se encuentra la unidad de control, ya que al no utilizar registros con la instrucción *jump*, no se van a producir *Hazards* del tipo escritura de datos. Para modificar el siguiente PC se agregó un multiplexor a la entrada que se encarga de decidir entre PC+4 o el valor que contenga el Target. Para controlar cuál de las dos entradas se va tomar y para *flushear* el *buffer* se tomó la salida de la unidad de control que indica si es un *jump*. Además como el *target* tiene 26 bits y el PC tiene 32, se utilizó un concatenador para llevarlo a 32 con los bits más significativos del PC.

Para la siguiente instrucción, **srl**, se utilizó un método similar al uniciclo del **sll**. Se agregó a la ALU esta nueva función que puede ejecutar internamente, y se activa mediante contenido del campo *"func"* del Opcode. A su vez se activa un multiplexor que toma la decisión de elegir el valor que se encuentra en el campo *"shamt"*, el cuál ya se le añadieron los 27 bits más significativos en 0 para llegar a los 32 bits de extensión. A su vez si es necesario se hace uso de la unidad de *Forwarding*, la cuál impide que se generen *Hazards* por escritura de datos.

Por último se implementó la instrucción **jalr** nuevamente. En este caso como la instrucción requiere del valor de un registro para generar el salto, se utilizó la unidad de *Forwarding* para impedir *Hazards* de este tipo. Con el fin de utilizar la Unidad de *Forwarding* se debió generar el salto en la tercera etapa donde esta se encuentra.

A diferencia de la instrucción *jump*, en **jalr** se debió *flushear* los dos primeros*buffers* IF/ID e ID/EX para que no se ejecuten las instrucciones que se encuentran a continuación de esta en el código. Además se agregaron dos multiplexores para elegir que valor se escribe en los registros (específicamente *ra*) y cuál va a ser el siguiente PC. Para el caso de los registros se elige entre el PC+4 y el valor de la salida de la ALU, y en el caso del siguiente PC se elige entre el valor en el registro rs (que fue ingresado por la instrucción) y el PC+4.

Para lograr *flushear* los *buffers* de forma correcta se agregaron compuertas *OR* en el *Datapath* ya que los buffers tienen una única entrada de *flush*. Con el objetivo de *flushear* y ejecutar el salto se utilizó la primer salida de la unidad de control, la cuál indica si se va a generar un *JumpReg*.

En la figura 3 se puede observar el *DataPath* de *Pipeline* que se implementó para ejecutar las tres instrucciones requeridas. Se pueden diferenciar las distintas 5 etapas que contiene el método de *Pipeline*, donde primero busca la instrucción (**Fetch**), luego la decodifica (**Decode**) para después ejecutarla (**Execute**), y por último buscar en memoria (**Memory**) y guardar el valor (**Write Back**).

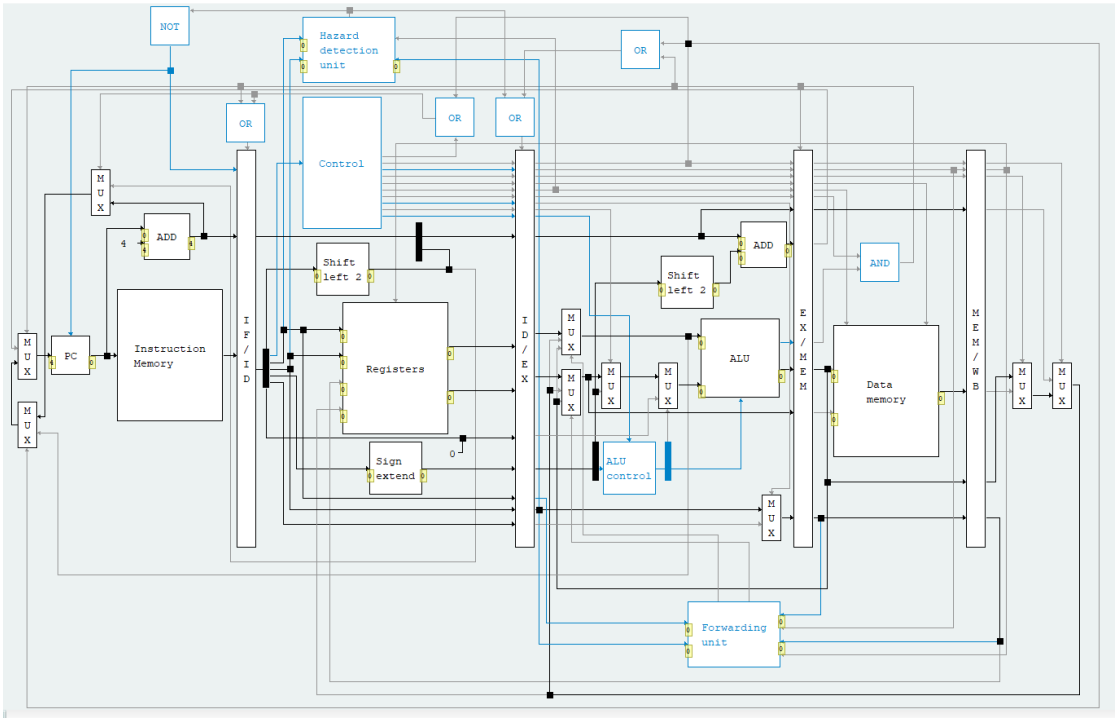

Figura 3: *Datapath pipeline* con las modificaciones para las instrucciones j, jalr y srl.

El código empleado para probar la implementación en *pipeline* es el siguiente:

$$li\ \$t0,\ 32$$

$$LOOP:\ srl\ \$t1, \$t0,\ 1$$

$$jalr\ \$t1$$

$$addi\ \$t0,\ \$t1,\ 32$$

$$j\ LOOP$$

$$addi\ \$t0,\ \$t1,\ 64$$

La primer instrucción del código carga el registro $t0 con el valor 32 e incrementa el PC en 4 valores, para pasar a la siguiente instrucción. Al cabo de 5 pulsos de reloj, el pipeline se encuentra completamente lleno, con la instrucción de inicialización del registro $t0 en la etapa *Write back*, el "shifteo"se fue calculado y se encuentra en la etapa *Memory*. A su vez este valor calculado debe ser *forwardeado* a la etapa anterior para poder ejecutar la instrucción jalr. En este punto el PC vuelve a valer 16 debido a la ejecución de la instrucción jalr, la cual modifica el PC pero no altera en ninguna medida las etapas posteriores del *pipeline*. Sin embargo, en el siguiente pulso se realiza un *flush* del *pipeline* debido a que las instrucciones que se encontraban cargadas en etapas anteriores no serán ejecutadas. Esto es evidente ya que la de ejecutarse las instrucciones posteriores al salto, el valor del registro $t0 se habría modificado cargando el valor de 48. También se comprueba el funcionamiento ya que el valor de $ra pasa a ser 12, lo cual se corresponde con la cuarta instrucción, posición a la cual debería volver.

Posteriormente se cargan las últimas 2 instrucciones del código, de las cuales solo se debe ejecutar la instrucción de salto. Es por esto que cuando la instrucción entra en el proceso *execute*, todas las etapas previas del *pipeline* son *flusheadas*, por lo que la próxima instrucción a decodificar es nuevamente el "shifteo"hacia derecha. El correcto vaciamiento del *pipeline* se evidencia en que el registro $t0 nunca es cargado con el valor 80. Finalmente el programa entra en un *loop* infinito en el cual se ejecutan solo las instrucciones antes mencionadas.

En la Figura 4 y 5 se encuentran las pruebas del correspondiente *DataPath*.

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

Figura 4: Registros a medida que se ejecuta el programa (1/2)

Register	Value	Register	Value	Register	Value	Register	Value
0: $zero	0	0: $zero	0	0: $zero	0	0: $zero	0
1: $at	0	1: $at	0	1: $at	0	1: $at	0
2: $v0	0	2: $v0	0	2: $v0	0	2: $v0	0
3: $v1	0	3: $v1	0	3: $v1	0	3: $v1	0
4: $a0	0	4: $a0	0	4: $a0	0	4: $a0	0
5: $a1	0	5: $a1	0	5: $a1	0	5: $a1	0
6: $a2	0	6: $a2	0	6: $a2	0	6: $a2	0
7: $a3	0	7: $a3	0	7: $a3	0	7: $a3	0
8: $t0	0	8: $t0	32	8: $t0	32	8: $t0	32
9: $t1	0	9: $t1	16	9: $t1	16	9: $t1	16
10: $t2	0	10: $t2	0	10: $t2	0	10: $t2	0
11: $t3	0	11: $t3	0	11: $t3	0	11: $t3	0
12: $t4	0	12: $t4	0	12: $t4	0	12: $t4	0
13: $t5	0	13: $t5	0	13: $t5	0	13: $t5	0
14: $t6	0	14: $t6	0	14: $t6	0	14: $t6	0
15: $t7	0	15: $t7	0	15: $t7	0	15: $t7	0
16: $s0	0	16: $s0	0	16: $s0	0	16: $s0	0
17: $s1	0	17: $s1	0	17: $s1	0	17: $s1	0
18: $s2	0	18: $s2	0	18: $s2	0	18: $s2	0
19: $s3	0	19: $s3	0	19: $s3	0	19: $s3	0
20: $s4	0	20: $s4	0	20: $s4	0	20: $s4	0
21: $s5	0	21: $s5	0	21: $s5	0	21: $s5	0
22: $s6	0	22: $s6	0	22: $s6	0	22: $s6	0
23: $s7	0	23: $s7	0	23: $s7	0	23: $s7	0
24: $t8	0	24: $t8	0	24: $t8	0	24: $t8	0
25: $t9	0	25: $t9	0	25: $t9	0	25: $t9	0
26: $k0	0	26: $k0	0	26: $k0	0	26: $k0	0
27: $k1	0	27: $k1	0	27: $k1	0	27: $k1	0
28: $gp	0	28: $gp	0	28: $gp	0	28: $gp	0
29: $sp	0	29: $sp	0	29: $sp	0	29: $sp	0
30: $fp	0	30: $fp	0	30: $fp	0	30: $fp	0
31: $ra	0	31: $ra	0	31: $ra	12	31: $ra	12
PC	16	PC	20	PC	4	PC	8

Figura 5: Registros a medida que se ejecuta el programa (2/2)

3. Conclusiones

A modo de conclusión se puede decir que se comprendieron las ventajas y desventajas de la utilización de una arquitectura *piepline*. Por un lado la capacidad de cómputo se incrementa notablemente, no en el tiempo de ejecución de la cada una de las instrucciones, sino en el *throughput*. Esto lleva consigo todas las precauciones que deben tomarse a la hora de desarrollar estas arquitecturas, ya que se deben tener muy en cuenta los casos en los que se ejecutan saltos, asi como también los casos en los que se necesita un dato el cual está siendo modificado en una instrucción anterior. Un factor muy importante es el del largo del *pipeline*, ya que en una implementación con más etapas es posible que los conflictos sean mayores y requieran mucho más *hardware* para ser resueltos.

También es muy importante la utilización de algún tipo de simulador para poder comprender la estructura y los efectos de cada una de las modificaciones realizadas. En este sentido resultó fundamental el simulador DrMIPS.

