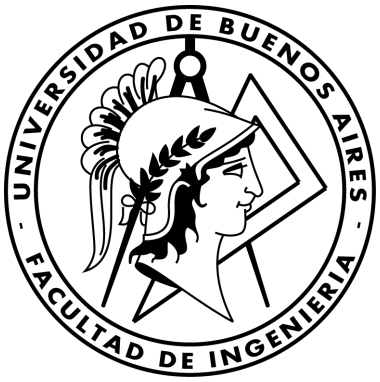

Universidad de Buenos Aires

Facultad de Ingeniería

2do Cuatrimestre de 2020

[86.37 / 66.20] Organización de Computadoras

Curso 2

Trabajo práctico 3

Data Path

Padrón	Alumno	Email
103442	Lovera, Daniel	dlovera@fi.uba.ar
102914	More, Agustín	amore@fi.uba.ar
99846	Torresetti, Lisandro	ltorresetti@fi.uba.ar

Repositorio:
https://github.com/DanieLovera/TPs_organizacion_computador

Índice

1. Introducción

Este trabajo consiste en la implementación de los tres tipos de saltos posibles en MIPS (j, jr, jalr), para esto se utilizó el programa DrMIPS el cual provee un datapath monociclo y otro con pipeline para ser utilizados con fines de aprendizaje. Los archivos Json utilizados y modificados se encuentran disponible en el link al repositorio de github anterior.

2. 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 agregaron instrucciones a diversas configuraciones de CPU provitas por el simulador **DrMIPS** [1].

3. Implementaciones

A continuación se presentan las CPU Monociclo y Pipeline con las que se trabajo principalmente sin ningún tipo de modificación.

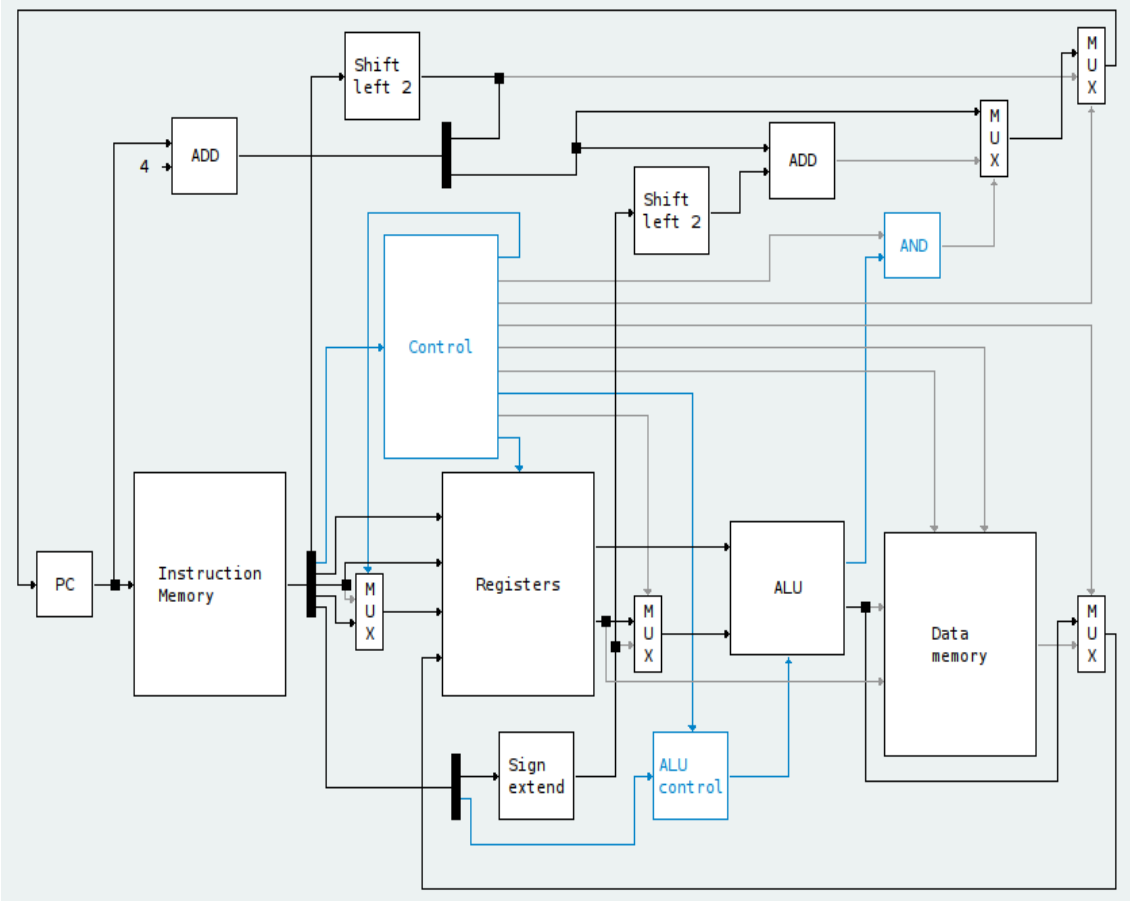

Figura 1: Datapath de CPU monociclo implementada en DrMIPS

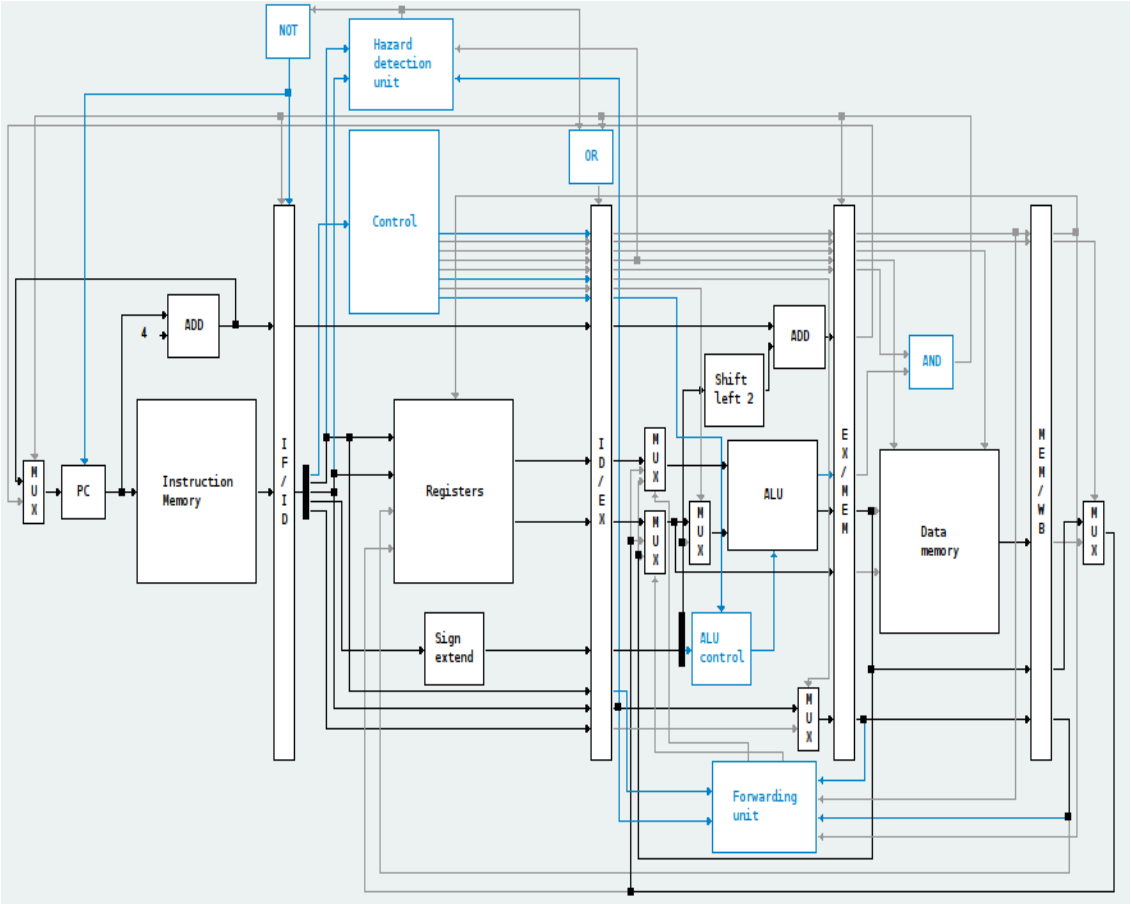

Figura 2: Datapath de CPU Pipeline implementada en DrMIPS

En base a estos dos modelos se realizaron extensiones sobre cada Datapath para incluir los tres tipos de saltos, Jump(j), Jump Register(jr) y Jump And Link Register(jalr) con la salvedad de que la instrucción Jump para el **Datapath Monociclo** [1] ya se encuentra implementada.

3.1. Monociclo

Estas implementaciones corresponden al **Datapath Monociclo** de DrMIPS.

3.1.1. Jump Register

Para esta instrucción unicamente fue necesario agregar dos nuevos componentes:

- **Fork**: Llamado **ForkReg1**, este permite bifurcar la salida de registro source proveniente del directorio de registros.
- **Multiplexer**: Llamado **MuxJumpReg** y diferencia entre la dirección proveniente de una instruccion branch o jump register y es seleccionado por la unidad de control con la señal JumpReg.

