

PRACTICA 4 - Algunas soluciones

Segmentación de cauce en procesador RISC

Ejercicio 2

El siguiente programa intercambia el contenido de dos palabras de la memoria de datos, etiquetadas A y B.

```
.data
A: .word 1
B: .word 2
.code
ld    r1, A(r0)
ld    r2, B(r0)
sd    r2, A(r0)
sd    r1, B(r0)
halt
```

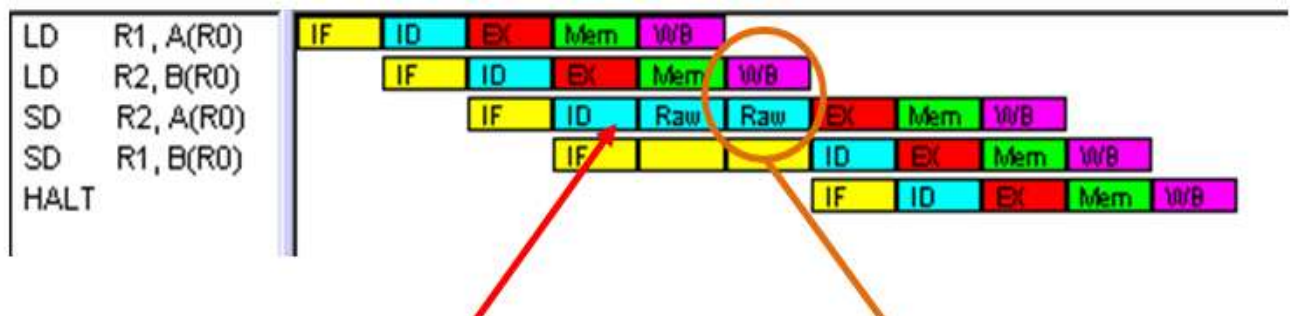
- a) Ejecutarlo en el simulador con la opción Configure/Enable Forwarding deshabilitada. Analizar paso a paso su funcionamiento, examinar las distintas ventanas que se muestran en el simulador y responder:

- ¿Qué instrucción está generando atascos (stalls) en el cauce (ó pipeline) y por qué?

La instrucción SD R2,A(r0) almacena en la dirección de memoria A el valor contenido en el registro R2, mientras que la instrucción anterior LD R2,B(r0) carga desde la dirección de memoria B el contenido de esa posición en el registro R2.

Cuando forwarding no está habilitado la instrucción SD R2,A(r0) que en la etapa ID trata de leer el contenido de R2, pero dicho contenido no estará disponible hasta que la instrucción anterior LD R2,B(r0) llegue a la etapa WB. Y Debido a esto se genera un atasco en la etapa ID donde se procesa la instrucción SD R2,A(r0) retrasando la salida de esta etapa (con RAWs) a la espera del contenido del registro. Esto además genera que la instrucción posterior SD R1, B(r0) deba permanecer en la etapa IF sin poder avanzar a la etapa ID (el pipeline se detuvo) como se puede ver en la imagen.

Tener en cuenta que en la primera mitad de la etapa WB se escribe el contenido en R2 y en el segundo RAW se lee el contenido de dicho registro.

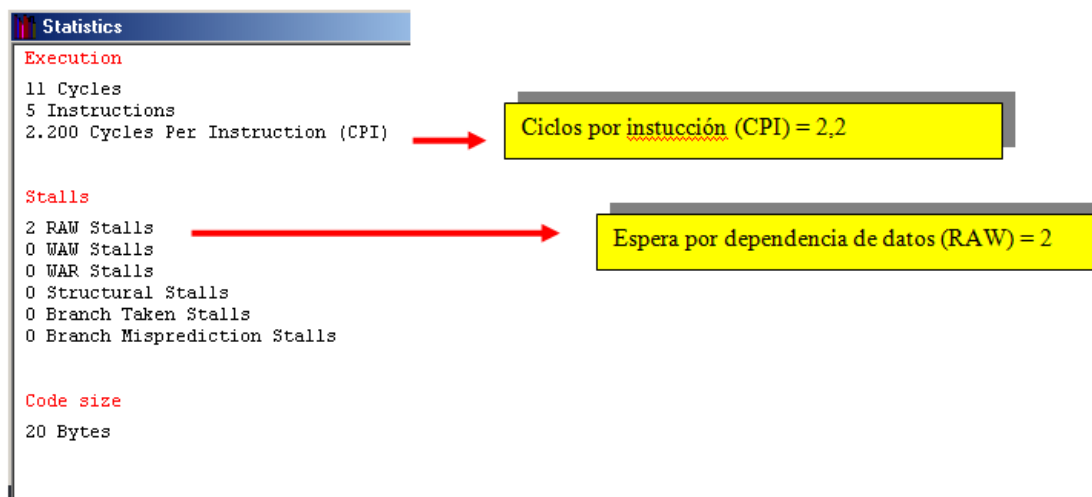


- ¿Qué tipo de 'stall' es el que aparece?

Aparecen atascos de tipo RAW (Read After Write) causado por una dependencia de datos, en este caso se intenta leer un dato antes que esté guardado en el registro. Serán dos atascos, equivalentes a dos ciclos hasta que LD R2, B(r0) salga de la etapa WB.

