Introducción

Suposiciones:

- Todas las tareas duran el mismo tiempo.
- Las instrucciones siempre pasan por todas las etapas.
- Todos las etapas pueden ser manejadas en paralelo.

Algunos Problemas

Problemas:

- No todas las instrucciones necesitan todas las etapas
  - *SW RT, inmed(RS)* - no utiliza W
  - En VonSim *MOV AX, mem* no requiere EX
- No todas las etapas pueden ser manejadas en paralelo.
  - F y M acceden a memoria
- No se tienen en cuenta los saltos de control.

Atascos

Llamamos *atasco* a la situación que impide a una o mas instrucciones seguir su camino en el cauce.

- Estructural
  - Provocados por conflictos con los recursos.
- Dependencia de Datos
  - Dos instrucciones se comunican por medio de un dato
- Dependencia de Control
  - La ejecución de una instrucción depende de cómo se ejecute otra

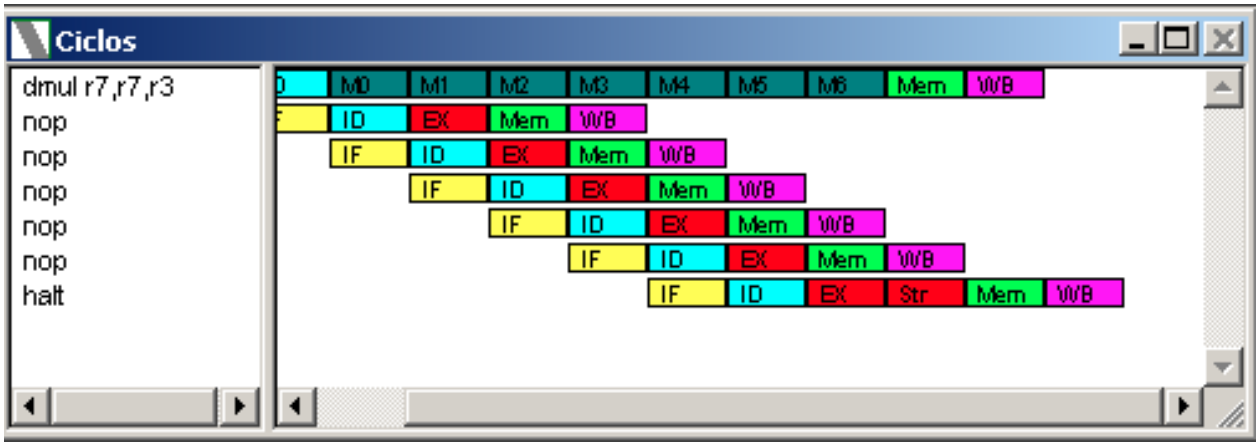
Si resolvemos con paradas del cauce, disminuye el rendimiento teórico

Atascos Estructurales

Dos o mas instrucciones necesitan utilizar el mismo recurso hardware en el mismo ciclo.

```
.code
dmul r7,r7,r3
nop
nop
nop
nop
nop
halt
```

Atasco Estructural



Atasco por Dependencia de Datos

Condición en la que los operandos fuente o destino de una instrucción no están disponibles en el momento en que se necesitan en una etapa determinada del cauce.

- Lectura después de Escritura (RAW, dependencia verdadera)
  - una instrucción genera un dato que lee otra posterior
- Escritura después de Escritura (WAW, dependencia en salida)
  - una instrucción escribe un dato después que otra posterior
  - sólo se da si se deja que las instrucciones se adelanten unas a otras
- Escritura después de Lectura (WAR, antidependencia)
  - una instrucción modifica un valor antes de que otra anterior que lo tiene que leer lo lea
  - Es el que menos suele darse

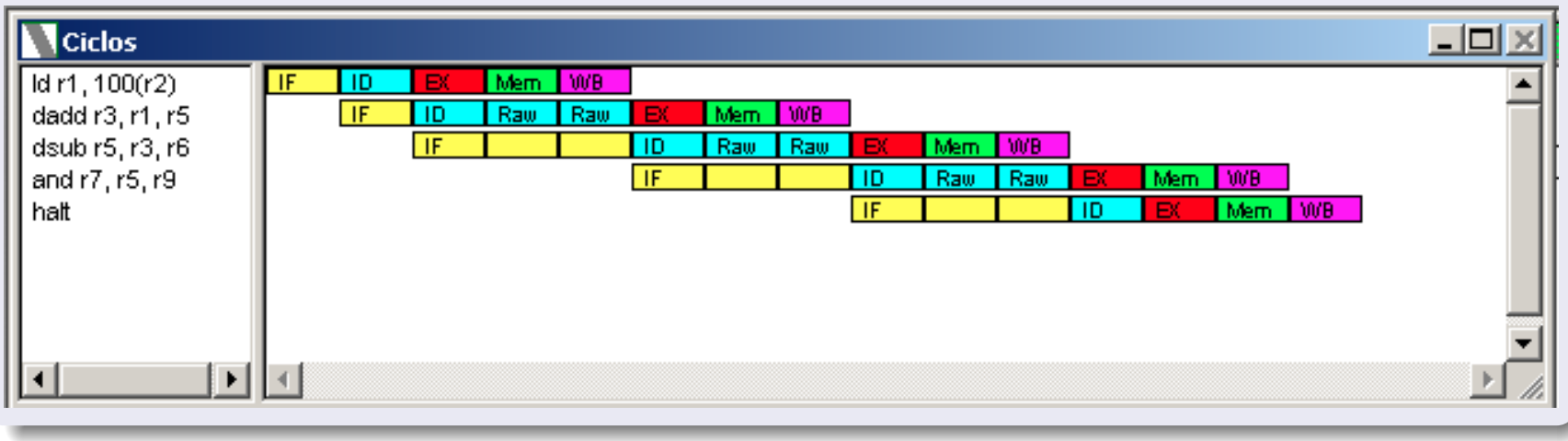
RAW

Una instrucción depende del resultado de otra instrucción que todavía no ha finalizado y debe esperar a que el resultado este disponible.

```
.code
LD   r1, 100(r2)
DADD r3, r1, r5
DSUB r5, r3, r6
AND  r7, r5, r9
```

RAW

Una instrucción depende del resultado de otra instrucción que todavía no ha finalizado y debe esperar a que el resultado este disponible.