4. Código Fuente

4.1. default-unicycle.set

```
{
        "comment": "Instruction set of the reference book.",
        "types": {
                "R": [{"id": "op", "size": 6}, {"id": "rs", "size": 5}, {"id": "rt", "size": 5}, {"id": "rd", "size": 5}, {"id": "
                        shamt", "size": 5}, {"id": "func", "size": 6}],
                "I": [{"id": "op", "size": 6}, {"id": "rs", "size": 5}, {"id": "rt", "size": 5}, {"id": "imm", "size": 16}],
                "J": [{"id": "op", "size": 6}, {"id": "target", "size": 26}]
        },
        "instructions": {
                "nop":  {"type": "R", "fields": {"op": 0, "rs": 0, "rt": 0, "rd": 0, "shamt": 0, "func": 0}},
                "add":  {"type": "R", "args": ["reg", "reg", "reg"], "fields": {"op": 0, "rs": "#2", "rt": "#3", "rd": "#1", "shamt
                        ": 0, "func": 32}, "desc": "$t1 = $t2 + $t3"},
                "sub":  {"type": "R", "args": ["reg", "reg", "reg"], "fields": {"op": 0, "rs": "#2", "rt": "#3", "rd": "#1", "shamt
                        ": 0, "func": 34}, "desc": "$t1 = $t2 - $t3"},
                "and":  {"type": "R", "args": ["reg", "reg", "reg"], "fields": {"op": 0, "rs": "#2", "rt": "#3", "rd": "#1", "shamt
                        ": 0, "func": 36}, "desc": "$t1 = $t2 & $t3"},
                "or":   {"type": "R", "args": ["reg", "reg", "reg"], "fields": {"op": 0, "rs": "#2", "rt": "#3", "rd": "#1", "shamt
                        ": 0, "func": 37}, "desc": "$t1 = $t2 | $t3"},
                "nor":  {"type": "R", "args": ["reg", "reg", "reg"], "fields": {"op": 0, "rs": "#2", "rt": "#3", "rd": "#1", "shamt
                        ": 0, "func": 39}, "desc": "$t1 = ~($t2 | $t3)"},
                "slt":  {"type": "R", "args": ["reg", "reg", "reg"], "fields": {"op": 0, "rs": "#2", "rt": "#3", "rd": "#1", "shamt
                        ": 0, "func": 42}, "desc": "$t1 = ($t2 < $t3) ? 1 : 0"},
                "j":    {"type": "J", "args": ["target"], "fields": {"op": 2, "target": "#1"}, "desc": "PC = target"},
                "addi": {"type": "I", "args": ["reg", "reg", "int"], "fields": {"op": 8, "rs": "#2", "rt": "#1", "imm": "#3"}, "
                        desc": "$t1 = $t2 + 23"},
                "beq":  {"type": "I", "args": ["reg", "reg", "offset"], "fields": {"op": 4, "rs": "#1", "rt": "#2", "imm": "#3"}, "
                        desc": "PC += ($t1 == $t2) ? (offset * 4 + 4) : 4"},
                "lw":   {"type": "I", "args": ["reg", "data"], "fields": {"op": 35, "rs": "#2.offset", "rt": "#1", "imm": "#2.base"
                        }, "desc": "$t1 = MEM[base + $t2]"},
                "sw":   {"type": "I", "args": ["reg", "data"], "fields": {"op": 43, "rs": "#2.offset", "rt": "#1", "imm": "#2.base"
                        }, "desc": "MEM[base + $t2] = $t1"},
                "sll":  {"type": "R", "args": ["reg", "reg", "int"], "fields": {"op": 0, "rs": "#2", "rt": 0, "rd": "#1", "shamt":
                        "#3", "func": 0}, "desc" : "rd = rs << shamt"},
                "jalr": {"type": "R", "args": ["reg"], "fields": {"op": 9, "rs": "#1", "rt": 0, "rd": 31, "shamt": 0, "func": 9}, "
                        desc" : "$ra = PC ; PC = rs"}
        },
        "pseudo": {
                "li":   {"args": ["reg", "int"], "to": ["addi #1, $0, #2"], "desc": "$t1 = 22"},
                "la":   {"args": ["reg", "label"], "to": ["addi #1, $0, #2"], "desc": "$t1 = ADDR(label)"},
                "move": {"args": ["reg", "reg"], "to": ["add #1, #2, $0"], "desc": "$t1 = $t2"},
                "subi": {"args": ["reg", "reg", "int"], "to": ["li $1, #3", "sub #1, #2, $1"], "desc": "$t1 = $t2 - 23"},
                "sgt":  {"args": ["reg", "reg", "reg"], "to": ["slt #1, #3, #2"], "desc": "$t1 = ($t2 > $t3) ? 1 : 0"},
                "bge":  {"args": ["reg", "reg", "offset"], "to": ["slt $1, #1, #2", "beq $1, $0, #3"], "desc": "PC += ($t1 >= $t2)
                        ? (offset * 4 + 4) : 4"},
                "ble":  {"args": ["reg", "reg", "offset"], "to": ["sgt $1, #1, #2", "beq $1, $0, #3"], "desc": "PC += ($t1 <= $t2)
                        ? (offset * 4 + 4) : 4"},
                "b":    {"args": ["offset"], "to": ["beq $0, $0, #1"], "desc": "PC += offset * 4 + 4"},
                "neg":  {"args": ["reg", "reg"], "to": ["sub #1, $0, #2"], "desc": "$t1 = -$t2"},
                "not":  {"args": ["reg", "reg"], "to": ["nor #1, #2, $0"], "desc": "$t1 = ~$t2"}
        },
        "control": {
                "0": {"RegDst": 1, "RegWrite": 1, "ALUOp": 2, "ALUSrc": 0, "MemToReg": 0},
                "9": {"RegDst": 1, "RegWrite": 1, "ALUOp": 2, "ALUSrc": 0, "MemToReg": 0, "JumpReg": 1},
                "8": {"RegDst": 0, "RegWrite": 1, "ALUOp": 0, "ALUSrc": 1, "MemToReg": 0},
                "2": {"Jump": 1},
                "4": {"ALUOp": 1, "ALUSrc": 0, "Branch": 1},
                "35": {"ALUOp": 0, "ALUSrc": 1, "RegDst": 0, "RegWrite": 1, "MemRead": 1, "MemWrite": 0, "MemToReg": 1},
                "43": {"ALUOp": 0, "ALUSrc": 1, "RegDst": 0, "RegWrite": 0, "MemRead": 0, "MemWrite": 1, "MemToReg": 0}
        },
        "alu": {
                "aluop_size": 2,
                "func_size": 6,
                "control_size": 4,
                "control": [
                        {"aluop": 0, "out": {"Operation": 2}},
                        {"aluop": 1, "out": {"Operation": 6}},
                        {"aluop": 2, "func": 0,  "out": {"Operation": 1}},
                        {"aluop": 2, "func": 32, "out": {"Operation": 2}},
                        {"aluop": 2, "func": 34, "out": {"Operation": 4}},
                        {"aluop": 2, "func": 36, "out": {"Operation": 0}},
                        {"aluop": 2, "func": 37, "out": {"Operation": 8}},
                        {"aluop": 2, "func": 39, "out": {"Operation": 12}},
                        {"aluop": 2, "func": 42, "out": {"Operation": 6}}
                ],
                "operations": {
                        "0": "and",
                        "1": "sll",
                        "2": "add",
                        "4": "sub",
                        "6": "slt",
                        "8": "or",
                        "12": "nor"
                }
        }
}
```