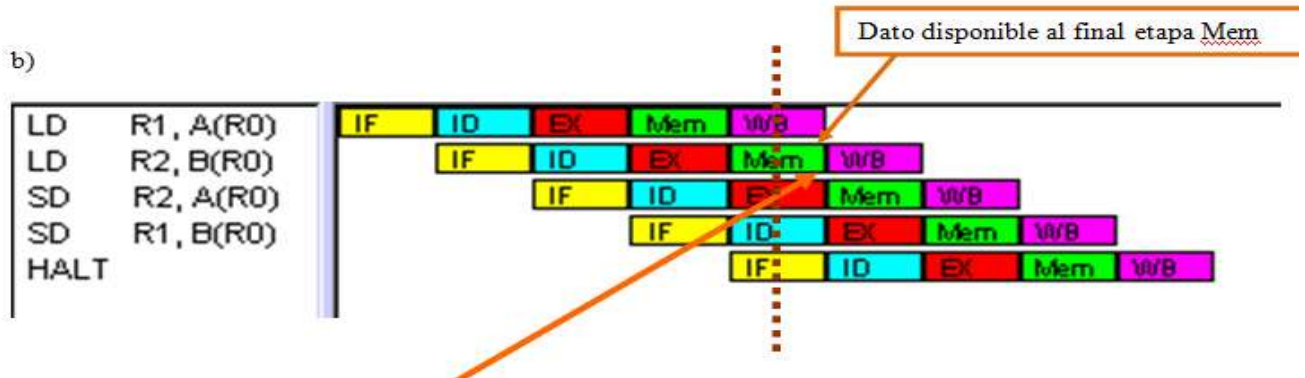
- ¿Cuál es el promedio de Ciclos Por Instrucción (CPI) en la ejecución de este programa bajo esta configuración?

El CPI de la ejecución de este programa es de 2.2



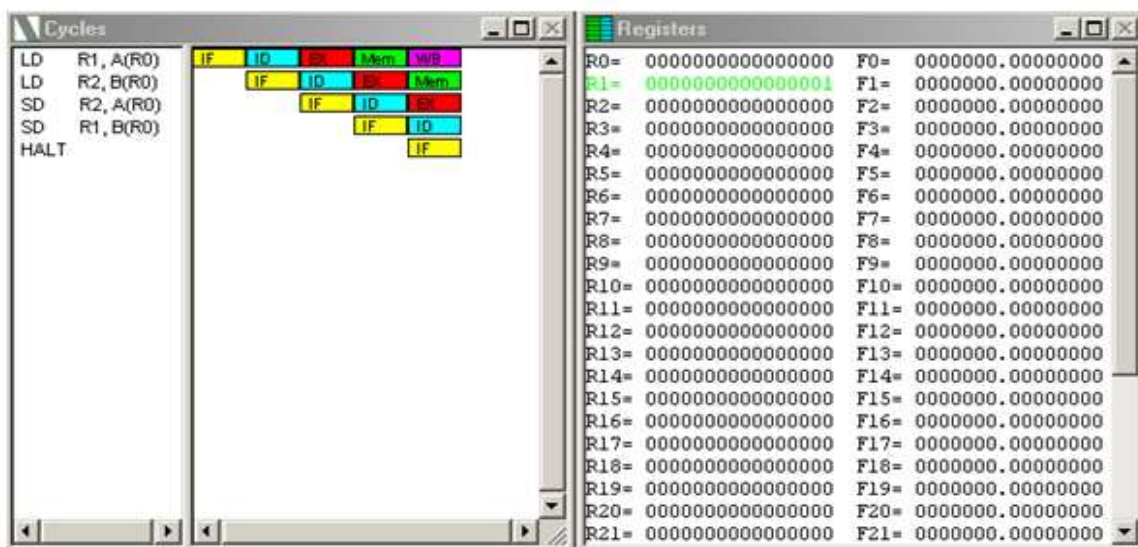
- b) Una forma de solucionar los atascos por dependencia de datos es utilizando el Adelantamiento de Operandos o Forwarding. Ejecutar nuevamente el programa anterior con la opción Enable Forwarding habilitada y responder:
- ¿Por qué no se presenta ningún atasco en este caso? Explicar la mejora.

Con la opción forwarding habilitada el dato contenido en el registro R2 podrá ser leído por la instrucción SD R2, A(R0) cuando la instrucción LD R2, B(R0) se encuentra finalizando la etapa MEM. La instrucción SD R2, A(R0) no tiene que esperar a que la instrucción LD R2, B(R0) salga de la etapa WB. De esta manera no aparecen atascos del tipo RAW.



- ¿Qué indica el color de los registros en la ventana Register durante la ejecución?