WAR

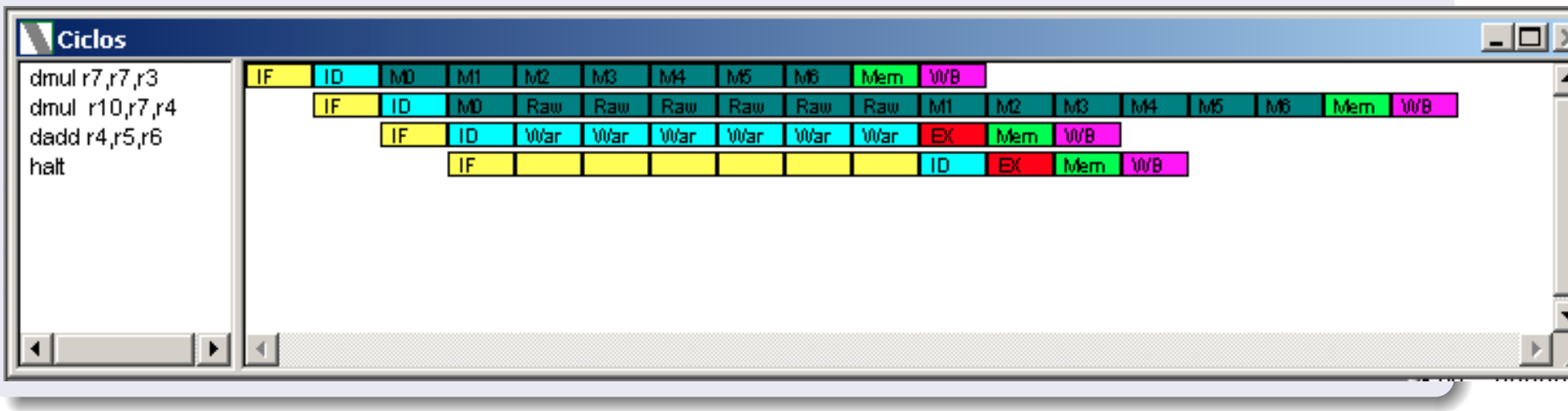
Una instrucción escribe el valor de un registro antes que otra anterior que lo tiene que leer lo lea

```
.code ; Activar Forwarding

dmul r7,r1,r3
dmul r10,r7,r4
dadd r4,r5,r6
halt
```

WAR

Una instrucción escribe el valor de un registro antes que otra anterior que lo tiene que leer lo lea



WAW

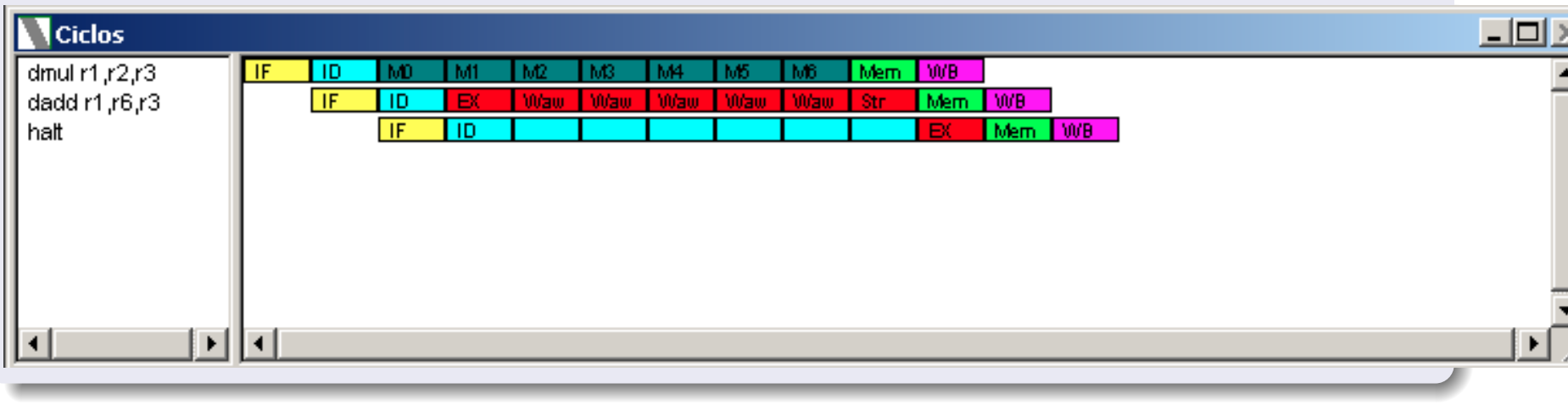
Una instrucción escribe un dato después que otra posterior

```
.code

dmul r1,r2,r3
dadd r1,r6,r3
halt
```

WAW

Una instrucción escribe un dato después que otra posterior



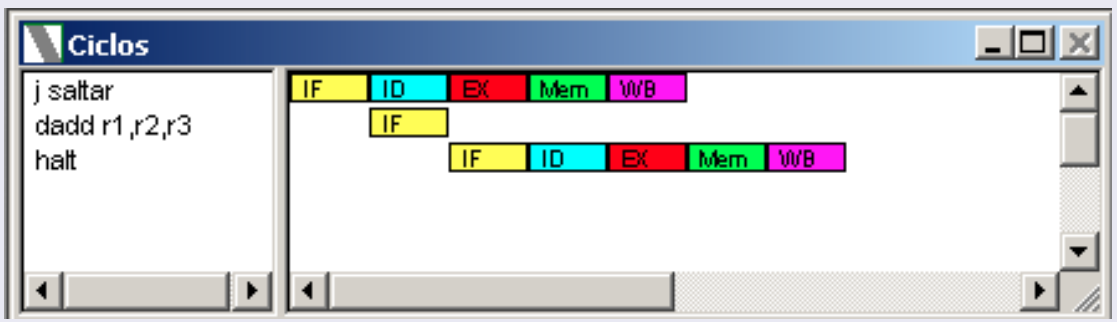
General

Una instrucción que modifica el valor del PC no lo ha hecho cuando se tiene que comenzar la siguiente.

- Estas instrucciones son saltos condicionales o incondicionales.
- La decodificación de la instrucción se hace en el segundo ciclo de una instrucción, la siguiente instrucción ya ingreso en el cause en la etapa *IF*.
- Si el salto realmente se ejecuta, la próxima instrucción no sera la inmediata siguiente sino la que se determine por la instrucción

```
        .code
        j saltar
        dadd r1,r2,r3
saltar: halt
```

Atasco por Dependencia de Control



Soluciones a riesgos estructurales

Replicar, segmentar ó realizar turnos para el acceso a las unidades funcionales en conflicto.