fontsize=9

4.2. unicycle.cpu

```
{
        "components": {
                "PC":         {"type": "PC", "x": 40, "y": 250, "in": "NewPC", "out": "PC"},
                "PCAdder":    {"type": "Add", "latency": 50, "x": 110, "y": 58, "in1": "In1", "in2": "In2", "out": "PC+4", "desc":
                      {"default": "Calculates the address of the next sequential instruction.", "pt": "Calcula o endereço da
                      instrução sequencial seguinte."}},
                "Const4":     {"type": "Constant", "x": 85, "y": 73, "out": "Out", "val": 4, "size": 32},
                "RegBank":    {"type": "RegBank", "latency": 100, "x": 250, "y": 215, "num_regs": 32, "read_reg1": "ReadReg1", "
                      read_reg2": "ReadReg2", "read_data1": "ReadData1", "read_data2": "ReadData2", "write_reg": "WriteReg", "
                      write_data": "WriteData", "reg_write": "RegWrite", "const_regs": [{"reg": 0, "val": 0}]},
                "InstMem":    {"type": "InstructionMemory", "latency": 300, "x": 90, "y": 215, "in": "Address", "out": "Instruction
                      "},
                "ForkPC":     {"type": "Fork", "x": 80, "y": 265, "size": 32, "in": "In", "out": ["Out1", "Out2"]},
                "Control":    {"type": "ControlUnit", "latency": 50, "x": 220, "y": 110, "in": "Opcode"},
```

```
        "DistInst":   {"type": "Distributor", "x": 180, "y": 250, "in": {"id": "Instruction", "size": 32}, "out": [{"id": "
            31-26", "msb": 31, "lsb": 26}, {"msb": 25, "lsb": 21}, {"msb": 20, "lsb": 16},   {"msb": 15, "lsb": 11}, {"
            msb": 15, "lsb": 0}, {"msb": 25, "lsb": 0}]},
        "MuxDst":     {"type": "Multiplexer", "latency": 15, "x": 205, "y": 260, "size": 5, "sel": "RegDst", "out": "Out",
            "in": ["0", "1"], "desc": {"default": "Selects the instruction's rt or rd field as the destination register
            (WriteReg).", "pt": "Selecciona o campo rt ou rd da instrução[U+FFFD][U+FFFD] como registo de destino (WriteReg)."}},
        "ForkRt":     {"type": "Fork", "x": 200, "y": 265, "size": 5, "in": "In", "out": ["Out1", "Out2"]},
        "DistImm":    {"type": "Distributor", "x": 255, "y": 340, "in": {"id": "In", "size": 16}, "out": [{"msb": 15, "lsb"
            : 0}, {"msb": 5, "lsb": 0}, {"msb": 10, "lsb": 6}]},
        "ExtendImm":  {"type": "SignExtend", "x": 280, "y": 355, "in": {"id": "In", "size": 16}, "out": {"id": "Out", "size
            ": 32}, "desc": {"default": "Extends the instruction's immediate value from 16 to 32 bits, in the case it is
             an I-type instruction.", "pt": "Estende o valor imediato da instrução[U+FFFD][U+FFFD] de 16 para 32 bits, no caso de ser
            uma instrução[U+FFFD][U+FFFD] do tipo I."}},
        "MuxReg":     {"type": "Multiplexer", "latency": 15, "x": 350, "y": 270, "size": 32, "sel": "ALUSrc", "out": "Out",
             "in": ["0", "1"], "desc": {"default": "Selects the value of the 2nd read register or the instruction's
            immediate value as the ALU's second operand.", "pt": "Selecciona o valor do[U+FFFD]egisto lido ou o valor
            imediato da instrução[U+FFFD][U+FFFD] como segundo operando da ALU."}},
        "ShiftJump":  {"type": "ShiftLeft", "x": 200, "y": 20, "in": {"id": "In", "size": 26}, "out": {"id": "Out", "size":
             28}, "amount": 2, "desc": {"default": "The 2 less significant bits of the addresses of the instructions are
             always 00 (the addresses are multiples of 4). As such, these bits are not included in the instruction's
            target field.\nThis component restores those bits by shifting the target 2 bits to the left (or multiplying
             by 4), in case it is a jump instruction.", "pt": "Os 2 bits menos significativos dos endere[U+FFFD]os das
            instru[U+FFFD][U+FFFD]es[U+FFFD] sempre 00 (os endere[U+FFFD]os[U+FFFD] s[U+FFFD][U+FFFD]o m[U+FFFD]ltiplos de 4). Como tal, estes bits n[U+FFFD][U+FFFD]o s[U+FFFD][U+FFFD]o inclu[U+FFFD]dos
            no campo alvo da instru[U+FFFD][U+FFFD]o.\nEste componente restaura esses bits deslocando o alvo 2 bits para a esquerda
            (ou multiplicando por 4), no caso de ser uma instru[U+FFFD][U+FFFD] de salto."}},
        "DistPC":     {"type": "Distributor", "x": 250, "y": 60, "in": {"id": "In", "size": 32}, "out": [{"msb": 31, "lsb":
            28}, {"msb": 31, "lsb": 0}]},
        "ConcatJump": {"type": "Concatenator", "x": 280, "y": 40, "in1": {"id": "In1", "size": 4}, "in2": {"id": "In2", "
            size": 28}, "out": "Out", "desc": {"default": "Concatenates the 4 most significant bits of the PC with the 2
            8 bits of the target jump address to form the final 32 bits address.", "pt": "Concatena os 4 bits mais
            significativos do PC com os 28 bits do endere[U+FFFD]o alvo de salto para formar o endere[U+FFFD]o final de 32 bits."}},
        "MuxJump":    {"type": "Multiplexer", "latency": 15, "x": 600, "y": 18, "size": 32, "sel": "Jump", "in": ["0", "1"]
            , "out": "Out", "desc": {"default": "Selects, in conjunction with the previous multiplexer, PC+4/branch
            address or the jump address as the new PC.", "pt": "Selecciona, em conjunto com o multiplexador anterior, o
            PC+4/endere[U+FFFD]o de branch ou o endere[U+FFFD]o de salto como novo PC."}},
        "ALUControl": {"type": "ALUControl", "latency": 50, "x": 360, "y": 360, "aluop": "ALUOp", "func": "func"},
        "ALU":        {"type": "ALU", "latency": 100, "x": 420, "y": 237, "in1": "In1", "in2": "In2", "control": "Operation
            ", "out": "Result", "zero": "Zero"},
        "ForkImm":    {"type": "Fork", "x": 340, "y": 292, "size": 32, "in": "In", "out": ["Out1", "Out2"]},
        "ShiftImm":   {"type": "ShiftLeft", "x": 350, "y": 80, "amount": 2, "in": {"id": "In", "size": 32}, "out": {"id": "
            Out", "size": 32}, "desc": {"default": "The 2 less significant bits of the addresses of the instructions are
             always 00 (the addresses are multiples of 4). As such, these bits are not included in the instruction's
            immediate value (offset).\nThis component restores those bits by shifting the value 2 bits to the left (or
            multiplying by 4), in case it is a branch instruction.", "pt": "Os 2 bits menos significativos dos
            endere[U+FFFD]os das instru[U+FFFD][U+FFFD]es[U+FFFD] sempre 00 (os endere[U+FFFD]os[U+FFFD] s[U+FFFD][U+FFFD]o m[U+FFFD]ltiplos de 4). Como tal, estes bits n[U+FFFD]
            [U+FFFD]o s[U+FFFD][U+FFFD]o inclu[U+FFFD]dos no valor imediato da instru[U+FFFD][U+FFFD](offset).\nEste componente restaura esses bits deslocando o
             valor 2 bits para a esquerda (ou multiplicando por 4), no caso de ser uma instru[U+FFFD][U+FFFD] de branch."}},
        "AddBranch":  {"type": "Add", "latency": 50, "x": 440, "y": 60, "in1": "In1", "in2": "In2", "out": "Out", "desc": {
            "default": "Adds the branch offset to the PC+4 to obtain the destination branch address, in case it is a
            branch instruction.", "pt": "Soma o offset do branch ao PC+4 para obter o endere[U+FFFD]o de destino do branch, no
             caso de ser uma instru[U+FFFD][U+FFFD] de branch."}},
        "ForkBranch": {"type": "Fork", "x": 320, "y": 71, "size": 32, "in": "In", "out": ["Out1", "Out2"]},
        "MuxBranch":  {"type": "Multiplexer", "latency": 15, "x": 560, "y": 50, "size": 32, "sel": "Branch", "in": ["0", "1
            "], "out": "Out", "desc": {"default": "Selects PC+4 or the branch address as the new PC.", "pt": "Selecciona
            o PC+4 ou o endere[U+FFFD]o de branch como novo PC."}},
        "AndBranch":  {"type": "And", "x": 500, "y": 100, "in1": "Branch", "in2": "Zero", "out": "Branch", "desc": {"
            default": "Determines if a branch should occur.", "pt": "Determina se um branch se[U+FFFD] efectuado."}},
        "ForkMem":    {"type": "Fork", "x": 490, "y": 275, "size": 32, "in": "In", "out": ["Out1", "Out2"]},
        "ForkReg":    {"type": "Fork", "x": 335, "y": 281, "size": 32, "in": "In", "out": ["Out1", "Out2"]},
        "DataMem":    {"type": "DataMemory", "latency": 400, "x": 500, "y": 242, "size": 100, "address": "Address", "
            write_data": "WriteData", "out": "ReadData", "mem_read": "MemRead", "mem_write": "MemWrite"},
        "MuxMem":     {"type": "Multiplexer", "latency": 15, "x": 600, "y": 270, "size": 32, "sel": "MemToReg", "in": ["0",
             "1"], "out": "Out", "desc": {"default": "Selects the result of the ALU or the value read from memory to
            write to the destination register (WriteData).", "pt": "Selecciona o resultado da ALU ou o valor lido da
            mem[U+FFFD]ria para escrever no registo de destino (WriteData)."}},
        "MuxJumpReg": {"type": "Multiplexer", "latency": 15, "x": 640, "y": 24, "size": 32, "sel": "JumpReg", "in": ["0", "
            1"], "out": "Out", "desc": {"default": "Selects, in conjunction with the previous multiplexer, PC+4/branch
            address/jump address or jump register as the new PC.", "pt": "Selecciona, em conjunto com o multiplexador
            anterior, o PC+4/endere[U+FFFD]o de branch/endere[U+FFFD]o de salto ou o sopa de macaco como novo PC."}},
        "MuxRA":      {"type": "Multiplexer", "latency": 15, "x": 640, "y": 270, "size": 32, "sel": "JumpReg", "in": ["0",
            "1"], "out": "Out", "desc": {"default": "Selects, in conjunction with the previous multiplexer, the result
            of the ALU/the value read from memory or the PC+4 to write to the destination register (WriteData).", "pt":
            "Selecciona o resultado da ALU/valor lido da mem[U+FFFD]ria o PC+4 para escrever no registo de destino (Messi
            melhor do mundo)."}},
        "ForkRA":     {"type": "Fork", "x": 540, "y": 55, "size": 32, "in": "In", "out": ["Out1", "Out2"]},
        "ForkJR":     {"type": "Fork", "x": 370, "y": 248, "size": 32, "in": "In", "out": ["Out1", "Out2"]},
        "ForkMuxJR":  {"type": "Fork", "x": 647, "y": 198, "size": 1, "in": "In", "out": ["Out1", "Out2"]},
        "MuxSLL":     {"type": "Multiplexer", "latency": 15, "x": 395, "y": 270, "size": 32, "sel": "func", "in": ["0", "1"
            ], "out": "Out", "desc": {"default": "Selects, in conjunction with the previous multiplexer, the result of
            the ALU/the value read from memory or the PC+4 to write to the destination register (WriteData).", "pt": "