Con estos componentes se pudo implementar la instrucción, ya que se codifico para que el registro seleccionado por el programador sea enviado a través del registro source y además envie una señal a la unidad de control llamada **JumpReg** que indica que se va a ejecutar un salto y que

la instrucción del nuevo PC será tomada directamente del dato almacenado en el registro source del directorio de registros.

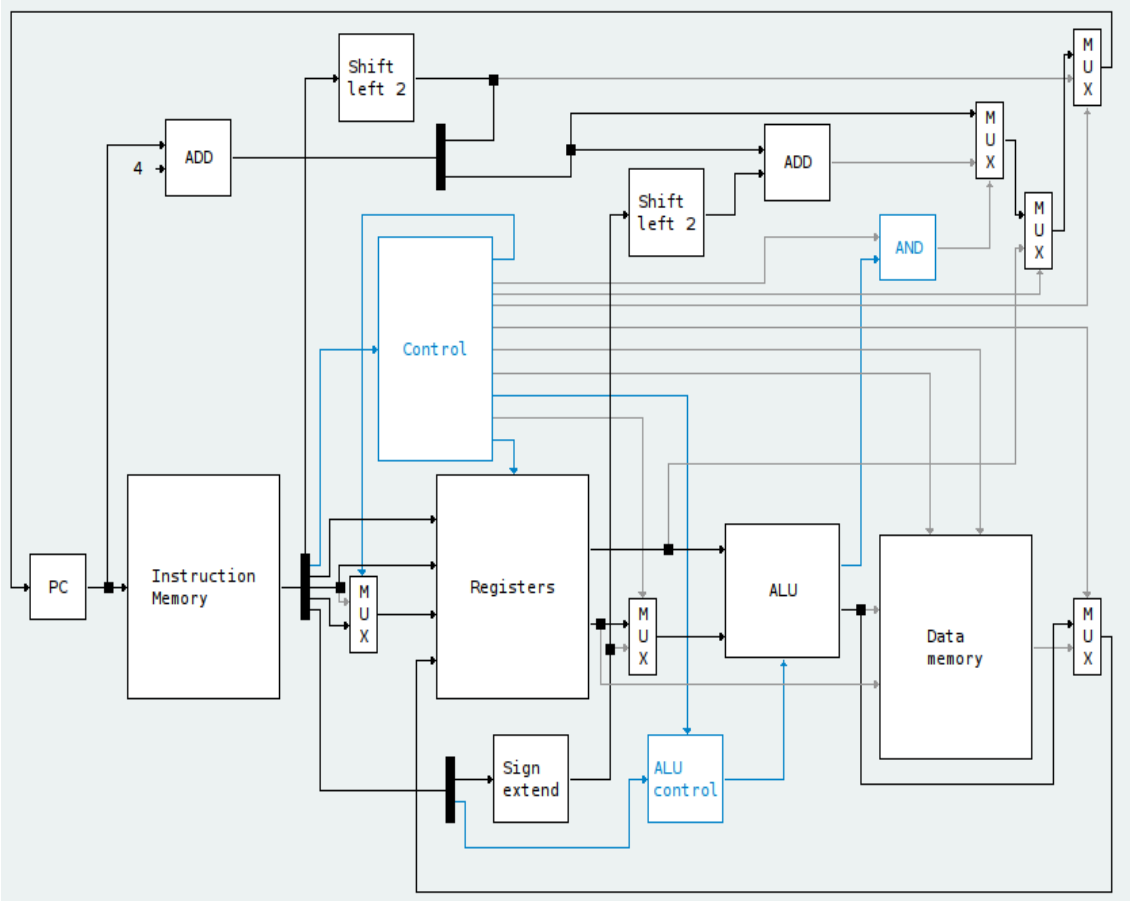

Figura 3: Datapath de CPU Monociclo extendido para sorportar la instrucción jump register.

3.1.2. Jump and Link Register

Nuevamente, extender el **Datapath Monociclo** para que soporte la nueva instrucción es sencillo, en este caso solamente se requirió añadir un componente extra de hardware:

- **Fork**: Se llamó **ForkPC** y permitió bifurcar la salida del (PC + 4) para que sea direccionada al multiplexor que se encarga de seleccionar el proximo dato a ser escrito en el directorio de registros.

Con este agregado, codificando la instrucción para que envie las señales correctas a la unidad de control (MemToReg = 2, Write = 1, RegDst = 1), seleccione a través del registro destiny el registro **ra** y extendiendo el multiplexor que selecciona la escritura en registro para que reciba 3 entradas, se pudo implementar la instrucción pedida ya que la instrucción es exactamente igual a jump register con la diferencia que debe guardar la siguiente instrucción a ser ejecutada en el registro de retorno, por lo cual, básicamente se realiza un jump register y se colocan las señales adecuadas para que también se almacene la siguiente instrucción a ejecutar en **ra**.

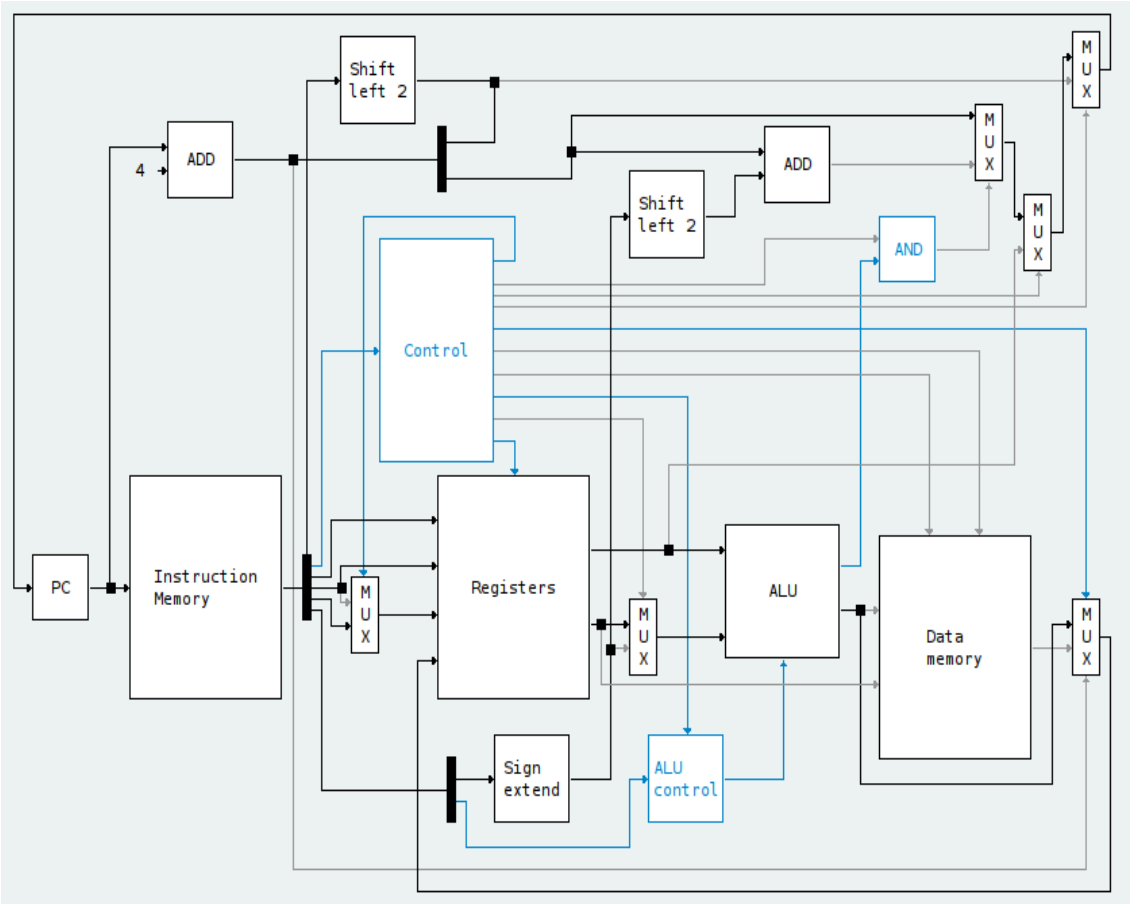

Figura 4: Datapath de CPU Monociclo extendido para sorportar la instrucción jump and link register.

3.2. Pipeline

Estas implementaciones corresponden al **Datapath Pipeline** de DrMIPS, las instrucciones jump en pipeline pueden tener riesgos de datos si, el salto que se debe tomar depende del resultado de una operación anterior, como ir a buscar la dirección de salto a memoria y luego querer saltar, o realizar una operación entre registros y luego realizar el salto, estos riesgos estan cubiertos por la implementación estandar de **DrMIPS** que provee una unidad de fordwarding para el caso de la dependencia de una operación de la ALU y por una unidad de hazards que agrega un stall cuando haya una dependencia con memoria, también pueden ocurrir riesgos de control ya que en pipeline al tener que realizar un salto las instrucciones siguientes que entraron no deben ser ejecutadas pues afectarían la lógica del programa, es por eso que en todas las implementaciones se saco provecho a la implementación de pipeline estandar que ya contaba con un sistema para realizar un flush de los registros interetapa en estos casos, por lo cual cualquier instrucción de tipo jump activa la señal de flush para evitar las instrucciones siguientes antes del salto tomado.