- Duplicación de recursos hardware
  - Unidades separadas para multiplicar, dividir además de la ALU
- Separación en memorias de instrucciones y datos
- Turnar el acceso al banco de registros
  - Escrituras en la primera mitad de los ciclos de reloj
  - Lecturas en la segunda mitad de los ciclos de reloj

General

- Se debe determinar cómo y cuando aparecen esos riesgos
- Hay dos tipos de soluciones
  - Software
    - Podemos agregar instrucciones *NOP*, o reordenar las instrucciones
  - Hardware
    - Adelantamiento de Operandos (Forwarding)

Agregar Instrucciones NOP

Cambiemos el ejemplo visto anteriormente

```
.code
ld r1, 100(r2)
nop
nop
dadd r3, r1, r5
nop
nop
dsub r5, r3, r6
nop
nop
and r7, r5, r9
halt
```

Reordenar instrucciones

Veamos otro ejemplo

```
.data
A:  .word 5
.code
ld r1, A(r0)
dadd r1, r1, r1
daddi r2, r0, 3
dadd r3, r0, r0
halt
```

Tenemos un atasco con el registro *r1*.

Reordenar instrucciones

Retrasando el *dadd* sobre el registro *r*1.

```
.data
A:  .word 5
.code
ld r1, A(r0)
daddi r2, r0, 3
dadd r3, r0, r0
dadd r1, r1, r1
halt
```

No hay atascos ahora.

Forwarding

Forwarding, Adelantamiento o Cortocircuito

- Consiste en pasar directamente el resultado obtenido con una instrucción a las instrucciones que lo necesitan como operando.
- Si el dato necesario está disponible antes se lleva a la entrada de la etapa correspondiente sin esperar a llegar a la etapa escritura de escritura del banco de registros.
- La idea es tener disponible el operando lo antes posible para no perder ciclos
- Usar esta técnica no implica la eliminación de todos los atascos.
- En WinMIPS64 podemos activarlo o desactivarlo en cualquier momento. Un cambio en esta configuración implica reiniciar la simulación.

Forwarding

Forwarding, Adelantamiento o Cortocircuito

- Cuando no existe Forwarding, una instrucción debe tener todos sus operandos (registros) disponibles en la etapa ID
- Cuando esta activo los registros pueden aelantarse una vez calculados o leidos a la etapa que los necesita. No siempre esa etapa es ID.
- Que instrucciones pueden adelantar un operando:
  - Si la instrucción es de lectura de memoria, puede adelantar el operando luego de ejecutar la etapa MEM.
  - Si es una instrucción aritmetico / lógica, luego de ejecutada la etapa EX.
- Si el operando se podia adelantar en EX, podrá también adelantarlo en MEM si una instrucción lo requiere.

Forwarding

Forwarding, Adelantamiento o Cortocircuito

- Cuando se requiere un operando:
  - Si la instrucción es de escritura de memoria, el operando a escribir, se necesita en MEM.
  - Si el operando se necesita para una operación aritmetico / lógica, se necesita en la etapa EX.
  - Si el operando se necesita para una instrucción de salto condicional, se necesita en la etapa ID.

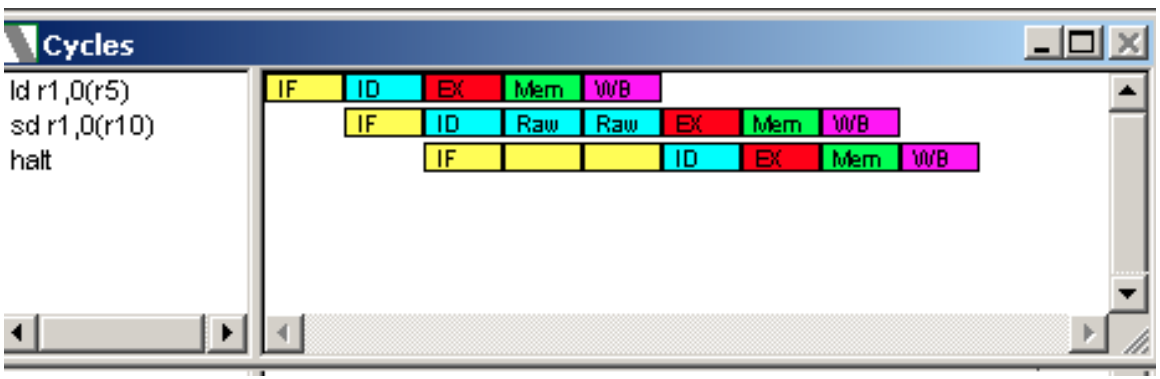
Forwarding

Forwarding, Adelantamiento o Cortocircuito

- Situaciones donde pueden ocurrir forwarding
  - Lectura seguida de Escritura
  - Lectura seguida de Aritmetico / Lógica
  - Lectura seguida de Salto Condicional
  - Aritmetica / Lógica seguida de Escritura
  - Aritmetica / Lógica seguida de Aritmetico / Logica
  - Aritmetica / Lógica seguida de Salto Condicional
  - Lectura seguida de Lectura
  - Aritmetica / Lógica seguida de Lectura

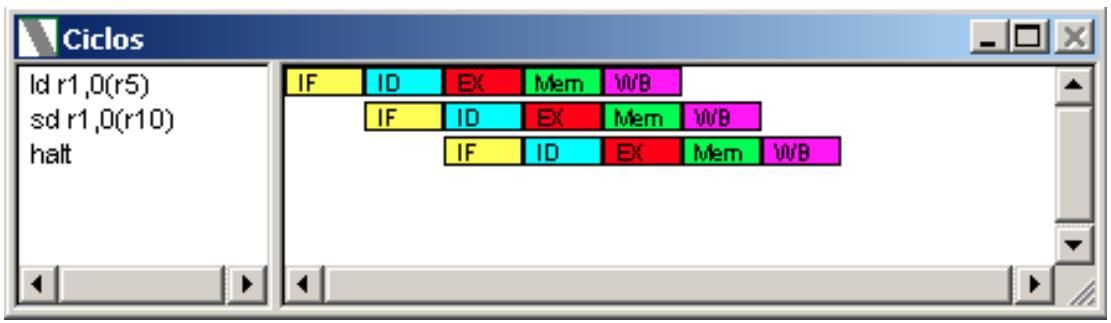
Forwarding - Lectura seguida de escritura