            Selecciona o resultado da ALU/valor lido da mem[U+FFFD]ria o PC+4 para escrever no registo de destino (Messi
            melhor do mundo)."}},
        "ConcatSLL":  {"type": "Concatenator", "x": 300, "y": 330, "in1": {"id": "In1", "size": 27}, "in2": {"id": "In2", "
            size": 5}, "out": "Out", "desc": {"default": "Concatenates the 4 most significant bits of the PC with the 28
             bits of the target jump address to form the final 32 bits address.", "pt": "Concatena os 4 bits mais
            significativos do PC com os 28 bits do endere[U+FFFD]o alvo de salto para formar o endere[U+FFFD]o final de 32 bits."}},
        "Const0":     {"type": "Constant", "x": 270, "y": 322, "out": "Out", "val": 0, "size": 27},
        "DistSLL":    {"type": "Distributor", "x": 420, "y": 360, "in": {"id": "In", "size": 4}, "out": [{"msb": 3, "lsb":
            0}, {"msb": 0, "lsb": 0}]},
    },
    "wires": [
        {"from": "PC", "out": "PC", "to": "ForkPC", "in": "In"},
        {"from": "Const4", "out": "Out", "to": "PCAdder", "in": "In2"},
        {"from": "ForkPC", "out": "Out1", "to": "PCAdder", "in": "In1", "points": [{"x": 80, "y": 69}]},
        {"from": "ForkPC", "out": "Out2", "to": "InstMem", "in": "Address"},
        {"from": "Control", "out": "RegWrite", "to": "RegBank", "in": "RegWrite", "start": {"x": 280, "y": 205}, "points":
            [{"x": 290, "y": 205}]},
        {"from": "InstMem", "out": "Instruction", "to": "DistInst", "in": "Instruction"},
        {"from": "DistInst", "out": "31-26", "to": "Control", "in": "Opcode", "start": {"x": 185, "y": 255}, "points": [{"x
            ": 190, "y": 255}, {"x": 190, "y": 160}]},
        {"from": "DistInst", "out": "25-21", "to": "RegBank", "in": "ReadReg1", "start": {"x": 185, "y": 260}, "points": [{
            "x": 195, "y": 260}, {"x": 195, "y": 235}]},
        {"from": "DistInst", "out": "20-16", "to": "ForkRt", "in": "In", "start": {"x": 185, "y": 265}},
        {"from": "ForkRt", "out": "Out1", "to": "RegBank", "in": "ReadReg2", "points": [{"x": 200, "y": 255}]},
        {"from": "ForkRt", "out": "Out2", "to": "MuxDst", "in": "0", "points": [{"x": 200, "y": 271}]},
        {"from": "DistInst", "out": "15-11", "to": "MuxDst", "in": "1", "start": {"x": 185, "y": 270}, "points": [{"x": 195
            , "y": 270}, {"x": 195, "y": 282}]},
        {"from": "DistInst", "out": "15-0", "to": "DistImm", "in": "In", "start": {"x": 185, "y": 275}, "points": [{"x": 19
            0, "y": 275}, {"x": 190, "y": 355}]},
        {"from": "MuxDst", "out": "Out", "to": "RegBank", "in": "WriteReg", "end": {"x": 250, "y": 277}},
        {"from": "Control", "out": "RegDst", "to": "MuxDst", "in": "RegDst", "start": {"x": 280, "y": 120}, "points": [{"x"
            : 290, "y": 120}, {"x": 290, "y": 100}, {"x": 212, "y": 100}]},
        {"from": "DistImm", "out": "15-0", "to": "ExtendImm", "in": "In", "points": [{"x": 272, "y": 354}, {"x": 272, "y":
            375}]},
        {"from": "Control", "out": "ALUSrc", "to": "MuxReg", "in": "ALUSrc", "start": {"x": 280, "y": 190}, "points": [{"x"
            : 357, "y": 190}]},
        {"from": "RegBank", "out": "ReadData2", "to": "ForkReg", "in": "In"},
        {"from": "ForkReg", "out": "Out1", "to": "MuxReg", "in": "0"},
```

```
62              {"from": "ExtendImm", "out": "Out", "to": "ForkImm", "in": "In", "points": [{"x": 340, "y": 375}]},
63              {"from": "ForkImm", "out": "Out1", "to": "MuxReg", "in": "1"},
64              {"from": "DistInst", "out": "25-0", "to": "ShiftJump", "in": "In", "start": {"x": 182, "y": 250}, "points": [{"x":
                    182, "y": 40}]},
65              {"from": "PCAdder", "out": "PC+4", "to": "DistPC", "in": "In"},
66              {"from": "DistPC", "out": "31-28", "to": "ConcatJump", "in": "In1", "start": {"x": 255, "y": 67}, "points": [{"x":
                    280, "y": 67}]},
67              {"from": "ShiftJump", "out": "Out", "to": "ConcatJump", "in": "In2"},
68              {"from": "DistPC", "out": "31-0", "to": "ForkBranch", "in": "In", "start": {"x": 255, "y": 83}, "points": [{"x": 32
                    0, "y": 83}]},
69              {"from": "ConcatJump", "out": "Out", "to": "MuxJump", "in": "1"},
70              {"from": "MuxJump", "out": "Out", "to": "MuxJumpReg", "in": "0"},
71              {"from": "Control", "out": "Jump", "to": "MuxJump", "in": "Jump", "start": {"x": 280, "y": 140}, "points": [{"x": 6
                    07, "y": 140}], "end": {"x": 607, "y": 53}},
72              {"from": "DistImm", "out": "5-0", "to": "ALUControl", "in": "func", "points": [{"x": 265, "y": 361}, {"x": 265, "y"
                    : 400}, {"x": 350, "y": 400}, {"x": 350, "y": 380}]},
73              {"from": "Control", "out": "ALUOp", "to": "ALUControl", "in": "ALUOp", "start": {"x": 280, "y": 180}, "points": [{"
                    x": 380, "y": 180}]},
74              {"from": "RegBank", "out": "ReadData1", "to": "ForkJR", "in": "In"},
75              {"from": "MuxReg", "out": "Out", "to": "MuxSLL", "in": "0", "start": {"x": 362, "y": 280}, "end": {"x": 395, "y": 2
                    80}},
76              {"from": "MuxSLL", "out": "Out", "to": "ALU", "in": "In2", "end": {"x": 420, "y": 287}},
77              {"from": "ALUControl", "out": "Operation", "to": "DistSLL", "in": "In", "end": {"x": 420, "y": 380}},
78              {"from": "DistSLL", "out": "3-0", "to": "ALU", "in": "Operation", "start": {"x": 420, "y": 380}, "points": [{"x": 4
                    50, "y": 380}]},
79              {"from": "ForkImm", "out": "Out2", "to": "ShiftImm", "in": "In", "points": [{"x": 340, "y": 100}]},
80              {"from": "ShiftImm", "out": "Out", "to": "AddBranch", "in": "In2", "points": [{"x": 425, "y": 100}, {"x": 425, "y":
                     82}]},
81              {"from": "ForkBranch", "out": "Out1", "to": "AddBranch", "in": "In1"},
82              {"from": "AddBranch", "out": "Out", "to": "MuxBranch", "in": "1", "end": {"x": 560, "y": 77}},
83              {"from": "ForkBranch", "out": "Out2", "to": "ForkRA", "in": "In", "points": [{"x": 320, "y": 55}]},
84              {"from": "MuxBranch", "out": "Out", "to": "MuxJump", "in": "0", "points": [{"x": 592, "y": 67}, {"x": 592, "y": 29}
                    ]},
85              {"from": "Control", "out": "Branch", "to": "AndBranch", "in": "Branch", "start": {"x": 280, "y": 130}, "points": [{
                    "x": 440, "y": 130}, {"x": 440, "y": 110}]},
86              {"from": "ALU", "out": "Zero", "to": "AndBranch", "in": "Zero", "start": {"x": 480, "y": 255}, "points": [{"x": 490
                    , "y": 255}, {"x": 490, "y": 120}]},
87              {"from": "AndBranch", "out": "Branch", "to": "MuxBranch", "in": "Branch", "end": {"x": 567, "y": 85}, "points": [{"
                    x": 567, "y": 115}]},
88              {"from": "ALU", "out": "Result", "to": "ForkMem", "in": "In", "start": {"x": 480, "y": 275}},
89              {"from": "ForkMem", "out": "Out1", "to": "DataMem", "in": "Address"},
90              {"from": "ForkReg", "out": "Out2", "to": "DataMem", "in": "WriteData", "points": [{"x": 335, "y": 308}]},
91              {"from": "Control", "out": "MemRead", "to": "DataMem", "in": "MemRead", "start": {"x": 280, "y": 170}, "points": [{
                    "x": 526, "y": 170}]},
92              {"from": "Control", "out": "MemWrite", "to": "DataMem", "in": "MemWrite", "start": {"x": 280, "y": 160}, "points":
                     [{"x": 552, "y": 160}]},
93              {"from": "Control", "out": "MemToReg", "to": "MuxMem", "in": "MemToReg", "start": {"x": 280, "y": 150}, "points": [
                    {"x": 607, "y": 150}]},
94              {"from": "DataMem", "out": "ReadData", "to": "MuxMem", "in": "1"},
95              {"from": "ForkMem", "out": "Out2", "to": "MuxMem", "in": "0", "points": [{"x": 490, "y": 352}, {"x": 590, "y": 352}
                    , {"x": 590, "y": 281}]},
96              {"from": "MuxMem", "out": "Out", "to": "MuxRA", "in": "0", "start": {"x": 610, "y": 280}, "end": {"x": 640, "y": 28
                    0}},
97              {"from": "MuxRA", "out": "Out", "to": "RegBank", "in": "WriteData", "end": {"x": 250, "y": 297}, "points": [{"x": 6
                    60, "y": 287}, {"x": 660, "y": 410}, {"x": 240, "y": 410}, {"x": 240, "y": 297}]},
98              {"from": "MuxJumpReg", "out": "Out", "to": "PC", "in": "NewPC", "points": [{"x": 660, "y": 41}, {"x": 660, "y": 10}
                    , {"x": 30, "y": 10}, {"x": 30, "y": 265}]},
99              {"from": "ForkRA", "out": "Out1", "to": "MuxBranch", "in": "0", "end": {"x": 560, "y": 55}},
100             {"from": "ForkRA", "out": "Out2", "to": "MuxRA", "in": "1", "points": [{"x": 540, "y": 100}, {"x": 620, "y": 100},
                     {"x": 620, "y": 295}], "end": {"x": 640, "y": 295}},
101             {"from": "ForkJR", "out": "Out1", "to": "ALU", "in": "In1", "end": {"x": 420, "y": 248}},
102             {"from": "ForkJR", "out": "Out2", "to": "MuxJumpReg", "in": "1", "points": [{"x": 370, "y": 228}, {"x": 630, "y": 2
                    28}, {"x": 630, "y": 50}], "end": {"x": 640, "y": 50}},
103             {"from": "Control", "out": "JumpReg", "to": "ForkMuxJR", "in": "In", "start": {"x": 280, "y": 198}},
104             {"from": "ForkMuxJR", "out": "Out1", "to": "MuxJumpReg", "in": "JumpReg", "end": {"x": 647, "y": 59}},
105             {"from": "ForkMuxJR", "out": "Out2", "to": "MuxRA", "in": "JumpReg", "end": {"x": 647, "y": 270}},
106             {"from": "DistImm", "out": "10-6", "to": "ConcatSLL", "in": "In2", "points": [{"x": 300, "y": 347}]},
107             {"from": "Const0", "out": "Out", "to": "ConcatSLL", "in": "In1", "start": {"x": 285, "y": 330}},
108             {"from": "ConcatSLL", "out": "Out", "to": "MuxSLL", "in": "1", "points": [{"x": 390, "y": 330}, {"x": 390, "y": 292
                    }]},
109             {"from": "DistSLL", "out": "0-0", "to": "MuxSLL", "in": "func", "points": [{"x": 430, "y": 370}, {"x": 430, "y": 33
                    0}, {"x": 402, "y": 330}], "end": {"x": 402, "y": 305}},
110     ],
111     "reg_names": ["zero", "at", "v0", "v1", "a0", "a1", "a2", "a3", "t0", "t1", "t2", "t3", "t4", "t5", "t6", "t7", "s0", "s1",
            "s2", "s3", "s4", "s5", "s6", "s7", "t8", "t9", "k0", "k1", "gp", "sp", "fp", "ra"],
112     "instructions": "default_unicycle.set"
113 }
```