3.2.1. Jump

Para agregar esta instrucción sin riesgos de hazards al pipeline fueron necesarios los siguientes componentes:

- **Multiplexer**: Llamado "MuxJump", este se agrego porque es necesario diferenciar entre

una instrucción de salto y una de branch por lo cual el multiplexor se encarga de seleccionar la dirección del PC adecuada para un jump.

- **ShiftLeft**: Llamado "ShiftJump", se encarga de desplazar a izquierda dos bits el inmediato de 26 bits recibido de la instruccion jump para que se alinee a posiciones de memoria multiplos de cuatro para las intrucciones de 32 bits de mips.
- **Distributor**: Llamado "DistInst", debe separar el valor del (PC + 4) en dos, una parte son sus 32 bits enteros y la otra son sus 4 bits mas significativos que son necesarios para terminar de formar la instrucción de salto del jump.
- **Concatenator**: Llamado ÇoncatJump", recibe los 4 bits entregados por el **Distributor** y los 28 bits provenientes del **ShifLeft** para formar la dirección del salto.
- **Fork**: Llamado "ForkJump", se encarga de bifurcar la señal de control que indica un salto en dos partes, una que va directamente al **MuxJump** y otra al **OrJumpFlush**, es esta la señal que habilita el flush de los registros interetapas.
- **Or**: Llamado OrJumpFlush", se agrego para extender el flush que ya realizaba la implementación original para instrucciones de tipo branch.

Además de estos componentes, la señal de control jump se envió a través de todos los registros hasta la etapa MEM que es en donde se define el nuevo PC, también se enviaron los 26 bits de la instrucción jump que indican la direccion del salto hasta esta etapa para evitar que sea sobreescrita por otras instrucciones y así se envia correctamente hasta el **ShiftJump**. Con estos detalles la instrucción de jump se ejecuta sin riesgos como en la CPU monociclo.

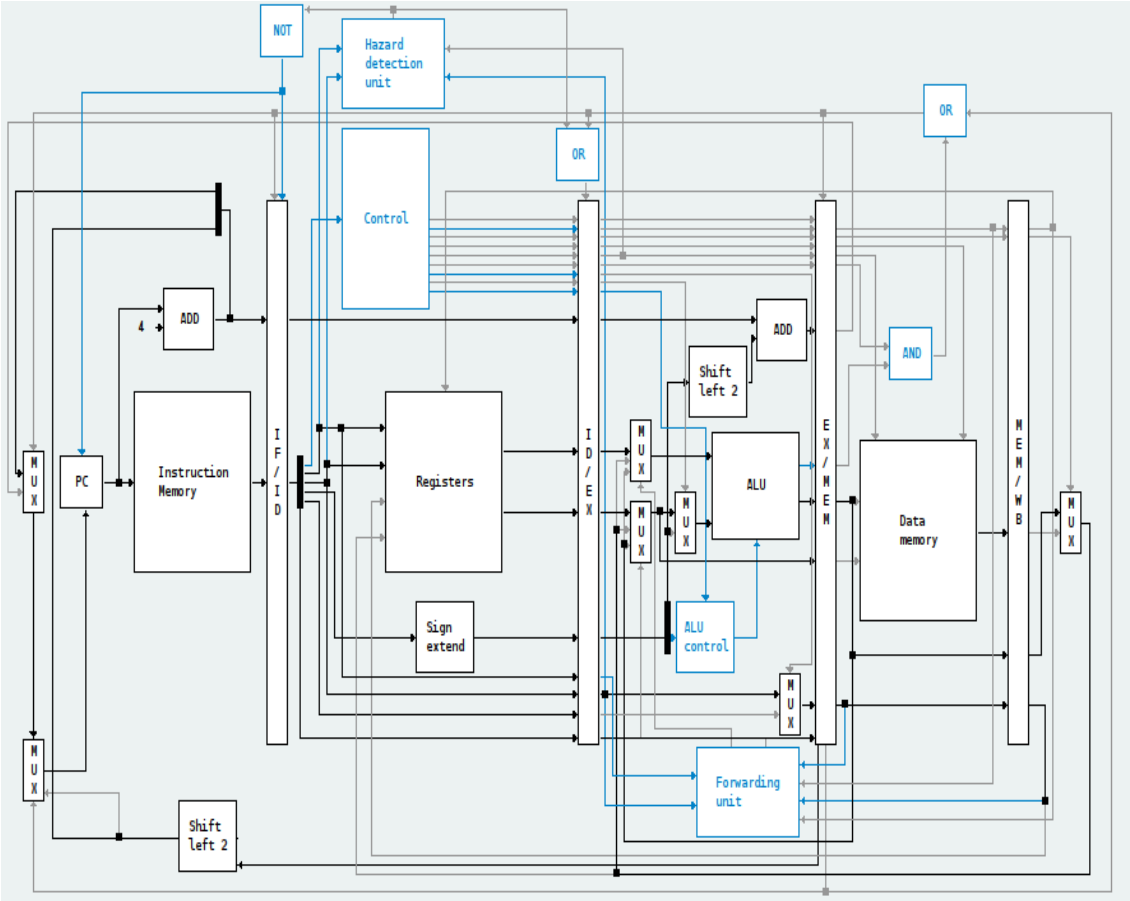

Figura 5: Datapath de CPU Pipeline extendido para sorportar la instrucción jump.

3.2.2. Jump Register

Implementar esta instrucción también es parecida al monociclo ya que se tiene que poder diferenciar una instruccion de branch a un jump register y jump register, los componentes agregados son parecidos ademas de una nueva compuerta or para evitar riesgos de control con un flush. Componentes agregados:

- **Or**: Llamado "`OrJumpFlush_`", Debido a la imposibilidad de hacer un or de tres entradas se incorporo uno nuevo para que pueda identificar el salto por jump register, ya se implemento nuevamente una señal de control **JumpReg** que indica cuando se esta ejecutando esta instrucción.
- **Multiplexer**: Llamado "MuxJumpReg", es un multiplexor extra como en la implementación monociclo para poder diferencia entre un jump a un jump reg.
- **Fork 1**: Llamado "ForkJumpReg", bifurca la señal de control de salto JumpReg en una parte para el **OrJumpFlush** y otra al **MuxJumpReg**.
- **Fork 2**: Llamado "`ForkEXR1_`", bifurca el resultado de la ALU en una parte que se dirige nuevamente a los registros fuente y destino de la ALU, y otra que va al multiplexor **MuxJumpReg** ya que el resultado del salto depende de la ALU.

En esta implementación se se decidió codificar la instrucción de forma tal que el registro source lleva el registro del salto y el registro destino es siempre cero, de manera que una operacion de

suma de la ALU devuelve nuevamente el registro fuente y a partir de aca, en etapa MEM se termina resolviendo la dirección enviandola a través del **MuxJumpReg**, y los riesgos no existen pues se aplico el mismo concepto que en la implementación de jump, el cual utiliza un flush de los registros interetapa.

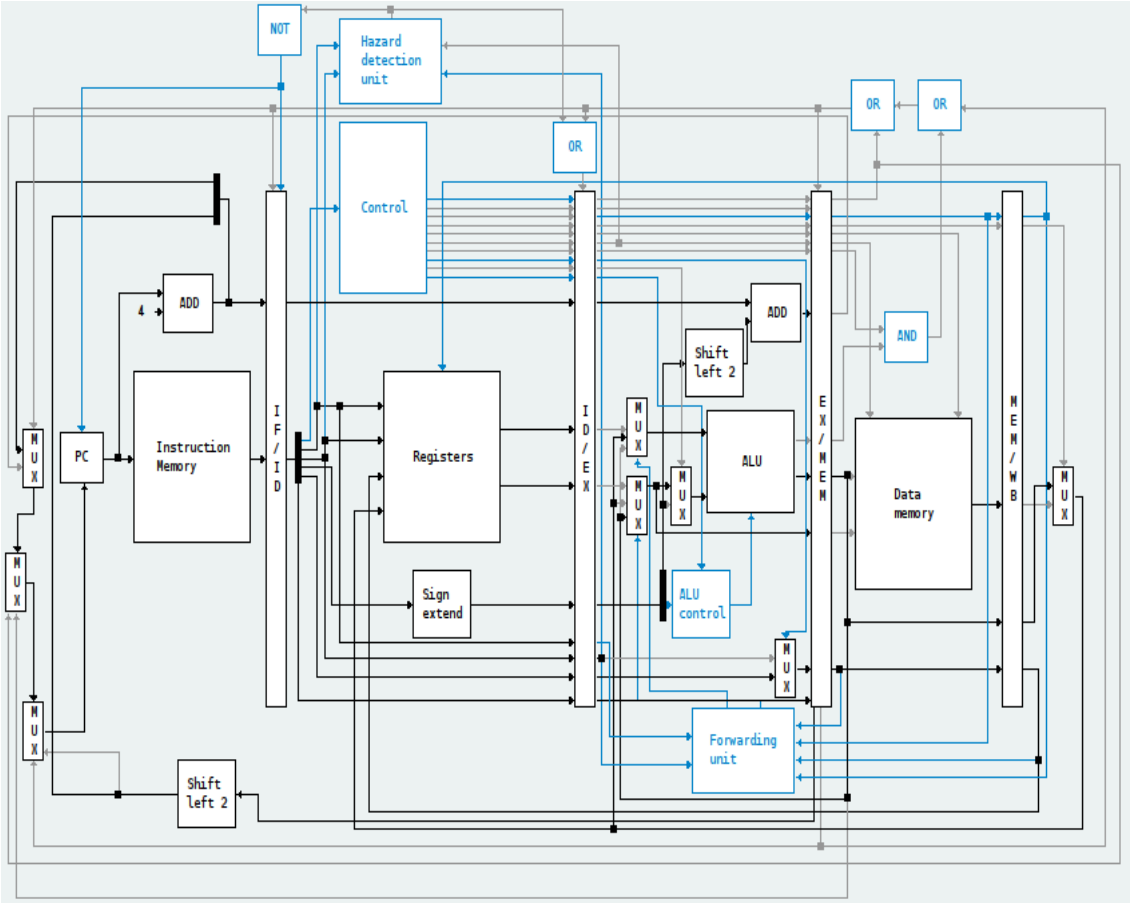

Figura 6: Datapath de CPU Pipeline extendido para sorportar la instrucción jump register.