- Lectura seguida de Escritura
  - El operando a escribir en memoria viene de una instrucción de lectura anterior.
  - La instrucción *sd* debe esperar a que la instrucción *ld* se complete, esperando en *ID*



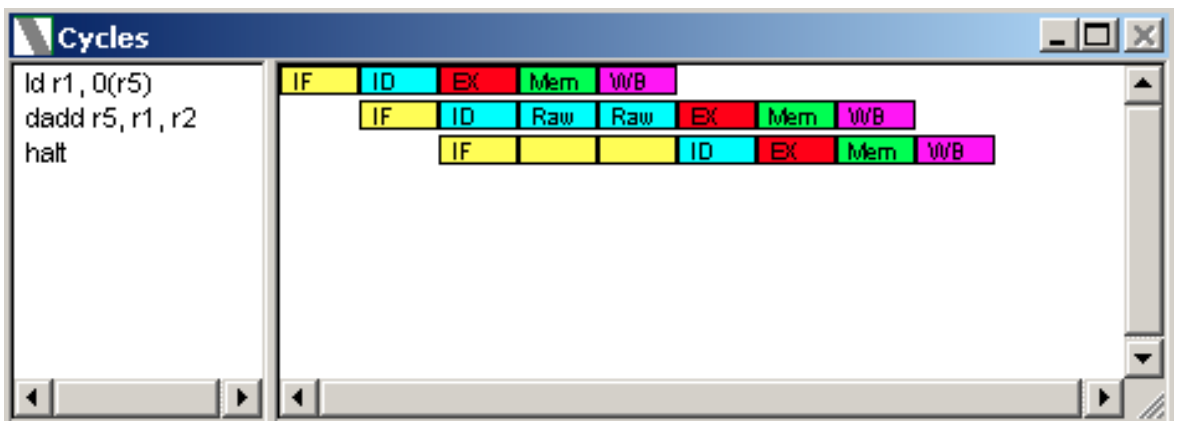
Forwarding - Lectura seguida de escritura

- Lectura seguida de Escritura
  - Con el adelantamiento activo, cuando la instrucción *ld* pasa la etapa *MEM*, ya puede adelantar el operando
  - La instrucción *sd* necesita el registro r1 en la etapa de *MEM*
  - Por lo tanto ya no hay atascos RAW.



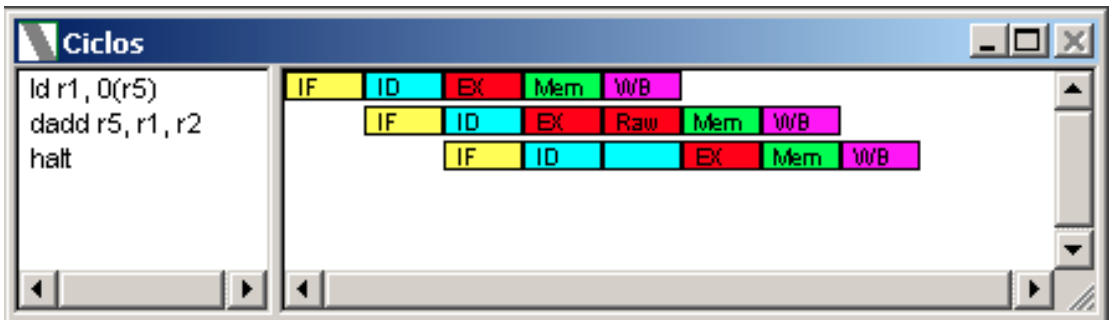
Forwarding - Lectura seguida de Aritmetico / Logica

- Lectura seguida de Aritmetico / Lógica
  - Uno de los operandos de una instrucción aritmetico lógico proviene de una lectura de memoria
  - La instrucción *daddi* debe esperar a que la instrucción *ld* se complete, esperando en *ID*



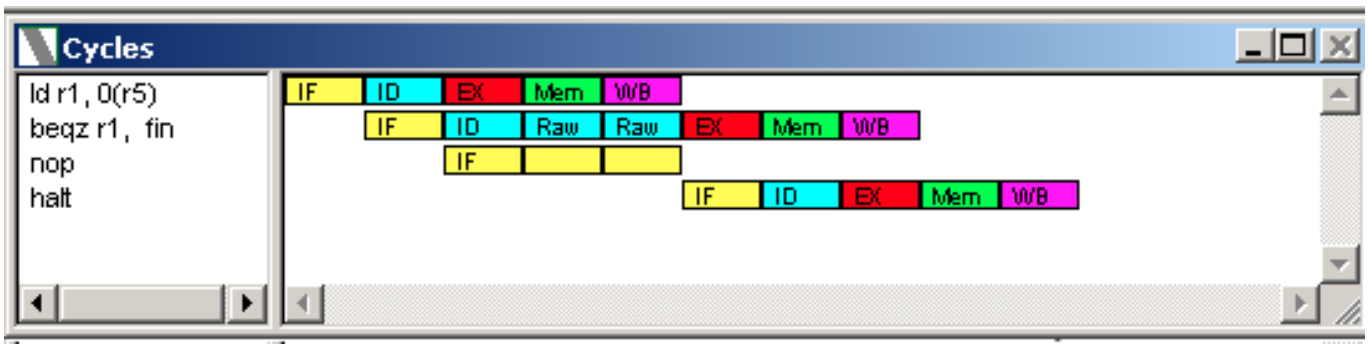
Forwarding - Lectura seguida de Aritmetico / Logica

- Lectura seguida de Aritmetico / Lógica
  - Con el adelantamiento activo, cuando la instrucción *ld* pasa la etapa *MEM*, ya puede adelantar el operando
  - La instrucción *daddi* necesita el registro r1 en la etapa de *EX*
  - Como *daddi* debe esperar en *EX* a que *ld* pase la etapa *MEM*, se produce un solo atasco RAW.