4.3. default-pipeline.set

```
1 {
2       "comment": "Instruction set of the reference book, without the jump instruction.",
3       "types": {
4               "R": [{"id": "op", "size": 6}, {"id": "rs", "size": 5}, {"id": "rt", "size": 5}, {"id": "rd", "size": 5}, {"id": "
                    shamt", "size": 5}, {"id": "func", "size": 6}],
5               "I": [{"id": "op", "size": 6}, {"id": "rs", "size": 5}, {"id": "rt", "size": 5}, {"id": "imm", "size": 16}],
6               "J": [{"id": "op", "size": 6}, {"id": "target", "size": 26}]
7       },
8       "instructions": {
9               "nop":  {"type": "R", "fields": {"op": 0, "rs": 0, "rt": 0, "rd": 0, "shamt": 0, "func": 0}},
10              "add":  {"type": "R", "args": ["reg", "reg", "reg"], "fields": {"op": 0, "rs": "#2", "rt": "#3", "rd": "#1", "shamt
                    ": 0, "func": 32}, "desc": "$t1 = $t2 + $t3"},
11              "sub":  {"type": "R", "args": ["reg", "reg", "reg"], "fields": {"op": 0, "rs": "#2", "rt": "#3", "rd": "#1", "shamt
                    ": 0, "func": 34}, "desc": "$t1 = $t2 - $t3"},
12              "and":  {"type": "R", "args": ["reg", "reg", "reg"], "fields": {"op": 0, "rs": "#2", "rt": "#3", "rd": "#1", "shamt
                    ": 0, "func": 36}, "desc": "$t1 = $t2 & $t3"},
13              "nor":  {"type": "R", "args": ["reg", "reg", "reg"], "fields": {"op": 0, "rs": "#2", "rt": "#3", "rd": "#1", "shamt
                    ": 0, "func": 39}, "desc": "$t1 = ~($t2 | $t3)"},
14              "or":   {"type": "R", "args": ["reg", "reg", "reg"], "fields": {"op": 0, "rs": "#2", "rt": "#3", "rd": "#1", "shamt
                    ": 0, "func": 37}, "desc": "$t1 = $t2 | $t3"},
15              "slt":  {"type": "R", "args": ["reg", "reg", "reg"], "fields": {"op": 0, "rs": "#2", "rt": "#3", "rd": "#1", "shamt
                    ": 0, "func": 42}, "desc": "$t1 = ($t2 < $t3) ? 1 : 0"},
16              "addi": {"type": "I", "args": ["reg", "reg", "int"], "fields": {"op": 8, "rs": "#2", "rt": "#1", "imm": "#3"}, "
                    desc": "$t1 = $t2 + 23"},
17              "srl":  {"type": "R", "args": ["reg", "reg", "int"], "fields": {"op": 0, "rs": "#2", "rt": 0, "rd": "#1", "shamt":
                     "#3", "func": 2}, "desc" : "rd = rs >> shamt"},
18              "beq":  {"type": "I", "args": ["reg", "reg", "offset"], "fields": {"op": 4, "rs": "#1", "rt": "#2", "imm": "#3"}, "
                    desc": "PC += ($t1 == $t2) ? (offset * 4 + 4) : 4"},
19              "lw":   {"type": "I", "args": ["reg", "data"], "fields": {"op": 35, "rs": "#2.offset", "rt": "#1", "imm": "#2.base"
                    }, "desc": "$t1 = MEM[base + $t2]"},
20              "sw":   {"type": "I", "args": ["reg", "data"], "fields": {"op": 43, "rs": "#2.offset", "rt": "#1", "imm": "#2.base"
                    }, "desc": "MEM[base + $t2] = $t1"},
```

```
21              "j":    {"type": "J", "args": ["target"], "fields": {"op": 2, "target": "#1"}, "desc": "PC = target"},
22              "jalr": {"type": "R", "args": ["reg"], "fields": {"op": 9, "rs": "#1", "rt": 0, "rd": 31, "shamt": 0, "func": 9}, "
                      desc" : "$ra = PC ; PC = rs"}
23      },
24      "pseudo": {
25              "li":   {"args": ["reg", "int"], "to": ["addi #1, $0, #2"], "desc": "$t1 = 22"},
26              "la":   {"args": ["reg", "label"], "to": ["addi #1, $0, #2"], "desc": "$t1 = ADDR(label)"},
27              "move": {"args": ["reg", "reg"], "to": ["add #1, #2, $0"], "desc": "$t1 = $t2"},
28              "subi": {"args": ["reg", "reg", "int"], "to": ["li $1, #3", "sub #1, #2, $1"], "desc": "$t1 = $t2 - 23"},
29              "sgt":  {"args": ["reg", "reg", "reg"], "to": ["slt #1, #3, #2"], "desc": "$t1 = ($t2 > $t3) ? 1 : 0"},
30              "bge":  {"args": ["reg", "reg", "offset"], "to": ["slt $1, #1, #2", "beq $1, $0, #3"], "desc": "PC += ($t1 >= $t2)
                      ? (offset * 4 + 4) : 4"},
31              "ble":  {"args": ["reg", "reg", "offset"], "to": ["sgt $1, #1, #2", "beq $1, $0, #3"], "desc": "PC += ($t1 <= $t2)
                      ? (offset * 4 + 4) : 4"},
32              "b":    {"args": ["offset"], "to": ["beq $0, $0, #1"], "desc": "PC += offset * 4 + 4"},
33              "neg":  {"args": ["reg", "reg"], "to": ["sub #1, $0, #2"], "desc": "$t1 = -$t2"},
34              "not":  {"args": ["reg", "reg"], "to": ["nor #1, #2, $0"], "desc": "$t1 = ~$t2"}
35      },
36      "control": {
37              "0": {"RegDst": 1, "RegWrite": 1, "ALUOp": 2, "ALUSrc": 0, "MemToReg": 0},
38              "9": {"RegDst": 1, "RegWrite": 1, "ALUOp": 2, "ALUSrc": 0, "MemToReg": 0, "JumpReg": 1},
39              "8": {"RegDst": 0, "RegWrite": 1, "ALUOp": 0, "ALUSrc": 1, "MemToReg": 0},
40              "4": {"ALUOp": 1, "ALUSrc": 0, "Branch": 1},
41              "2": {"Jump": 1},
42              "35": {"ALUOp": 0, "ALUSrc": 1, "RegDst": 0, "RegWrite": 1, "MemRead": 1, "MemWrite": 0, "MemToReg": 1},
43              "43": {"ALUOp": 0, "ALUSrc": 1, "RegDst": 0, "RegWrite": 0, "MemRead": 0, "MemWrite": 1, "MemToReg": 0}
44      },
45      "alu": {
46              "aluop_size": 2,
47              "func_size": 6,
48              "control_size": 4,
49              "control": [
50                      {"aluop": 0, "out": {"Operation": 2}},
51                      {"aluop": 1, "out": {"Operation": 6}},
52                      {"aluop": 2, "func": 2, "out":  {"Operation": 1}},
53                      {"aluop": 2, "func": 32, "out": {"Operation": 2}},
54                      {"aluop": 2, "func": 34, "out": {"Operation": 6}},
55                      {"aluop": 2, "func": 36, "out": {"Operation": 0}},
56                      {"aluop": 2, "func": 37, "out": {"Operation": 8}},
57                      {"aluop": 2, "func": 42, "out": {"Operation": 6}},
58                      {"aluop": 2, "func": 39, "out": {"Operation": 12}}
59              ],
60              "operations": {
61                      "0": "and",
62                      "1": "srl",
63                      "2": "add",
64                      "4": "sub",
65                      "6": "slt",
66                      "8": "or",
67                      "12": "nor"
68              }
69      }
70 }
```