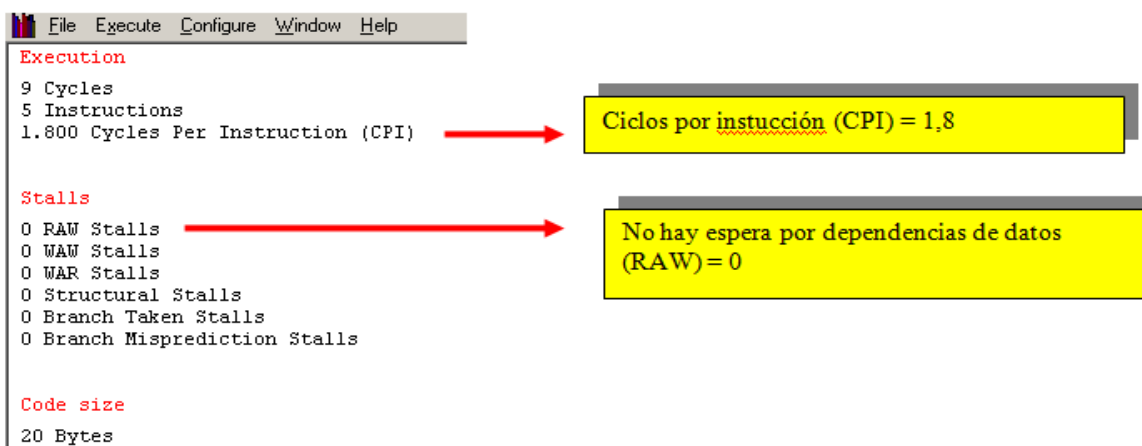
El color verde en el registro R1 significa que el dato está disponible en etapa MEM para adelantamiento.



Además, los registros pueden tener color Rojo indicando que el resultado está disponible en EX y puede ser adelantado. Si el color es Gris: el valor no está disponible en este ciclo para adelantamiento.

- ¿Cuál es el promedio de Ciclos Por Instrucción (CPI) en este caso? Comparar con el anterior.

En este caso el CPI es de 1,8 y por estar más cercana a 1 (valor ideal) la ejecución del programa está más cerca de un desempeño óptimo.



Ejercicio 3

Analizar el siguiente programa con el simulador MIPS64:

```
.data
A:    .word 1
B:    .word 3
.code
ld     r1, A(r0)
ld     r2, B(r0)
loop: dsll  r1, r1, 1
      daddi r2, r2, -1
      bnez  r2, loop
      halt
```

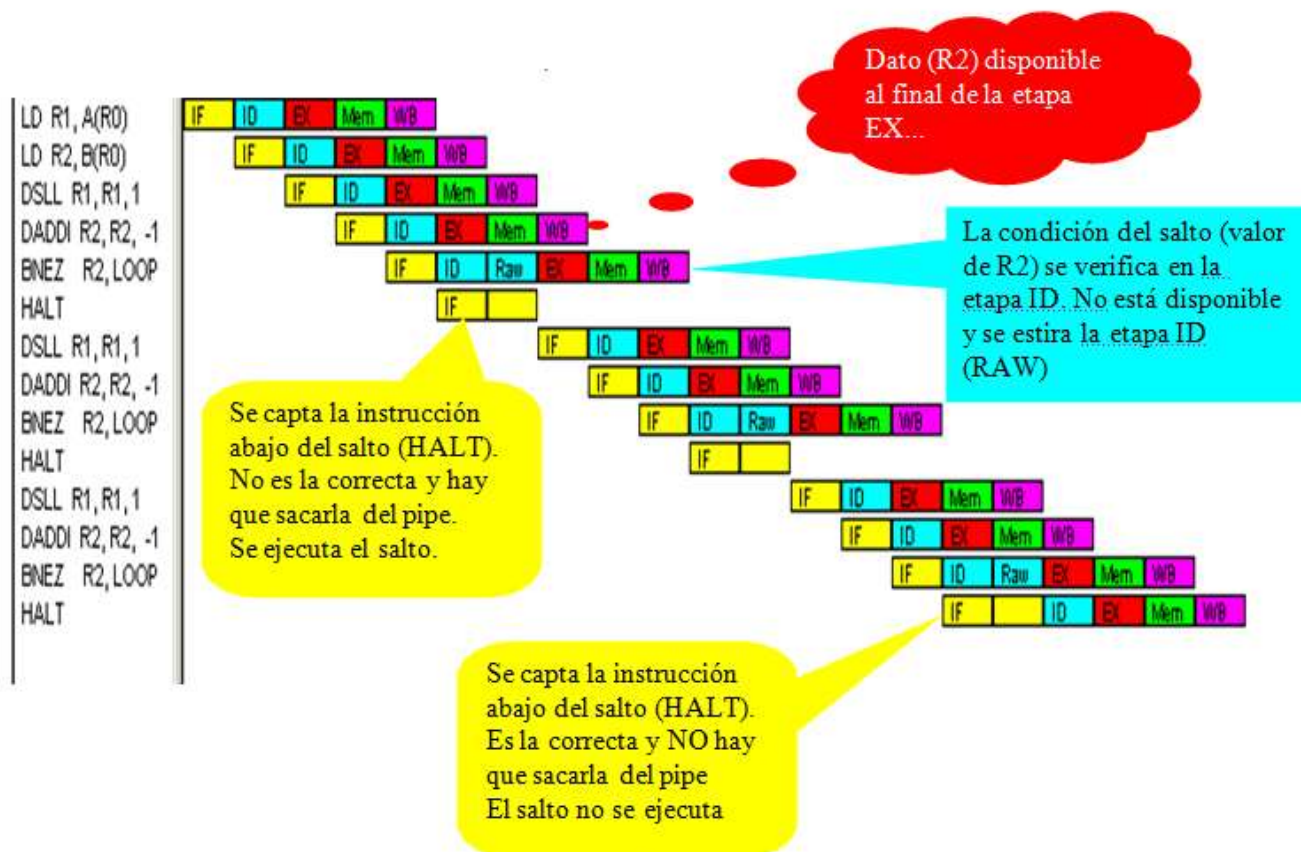
a) Ejecutar el programa con Forwarding habilitado y responder:

- ¿Por qué se presentan atascos tipo RAW?