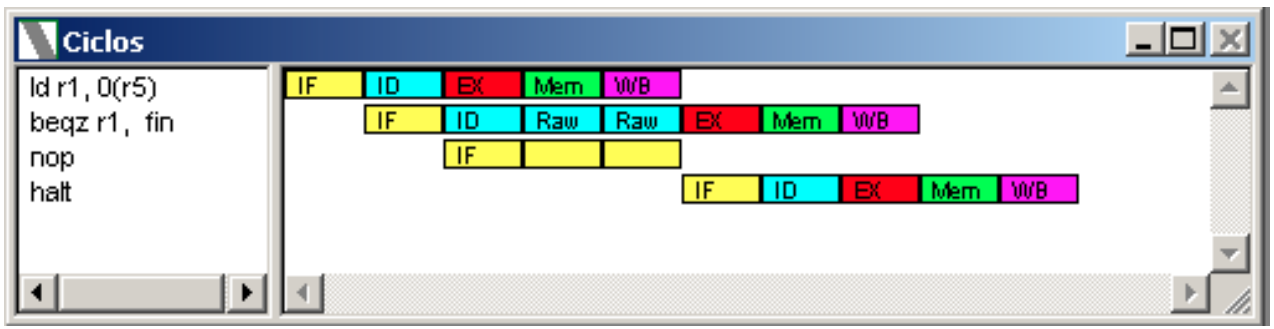
Forwarding - Lectura seguida de Salto Condicional

- Lectura seguida de Salto Condicional
  - Uno de los operandos de una instrucción de salto condicional proviene de una lectura de memoria
  - La instrucción *beqz* debe esperar a que la instrucción *ld* se complete, esperando en *ID*



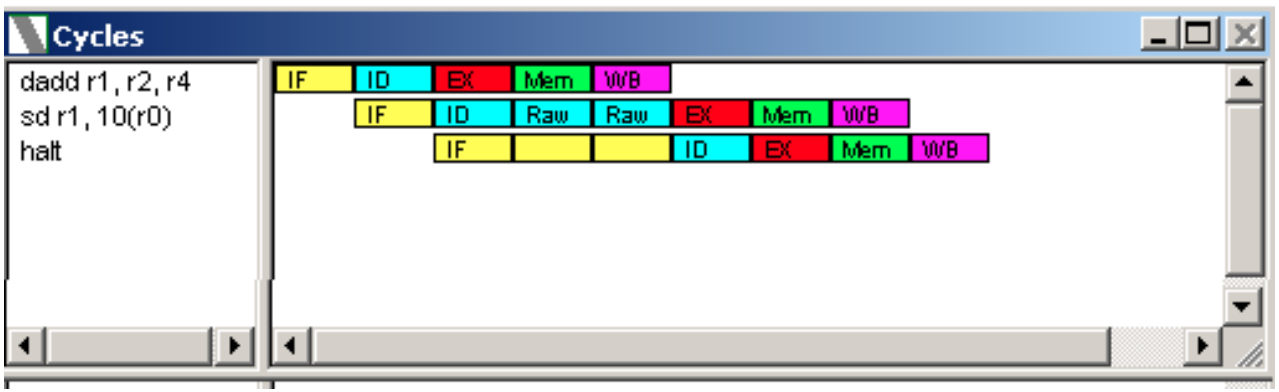
Forwarding - Lectura seguida de Salto Condicional

- Lectura seguida de Salto Condicional
  - Con el adelantamiento activo, cuando la instrucción *ld* pasa a la etapa *MEM*, ya puede adelantar el operando
  - La instrucción *beqz* necesita el registro r1 en la etapa de *ID*
  - Por lo tanto esta situación no cambia



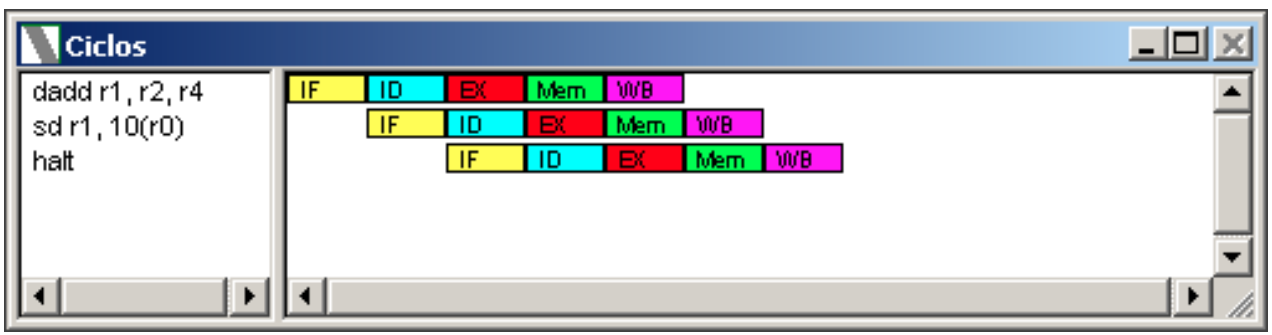
Forwarding - Aritmetica / Lógica seguida de Escritura

- Aritmetica / Lógica seguida de Escritura
  - El operando a escribir en memoria viene de una instrucción aritmetico lógica anterior.
  - La instrucción *sd* debe esperar a que la instrucción *daddi* se complete, esperando en *ID*



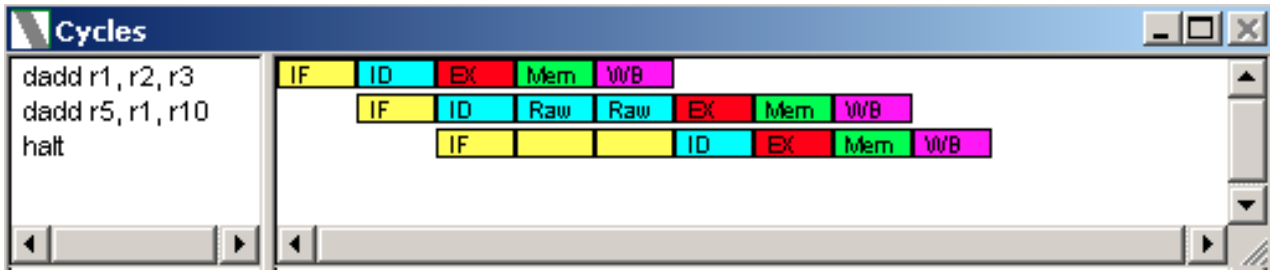
Forwarding - Aritmetica / Lógica seguida de Escritura

- Aritmetica / Lógica seguida de Escritura
  - Con el adelantamiento activo, cuando la instrucción *daddi* pasa la etapa *EX*, ya puede adelantar el operando
  - La instrucción *sd* necesita el registro r1 en la etapa de *MEM*
  - Por lo tanto ya no hay atascos RAW.