4.4. pipeline.cpu

```
1 {
2       "components": {
3               "PC":          {"type": "PC", "x": 40, "y": 250, "in": "NewPC", "out": "PC", "write": "Write", "desc": {"default": "
                      Contains the address of the instruction being executed.\nThe address is updated (with the value at the input
                      ) at the clock transition if Write is active.", "pt": "Contém o endereço da instrução a ser executada.\
                      nO endereço actualizado (com o valor da entrada) na transição do relógio se Write estiver activo."}},
4               "ForkPC":      {"type": "Fork", "x": 82, "y": 265, "size": 32, "in": "In", "out": ["Out1", "Out2"]},
5               "PCAdder":     {"type": "Add", "latency": 50, "x": 110, "y": 158, "in1": "In1", "in2": "In2", "out": "PC+4", "desc":
                       {"default": "Calculates the address of the next sequential instruction.", "pt": "Calcula o endereço da
                      instrução sequencial seguinte."}},
6               "Const4":      {"type": "Constant", "x": 85, "y": 173, "out": "Out", "val": 4, "size": 32},
7               "ForkPCAdder":{"type": "Fork", "x": 155, "y": 175, "size": 32, "in": "In", "out": ["Out1", "Out2"]},
8               "MuxPC":       {"type": "Multiplexer", "latency": 15, "x": 15, "y": 248, "size": 32, "sel": "PCSrc", "in": ["0", "1"
                      ], "out": "Out", "desc": {"default": "Selects PC+4 or the branch address as the new PC.", "pt": "Selecciona
                      o PC+4 ou o endereço de branch como novo PC."}},
9               "InstMem":     {"type": "InstructionMemory", "latency": 300, "x": 90, "y": 215, "in": "Address", "out": "Instruction
                      "},
10              "MuxJump":     {"type": "Multiplexer", "latency": 15, "x": 70, "y": 125, "size": 32, "sel": "Jump", "in": ["0", "1"]
                      , "out": "Out", "desc": {"default": "Selects PC+4 or the branch address as the new PC.", "pt": "Selecciona o
                       PC+4 ou o endereço de branch como novo PC."}},
11              "MuxJumpReg": {"type": "Multiplexer", "latency": 15, "x": 15, "y": 300, "size": 32, "sel": "JumpReg", "in": ["0", "
                      1"], "out": "Out", "desc": {"default": "Selects PC+4 or the branch address as the new PC.", "pt": "
                      Selecciona o PC+4 ou o endereço de branch como novo PC."}},
12
13              "IF/ID":       {"type": "PipelineRegister", "x": 180, "y": 110, "write": "Write", "flush": "Flush", "regs": {"NewPC"
                      : 32, "Instruction": 32}},
14
15              "DistInst":    {"type": "Distributor", "x": 200, "y": 260, "in": {"id": "Instruction", "size": 32}, "out": [{"msb":
                      31, "lsb": 26}, {"msb": 25, "lsb": 21}, {"msb": 20, "lsb": 16},   {"msb": 15, "lsb": 11}, {"msb": 15, "lsb":
                       0}, {"msb": 25, "lsb": 0}, {"msb": 10, "lsb": 6}]},
16              "ForkRt":      {"type": "Fork", "x": 220, "y": 265, "size": 5, "in": "In", "out": ["Out1", "Out2"]},
17              "RegBank":     {"type": "RegBank", "latency": 100, "x": 260, "y": 225, "num_regs": 32, "read_reg1": "ReadReg1", "
                      read_reg2": "ReadReg2", "read_data1": "ReadData1", "read_data2": "ReadData2", "write_reg": "WriteReg", "
                      write_data": "WriteData", "reg_write": "RegWrite", "forwarding": true, "const_regs": [{"reg": 0, "val": 0}],
                       "desc": {"default": "Holds all the MIPS registers and provides read/write to them.\nThe values of the
                      ReadReg1 and ReadReg2 registers are read to the outputs.\nWriteData is written to the WriteReg register at
                      the clock transition if RegWrite is enabled.\nWhen the same register is read from and written to in the same
                       clock cycle, this register bank also forwards the written value to the output.", "pt": "Contém todos os
                      registos do MIPS e fornece acesso de leitura/escrita aos mesmos.\nOs valores dos registos ReadReg1 e ReadReg
                      2 são lidos para as saídas.\nWriteData é escrito para o registo WriteReg na transição do relógio se
                      RegWrite estiver activo.\nQuando o mesmo registo é lido e escrito no mesmo ciclo de relógio, este banco de
                       registos também encaminha o valor escrito para a saída."}},
18              "Control":     {"type": "ControlUnit", "latency": 50, "x": 230, "y": 70, "in": "Opcode"},
19              "ExtendImm":   {"type": "SignExtend", "x": 280, "y": 330, "in": {"id": "In", "size": 16}, "out": {"id": "Out", "size
                      ": 32}, "desc": {"default": "Extends the instruction's immediate value from 16 to 32 bits, in the case it is
                       an I-type instruction.", "pt": "Estende o valor imediato da instrução de 16 para 32 bits, no caso de ser
                      uma instrução do tipo I."}},
20              "ForkRs":      {"type": "Fork", "x": 230, "y": 245, "size": 5, "in": "In", "out": ["Out1", "Out2"]},
21              "HazardUnit": {"type": "HazardDetectionUnit", "latency": 50, "x": 230, "y": 10, "id_ex_mem_read": "ID/EX.MemRead",
                      "id_ex_rt": "ID/EX.Rt", "if_id_rs": "IF/ID.Rs", "if_id_rt": "IF/ID.Rt", "stall": "Stall"},
22              "ForkStall":   {"type": "Fork", "x": 265, "y": 5, "size": 1, "in": "In", "out": ["Out1", "Out2"]},
23              "ForkRs2":     {"type": "Fork", "x": 215, "y": 245, "size": 5, "in": "In", "out": ["Out1", "Out2"]},
24              "ForkRt3":     {"type": "Fork", "x": 220, "y": 275, "size": 5, "in": "In", "out": ["Out1", "Out2"]},
25              "NotStall":    {"type": "Not", "x": 115, "y": 2, "in": "Stall", "out": "Write"},
26              "ForkWrite":   {"type": "Fork", "x": 130, "y": 50, "size": 1, "in": "In", "out": ["Out1", "Out2"]},
```

```
27          "ShiftJump":  {"type": "ShiftLeft", "x": 240, "y": 180, "in": {"id": "In", "size": 26}, "out": {"id": "Out", "size"
                : 28}, "amount": 2, "desc": {"default": "The 2 less significant bits of the addresses of the instructions
                are always 00 (the addresses are multiples of 4). As such, these bits are not included in the instruction's
                immediate value (offset).\nThis component restores those bits by shifting the value 2 bits to the left (or
                multiplying by 4), in case it is a branch instruction.", "pt": "Os 2 bits menos significativos dos
                endereços das instruções são sempre 00 (os endereços são múltiplos de 4). Como tal, estes bits são
                omitidos no valor imediato da instrução (offset).\nEste componente restaura esses bits deslocando o
                valor 2 bits para a esquerda (ou multiplicando por 4), no caso de ser uma instrução de branch."}},
28          "DistJump":   {"type": "Distributor", "x": 315, "y": 165, "in": {"id": "PC+4", "size": 32}, "out": [{"msb": 31, "
                lsb": 0}, {"msb": 31, "lsb": 28}]},
29          "ConcatJump": {"type": "Concatenator", "x": 340, "y": 200, "in1": {"id": "In1", "size": 4}, "in2": {"id": "In2", "
                size": 28}, "out": "Out", "desc": {"default": "Concatenates the 4 most significant bits of the PC with the 2
                8 bits of the target jump address to form the final 32 bits address.", "pt": "Concatena os 4 bits mais
                significativos do PC com os 28 bits do endereço alvo de salto para formar o endereço final de 32 bits."}},
30          "OrFlush1":   {"type": "Or", "x": 172, "y": 75, "in1": "Jump", "in2": "Branch", "out": "Flush"},
31          "ForkJump":   {"type": "Fork", "x": 193, "y": 68, "size": 1, "in": "In", "out": ["Out1", "Out2"]},
32          "ConcatSRL":  {"type": "Concatenator", "x": 350, "y": 327, "in1": {"id": "In1", "size": 27}, "in2": {"id": "In2", "
                size": 5}, "out": "Out", "desc": {"default": "Concatenates the 4 most significant bits of the PC with the 28
                 bits of the target jump address to form the final 32 bits address.", "pt": "Concatena os 4 bits mais
                significativos do PC com os 28 bits do endereço alvo de salto para formar o endereço final de 32 bits."}},
33          "Const0":     {"type": "Constant", "x": 333, "y": 331, "out": "Out", "val": 0, "size": 27},
34          "OrFlush3":   {"type": "Or", "x": 330, "y": 71, "in1": "Jump", "in2": "JumpReg", "out": "Flush"},
35
36          "ID/EX":      {"type": "PipelineRegister", "x": 390, "y": 110, "regs": {"ReadData1": 32, "Shamt": 32, "ReadData2":
                32, "NewPC": 32, "Imm": 32, "Rs": 5, "Rt": 5, "Rd": 5, "RegDst": 1, "ALUOp": 2, "ALUSrc": 1, "Branch": 1, "
                MemRead": 1, "MemWrite": 1, "MemToReg": 1, "RegWrite": 1, "JumpReg": 1}, "desc": {"default": "Register that
                separates two pipeline stages.\nThe values that transition to the next stage are stored here temporarily.\
                nIf Flush is active, all values are set to zero, inserting a NOP instruction.", "pt": "Registo que separa
                duas etapas do pipeline.\nOs valores que transitam para a próxima etapa são armazenados aqui
                temporariamente.\nSe Flush estiver activo, todos os valores são colocados a zero, inserindo uma instrução
                 NOP."}},
37
38
39          "ForkReg":    {"type": "Fork", "x": 445, "y": 281, "size": 32, "in": "In", "out": ["Out1", "Out2"]},
            "MuxFwdA":    {"type": "Multiplexer", "latency": 15, "x": 425, "y": 230, "size": 32, "sel": "ForwardA", "out": "Out
                ", "in": ["0", "1", "2"], "desc": {"default": "Selects, in conjunction with the forwarding unit, if the
                value of the 1st register comes from the register bank or if it's forwarded from one of the next stages.",
                "pt": "Selecciona, em conjunto com a unidade de atalhos, se o valor do registo vem do banco de registos
                ou se é encaminhado de uma das etapas seguintes."}},
40          "MuxFwdB":    {"type": "Multiplexer", "latency": 15, "x": 425, "y": 275, "size": 32, "sel": "ForwardB", "out": "Out
                ", "in": ["0", "1", "2"], "desc": {"default": "Selects, in conjunction with the forwarding unit, if the
                value of the 2nd register comes from the register bank or if it's forwarded from one of the next stages.",
                "pt": "Selecciona, em conjunto com a unidade de atalhos, se o valor do registo vem do banco de registos
                ou se é encaminhado de uma das etapas seguintes."}},
41          "ForkEXR2":   {"type": "Fork", "x": 421, "y": 299, "size": 32, "in": "In", "out": ["Out1", "Out2"]},
42          "ForkMEMR2":  {"type": "Fork", "x": 416, "y": 291, "size": 32, "in": "In", "out": ["Out1", "Out2"]},
43          "MuxReg":     {"type": "Multiplexer", "latency": 15, "x": 455, "y": 270, "size": 32, "sel": "ALUSrc", "out": "Out",
                 "in": ["0", "1"], "desc": {"default": "Selects the value of the 2nd read register or the instruction's
                immediate value as the ALU's second operand.", "pt": "Selecciona o valor do registo lido ou o valor
                imediato da instrução como segundo operando da ALU."}},
44          "DistImm":    {"type": "Distributor", "x": 448, "y": 330, "in": {"id": "In", "size": 32}, "out": [{"msb": 31, "lsb"
                : 0}, {"msb": 5, "lsb": 0}]},
45          "ALUControl": {"type": "ALUControl", "latency": 50, "x": 456, "y": 330, "aluop": "ALUOp", "func": "func"},
46          "ALU":        {"type": "ALU", "latency": 100, "x": 530, "y": 237, "in1": "In1", "in2": "In2", "control": "Operation
                ", "out": "Result", "zero": "Zero"},
47          "MuxDst":     {"type": "Multiplexer", "latency": 15, "x": 576, "y": 370, "size": 5, "sel": "RegDst", "in": ["0", "1
                "], "out": "Out", "desc": {"default": "Selects the instruction's rt or rd field as the destination register
                (WriteReg).", "pt": "Selecciona o campo rt ou rd da instrução como registo de destino (WriteReg)."}},
48          "ShiftImm":   {"type": "ShiftLeft", "x": 500, "y": 190, "in": {"id": "In", "size": 32}, "out": {"id": "Out", "size"
                : 32}, "amount": 2, "desc": {"default": "The 2 less significant bits of the addresses of the instructions
                are always 00 (the addresses are multiples of 4). As such, these bits are not included in the instruction's
                immediate value (offset).\nThis component restores those bits by shifting the value 2 bits to the left (or
                multiplying by 4), in case it is a branch instruction.", "pt": "Os 2 bits menos significativos dos
                endereços das instruções são sempre 00 (os endereços são múltiplos de 4). Como tal, estes bits são
                omitidos no valor imediato da instrução (offset).\nEste componente restaura esses bits deslocando o
                valor 2 bits para a esquerda (ou multiplicando por 4), no caso de ser uma instrução de branch."}},
49          "AddBranch":  {"type": "Add", "latency": 50, "x": 560, "y": 164, "in1": "In1", "in2": "In2", "out": "Out", "desc":
                {"default": "Adds the branch offset to the PC+4 to obtain the destination branch address, in case it is a
                branch instruction.", "pt": "Soma o offset do branch ao PC+4 para obter o endereço de destino do branch, no
                 caso de ser uma instrução de branch."}},
50          "ForkImm":    {"type": "Fork", "x": 450, "y": 292, "size": 32, "in": "In", "out": ["Out1", "Out2"]},