3.2.3. Jump and Link Register

Una vez implementada la instruccion Jump Register, está es trivial debido a que el hardware adicional que hay que agregar para hacer el link es simple. Solo se añadio:

- **Fork**: Llamado `"ForkPCAdder_"`, se utiliza para bifurcar el (PC + 4) y enviarla hasta el registro WB.

En este caso el salto se resuelve recien en etapa de Write Back que es donde se ubica el multiplexor que envia los datos a escribirse en el directorio de registros, para lograr esto se convirtio el multiplexor en uno de tres entradas extendiendo la señal de contro **MemToReg** para que la opcion 2 seleccione el (PC+4) para escribirse en el registro de retorno. La elección del registro que tiene la dirección del salto se vuelve a generar a través de la ALU y esta vez se habilitan la señal Write para permitir que se guarda el retorno del salto.

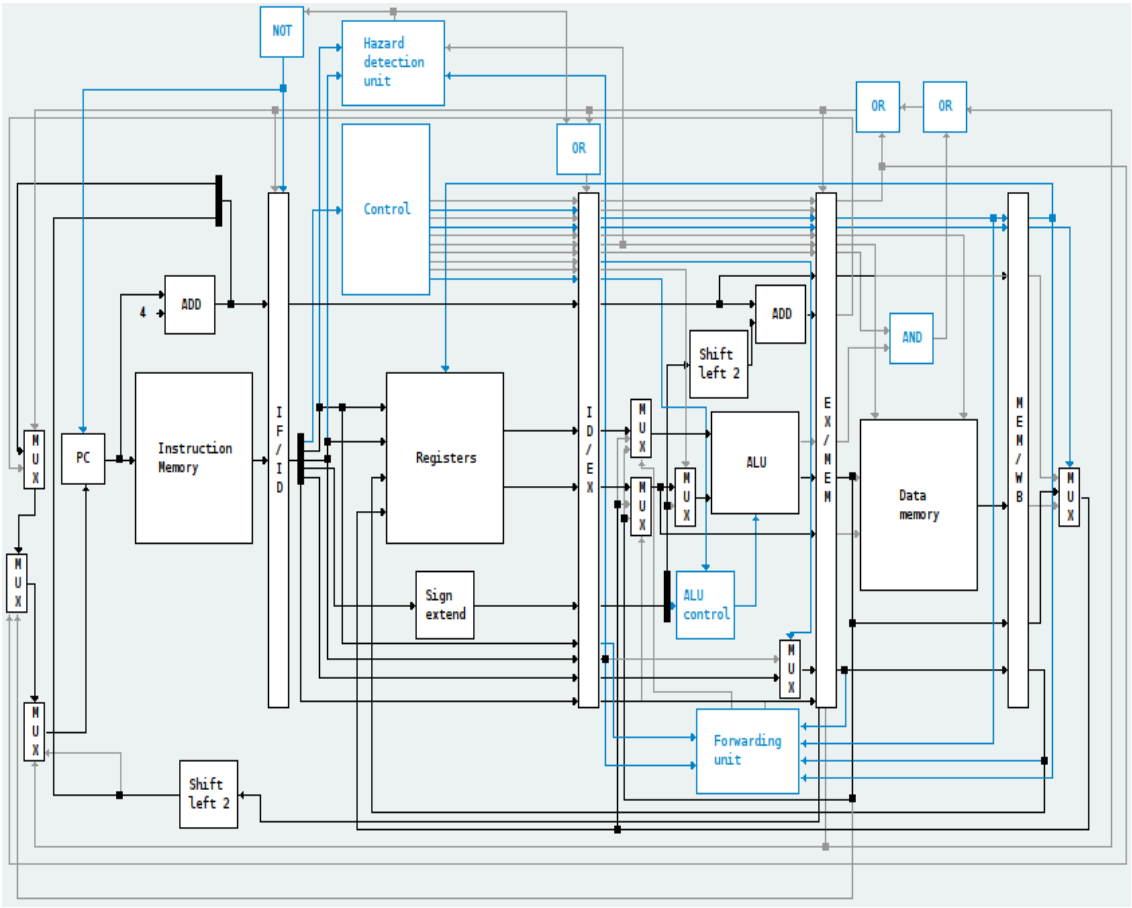

Figura 7: Datapath de CPU Pipeline extendido para sorportar la instrucción jump and link register.

4. Pruebas

4.0.1. Jump Pipeline

```
1  #Salto incondicional de mas de tres instrucciones no debe ejecutar
       ↪ las instrucciones posteriores al salto
2  addi $t0, $zero, 5
3  j test_1
4  addi $t1, $zero, 10    #NO SE DEBE EJECUTAR
5  and $t2, $t1, $t0    #NO SE DEBE EJECUTAR
6  add $t3, $t1, $t0    #NO SE DEBE EJECUTAR
7  add $t4, $t3, $t1    #NO SE DEBE EJECUTAR
8  test_1:
9    addi $t5, $zero, 55
10
11 #Salto incondicional de menos de tres instrucciones no debe ejecutar
       ↪ las instrucciones posteriores al salto
12 addi $t0, $zero, 77
13 j test_2
14 addi $t1, $zero, 10    #NO SE DEBE EJECUTAR
15 test_2:
```

```
16    add $t2, $t0, $t0
17    add $t3, $t2, $t0
18    addi $t4, $zero, 101
```

4.0.2. Jump Register Monociclo

```
1 # Salto incondicional a la instruccion 5 del programa.
2 addi $t0, $zero, 20
3 addi $t1, $zero, 8
4 jr $t0
5 addi $t2, $zero, 12
6 addi $t3, $zero, 16
7 addi $t4, $zero, 20
8 add $t5, $t0, $zero
```

4.0.3. Jump Register Pipeline

```
1  # Salto incondicional a la instruccion 4 del programa.
2  addi $t0, $zero, 12
3  addi $t1, $zero, 8
4  jr $t0
5  addi $t2, $zero, 12   # Linea del salto jr $t0
6  addi $t3, $zero, 16
7  addi $t4, $zero, 20   # Linea del salto jr $t2
8  add $t5, $t0, $zero
9
10 # Salto incondicional a la instruccion indicada por la suma de dos
       ↪ registros.
11 # No ocurre riesgos de datos porque ya venia implementada la unidad
       ↪ de fordwarding
12 # no hacia falta resolver nada
13 addi $t0, $zero, 16
14 addi $t1, $zero, 4
15 add $t2, $t0, $t1
16 jr $t2
```

4.0.4. Jump And Link Register Monociclo

```
1 # Salto incondicional a la instruccion 9 del programa.
2 # El 32 del salto es porque la instrucciones son de [0-n] y como
      ↪ quiero la
3 # instruccion 8 se multiplica por 4 y da de resultado la direccion 32
      ↪ para ese salto.
4 addi $t0, $zero, 32
5 addi $t1, $zero, 8
6 jalr $t0
7 addi $t2, $zero, 12
```

```
8  addi $t3, $zero, 16
9  addi $t4, $zero, 20
10 add $t5, $t0, $zero
11 j end
12
13 addi $t0, $zero, 4
14 addi $t1, $zero, 1
15 add $t2, $t0, $t1
16 addi $t2, $zero, 12
17 addi $t3, $zero, 16
18 addi $t4, $zero, 20
19 add $t5, $t0, $zero
20 jr $ra
21
22 end:
23   addi $t7, $zero, 50
```

4.0.5. Jump And Link Register Pipeline

```
1  # Salto incondicional a la instruccion 9 del programa y enlace de la
       ↪ vuelta al $ra
2  # Misma prueba que en unicycle, se mantiene el resultado.
3  addi $t0, $zero, 32
4  addi $t1, $zero, 8
5  jalr $t0
6  addi $t2, $zero, 12
7  addi $t3, $zero, 16
8  addi $t4, $zero, 20
9  add $t5, $t0, $zero
10 j end
11
12 addi $t0, $zero, 4
13 addi $t1, $zero, 1
14 add $t2, $t0, $t1
15 addi $t2, $zero, 12
16 addi $t3, $zero, 16
17 addi $t4, $zero, 20
18 add $t5, $t0, $zero
19 jr $ra
20
21 end:
22   addi $t7, $zero, 50
23
24 # Salto indoncidicional y ejecucion de una instruccion dependiente
       ↪ del $ra
25 # Esto no tiene mucho sentido en realidad pero para probar no hay
       ↪ riesgos
26 # por dependencias, el salto de retorno se calcula bien porque cuando
       ↪ se
27 # decide la direccion del salto jalr $t0 (en MEM) se actualiza el PC,
       ↪ y se busca
```

```
28 # la instruccion jr que es la correcta, en ese momento ya jalr esta
       ↪ en etapa WB
29 # en el siguiente ciclo de reloj se habra actualizado en directorio
       ↪ de registros y
30 # tendra el $ra correcto y entra en un bucle siempre saltando sobre
       ↪ si mismo.
31 addi $t0, $zero, 80
32 addi $t1, $zero, 8
33 jalr $t0
34 jr $ra
```