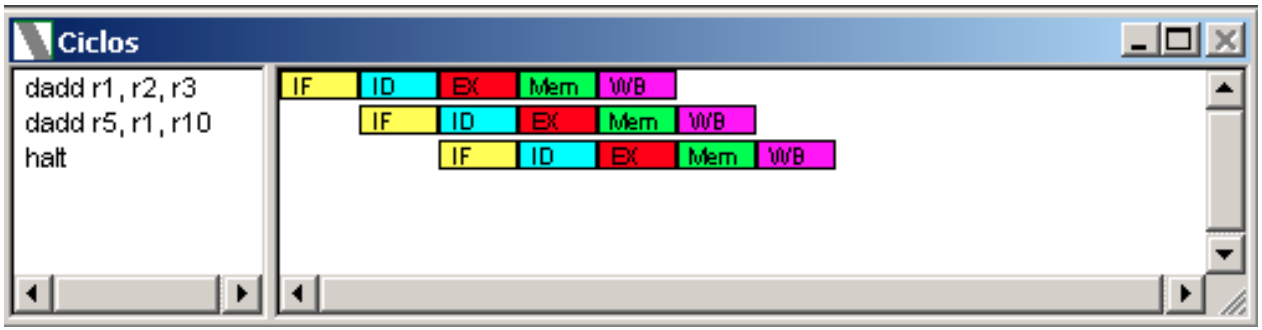
Forwarding - Aritmetica / Lógica seguida de Aritmetico / Logica

- Aritmetica / Lógica seguida de Aritmetico / Logica
  - Un operando de una instrucción aritmetico lógica viene de una instrucción aritmetico lógica anterior.
  - La segunda instrucción *daddi* debe esperar a que la primer instrucción *daddi* se complete, esperando en *ID*



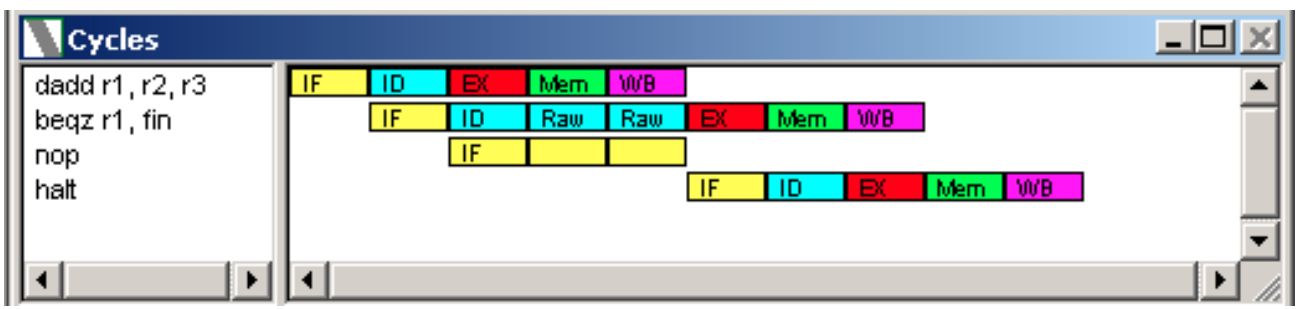
Forwarding - Aritmetica / Lógica seguida de Aritmetico / Logica

- Aritmetica / Lógica seguida de Aritmetico / Logica
  - Con el adelantamiento activo, cuando la primer instrucción *daddi* pasa la etapa *EX*, ya puede adelantar el operando
  - La segunda instrucción *daddi* necesita el registro r1 en la etapa de *EX*
  - Por lo tanto ya no hay atascos RAW.



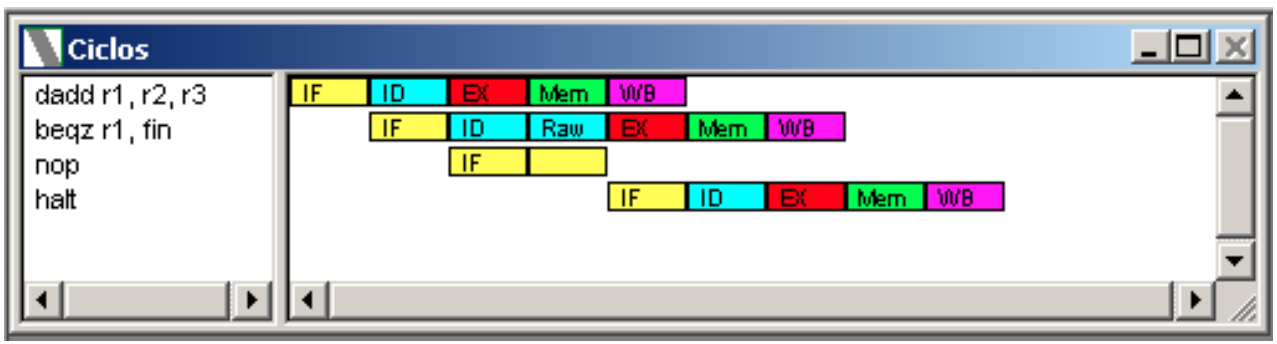
Forwarding - Aritmetica / Lógica seguida de Salto Condicional

- Aritmetica / Lógica seguida de Salto Condicional
  - Un operando de una instrucción de salto condicional viene de una instrucción aritmetico lógica anterior.
  - La instrucción *beqz* debe esperar a que la instrucción *daddi* se complete, esperando en *ID*