51          "ForkRt2":    {"type": "Fork", "x": 408, "y": 381, "size": 5, "in": "In", "out": ["Out1", "Out2", "Out3"]},
52          "ForwardingUnit":{"type": "ForwardingUnit", "latency": 50, "x": 520, "y": 450, "ex_mem_reg_write": "EX/MEM.RegWrite
                ", "mem_wb_reg_write": "MEM/WB.RegWrite", "ex_mem_rd": "EX/MEM.Rd", "mem_wb_rd": "MEM/WB.Rd", "id_ex_rs": "
                ID/EX.Rs", "id_ex_rt": "ID/EX.Rt", "fwd_a": "ForwardA", "fwd_b": "ForwardB"},
53          "MuxSRL":     {"type": "Multiplexer", "latency": 15, "x": 498, "y": 270, "size": 32, "sel": "SRL", "out": "Out", "
                in": ["0", "1"], "desc": {"default": "Selects the value of the 2nd read register or the instruction's
                immediate value as the ALU's second operand.", "pt": "Selecciona o valor do registo lido ou o valor
                imediato da instrução como segundo operando da ALU."}},
54          "DistSRL":    {"type": "Distributor", "x": 503, "y": 330, "in": {"id": "In", "size": 4}, "out": [{"msb": 3, "lsb":
                0}, {"msb": 0, "lsb": 0}]},
55          "ForkPC+4":   {"type": "Fork", "x": 530, "y": 175, "size": 32, "in": "In", "out": ["Out1", "Out2"]},
56          "ForkFwd":    {"type": "Fork", "x": 520, "y": 250, "size": 32, "in": "In", "out": ["Out1", "Out2"]},
57          "ForkJumpReg":{"type": "Fork", "x": 520, "y": 120, "size": 1, "in": "In", "out": ["Out1", "Out2", "Out3", "Out4"]},
58          "OrFlushJump":{"type": "Or", "x": 470, "y": 20, "in1": "Branch", "in2": "JumpReg", "out": "Flush"},
59          "ForkJR":     {"type": "Fork", "x": 520, "y": 30, "size": 1, "in": "In", "out": ["Out1", "Out2", "Out3"]},
60
61          "EX/MEM":     {"type": "PipelineRegister", "x": 600, "y": 110, "regs": {"Result": 32, "ReadData2": 32, "NewPC": 32,
                 "Zero": 1, "RegBankDst": 5, "Target": 32, "Branch": 1, "MemRead": 1, "MemWrite": 1, "MemToReg": 1, "
                RegWrite": 1, "JumpReg": 1}, "desc": {"default": "Register that separates two pipeline stages.\nThe values
                that transition to the next stage are stored here temporarily.\nIf Flush is active, all values are set to
                zero, inserting a NOP instruction.", "pt": "Registo que separa duas etapas do pipeline.\nOs valores que
                transitam para a próxima etapa são armazenados aqui temporariamente.\nSe Flush estiver activo, todos os
                valores são colocados a zero, inserindo uma instrução NOP."}},
62
63          "ForkMem":    {"type": "Fork", "x": 625, "y": 275, "size": 32, "in": "In", "out": ["Out1", "Out2"]},
64          "ForkEXR1":   {"type": "Fork", "x": 625, "y": 360, "size": 32, "in": "In", "out": ["Out1", "Out2"]},
65          "DataMem":    {"type": "DataMemory", "latency": 400, "x": 630, "y": 242, "size": 100, "address": "Address", "
                write_data": "WriteData", "out": "ReadData", "mem_read": "MemRead", "mem_write": "MemWrite"},
66          "AndBranch":  {"type": "And", "x": 650, "y": 180, "in1": "Branch", "in2": "Zero", "out": "Branch", "desc": {"
                default": "Determines if a branch should occur.", "pt": "Determina se um branch é efectuado."}},
67          "ForkDst1":   {"type": "Fork", "x": 620, "y": 387, "size": 5, "in": "In", "out": ["Out1", "Out2"]},
68          "ForkRegWR1": {"type": "Fork", "x": 720, "y": 125, "size": 1, "in": "In", "out": ["Out1", "Out2"]},
69          "ForkMemRd":  {"type": "Fork", "x": 420, "y": 140, "size": 1, "in": "In", "out": ["Out1", "Out2"]},
70          "ForkBr1":    {"type": "Fork", "x": 605, "y": 62, "size": 1, "in": "In", "out": ["Out1", "Out2"]},
71          "ForkBr2":    {"type": "Fork", "x": 510, "y": 62, "size": 1, "in": "In", "out": ["Out1", "Out2"]},
72          "ForkBr3":    {"type": "Fork", "x": 183, "y": 62, "size": 1, "in": "In", "out": ["Out1", "Out2"]},
73          "OrFlush2":   {"type": "Or", "x": 375, "y": 71, "in1": "Stall", "in2": "Branch", "out": "Flush"},
74
75          "MEM/WB":     {"type": "PipelineRegister", "x": 730, "y": 110, "regs": {"Result": 32, "ReadData": 32, "NewPC": 32,
                "RegBankDst": 5, "MemToReg": 1, "RegWrite": 1, "JumpReg": 1}, "desc": {"default": "Register that separates
                two pipeline stages.\nThe values that transition to the next stage are stored here temporarily.", "pt": "
                Registo que separa duas etapas do pipeline.\nOs valores que transitam para a próxima etapa são armazenados
                 aqui temporariamente."}},
76
77          "MuxMem":     {"type": "Multiplexer", "latency": 15, "x": 765, "y": 270, "size": 32, "sel": "MemToReg", "in": ["0",
                 "1"], "out": "Out", "desc": {"default": "Selects the result of the ALU or the value read from memory to
                write to the destination register (WriteData).", "pt": "Selecciona o resultado da ALU ou o valor lido da
                memória para escrever no registo de destino (WriteData)."}},
```

```
            "ForkRegWR2": {"type": "Fork", "x": 760, "y": 125, "size": 1, "in": "In", "out": ["Out1", "Out2"]},
            "ForkDst2":   {"type": "Fork", "x": 755, "y": 480, "size": 5, "in": "In", "out": ["Out1", "Out2"]},
            "ForkMemR1":  {"type": "Fork", "x": 416, "y": 520, "size": 32, "in": "In", "out": ["Out1", "Out2"]},
            "MuxJALR":    {"type": "Multiplexer", "latency": 15, "x": 795, "y": 270, "size": 32, "sel": "JALR", "in": ["0", "1"
                ], "out": "Out", "desc": {"default": "Selects the result of the ALU or the value read from memory to write
                to the destination register (WriteData).", "pt": "Selecciona o resultado da ALU ou o valor lido da memória
                para escrever no registo de destino (WriteData)."}},
        },
        "wires": [
            {"from": "PC", "out": "PC", "to": "ForkPC", "in": "In"},
            {"from": "ForkPC", "out": "Out1", "to": "InstMem", "in": "Address"},
            {"from": "ForkPC", "out": "Out2", "to": "PCAdder", "in": "In1", "points": [{"x": 82, "y": 169}]},
            {"from": "Const4", "out": "Out", "to": "PCAdder", "in": "In2"},
            {"from": "PCAdder", "out": "PC+4", "to": "ForkPCAdder", "in": "In"},
            {"from": "ForkPCAdder", "out": "Out1", "to": "MuxJump", "in": "0", "points": [{"x": 155, "y": 150}], "end": {"x": 8
                5, "y": 150}},
            {"from": "MuxJump", "out": "Out", "to": "MuxJumpReg", "in": "0","start": {"x": 70, "y": 143}, "points": [{"x": 35,
                "y": 143}, {"x": 35, "y": 311}], "end": {"x": 30, "y": 311}},
            {"from": "MuxPC", "out": "Out", "to": "PC", "in": "NewPC"},
            {"from": "ForkPCAdder", "out": "Out2", "to": "IF/ID", "in": "NewPC", "end": {"x": 180, "y": 175}},
            {"from": "InstMem", "out": "Instruction", "to": "IF/ID", "in": "Instruction", "end": {"x": 180, "y": 265}},
            {"from": "MuxJumpReg", "out": "Out", "to": "MuxPC", "in": "0", "start": {"x": 15, "y": 317}, "points": [{"x": 10, "
                y": 317}, {"x": 10, "y": 270}], "end": {"x": 15, "y": 270}},

            {"from": "IF/ID", "out": "NewPC", "to": "ID/EX", "in": "NewPC", "start": {"x": 195, "y": 175}, "end": {"x": 390, "y
                ": 175}},
            {"from": "IF/ID", "out": "Instruction", "to": "DistInst", "in": "Instruction", "start": {"x": 195, "y": 275}},
            {"from": "DistInst", "out": "31-26", "to": "Control", "in": "Opcode", "start": {"x": 205, "y": 265}, "points": [{"x
                ": 210, "y": 265}, {"x": 210, "y": 120}]},
            {"from": "DistInst", "out": "25-21", "to": "ForkRs2", "in": "In", "start": {"x": 205, "y": 270}, "points": [{"x": 2
                15, "y": 270}]},
            {"from": "ForkRs2", "out": "Out1", "to": "ForkRs", "in": "In", "points": [{"x": 215, "y": 245}]},
            {"from": "ForkRs2", "out": "Out2", "to": "HazardUnit", "in": "IF/ID.Rs", "points": [{"x": 215, "y": 26}]},
            {"from": "ForkRs", "out": "Out1", "to": "RegBank", "in": "ReadReg1"},
            {"from": "ForkRs", "out": "Out2", "to": "ID/EX", "in": "Rs", "points": [{"x": 230, "y": 372}], "end": {"x": 390, "y
                ": 372}},
            {"from": "DistInst", "out": "20-16", "to": "ForkRt3", "in": "In", "start": {"x": 205, "y": 275}},
            {"from": "ForkRt3", "out": "Out1", "to": "ForkRt", "in": "In"},
            {"from": "ForkRt", "out": "Out1", "to": "RegBank", "in": "ReadReg2"},
            {"from": "ForkRt", "out": "Out2", "to": "HazardUnit", "in": "IF/ID.Rt", "points": [{"x": 220, "y": 42}]},
            {"from": "ForkRt3", "out": "Out2", "to": "ID/EX", "in": "Rt", "points": [{"x": 220, "y": 381}], "end": {"x": 390, "
                y": 381}},
            {"from": "DistInst", "out": "15-0", "to": "ExtendImm", "in": "In", "start": {"x": 205, "y": 280}, "points": [{"x":
                225, "y": 280}, {"x": 225, "y": 350}]},
            {"from": "DistInst", "out": "15-11", "to": "ID/EX", "in": "Rd", "start": {"x": 205, "y": 285}, "points": [{"x": 215
                , "y": 285}, {"x": 215, "y": 392}], "end": {"x": 390, "y": 392}},
            {"from": "RegBank", "out": "ReadData1", "to": "ID/EX", "in": "ReadData1", "end": {"x": 390, "y": 258}},
            {"from": "RegBank", "out": "ReadData2", "to": "ID/EX", "in": "ReadData2", "end": {"x": 390, "y": 291}},
            {"from": "ExtendImm", "out": "Out", "to": "ID/EX", "in": "Imm", "end": {"x": 390, "y": 350}},
            {"from": "Control", "out": "ALUOp", "to": "ID/EX", "in": "ALUOp", "start": {"x": 290, "y": 160}, "end": {"x": 390,
                "y": 160}},
            {"from": "Control", "out": "ALUSrc", "to": "ID/EX", "in": "ALUSrc", "start": {"x": 290, "y": 155}, "end": {"x": 390
                , "y": 155}},
            {"from": "Control", "out": "RegDst", "to": "ID/EX", "in": "RegDst", "start": {"x": 290, "y": 150}, "end": {"x": 390
                , "y": 150}},
            {"from": "Control", "out": "Branch", "to": "ID/EX", "in": "Branch", "start": {"x": 290, "y": 145}, "end": {"x": 390
                , "y": 145}},
            {"from": "Control", "out": "MemRead", "to": "ID/EX", "in": "MemRead", "start": {"x": 290, "y": 140}, "end": {"x": 3
                90, "y": 140}},
            {"from": "Control", "out": "MemWrite", "to": "ID/EX", "in": "MemWrite", "start": {"x": 290, "y": 135}, "end": {"x":
                 390, "y": 135}},
            {"from": "Control", "out": "MemToReg", "to": "ID/EX", "in": "MemToReg", "start": {"x": 290, "y": 130}, "end": {"x":
                 390, "y": 130}},
            {"from": "Control", "out": "RegWrite", "to": "ID/EX", "in": "RegWrite", "start": {"x": 290, "y": 125}, "end": {"x":
                 390, "y": 125}},
            {"from": "HazardUnit", "out": "Stall", "to": "ForkStall", "in": "In"},
            {"from": "ForkStall", "out": "Out1", "to": "OrFlush2", "in": "Stall", "points": [{"x": 382, "y": 5}], "end": {"x":
                382, "y": 70}},
            {"from": "OrFlush2", "out": "Flush", "to": "ID/EX", "in": "Flush", "start": {"x": 395, "y": 100}},
            {"from": "ForkStall", "out": "Out2", "to": "NotStall", "in": "Stall", "end": {"x": 145, "y": 5}},
            {"from": "NotStall", "out": "Write", "to": "ForkWrite", "in": "In", "start": {"x": 130, "y": 32}},
            {"from": "ForkWrite", "out": "Out1", "to": "IF/ID", "in": "Write", "end": {"x": 180,"y": 125}, "points": [{"x": 130
                ,"y": 125}]},
            {"from": "ForkWrite", "out": "Out2", "to": "PC", "in": "Write", "points": [{"x": 55, "y": 50}]},
            {"from": "DistInst", "out": "25-0", "to": "ShiftJump", "in": "In", "start": {"x": 202, "y": 260}, "points": [{"x":
                202, "y": 200}]},
            {"from": "ShiftJump", "out": "Out", "to": "ConcatJump", "in": "In2"},
            {"from": "DistJump", "out": "31-28", "to": "ConcatJump", "in": "In1", "points": [{"x": 340, "y": 185}]},
            {"from": "Control", "out": "Jump", "to": "OrFlush3", "in": "Jump", "start": {"x": 290, "y": 115}, "points": [{"x":
                345, "y": 115}], "end": {"x": 345, "y": 101}},
            {"from": "OrFlush3", "out": "Flush", "to": "ForkJump", "in": "In", "start": {"x": 330, "y": 86}, "points": [{"x": 3
                20, "y": 86}, {"x": 320, "y": 68}]},
            {"from": "ForkJump", "out": "Out1", "to": "OrFlush1", "in": "Jump", "end": {"x": 193, "y":75}},
            {"from": "ForkJump", "out": "Out2", "to": "MuxJump", "in": "Jump", "points": [{"x": 77, "y": 68}]},
            {"from": "ConcatJump", "out": "Out", "to": "MuxJump", "in": "1", "points": [{"x": 360, "y": 200}, {"x": 360, "y": 4
                20}, {"x": 175, "y": 420}, {"x": 175, "y": 137}], "end": {"x": 85, "y": 137}},
            {"from": "OrFlush1", "out": "Flush", "to": "IF/ID", "in": "Flush", "start": {"x": 188, "y": 100}, "end": {"x": 188,
                 "y":110}},
            {"from": "DistInst", "out": "10-6", "to": "ConcatSRL", "in": "In2", "start": {"x": 202, "y": 290}, "points": [{"x":
                 202, "y": 327}]},
            {"from": "Const0", "out": "Out", "to": "ConcatSRL", "in": "In1", "start": {"x": 347, "y": 338},  "points": [{"x": 3
                50, "y": 338}]},
            {"from": "ConcatSRL", "out": "Out", "to": "ID/EX", "in": "Shamt", "end": {"x": 390, "y": 327}},
            {"from": "Control", "out": "JumpReg", "to": "ID/EX", "in": "JumpReg", "start": {"x": 290, "y": 120}, "end": {"x": 3
                90, "y": 120}},