5. Conclusiones

Se puede decir que las implementaciones de conjuntos de instrucciones en una CPU monociclo son mucho mas simples a nivel de hardware y no involucran tantos riesgos como una implementacion de pipeline, unicamente hay que tener cuidado en que el componente mas lento de hardware no sea mas lento que un clock de reloj, así todo en el datapath podria responder a tiempo sin que ocurras efectos no deseados.
Por otro lado la implementación de un pipeline hace complejo el hardware y se generan problemas debido a inconsistencias en valores de registros o saltos en el programa, resolverlos involucra componentes adicionales como detectores de riesgos o unidades de fordwarding.
Finalmente, a pesar de la complejidad de un CPU implementado con pipeline siempre sera preferible por sobre uno monociclo o multiciclo.

6. Anexo

6.1. Enunciado del trabajo práctico

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

1. Objetivos

El objetivo de este trabajo es familiarizarse con la arquitectura de una CPU MIPS, específicamente con el datapath y la implementación de instrucciones. Para ello, se deberán agregar instrucciones a diversas configuraciones de CPU provistas por el simulador DrMIPS [1]

2. Alcance

Este trabajo práctico es de elaboración grupal, evaluación individual, y de carácter obligatorio para todos alumnos del curso.

3. Requisitos

El trabajo deberá ser entregado personalmente, en la fecha estipulada, con una carátula que contenga los datos completos de todos los integrantes.

Además, es necesario que el trabajo práctico incluya (entre otras cosas, ver sección 8), la presentación de los resultados obtenidos, explicando, cuando corresponda, con fundamentos reales, las causas o razones de cada resultado obtenido.

El informe deberá respetar el modelo de referencia que se encuentra en el grupo[1], y se valorarán aquellos escritos usando la herramienta TEX / LATEX.

4. Recursos

Usaremos el programa DrMIPS [1] para configurar y simular el data path de un procesador MIPS [4], tanto uniciclo como multiciclo.

5. Descripción.

5.1. Introducción

El programa DrMIPS nos permite evaluar distintos diseños de datapath para procesadores MIPS32, al darnos la posibilidad de organizarlo como queramos. Si bien sólo puede haber uno de algunos de los componentes del DP (como el registro de PC o la unidad de control), podemos poner sumadores, multiplexores, extensores de signo y conexiones arbitrariamente. También es

[1] http://groups.yahoo.com/group/orga6620

posible modificar el conjunto de instrucciones. Además de la estructura lógica del DP, DrMips nos permite escribir programas simples y simular su ejecución en el DP, mostrando los valores que toman las diversas entradas y salidas de cada elemento. El programa se puede conseguir en `https://bitbucket.org/brunonova/drmips/wiki/Home`, o se puede descargar para Ubuntu, ya sea desde el repositorio de Ubuntu (aunque la versión está desactualizada) o autorizando un repositorio externo (ver [2]).

5.2. Datapaths

El programa viene con algunos DP ya implementados, a saber:
Uniciclo:

- `unycicle.cpu`: El DP uniciclo por defecto.
- `unycicle-no-jump.cpu`: Variante más simple del DP uniciclo que no soporta la instrucción `j`.
- `unycicle-no-jump-branch.cpu`: Una variante aún más simple que no soporta `jump` ni `branch`.
- `unycicle-extended.cpu`: Una variante que soporta instrucciones adicionales, como multiplicación y división.

Multiciclo:

- `pipeline.cpu`: El DP de pipeline por defecto, implementa detección de hazards. Los DP de pipeline no soportan la instrucción `j` (salto).
- `pipeline-only-forwarding.cpu`: Variante del DP de pipeline que implementa forwarding pero no genera stalls (genera resultados incorrectos).
- `pipeline-no-hazard-detection.cpu`: Otra variante que no hace hazard detection de ninguna manera (genera resultados incorrectos).
- `pipeline-extended.cpu`: Una variante que soporta instrucciones adicionales, como multiplicación y división, como `unycicle-extended.cpu`.

5.3. Instrucciones a implementar

1. Implementar la instrucción `j` en el DP `pipeline.cpu`. Verificar que no se produzcan hazards.
2. Implementar la instrucción `jr` (Jump Register) en el DP `unicycle.cpu`.
3. Implementar la instrucción `jr` 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 pipeline 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.
- Los DP, los programas de prueba y los conjuntos de instrucciones usados en cada caso.
- Para los datapath de pipeline, explicar cómo se verificó que no hubiera hazards.
- Este enunciado.

9. Fechas de entrega.

La fecha de entrega de este trabajo práctico es el Jueves 4 de Marzo de 2021.

Referencias

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

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

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

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

6.2. Conjuntos de instrucciones

6.3. Jump

- Instrucciones para un CPU pipeline

```
{
"comment": "Instruction set of the reference book, without the jump instruction.",
"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": {
        "j":    {"type": "J", "args": ["target"], "fields": {"op": 2, "target": "#1"}, "desc": "PC = target"},

        "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 & $t3"},
        "nor":   {"type": "R", "args": ["reg", "reg", "reg"], "fields": {"op": 0, "rs": "#2", "rt": "#3", "rd": "#1", "shamt": 0, "func": 39}, "desc": "$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 | $t3"},
        "slt":  {"type": "R", "args": ["reg", "reg", "reg"], "fields": {"op": 0, "rs": "#2", "rt": "#3", "rd": "#1", "shamt": 0, "func": 42}, "desc": "$t1 = ($t2 < $t3) ? 1 : 0"},
        "addi": {"type": "I", "args": ["reg", "reg", "int"], "fields": {"op": 8, "rs": "#2", "rt": "#1", "imm": "#3"}, "desc": "$t1 = $t2 +
```

```
                    23"},
19                  "beq":  {"type": "I", "args": ["reg", "reg", "
                    offset"], "fields": {"op": 4, "rs": "#1", "
                    rt": "#2", "imm": "#3"}, "desc": "PC += ($t1
                     == $t2) ? (offset * 4 + 4) : 4"},
20                  "lw":   {"type": "I", "args": ["reg", "data"],
                    "fields": {"op": 35, "rs": "#2.offset", "rt
                    ": "#1", "imm": "#2.base"}, "desc": "$t1 =
                    MEM[base + $t2]"},
21                  "sw":   {"type": "I", "args": ["reg", "data"],
                    "fields": {"op": 43, "rs": "#2.offset", "rt
                    ": "#1", "imm": "#2.base"}, "desc": "MEM[
                    base + $t2] = $t1"}
22          },
23          "pseudo": {
24                  "li":   {"args": ["reg", "int"], "to": ["addi
                    #1, $0, #2"], "desc": "$t1 = 22"},
25                  "la":   {"args": ["reg", "label"], "to": ["addi
                     #1, $0, #2"], "desc": "$t1 = ADDR(label)"},
26                  "move": {"args": ["reg", "reg"], "to": ["add
                    #1, #2, $0"], "desc": "$t1 = $t2"},
27                  "subi": {"args": ["reg", "reg", "int"], "to":
                    ["li $1, #3", "sub #1, #2, $1"], "desc": "
                    $t1 = $t2 - 23"},
28                  "sgt":  {"args": ["reg", "reg", "reg"], "to":
                    ["slt #1, #3, #2"], "desc": "$t1 = ($t2 >
                    $t3) ? 1 : 0"},
29                  "bge":  {"args": ["reg", "reg", "offset"], "to
                    ": ["slt $1, #1, #2", "beq $1, $0, #3"], "
                    desc": "PC += ($t1 >= $t2) ? (offset * 4 +
                    4) : 4"},
30                  "ble":  {"args": ["reg", "reg", "offset"], "to
                    ": ["sgt $1, #1, #2", "beq $1, $0, #3"], "
                    desc": "PC += ($t1 <= $t2) ? (offset * 4 +
                    4) : 4"},
31                  "b":    {"args": ["offset"], "to": ["beq $0, $0
                    , #1"], "desc": "PC += offset * 4 + 4"},
32                  "neg":  {"args": ["reg", "reg"], "to": ["sub
                    #1, $0, #2"], "desc": "$t1 = -$t2"},
33                  "not":  {"args": ["reg", "reg"], "to": ["nor
                    #1, #2, $0"], "desc": "$t1 = ~$t2"}
34          },
35          "control": {
36                  "2": {"Jump": 1},
37
38                  "0": {"RegDst": 1, "RegWrite": 1, "ALUOp": 2, "
                    ALUSrc": 0, "MemToReg": 0},
39                  "8": {"RegDst": 0, "RegWrite": 1, "ALUOp": 0, "
                    ALUSrc": 1, "MemToReg": 0},
40                  "4": {"ALUOp": 1, "ALUSrc": 0, "Branch": 1},
41                  "35": {"ALUOp": 0, "ALUSrc": 1, "RegDst": 0, "
                    RegWrite": 1, "MemRead": 1, "MemWrite": 0, "
                    MemToReg": 1},
42                  "43": {"ALUOp": 0, "ALUSrc": 1, "RegDst": 0, "
```

```
                        RegWrite": 0, "MemRead": 0, "MemWrite": 1, "
                        MemToReg": 0}
43          },
44          "alu": {
45                  "aluop_size": 2,
46                  "func_size": 6,
47                  "control_size": 4,
48                  "control": [
49                          {"aluop": 0, "out": {"Operation": 2}},
50                          {"aluop": 1, "out": {"Operation": 6}},
51                          {"aluop": 2, "func": 32, "out": {"
                               Operation": 2}},
52                          {"aluop": 2, "func": 34, "out": {"
                               Operation": 6}},
53                          {"aluop": 2, "func": 36, "out": {"
                               Operation": 0}},
54                          {"aluop": 2, "func": 37, "out": {"
                               Operation": 1}},
55                          {"aluop": 2, "func": 42, "out": {"
                               Operation": 7}},
56                          {"aluop": 2, "func": 39, "out": {"
                               Operation": 12}}
57                  ],
58                  "operations": {
59                          "0": "and",
60                          "1": "or",
61                          "2": "add",
62                          "6": "sub",
63                          "7": "slt",
64                          "12": "nor"
65                  }
66          }
67  }
```