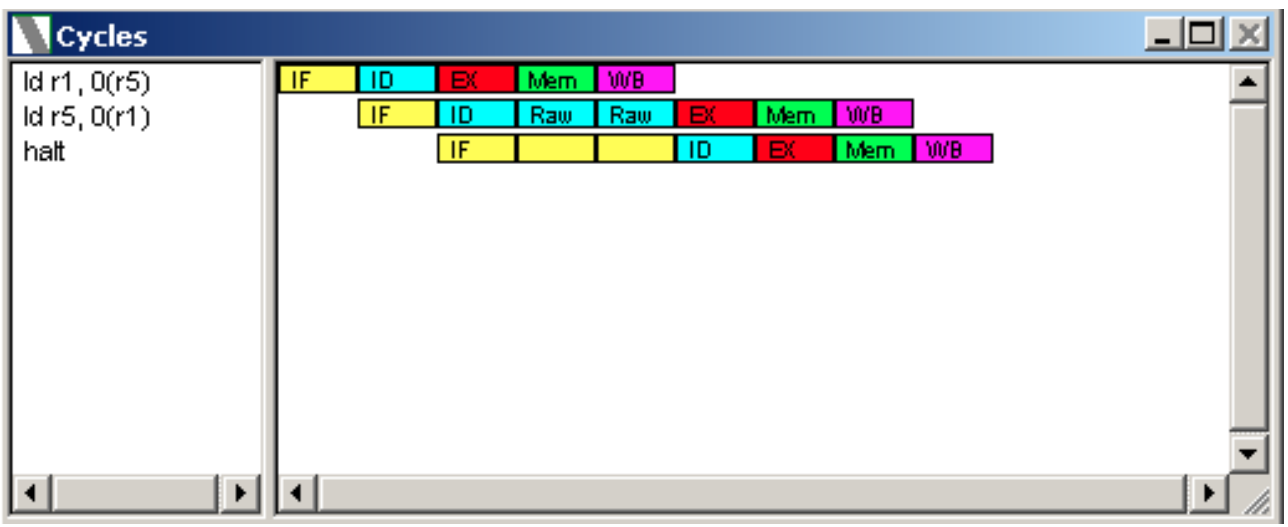
Forwarding - Aritmetica / Lógica seguida de Salto Condicional

- Aritmetica / Lógica seguida de Salto Condicional
  - Con el adelantamiento activo, cuando la instrucción *daddi* pasa la etapa *EX*, ya puede adelantar el operando
  - La instrucción *beqz* necesita el registro r1 en la etapa de *ID*
  - Como *beqz* debe esperar en *ID* a que *daddi* pase la etapa *EX*, se produce un solo atasco RAW.



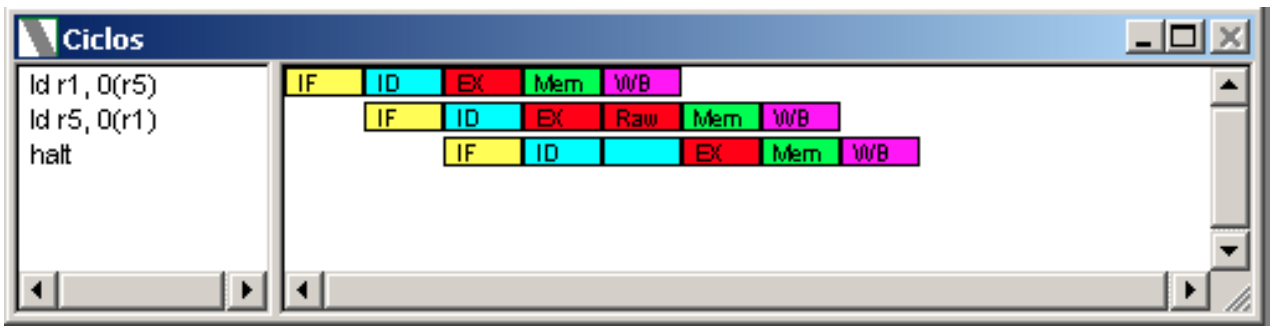
Forwarding - Lectura seguida de Lectura

- Lectura seguida de Lectura
  - El operando que calcula el desplazamiento en memoria de una escritura viene de una instrucción de lectura anterior.
  - La segunda instrucción *ld* debe esperar a que la primer instrucción *ld* se complete, esperando en *ID*



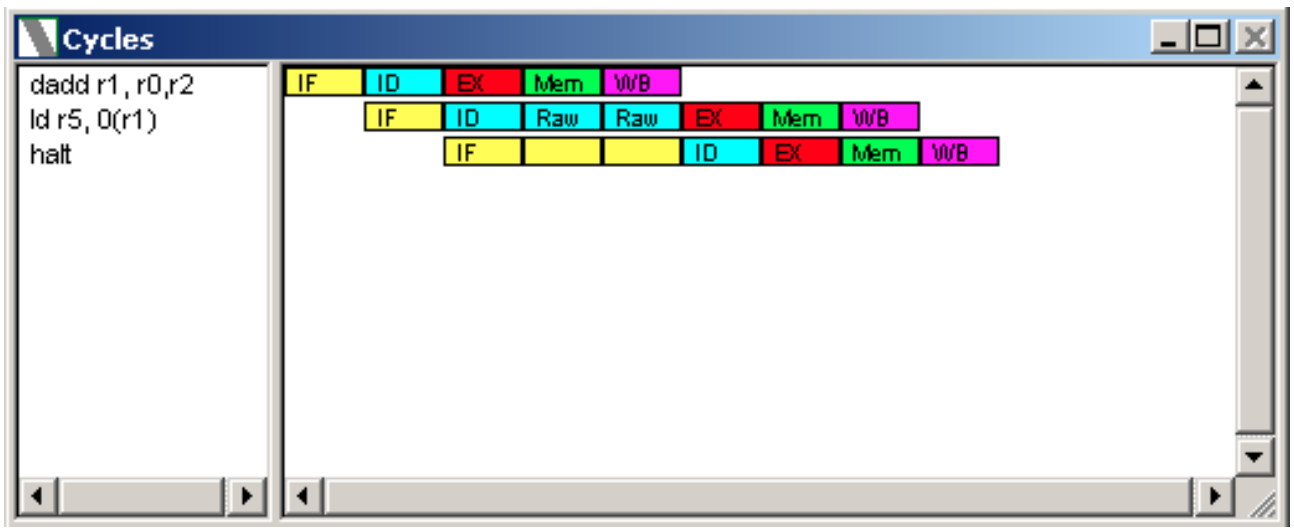
Forwarding - Lectura seguida de Lectura

- Lectura seguida de Lectura
  - Con el adelantamiento activo, cuando la primer instrucción *ld* pasa la etapa *MEM*, ya puede adelantar el operando
  - La segunda instrucción *ld* necesita acceder a r1 en la etapa *EX*
  - Como la segunda instrucción *ld* debe esperar en *EX* a que la primer instrucción *ld* pase la etapa *MEM*, se produce un solo atasco RAW.