            {"from": "ID/EX", "out": "ReadData1", "to": "MuxFwdA", "in": "0", "start": {"x": 405, "y": 248}, "end": {"x": 425,
                 "y": 248}},
            {"from": "MuxFwdA", "out": "Out", "to": "ForkFwd", "in": "In", "start": {"x": 440, "y": 250}},
            {"from": "ForkFwd", "out": "Out1", "to": "ALU", "in": "In1", "end": {"x": 530, "y": 250}},
            {"from": "ID/EX", "out": "ReadData2", "to": "MuxFwdB", "in": "0", "start": {"x": 405, "y": 281}, "end": {"x": 425,
                 "y": 281}},
            {"from": "MuxFwdB", "out": "Out", "to": "ForkReg", "in": "In", "start": {"x": 440, "y": 281}},
            {"from": "ForkEXR2", "out": "Out2", "to": "MuxFwdB", "in": "2"},
            {"from": "ForkMEMR2", "out": "Out2", "to": "MuxFwdB", "in": "1"},
            {"from": "ForkReg", "out": "Out1", "to": "MuxReg", "in": "0"},
            {"from": "ForkReg", "out": "Out2", "to": "EX/MEM", "in": "ReadData2", "points": [{"x": 445, "y": 308}], "end": {"x"
                : 600, "y": 308}},
            {"from": "MuxReg", "out": "Out", "to": "MuxSRL", "in": "0", "start": {"x": 470, "y": 281}},
            {"from": "MuxSRL", "out": "Out", "to": "ALU", "in": "In2", "end": {"x": 530, "y": 287}},
            {"from": "ID/EX", "out": "ALUSrc", "to": "MuxReg", "in": "ALUSrc", "start": {"x": 405, "y": 155}, "points": [{"x":
                462, "y": 155}]},
            {"from": "ID/EX", "out": "Imm", "to": "DistImm", "in": "In", "start": {"x": 405, "y": 350}, "end": {"x": 448, "y":
                350}},
            {"from": "DistImm", "out": "5-0", "to": "ALUControl", "in": "func", "start": {"x": 448, "y": 350}},
```

```
{"from": "ID/EX", "out": "ALUOp", "to": "ALUControl", "in": "ALUOp", "start": {"x": 405, "y": 160}, "points": [{"x": 446, "y": 160}, {"x": 446, "y": 235}, {"x": 476, "y": 235}]},
{"from": "ALU", "out": "Zero", "to": "EX/MEM", "in": "Zero", "start": {"x": 590, "y": 255}, "end": {"x": 600, "y": 255}},
{"from": "ALU", "out": "Result", "to": "EX/MEM", "in": "Result", "start": {"x": 590, "y": 275}, "end": {"x": 600, "y": 275}},
{"from": "ID/EX", "out": "Rt", "to": "ForkRt2", "in": "In", "start": {"x": 405, "y": 381}},
{"from": "ForkRt2", "out": "Out1", "to": "MuxDst", "in": "0"},
{"from": "ForkRt2", "out": "Out2", "to": "ForwardingUnit", "in": "ID/EX.Rt", "points": [{"x": 408, "y": 482}]},
{"from": "ForkRt2", "out": "Out3", "to": "HazardUnit", "in": "ID/EX.Rt", "points": [{"x": 408, "y": 42}]},
{"from": "ID/EX", "out": "Rd", "to": "MuxDst", "in": "1", "start": {"x": 405, "y": 392}},
{"from": "ID/EX", "out": "RegDst", "to": "MuxDst", "in": "RegDst", "start": {"x": 405, "y": 150}, "points": [{"x": 597, "y": 150}, {"x": 597, "y": 365}, {"x": 583, "y": 365}]},
{"from": "MuxDst", "out": "Out", "to": "EX/MEM", "in": "RegBankDst", "end": {"x": 600, "y": 387}},
{"from": "DistImm", "out": "31-0", "to": "ForkImm", "in": "In", "start": {"x": 450, "y": 330}},
{"from": "ForkImm", "out": "Out1", "to": "MuxReg", "in": "1"},
{"from": "ForkImm", "out": "Out2", "to": "ShiftImm", "in": "In", "points": [{"x": 450, "y": 210}]},
{"from": "ShiftImm", "out": "Out", "to": "AddBranch", "in": "In2", "points": [{"x": 552, "y": 210}, {"x": 552, "y": 186}]},
{"from": "AddBranch", "out": "Out", "to": "EX/MEM", "in": "Target", "end": {"x": 600, "y": 181}},
{"from": "ID/EX", "out": "Branch", "to": "EX/MEM", "in": "Branch", "start": {"x": 405, "y": 145}, "end": {"x": 600, "y": 145}},
{"from": "ID/EX", "out": "MemRead", "to": "ForkMemRd", "in": "In", "start": {"x": 405, "y": 140}},
{"from": "ForkMemRd", "out": "Out1", "to": "EX/MEM", "in": "MemRead", "end": {"x": 600, "y": 140}},
{"from": "ForkMemRd", "out": "Out2", "to": "HazardUnit", "in": "ID/EX.MemRead", "points": [{"x": 420, "y": 26}]},
{"from": "ID/EX", "out": "MemWrite", "to": "EX/MEM", "in": "MemWrite", "start": {"x": 405, "y": 135}, "end": {"x": 600, "y": 135}},
{"from": "ID/EX", "out": "MemToReg", "to": "EX/MEM", "in": "MemToReg", "start": {"x": 405, "y": 130}, "end": {"x": 600, "y": 130}},
{"from": "ID/EX", "out": "RegWrite", "to": "EX/MEM", "in": "RegWrite", "start": {"x": 405, "y": 125}, "end": {"x": 600, "y": 125}},
{"from": "ID/EX", "out": "Rs", "to": "ForwardingUnit", "in": "ID/EX.Rs", "start": {"x": 405, "y": 372}, "points": [{"x": 413, "y": 372}, {"x": 413, "y": 466}]},
{"from": "ForwardingUnit", "out": "ForwardA", "to": "MuxFwdA", "in": "ForwardA", "points": [{"x": 543, "y": 400}, {"x": 441, "y": 400}, {"x": 441, "y": 270}, {"x": 432, "y": 270}], "end": {"x": 432, "y": 265}},
{"from": "ForwardingUnit", "out": "ForwardB", "to": "MuxFwdB", "in": "ForwardB", "points": [{"x": 566, "y": 405}, {"x": 432, "y": 405}], "end": {"x": 432, "y": 310}},
{"from": "ID/EX", "out": "Shamt", "to": "MuxSRL", "in": "1", "start": {"x": 405, "y": 325}, "points": [{"x": 490, "y": 325}, {"x": 490, "y": 297}], "end": {"x": 498, "y": 297}},
{"from": "DistSRL", "out": "0-0", "to": "MuxSRL", "in": "SRL", "start": {"x": 505, "y": 330}, "end": {"x": 505, "y": 305}},
{"from": "ALUControl", "out": "Operation", "to": "DistSRL", "in": "In", "end": {"x": 503, "y": 350}},
{"from": "DistSRL", "out": "3-0", "to": "ALU", "in": "Operation", "points": [{"x": 560, "y": 350}]},
{"from": "ID/EX", "out": "JumpReg", "to": "ForkJumpReg", "in": "In", "start": {"x": 405, "y": 120}},
{"from": "ForkJumpReg", "out": "Out1", "to": "EX/MEM", "in": "JumpReg", "end": {"x": 600, "y": 120}},
{"from": "ID/EX", "out": "NewPC", "to": "ForkPC+4", "in": "In", "start": {"x": 405, "y": 175}},
{"from": "ForkPC+4", "out": "Out1", "to": "AddBranch", "in": "In1"},
{"from": "ForkPC+4", "out": "Out2", "to": "EX/MEM", "in": "NewPC", "points": [{"x": 530, "y": 155}], "end": {"x": 600, "y": 155}},
{"from": "ForkFwd", "out": "Out2", "to": "MuxJumpReg", "in": "1", "points": [{"x": 520, "y": 430}, {"x": 45, "y": 430}, {"x": 45, "y": 323}], "end": {"x": 30, "y": 323}},
{"from": "ForkJumpReg", "out": "Out2", "to": "MuxJumpReg", "in": "JumpReg", "points": [{"x": 520, "y": 30}, {"x": 830, "y": 30}, {"x": 830, "y": 530}, {"x": 22, "y": 530}], "end": {"x": 22, "y": 335}},
{"from": "ForkJumpReg", "out": "Out3", "to": "OrFlushJump", "in": "JumpReg", "points": [{"x": 520, "y": 30}], "end": {"x": 500, "y": 30}},
{"from": "ForkJumpReg", "out": "Out4", "to": "OrFlush3", "in": "JumpReg", "points": [{"x": 520, "y": 10}, {"x": 345, "y": 10}], "end": {"x": 345, "y": 71}},

{"from": "EX/MEM", "out": "RegBankDst", "to": "ForkDst1", "in": "In", "start": {"x": 615, "y": 387}},
{"from": "ForkDst1", "out": "Out1", "to": "MEM/WB", "in": "RegBankDst", "end": {"x": 730, "y": 387}},
{"from": "ForkDst1", "out": "Out2", "to": "ForwardingUnit", "in": "EX/MEM.Rd", "points": [{"x": 620, "y": 460}]},
{"from": "EX/MEM", "out": "Result", "to": "ForkMem", "in": "In", "start": {"x": 615, "y": 275}},
{"from": "ForkMem", "out": "Out1", "to": "DataMem", "in": "Address"},
{"from": "ForkMem", "out": "Out2", "to": "ForkEXR1", "in": "In"},
{"from": "ForkEXR1", "out": "Out1", "to": "MEM/WB", "in": "Result", "end": {"x": 730, "y": 360}},
{"from": "ForkEXR1", "out": "Out2", "to": "ForkEXR2", "in": "In", "points": [{"x": 625, "y": 440}, {"x": 421, "y": 440}]},
{"from": "ForkEXR2", "out": "Out1", "to": "MuxFwdA", "in": "2", "points": [{"x": 421, "y": 259}], "end": {"x": 425, "y": 259}},
{"from": "EX/MEM", "out": "ReadData2", "to": "DataMem", "in": "WriteData", "start": {"x": 615, "y": 308}},
{"from": "DataMem", "out": "ReadData", "to": "MEM/WB", "in": "ReadData", "end": {"x": 730, "y": 292}},
{"from": "EX/MEM", "out": "Target", "to": "MuxPC", "in": "1", "start": {"x": 615, "y": 181}, "points": [{"x": 625, "y": 181}, {"x": 625, "y": 65}, {"x": 10, "y": 65}, {"x": 10, "y": 259}], "end": {"x": 15, "y": 259}},
{"from": "EX/MEM", "out": "Zero", "to": "AndBranch", "in": "Zero", "start": {"x": 615, "y": 255}, "points": [{"x": 622, "y": 255}, {"x": 622, "y": 200}]},
{"from": "EX/MEM", "out": "Branch", "to": "AndBranch", "in": "Branch", "start": {"x": 615, "y": 145}, "points": [{"x": 630, "y": 145}, {"x": 630, "y": 190}]},
{"from": "AndBranch", "out": "Branch", "to": "ForkBr1", "in": "In", "points": [{"x": 690, "y": 195}, {"x": 690, "y": 62}]},
{"from": "ForkBr1", "out": "Out1", "to": "ForkBr2", "in": "In"},
{"from": "ForkBr2", "out": "Out1", "to": "OrFlushJump", "in": "Branch", "points": [{"x": 510, "y": 40}], "end": {"x": 500, "y": 40}},
{"from": "OrFlushJump", "out": "Flush", "to": "OrFlush2", "in": "Branch", "points": [{"x": 397, "y": 35}], "start": {"x": 470, "y": 35}, "end": {"x": 397, "y": 70}},
{"from": "ForkBr1", "out": "Out2", "to": "EX/MEM", "in": "Flush"},
{"from": "ForkBr2", "out": "Out2", "to": "ForkBr3", "in": "In"},
{"from": "ForkBr3", "out": "Out1", "to": "MuxPC", "in": "PCSrc", "points": [{"x": 22, "y": 62}]},
{"from": "ForkBr3", "out": "Out2", "to": "OrFlush1", "in": "Branch", "end": {"x": 183, "y": 75}},
{"from": "EX/MEM", "out": "MemRead", "to": "DataMem", "in": "MemRead", "start": {"x": 615, "y": 140}, "points": [{"x": 640, "y": 140}], "end": {"x": 640, "y": 242}},
{"from": "EX/MEM", "out": "MemWrite", "to": "DataMem", "in": "MemWrite", "start": {"x": 615, "y": 135}, "points": [{"x": 700, "y": 135}], "end": {"x": 700, "y": 242}},
{"from": "EX/MEM", "out": "MemToReg", "to": "MEM/WB", "in": "MemToReg", "start": {"x": 615, "y": 130}, "end": {"x": 730, "y": 130}},
{"from": "EX/MEM", "out": "RegWrite", "to": "ForkRegWR1", "in": "In", "start": {"x": 615, "y": 125}},
{"from": "ForkRegWR1", "out": "Out1", "to": "MEM/WB", "in": "RegWrite", "end": {"x": 730, "y": 125}},
{"from": "ForkRegWR1", "out": "Out2", "to": "ForwardingUnit", "in": "EX/MEM.RegWrite", "points": [{"x": 720, "y": 470}]},
{"from": "EX/MEM", "out": "JumpReg", "to": "MEM/WB", "in": "JumpReg", "start": {"x": 615, "y": 120}, "end": {"x": 730, "y": 120}},
{"from": "EX/MEM", "out": "NewPC", "to": "MEM/WB", "in": "NewPC", "start": {"x": 615, "y": 155}, "end": {"x": 730, "y": 155}},

{"from": "MEM/WB", "out": "ReadData", "to": "MuxMem", "in": "1", "start": {"x": 745, "y": 292}},
{"from": "MEM/WB", "out": "Result", "to": "MuxMem", "in": "0", "start": {"x": 745, "y": 360}, "points": [{"x": 752, "y": 360}, {"x": 752, "y": 281}]},
{"from": "MEM/WB", "out": "MemToReg", "to": "MuxMem", "in": "MemToReg", "start": {"x": 745, "y": 130}, "points": [{"x": 772, "y": 130}]},
{"from": "MEM/WB", "out": "RegBankDst", "to": "ForkDst2", "in": "In", "start": {"x": 745, "y": 387}, "points": [{"x": 755, "y": 387}]},
{"from": "ForkDst2", "out": "Out1", "to": "RegBank", "in": "WriteReg", "points": [{"x": 755, "y": 510}, {"x": 250, "y": 510}, {"x": 250, "y": 285}], "end": {"x": 260, "y": 285}},
{"from": "ForkDst2", "out": "Out2", "to": "ForwardingUnit", "in": "MEM/WB.Rd"},
{"from": "MuxMem", "out": "Out", "to": "MuxJALR", "in": "0", "start": {"x": 780, "y": 295}, "end": {"x": 795, "y": 295}},
{"from": "MuxJALR", "out": "Out", "to": "ForkMemR1", "in": "In", "points": [{"x": 815, "y": 287}, {"x": 815, "y": 520}]},
```

```
235             {"from": "ForkMemR1", "out": "Out1", "to": "RegBank", "in": "WriteData", "points": [{"x": 240, "y": 520}, {"x": 240
                    , "y": 305}], "end": {"x": 260, "y": 305}},
236             {"from": "ForkMemR1", "out": "Out2", "to": "ForkMEMR2", "in": "In"},
237             {"from": "ForkMEMR2", "out": "Out1", "to": "MuxFwdA", "in": "1", "points": [{"x": 416, "y": 253}], "end": {"x": 425
                    , "y": 253}},
238             {"from": "MEM/WB", "out": "RegWrite", "to": "ForkRegWR2", "in": "In", "start": {"x": 745, "y": 125}},
239             {"from": "ForkRegWR2", "out": "Out1", "to": "RegBank", "in": "RegWrite", "points": [{"x": 760, "y": 105}, {"x": 300
                    , "y": 105}]},
240             {"from": "ForkRegWR2", "out": "Out2", "to": "ForwardingUnit", "in": "MEM/WB.RegWrite", "points": [{"x": 760, "y": 4
                    90}]},
241             {"from": "MEM/WB", "out": "JumpReg", "to": "MuxJALR", "in": "JALR", "start": {"x": 745, "y": 120}, "points": [{"x":
                     802, "y": 120}]},
242             {"from": "MEM/WB", "out": "NewPC", "to": "MuxJALR", "in": "1", "start": {"x": 745, "y": 155}, "points": [{"x": 788,
                     "y": 155}, {"x": 788, "y": 283}], "end": {"x": 795, "y": 283}},
243         ],
244         "reg_names": ["zero", "at", "v0", "v1", "a0", "a1", "a2", "a3", "t0", "t1", "t2", "t3", "t4", "t5", "t6", "t7", "s0", "s1",
                "s2", "s3", "s4", "s5", "s6", "s7", "t8", "t9", "k0", "k1", "gp", "sp", "fp", "ra"],
245         "instructions": "default_pipeline.set"
246 }
```

5. Enunciado

66:20 Organización de computadoras
Trabajo práctico 3: Datapath 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 [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 [1], o se puede descargar para Ubuntu, ya sea desde el repositorio de Ubuntu (aunque la versión a veces 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`. Verificar que no se produzcan hazards.
2. Implementar la instrucción `sll` (Shift Left Logical) en el DP `unicycle.cpu`.
3. Implementar la instrucción `srl` (Shift Right Logical)en el DP `pipeline.cpu`.
4. Implementar la instrucción `jalr` (Jump and Link Register) en el DP `unicycle.cpu`.
5. Implementar la instrucción `jalr` 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 [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 conjuntos de instrucciones usados en cada caso.
- Este enunciado.

9. Fechas de entrega.

- Primera entrega: Jueves 20 de Junio.
- Vencimiento: Jueves 27 de Junio.

Referencias

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

[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.