6.3.1. Jump Register

- **Instrucciones para un CPU monociclo**

```
1           {
2           "comment": "Instruction set of the reference book.",
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                   "jr": {"type": "R", "args": ["reg"], "fields":
                        {"op": 1, "rs": "#1", "rt": 0, "rd": 0, "
```

```
                    shamt": 0, "func": 8}, "desc": "PC = $t1"},
10
11              "nop":  {"type": "R", "fields": {"op": 0, "rs":
                     0, "rt": 0, "rd": 0, "shamt": 0, "func":
                    0}},
12              "add":  {"type": "R", "args": ["reg", "reg", "
                    reg"], "fields": {"op": 0, "rs": "#2", "rt":
                     "#3", "rd": "#1", "shamt": 0, "func": 32},
                    "desc": "$t1 = $t2 + $t3"},
13              "sub":  {"type": "R", "args": ["reg", "reg", "
                    reg"], "fields": {"op": 0, "rs": "#2", "rt":
                     "#3", "rd": "#1", "shamt": 0, "func": 34},
                    "desc": "$t1 = $t2 - $t3"},
14              "and":  {"type": "R", "args": ["reg", "reg", "
                    reg"], "fields": {"op": 0, "rs": "#2", "rt":
                     "#3", "rd": "#1", "shamt": 0, "func": 36},
                    "desc": "$t1 = $t2 & $t3"},
15              "or":   {"type": "R", "args": ["reg", "reg", "
                    reg"], "fields": {"op": 0, "rs": "#2", "rt":
                     "#3", "rd": "#1", "shamt": 0, "func": 37},
                    "desc": "$t1 = $t2 | $t3"},
16              "nor":   {"type": "R", "args": ["reg", "reg", "
                    reg"], "fields": {"op": 0, "rs": "#2", "rt":
                     "#3", "rd": "#1", "shamt": 0, "func": 39},
                    "desc": "$t1 = ~($t2 | $t3)"},
17              "slt":  {"type": "R", "args": ["reg", "reg", "
                    reg"], "fields": {"op": 0, "rs": "#2", "rt":
                     "#3", "rd": "#1", "shamt": 0, "func": 42},
                    "desc": "$t1 = ($t2 < $t3) ? 1 : 0"},
18              "j":    {"type": "J", "args": ["target"], "
                    fields": {"op": 2, "target": "#1"}, "desc":
                    "PC = target"},
19              "addi": {"type": "I", "args": ["reg", "reg", "
                    int"], "fields": {"op": 8, "rs": "#2", "rt":
                     "#1", "imm": "#3"}, "desc": "$t1 = $t2 +
                    23"},
20              "beq":  {"type": "I", "args": ["reg", "reg", "
                    offset"], "fields": {"op": 4, "rs": "#1", "
                    rt": "#2", "imm": "#3"}, "desc": "PC += ($t1
                     == $t2) ? (offset * 4 + 4) : 4"},
21              "lw":   {"type": "I", "args": ["reg", "data"],
                    "fields": {"op": 35, "rs": "#2.offset", "rt
                    ": "#1", "imm": "#2.base"}, "desc": "$t1 =
                    MEM[base + $t2]"},
22              "sw":   {"type": "I", "args": ["reg", "data"],
                    "fields": {"op": 43, "rs": "#2.offset", "rt
                    ": "#1", "imm": "#2.base"}, "desc": "MEM[
                    base + $t2] = $t1"}
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                  "1": {"RegDst": 1, "RegWrite": 0, "ALUop": 2, "
                        AluSrc": 0, "MemToReg":0, "JumpReg": 1},
38
39                  "0": {"RegDst": 1, "RegWrite": 1, "ALUOp": 2, "
                        ALUSrc": 0, "MemToReg": 0},
40                  "8": {"RegDst": 0, "RegWrite": 1, "ALUOp": 0, "
                        ALUSrc": 1, "MemToReg": 0},
41                  "2": {"Jump": 1},
42                  "4": {"ALUOp": 1, "ALUSrc": 0, "Branch": 1},
43                  "35": {"ALUOp": 0, "ALUSrc": 1, "RegDst": 0, "
                        RegWrite": 1, "MemRead": 1, "MemWrite": 0, "
                        MemToReg": 1},
44                  "43": {"ALUOp": 0, "ALUSrc": 1, "RegDst": 0, "
                        RegWrite": 0, "MemRead": 0, "MemWrite": 1, "
                        MemToReg": 0}
45          },
46          "alu": {
47                  "aluop_size": 2,
48                  "func_size": 6,
49                  "control_size": 4,
50                  "control": [
51                          {"aluop": 2, "func": 8, "out": {"
                                Operation": 2}},
52
53                          {"aluop": 0, "out": {"Operation": 2}},
54                          {"aluop": 1, "out": {"Operation": 6}},
55                          {"aluop": 2, "func": 32, "out": {"
                                Operation": 2}},
56                          {"aluop": 2, "func": 34, "out": {"
                                Operation": 6}},
```

```
57                          {"aluop": 2, "func": 36, "out": {"
                               Operation": 0}},
58                          {"aluop": 2, "func": 37, "out": {"
                               Operation": 1}},
59                          {"aluop": 2, "func": 39, "out": {"
                               Operation": 12}},
60                          {"aluop": 2, "func": 42, "out": {"
                               Operation": 7}}
61                  ],
62                  "operations": {
63                          "0": "and",
64                          "1": "or",
65                          "2": "add",
66                          "6": "sub",
67                          "7": "slt",
68                          "12": "nor"
69                  }
70          }
71  }
```