Se presentan atasco por dependencia de datos de tipo RAW causado por la instrucción BNEZ R2, loop al procesarse en la etapa ID. Esta instrucción necesita del contenido del registro R2 que está siendo utilizado por la instrucción DADDI R2,R2,-1 en la etapa EX sin salir aún de esta.

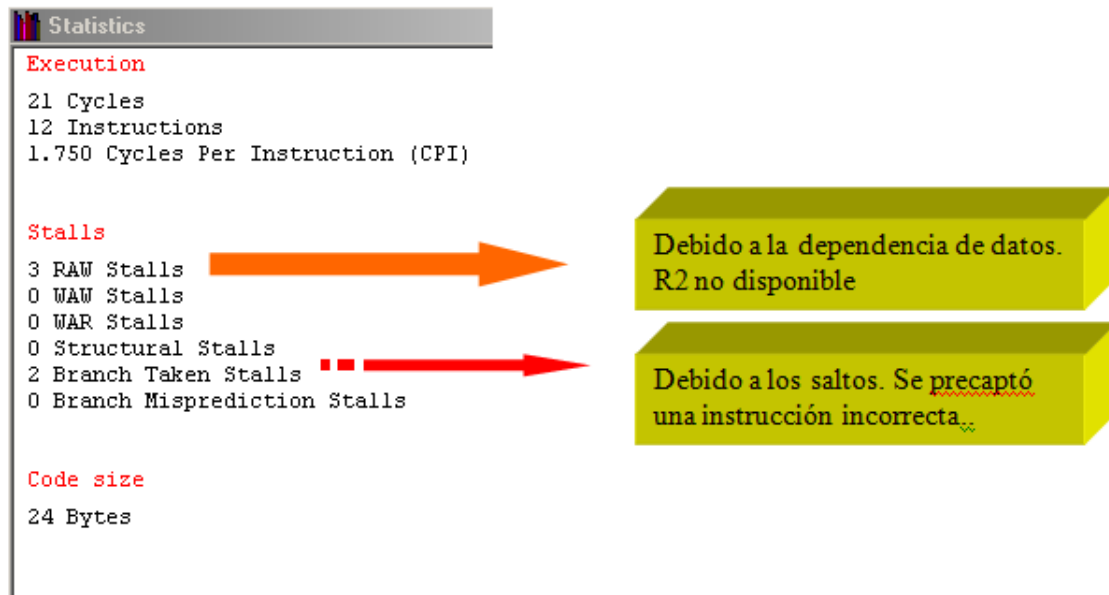
- Branch Taken es otro tipo de atasco que aparece. ¿Qué significa? ¿Por qué se produce?

El atasco de tipo Branch Taken Stalls (BTS), ocurre como consecuencia de la ejecución incorrecta de la instrucción siguiente a una instrucción condicional. Esto se debe a que la condición a evaluar tarda algunos ciclos en ser ejecutada, mientras que durante esos ciclos siguen entrando nuevas instrucciones al pipeline. Luego de evaluada la condición si la instrucción posterior a ésta que se ejecutó no es la que debía ser ejecutada, su ejecución se trunca y se ejecuta la que está en el lugar de memoria indicada por la etiqueta en la instrucción condicional.



- ¿Cuántos CPI tiene la ejecución de este programa? Tomar nota del número de ciclos, cantidad de instrucciones y CPI.

Se ejecutan 12 instrucciones en 21 ciclos dando un CPI de 1.750



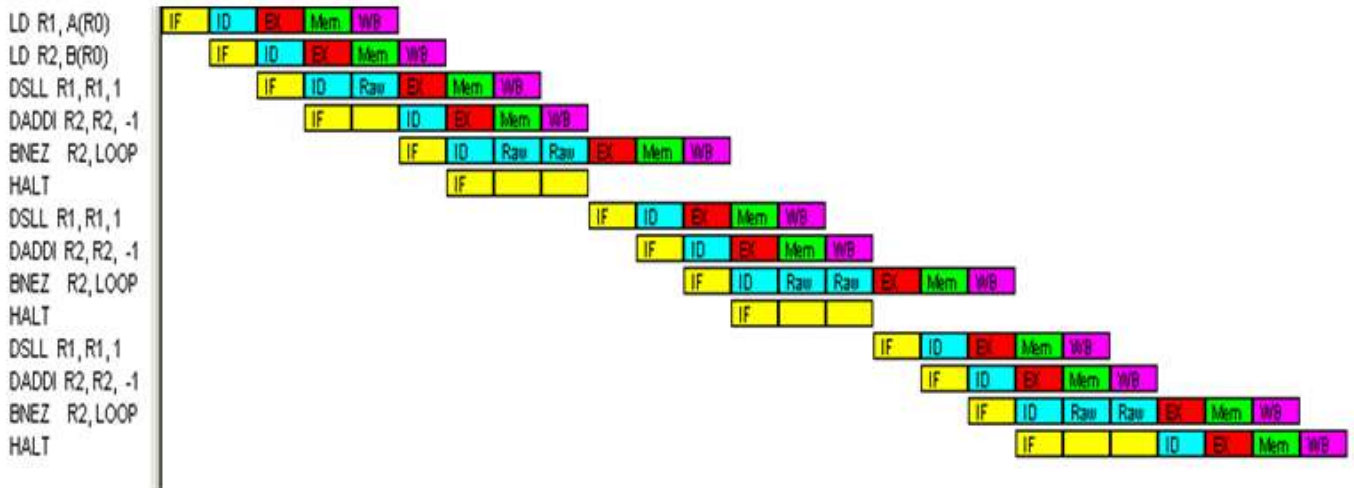
b) Ejecutar ahora el programa deshabilitando el Forwarding y responder:

- ¿Qué instrucciones generan los atascos tipo RAW y por qué? ¿En qué etapa del cauce se produce el atasco en cada caso y durante cuántos ciclos?

Las instrucciones que generan los atascos RAW son:

la instrucción DSLL R1,R1,1 (ver registro R1 coloreado en gris), que trata de leer el contenido del registro R1, mientras que la instrucción LD R1,A(r0) todavía no copio el contenido de la dirección de memoria A en R1 y permanece aún en la etapa WB (RAW durante 1 ciclo).

Y la instrucción BNEZ R2,loop (ver registro R2 coloreado en gris), que trata de leer el contenido del registro R2, mientras que DADDI R2,R2,-1 está buscando copiar el resultado de la operación en dicho registro, permaneciendo en la etapa MEM y posteriormente en la etapa WB (RAW durante 2 ciclos).



- Los Branch Taken Stalls se siguen generando. ¿Qué cantidad de ciclos dura este atasco en cada vuelta del lazo 'loop'? Comparar con la ejecución con Forwarding y explicar la diferencia.

Con forwarding deshabilitada, Los atascos por Branch Taken Stalls duran 2 ciclos en cada vuelta del lazo loop, mientras que con dicha opción habilitada se reducen a 1 ciclo por vuelta de lazo.

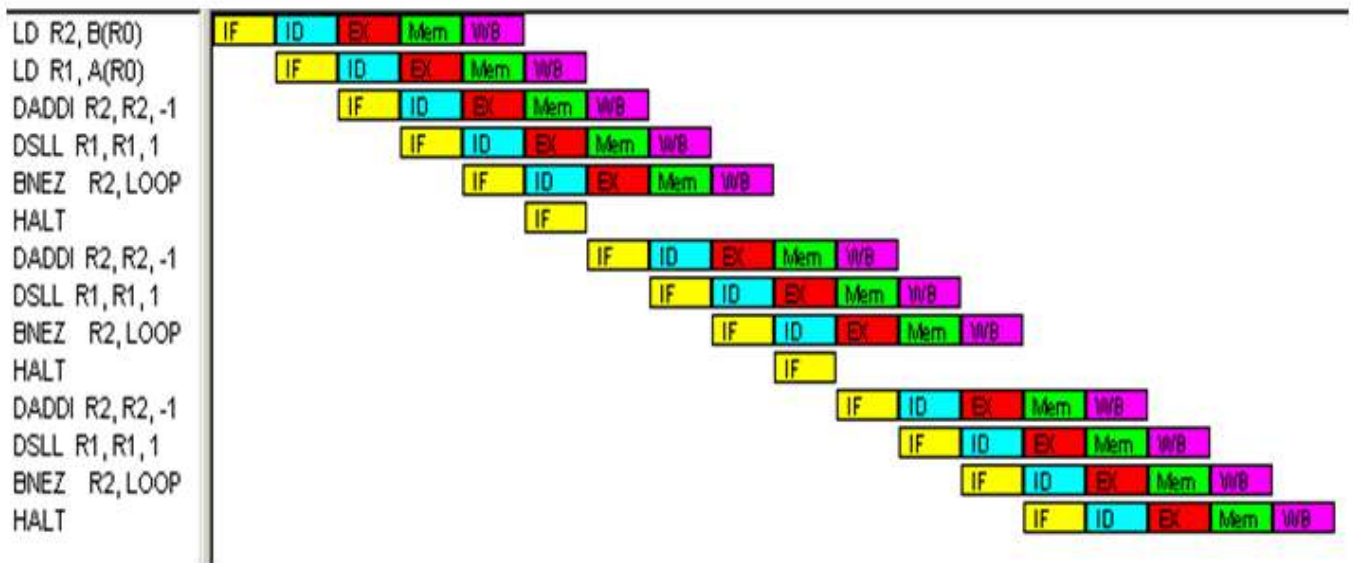
Esta diferencia tiene su causa en la instrucción condicional que es la que está generando los atascos RAW; entonces al disminuir la cantidad de RAWs producidos por esta, también disminuyen los ciclos de espera de la instrucción siguiente, que además se dejara de ejecutar si la condicional así se lo indica al procesador.

- ¿Cuántos CPI tiene la ejecución del programa en este caso? Comparar número de ciclos, cantidad de instrucciones y CPI con el caso con Forwarding.