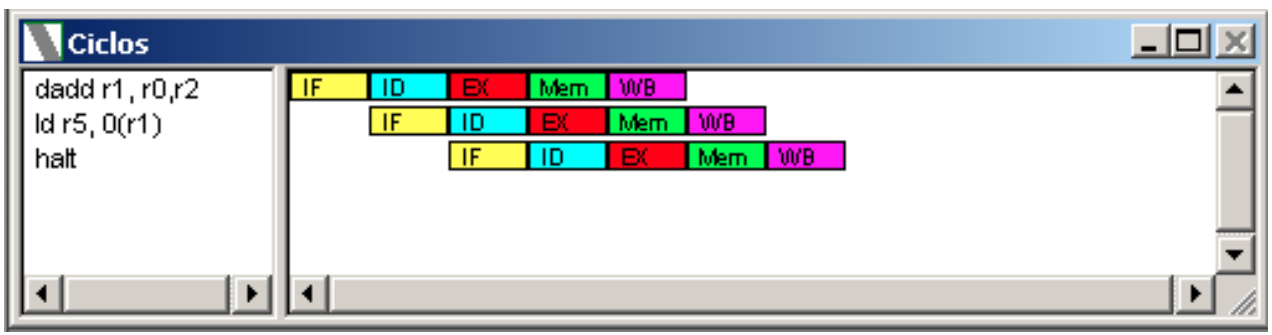
Forwarding - Aritmetica / Lógica seguida de Lectura

- Aritmetica / Lógica seguida de Lectura
  - El operando que calcula el desplazamiento en memoria de una escritura viene de una instrucción aritmetico lógica anterior.
  - La instrucción *ld* debe esperar a que la instrucción *daddi* se complete, esperando en *ID*



Forwarding - Aritmetica / Lógica seguida de Lectura

- Aritmetica / Lógica seguida de Lectura
  - Con el adelantamiento activo, cuando la instrucción *daddi* pasa la etapa *EX*, ya puede adelantar el operando
  - La instrucción *ld* necesita acceder a r1 en la etapa *EX*
  - Por lo tanto ya no hay atascos RAW.



General

Existe una Penalización por salto, ya que se empieza a analizar la instrucción inmediata siguiente, mientras estamos decodificando el salto. Las instrución de salto puede ser:

- Incondicional: La dirección de destino se debe determinar lo más pronto posible, dentro del cauce, para reducir la penalización.
- Condicional: Introduce riesgo adicional por la dependencia entre la condición de salto y el resultado de una instrucción previa.

Deberiamos encontrar alguna manera de saber cual es la próxima instrucción lo antes posible.

Branch-Target-Buffer

Podemos ir guardando un historial de saltos para poder "predecir" si un salto se produce o no.

- Cada vez que se ejecuta una instrucción de salto se guarda un registro si el salto se realizo o no (un registro para cada instrucción de salto). Ejemplo:
  - Tengo un contador, inicialmente en 0. Sumo 1 por salto realizado, resto 1 si el salto no se realiza.
  - La próxima vez que pasé por esa instrucción puedo *predecir* si el salto se realizará o no.
  - Si el contador es positivo asumo se realizara el salto, la próxima instrucción será la que resulte de ejecutar el salto
  - Si el contador fuera negativo asumo no se realiza el salto, la próxima instrucción será la siguiente.
- Que se hace la primera vez que se evalua un salto depende de la arquitectura.

Branch-Target-Buffer en el Simulador

- El simulador WinMIPS64 permite activar o desactivar la predicción de Saltos
- La predicción de saltos usado en el simulador funciona de la siguiente manera:
  - Si es la primera vez que se realiza el salto, se predice que el salto no va a ocurrir.
  - Si la instrucción se ejecuto por lo menos una vez, se asume que va a ocurrir lo último que ocurrió.
  - Si el resultado de la predicción cambio, se actualiza ese estado.
- Si una predicción no ocurre, hay una penalidad de un ciclo perdido adicional.

Branch-Target-Buffer en el Simulador

- Si un salto se predice que va a ocurrir y no ocurre. La penalidad son 2 atascos Branch Misprediction.
- Si un salto se predice que no va a ocurrir y ocurre. La penalidad son 2 atascos Branch Taken.
- Los 2 atascos corresponden a:
  - 1 Ciclo para la actualización de la tabla BTB
  - 1 Ciclo para la carga de la nueva instrucción (como ocurria sin BTB)

Branch-Target-Buffer en el Simulador

- Si un salto se predice que va a ocurrir, podemos ver un simbolo después de la dirección de memoria en la ventana del programa

```
0000 dc010058          ld r1, long(r0)
0004 dc020050          ld r2, num(r0)
0008 0000182c          dadd r3, r0, r0
000c 0000502c          dadd r10, r0, r0
0010 dc640000 loop:    ld r4, tabla(r3)
0014 10440004          beq r4, r2, listo
0018 6021ffff          daddi r1, r1, -1
001c 60630008          daddi r3, r3, 8
0020 1c01fffb          bnez r1, loop
0024 08000001          j fin
0028 600a0001 listo:   daddi r10, r0, 1
002c 04000000 fin:     halt
```

Delay Slot

Salto retardado o de relleno de ranura de retardo

- Es un método alternativo de atacar el problema de los atascos por dependencia de Control
- El problema es la predicción de la siguiente instrucción luego de un salto.
- Delay Slot ofrece como solución que la siguiente instrucción después de un salto *SIEMPRE* se ejecute.
- Esto elimina el problema de la predicción de instrucciones...
- ... pero hay que tener en cuenta a la hora de programar.

Delay Slot

¿Es lo mismo con o sin *Delay Slot*?

```
        .data
A:    .word 2
        .code
        ld     r1, A(r0)
        daddi  r2, r0, 3
        dadd   r3, r0, r0
salto:  dadd   r3, r3, r1
        daddi  r2, r2, -1
        bnez   r2, salto
        halt
```

Delay Slot - Solución 1

Podemos "arreglar" nuestro programa poniendo un *NOP*, luego del salto.

```
        .data
A:    .word 2
        .code
        ld     r1, A(r0)
        daddi  r2, r0, 3
        dadd   r3, r0, r0
salto:  dadd   r3, r3, r1
        daddi  r2, r2, -1
        bnez   r2, salto
        nop
        halt
```

Delay Slot - Solución 2

O podriamos reordenar las instrucciones...

```
        .data
A:    .word 2
        .code
        ld     r1, A(r0)
        daddi  r2, r0, 3
        dadd   r3, r0, r0
salto:  daddi   r2, r2, -1
        bnez   r2, salto
        dadd r3, r3, r1
        halt
```

Delay Slot en el Simulador

- El simulador WinMIPS64 permite activar o desactivar el Delay Slot
- No se pueden tener activos Delay-Slot y Branch-Target-Buffer al mismo tiempo
- Recordar que cualquier cambio en estas opciones involucra un reinicio en la simulación