- **Instrucciones para un CPU Pipeline**

```
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                   "j":    {"type": "J", "args": ["target"], "
                       fields": {"op": 2, "target": "#1"}, "desc":
                       "PC = target"},
10                  "jr": {"type": "R", "args": ["reg"], "fields":
                       {"op": 1, "rs": "#1", "rt": 0, "rd": 0, "
                       shamt": 0, "func": 8}, "desc": "PC = $t1"},
11
12                  "nop":  {"type": "R", "fields": {"op": 0, "rs":
                        0, "rt": 0, "rd": 0, "shamt": 0, "func":
                       0}},
13                  "add":  {"type": "R", "args": ["reg", "reg", "
                       reg"], "fields": {"op": 0, "rs": "#2", "rt":
                        "#3", "rd": "#1", "shamt": 0, "func": 32},
                       "desc": "$t1 = $t2 + $t3"},
14                  "sub":  {"type": "R", "args": ["reg", "reg", "
                       reg"], "fields": {"op": 0, "rs": "#2", "rt":
                        "#3", "rd": "#1", "shamt": 0, "func": 34},
```

```
                "desc": "$t1 = $t2 - $t3"},
15              "and":  {"type": "R", "args": ["reg", "reg", "
                    reg"], "fields": {"op": 0, "rs": "#2", "rt":
                     "#3", "rd": "#1", "shamt": 0, "func": 36},
                    "desc": "$t1 = $t2 & $t3"},
16              "nor":   {"type": "R", "args": ["reg", "reg", "
                    reg"], "fields": {"op": 0, "rs": "#2", "rt":
                     "#3", "rd": "#1", "shamt": 0, "func": 39},
                    "desc": "$t1 = ~($t2 | $t3)"},
17              "or":   {"type": "R", "args": ["reg", "reg", "
                    reg"], "fields": {"op": 0, "rs": "#2", "rt":
                     "#3", "rd": "#1", "shamt": 0, "func": 37},
                    "desc": "$t1 = $t2 | $t3"},
18              "slt":  {"type": "R", "args": ["reg", "reg", "
                    reg"], "fields": {"op": 0, "rs": "#2", "rt":
                     "#3", "rd": "#1", "shamt": 0, "func": 42},
                    "desc": "$t1 = ($t2 < $t3) ? 1 : 0"},
19              "addi": {"type": "I", "args": ["reg", "reg", "
                    int"], "fields": {"op": 8, "rs": "#2", "rt":
                     "#1", "imm": "#3"}, "desc": "$t1 = $t2 +
                    23"},
20              "beq":  {"type": "I", "args": ["reg", "reg", "
                    offset"], "fields": {"op": 4, "rs": "#1", "
                    rt": "#2", "imm": "#3"}, "desc": "PC += ($t1
                     == $t2) ? (offset * 4 + 4) : 4"},
21              "lw":   {"type": "I", "args": ["reg", "data"],
                    "fields": {"op": 35, "rs": "#2.offset", "rt
                    ": "#1", "imm": "#2.base"}, "desc": "$t1 =
                    MEM[base + $t2]"},
22              "sw":   {"type": "I", "args": ["reg", "data"],
                    "fields": {"op": 43, "rs": "#2.offset", "rt
                    ": "#1", "imm": "#2.base"}, "desc": "MEM[
                    base + $t2] = $t1"}
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                  "2": {"Jump": 1},
38                  "1": {"RegDst": 1, "RegWrite": 0, "ALUop": 2, "
                    AluSrc": 0, "MemToReg":0, "JumpReg": 1},
39
40                  "0": {"RegDst": 1, "RegWrite": 1, "ALUOp": 2, "
                    ALUSrc": 0, "MemToReg": 0},
41                  "8": {"RegDst": 0, "RegWrite": 1, "ALUOp": 0, "
                    ALUSrc": 1, "MemToReg": 0},
42                  "4": {"ALUOp": 1, "ALUSrc": 0, "Branch": 1},
43                  "35": {"ALUOp": 0, "ALUSrc": 1, "RegDst": 0, "
                    RegWrite": 1, "MemRead": 1, "MemWrite": 0, "
                    MemToReg": 1},
44                  "43": {"ALUOp": 0, "ALUSrc": 1, "RegDst": 0, "
                    RegWrite": 0, "MemRead": 0, "MemWrite": 1, "
                    MemToReg": 0}
45          },
46          "alu": {
47                  "aluop_size": 2,
48                  "func_size": 6,
49                  "control_size": 4,
50                  "control": [
51                          {"aluop": 2, "func": 8, "out": {"
                            Operation": 2}},
52
53                          {"aluop": 0, "out": {"Operation": 2}},
54                          {"aluop": 1, "out": {"Operation": 6}},
55                          {"aluop": 2, "func": 32, "out": {"
                            Operation": 2}},
56                          {"aluop": 2, "func": 34, "out": {"
                            Operation": 6}},
57                          {"aluop": 2, "func": 36, "out": {"
                            Operation": 0}},
58                          {"aluop": 2, "func": 37, "out": {"
                            Operation": 1}},
59                          {"aluop": 2, "func": 42, "out": {"
                            Operation": 7}},
60                          {"aluop": 2, "func": 39, "out": {"
                            Operation": 12}}
61                  ],
62                  "operations": {
63                          "0": "and",
64                          "1": "or",
65                          "2": "add",
66                          "6": "sub",
67                          "7": "slt",
```

```
68                              "12": "nor"
69                      }
70              }
71      }
```

6.3.2. Jump And Link Register

- **Instrucciones para un CPU monociclo**

```
1       {
2       "comment": "Instruction set of the reference book.",
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               "jr": {"type": "R", "args": ["reg"], "fields":
                   {"op": 1, "rs": "#1", "rt": 0, "rd": 0, "
                   shamt": 0, "func": 8}, "desc": "PC = $t1"},
10              "jalr": {"type": "R", "args": ["reg"], "fields
                   ": {"op": 3, "rs": "#1", "rt": 0, "rd": 31,
                   "shamt": 0, "func": 8}, "desc": "PC = $t1 &
                   $ra = PC"},
11
12              "nop":  {"type": "R", "fields": {"op": 0, "rs":
                    0, "rt": 0, "rd": 0, "shamt": 0, "func":
                   0}},
13              "add":  {"type": "R", "args": ["reg", "reg", "
                   reg"], "fields": {"op": 0, "rs": "#2", "rt":
                    "#3", "rd": "#1", "shamt": 0, "func": 32},
                   "desc": "$t1 = $t2 + $t3"},
14              "sub":  {"type": "R", "args": ["reg", "reg", "
                   reg"], "fields": {"op": 0, "rs": "#2", "rt":
                    "#3", "rd": "#1", "shamt": 0, "func": 34},
                   "desc": "$t1 = $t2 - $t3"},
15              "and":  {"type": "R", "args": ["reg", "reg", "
                   reg"], "fields": {"op": 0, "rs": "#2", "rt":
                    "#3", "rd": "#1", "shamt": 0, "func": 36},
                   "desc": "$t1 = $t2 & $t3"},
16              "or":   {"type": "R", "args": ["reg", "reg", "
                   reg"], "fields": {"op": 0, "rs": "#2", "rt":
                    "#3", "rd": "#1", "shamt": 0, "func": 37},
                   "desc": "$t1 = $t2 | $t3"},
17              "nor":   {"type": "R", "args": ["reg", "reg", "
                   reg"], "fields": {"op": 0, "rs": "#2", "rt":
                    "#3", "rd": "#1", "shamt": 0, "func": 39},
                   "desc": "$t1 = ~($t2 | $t3)"},
```

```
18                  "slt":  {"type": "R", "args": ["reg", "reg", "
                       reg"], "fields": {"op": 0, "rs": "#2", "rt":
                        "#3", "rd": "#1", "shamt": 0, "func": 42},
                       "desc": "$t1 = ($t2 < $t3) ? 1 : 0"},
19                  "j":    {"type": "J", "args": ["target"], "
                       fields": {"op": 2, "target": "#1"}, "desc":
                       "PC = target"},
20                  "addi": {"type": "I", "args": ["reg", "reg", "
                       int"], "fields": {"op": 8, "rs": "#2", "rt":
                        "#1", "imm": "#3"}, "desc": "$t1 = $t2 +
                       23"},
21                  "beq":  {"type": "I", "args": ["reg", "reg", "
                       offset"], "fields": {"op": 4, "rs": "#1", "
                       rt": "#2", "imm": "#3"}, "desc": "PC += ($t1
                        == $t2) ? (offset * 4 + 4) : 4"},
22                  "lw":   {"type": "I", "args": ["reg", "data"],
                       "fields": {"op": 35, "rs": "#2.offset", "rt
                       ": "#1", "imm": "#2.base"}, "desc": "$t1 =
                       MEM[base + $t2]"},
23                  "sw":   {"type": "I", "args": ["reg", "data"],
                       "fields": {"op": 43, "rs": "#2.offset", "rt
                       ": "#1", "imm": "#2.base"}, "desc": "MEM[
                       base + $t2] = $t1"}
24          },
25          "pseudo": {
26                  "li":   {"args": ["reg", "int"], "to": ["addi
                       #1, $0, #2"], "desc": "$t1 = 22"},
27                  "la":   {"args": ["reg", "label"], "to": ["addi
                        #1, $0, #2"], "desc": "$t1 = ADDR(label)"},
28                  "move": {"args": ["reg", "reg"], "to": ["add
                       #1, #2, $0"], "desc": "$t1 = $t2"},
29                  "subi": {"args": ["reg", "reg", "int"], "to":
                       ["li $1, #3", "sub #1, #2, $1"], "desc": "
                       $t1 = $t2 - 23"},
30                  "sgt":  {"args": ["reg", "reg", "reg"], "to":
                       ["slt #1, #3, #2"], "desc": "$t1 = ($t2 >
                       $t3) ? 1 : 0"},
31                  "bge":  {"args": ["reg", "reg", "offset"], "to
                       ": ["slt $1, #1, #2", "beq $1, $0, #3"], "
                       desc": "PC += ($t1 >= $t2) ? (offset * 4 +
                       4) : 4"},
32                  "ble":  {"args": ["reg", "reg", "offset"], "to
                       ": ["sgt $1, #1, #2", "beq $1, $0, #3"], "
                       desc": "PC += ($t1 <= $t2) ? (offset * 4 +
                       4) : 4"},
33                  "b":    {"args": ["offset"], "to": ["beq $0, $0
                       , #1"], "desc": "PC += offset * 4 + 4"},
34                  "neg":  {"args": ["reg", "reg"], "to": ["sub
                       #1, $0, #2"], "desc": "$t1 = -$t2"},
35                  "not":  {"args": ["reg", "reg"], "to": ["nor
                       #1, #2, $0"], "desc": "$t1 = ~$t2"}
36          },
37          "control": {
38                  "1": {"RegDst": 1, "RegWrite": 0, "ALUop": 2, "
```

```
                    AluSrc": 0, "MemToReg":0, "JumpReg": 1},
39                  "3": {"RegDst": 1, "RegWrite": 1, "ALUop": 2, "
                    AluSrc": 0, "MemToReg": 2, "JumpReg": 1},
40
41                  "0": {"RegDst": 1, "RegWrite": 1, "ALUOp": 2, "
                    ALUSrc": 0, "MemToReg": 0},
42                  "8": {"RegDst": 0, "RegWrite": 1, "ALUOp": 0, "
                    ALUSrc": 1, "MemToReg": 0},
43                  "2": {"Jump": 1},
44                  "4": {"ALUOp": 1, "ALUSrc": 0, "Branch": 1},
45                  "35": {"ALUOp": 0, "ALUSrc": 1, "RegDst": 0, "
                    RegWrite": 1, "MemRead": 1, "MemWrite": 0, "
                    MemToReg": 1},
46                  "43": {"ALUOp": 0, "ALUSrc": 1, "RegDst": 0, "
                    RegWrite": 0, "MemRead": 0, "MemWrite": 1, "
                    MemToReg": 0}
47          },
48          "alu": {
49                  "aluop_size": 2,
50                  "func_size": 6,
51                  "control_size": 4,
52                  "control": [
53                          {"aluop": 2, "func": 8, "out": {"
                            Operation": 2}},
54                          {"aluop": 0, "out": {"Operation": 2}},
55                          {"aluop": 1, "out": {"Operation": 6}},
56                          {"aluop": 2, "func": 32, "out": {"
                            Operation": 2}},
57                          {"aluop": 2, "func": 34, "out": {"
                            Operation": 6}},
58                          {"aluop": 2, "func": 36, "out": {"
                            Operation": 0}},
59                          {"aluop": 2, "func": 37, "out": {"
                            Operation": 1}},
60                          {"aluop": 2, "func": 39, "out": {"
                            Operation": 12}},
61                          {"aluop": 2, "func": 42, "out": {"
                            Operation": 7}}
62                  ],
63                  "operations": {
64                          "0": "and",
65                          "1": "or",
66                          "2": "add",
67                          "6": "sub",
68                          "7": "slt",
69                          "12": "nor"
70                  }
71          }
72  }
```