A REALIZAR POR EL ALUMNO

c) Reordenar las instrucciones para que la cantidad de RAW sea '0' en la ejecución del programa (Forwarding habilitado)

Con la opción forwarding habilitada aún siguen apareciendo atascos RAW. Existe la posibilidad de tratar de eliminarlos reordenando las instrucciones en el programa y Dicho reordenamiento no debe afectar la lógica de este. El resultado de este método puede observarse en la imagen siguiente.



Statistics	
Execution	
18 Cycles	
12 Instructions	
1.500 Cycles Per Instruction (CPI)	
Stalls	
0 RAW Stalls	
0 WAW Stalls	
0 WAR Stalls	
0 Structural Stalls	
2 Branch Taken Stalls	
0 Branch Misprediction Stalls	
Code size	
24 Bytes	

- d) Modificar el programa para que almacene en un arreglo en memoria de datos los contenidos parciales del registro r1 ¿Qué significado tienen los elementos de la tabla que se genera?

A REALIZAR POR EL ALUMNO

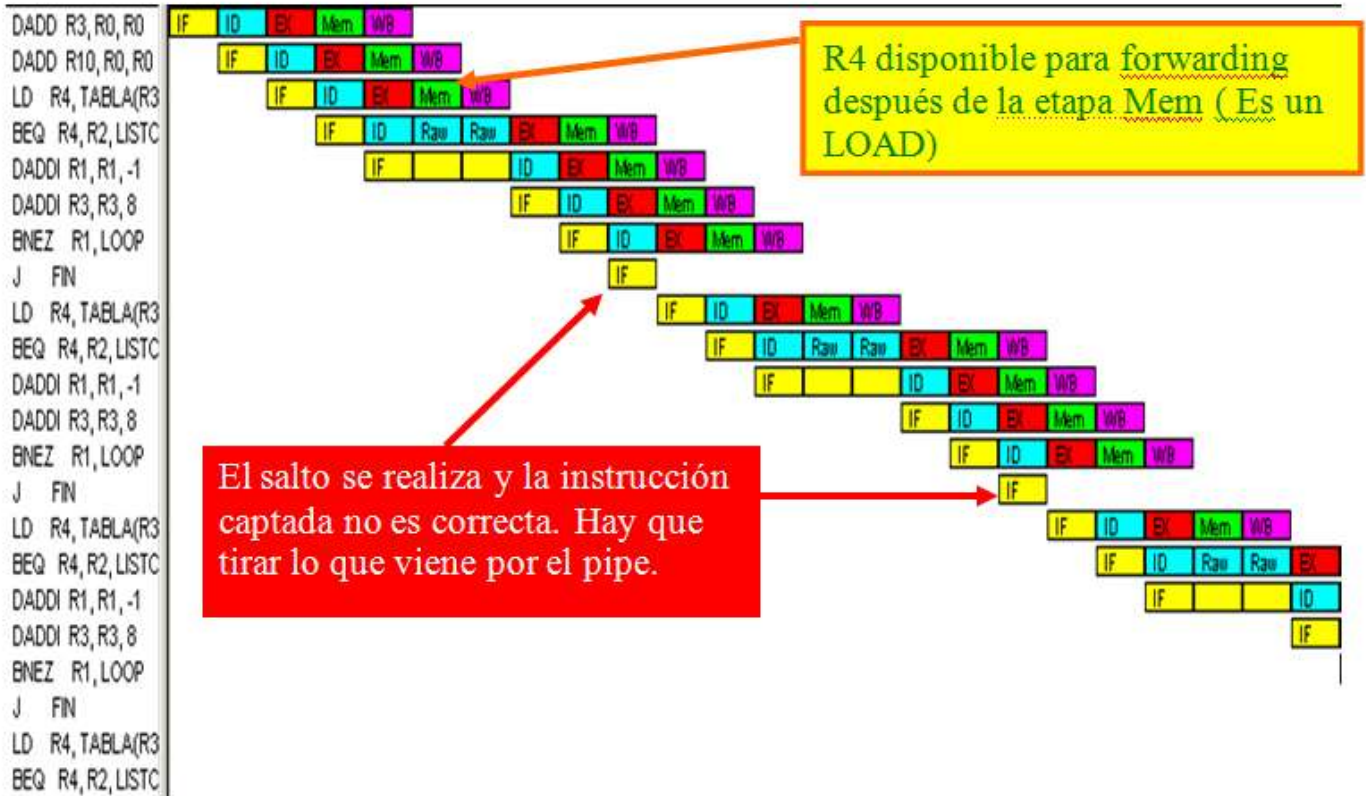
Ejercicio 4

Dado el siguiente programa:

```
.data
tabla: .word 20, 1, 14, 3, 2, 58, 18, 7, 12, 11
num: .word 7
long: .word 10
.code
ld    r1, long(r0)
ld    r2, num(r0)
dadd  r3, r0, r0
dadd  r10, r0, r0
loop: ld    r4, tabla(r3)
      beq   r4, r2, listo
      daddi r1, r1, -1
      daddi r3, r3, 8
      bnez  r1, loop
      j     fin
listo: daddi r10, r0, 1
fin:  halt
```

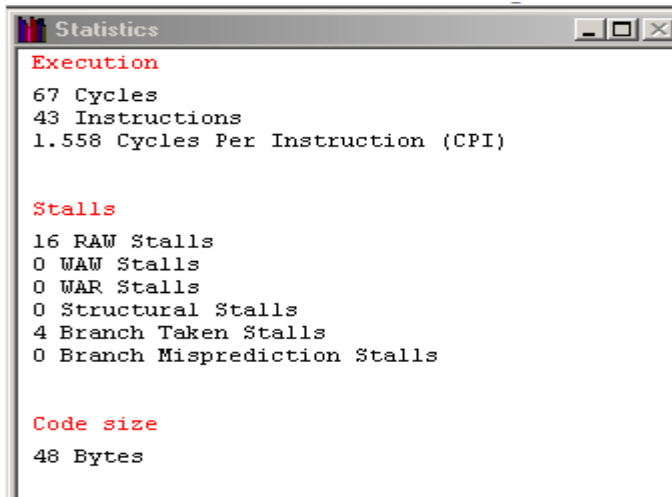
- Ejecutar en simulador con Forwarding habilitado. ¿Qué tarea realiza? ¿Cuál es el resultado y dónde queda indicado?
- Re-Ejecutar el programa con la opción Configure/Enable Branch Target Buffer habilitada. Explicar la ventaja de usar este método y cómo trabaja.

c) Confeccionar una tabla que compare número de ciclos, CPI, RAWs y Branch Taken Stalls para los dos casos anteriores. El programa busca en TABLA un elemento igual al contenido en la dirección de memoria NUM. En este caso dicha coincidencia se produce cuando el contenido del registro R4 es igual al contenido del registro R2 ($R4=R2$), razón por la cual luego de evaluada esta condición y de resultar verdadera se salta a la posición de memoria indicada por la etiqueta "listo". Cuando hay coincidencia la línea de programa en listo suma al registro R10 un 1, caso contrario el contenido del registro R10 queda en 0. Este es el resultado y queda almacenado en el registro R10. El registro R3 se utiliza como índice para recorrer la TABLA. El contenido del registro R3 se incrementa de 8 porque cada elemento de tabla es del tamaño word, es decir de 64 bits (8 bytes).



Statistics	
Execution	
71 Cycles	
43 Instructions	
1.651 Cycles Per Instruction (CPI)	
Stalls	
16 RAW Stalls	
0 WAW Stalls	
0 WAR Stalls	
0 Structural Stalls	
8 Branch Taken Stalls	
0 Branch Misprediction Stalls	
Code size	
48 Bytes	

Habilitando la opción Branch Target Buffer (BTB) logramos reducir los atascos Branch Taken stalls a la mitad. Tener en cuenta que esta opción es útil cuando aumenta la cantidad de iteraciones de un lazo. Como vemos también esta opción no actúa sobre los atascos por dependencia de datos (RAW en este caso) que no se modifican.



```

Statistics
Execution
67 Cycles
43 Instructions
1.558 Cycles Per Instruction (CPI)

Stalls
16 RAW Stalls
0 WAW Stalls
0 WAR Stalls
0 Structural Stalls
4 Branch Taken Stalls
0 Branch Misprediction Stalls

Code size
48 Bytes

```

Ejercicio 7

Escribir un programa que recorra una TABLA de diez números enteros y determine cuántos elementos son mayores que X. El resultado debe almacenarse en una dirección etiquetada CANT. El programa debe generar además otro arreglo llamado RES cuyos elementos sean ceros y unos. Un '1' indicará que el entero correspondiente en el arreglo TABLA es mayor que X, mientras que un '0' indicará que es menor o igual.

Definir en la zona de memoria de datos una tabla (TABLA) con 10 números enteros, a continuación un arreglo (RES) donde se cargaran 1s o 0s según los números definidos en tabla sean "mayor" o "menor o igual" que un número (X) que en memoria a continuación del arreglo RES. También definiremos en memoria una variable LONG que valdrá 10 y otra CANT inicializada en 0. Todas estas variables serán de tamaño word. Faltaría definir una máscara para detectar si el número verificado es mayor o menor, para este fin utilizaremos la 0x8000 como MASK que también definiremos en memoria (con una operación AND logramos este cometido).

Recordar que se debe contabilizar la cantidad de números que son mayores que el número "x" y guardar ese resultado en la variable en memoria CANT.

Ejercicio 8

Escribir un programa que multiplique dos números enteros utilizando sumas repetidas (similar a Ejercicio 6 o 7 de la Práctica 1). El programa debe estar optimizado para su ejecución con la opción Delay Slot habilitada.