- **Instrucciones para un CPU pipeline**

```
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                 "j":    {"type": "J", "args": ["target"], "
                      fields": {"op": 2, "target": "#1"}, "desc":
                      "PC = target"},
10                "jr": {"type": "R", "args": ["reg"], "fields":
                      {"op": 1, "rs": "#1", "rt": 0, "rd": 0, "
                      shamt": 0, "func": 8}, "desc": "PC = $t1"},
11                "jalr": {"type": "R", "args": ["reg"], "fields
                      ": {"op": 3, "rs": "#1", "rt": 0, "rd": 31,
                      "shamt": 0, "func": 8}, "desc": "PC = $t1 &
                      $ra = PC + 4"},
12
13                "nop":  {"type": "R", "fields": {"op": 0, "rs":
                       0, "rt": 0, "rd": 0, "shamt": 0, "func":
                      0}},
14                "add":  {"type": "R", "args": ["reg", "reg", "
                      reg"], "fields": {"op": 0, "rs": "#2", "rt":
                       "#3", "rd": "#1", "shamt": 0, "func": 32},
                      "desc": "$t1 = $t2 + $t3"},
15                "sub":  {"type": "R", "args": ["reg", "reg", "
                      reg"], "fields": {"op": 0, "rs": "#2", "rt":
                       "#3", "rd": "#1", "shamt": 0, "func": 34},
                      "desc": "$t1 = $t2 - $t3"},
16                "and":  {"type": "R", "args": ["reg", "reg", "
                      reg"], "fields": {"op": 0, "rs": "#2", "rt":
                       "#3", "rd": "#1", "shamt": 0, "func": 36},
                      "desc": "$t1 = $t2 & $t3"},
17                "nor":   {"type": "R", "args": ["reg", "reg", "
                      reg"], "fields": {"op": 0, "rs": "#2", "rt":
                       "#3", "rd": "#1", "shamt": 0, "func": 39},
                      "desc": "$t1 = ~($t2 | $t3)"},
18                "or":   {"type": "R", "args": ["reg", "reg", "
                      reg"], "fields": {"op": 0, "rs": "#2", "rt":
                       "#3", "rd": "#1", "shamt": 0, "func": 37},
                      "desc": "$t1 = $t2 | $t3"},
19                "slt":  {"type": "R", "args": ["reg", "reg", "
                      reg"], "fields": {"op": 0, "rs": "#2", "rt":
                       "#3", "rd": "#1", "shamt": 0, "func": 42},
                      "desc": "$t1 = ($t2 < $t3) ? 1 : 0"},
20                "addi": {"type": "I", "args": ["reg", "reg", "
                      int"], "fields": {"op": 8, "rs": "#2", "rt":
```

```
                    "#1", "imm": "#3"}, "desc": "$t1 = $t2 +
                    23"},
21              "beq":  {"type": "I", "args": ["reg", "reg", "
                    offset"], "fields": {"op": 4, "rs": "#1", "
                    rt": "#2", "imm": "#3"}, "desc": "PC += ($t1
                     == $t2) ? (offset * 4 + 4) : 4"},
22              "lw":   {"type": "I", "args": ["reg", "data"],
                    "fields": {"op": 35, "rs": "#2.offset", "rt
                    ": "#1", "imm": "#2.base"}, "desc": "$t1 =
                    MEM[base + $t2]"},
23              "sw":   {"type": "I", "args": ["reg", "data"],
                    "fields": {"op": 43, "rs": "#2.offset", "rt
                    ": "#1", "imm": "#2.base"}, "desc": "MEM[
                    base + $t2] = $t1"}
24      },
25      "pseudo": {
26              "li":   {"args": ["reg", "int"], "to": ["addi
                    #1, $0, #2"], "desc": "$t1 = 22"},
27              "la":   {"args": ["reg", "label"], "to": ["addi
                     #1, $0, #2"], "desc": "$t1 = ADDR(label)"},
28              "move": {"args": ["reg", "reg"], "to": ["add
                    #1, #2, $0"], "desc": "$t1 = $t2"},
29              "subi": {"args": ["reg", "reg", "int"], "to":
                    ["li $1, #3", "sub #1, #2, $1"], "desc": "
                    $t1 = $t2 - 23"},
30              "sgt":  {"args": ["reg", "reg", "reg"], "to":
                    ["slt #1, #3, #2"], "desc": "$t1 = ($t2 >
                    $t3) ? 1 : 0"},
31              "bge":  {"args": ["reg", "reg", "offset"], "to
                    ": ["slt $1, #1, #2", "beq $1, $0, #3"], "
                    desc": "PC += ($t1 >= $t2) ? (offset * 4 +
                    4) : 4"},
32              "ble":  {"args": ["reg", "reg", "offset"], "to
                    ": ["sgt $1, #1, #2", "beq $1, $0, #3"], "
                    desc": "PC += ($t1 <= $t2) ? (offset * 4 +
                    4) : 4"},
33              "b":    {"args": ["offset"], "to": ["beq $0, $0
                    , #1"], "desc": "PC += offset * 4 + 4"},
34              "neg":  {"args": ["reg", "reg"], "to": ["sub
                    #1, $0, #2"], "desc": "$t1 = -$t2"},
35              "not":  {"args": ["reg", "reg"], "to": ["nor
                    #1, #2, $0"], "desc": "$t1 = ~$t2"}
36      },
37      "control": {
38              "2": {"Jump": 1},
39              "1": {"RegDst": 1, "RegWrite": 0, "ALUop": 2, "
                    AluSrc": 0, "MemToReg":0, "JumpReg": 1},
40              "3": {"RegDst": 1, "RegWrite": 1, "ALUop": 2, "
                    AluSrc": 0, "MemToReg": 2, "JumpReg": 1},
41
42              "0": {"RegDst": 1, "RegWrite": 1, "ALUOp": 2, "
                    ALUSrc": 0, "MemToReg": 0},
43              "8": {"RegDst": 0, "RegWrite": 1, "ALUOp": 0, "
                    ALUSrc": 1, "MemToReg": 0},
```

```
44                  "4": {"ALUOp": 1, "ALUSrc": 0, "Branch": 1},
45                  "35": {"ALUOp": 0, "ALUSrc": 1, "RegDst": 0, "
                       RegWrite": 1, "MemRead": 1, "MemWrite": 0, "
                       MemToReg": 1},
46                  "43": {"ALUOp": 0, "ALUSrc": 1, "RegDst": 0, "
                       RegWrite": 0, "MemRead": 0, "MemWrite": 1, "
                       MemToReg": 0}
47          },
48          "alu": {
49                  "aluop_size": 2,
50                  "func_size": 6,
51                  "control_size": 4,
52                  "control": [
53                          {"aluop": 2, "func": 8, "out": {"
                               Operation": 2}},
54                          {"aluop": 0, "out": {"Operation": 2}},
55                          {"aluop": 1, "out": {"Operation": 6}},
56                          {"aluop": 2, "func": 32, "out": {"
                               Operation": 2}},
57                          {"aluop": 2, "func": 34, "out": {"
                               Operation": 6}},
58                          {"aluop": 2, "func": 36, "out": {"
                               Operation": 0}},
59                          {"aluop": 2, "func": 37, "out": {"
                               Operation": 1}},
60                          {"aluop": 2, "func": 42, "out": {"
                               Operation": 7}},
61                          {"aluop": 2, "func": 39, "out": {"
                               Operation": 12}}
62                  ],
63                  "operations": {
64                          "0": "and",
65                          "1": "or",
66                          "2": "add",
67                          "6": "sub",
68                          "7": "slt",
69                          "12": "nor"
70                  }
71          }
72  }
```

7. Referencias

1. DrMIPS, https://brunonova.github.io/drmips