El programa podría ser así:

```

.data
NUM1: .word 3
NUM2: .word 8
RES:  .word 0
.code
    ld      R1, NUM1(r0)
    ld      R2, NUM2(r0)
    dadd    R3, R0, R0
loop:  dadd  R3, R3, R1
    daddi   R2, R2, -1
    bnez   R2, loop
    sd     R3, RES(r0)
    halt

```

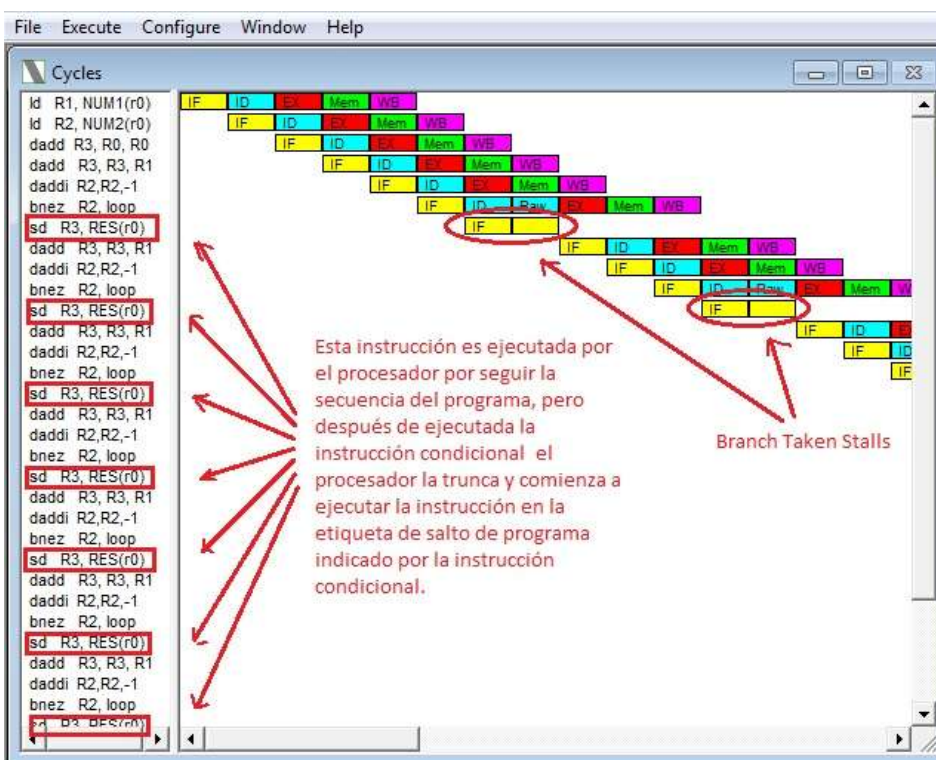
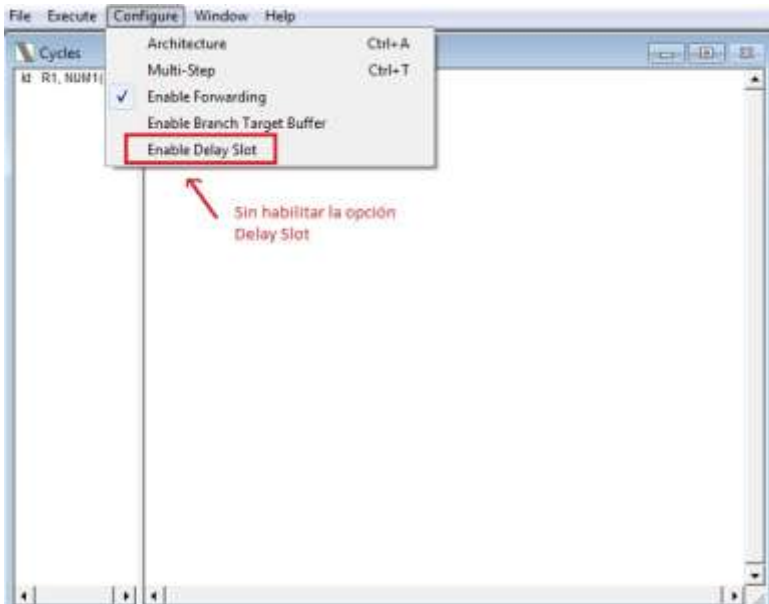
Otra versión para que puedan analizar es:

```

.data
num1: .word 3
num2: .word 5
res:  .word 0
.code
    LD      r1, num1(r0)
    LD      r2, num2(r0)
    DADD    r10, r0, r0
LOOP:  DADDI r2, r2, -1
    BNEZ    r2, LOOP
    DADD    r10, r10, r1
    SD      r10, res(r0)
    HALT

```

Su ejecución en el simulador con la opción Delay Slot deshabilitada mostraría lo siguiente:



Statistics	
Execution	
48 Cycles	
29 Instructions	
1.655 Cycles Per Instruction (CPI)	
Stalls	
8 RAW Stalls	
0 WAW Stalls	
0 WAR Stalls	
0 Structural Stalls	
7 Branch Taken Stalls	
0 Branch Misprediction Stalls	
Code size	
32 Bytes	

Con opción Delay Slot deshabilitada

Los atascos por dependencia de datos RAW, son independientes de estos atascos Branch Taken Stalls.

Con la opción Delay Slot habilitada se podrá observar que eliminamos los atascos Branch Taken Stalls, logrando optimizar el CPI de 1.655 a 1.333.

Tener presente que habilitar esta opción equivale a ejecutar completamente la instrucción siguiente a una

instrucción condicional, razón por la cual nunca debe haber una instrucción HALT inmediatamente después de una instrucción condicional.

Por otro lado se debe aprovechar esta bondad de la opción, para colocar luego de las instrucciones condicionales aquellas que deban ejecutarse en algún momento del programa y que al permutarlas a esa posición no cambien la lógica de este.

