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

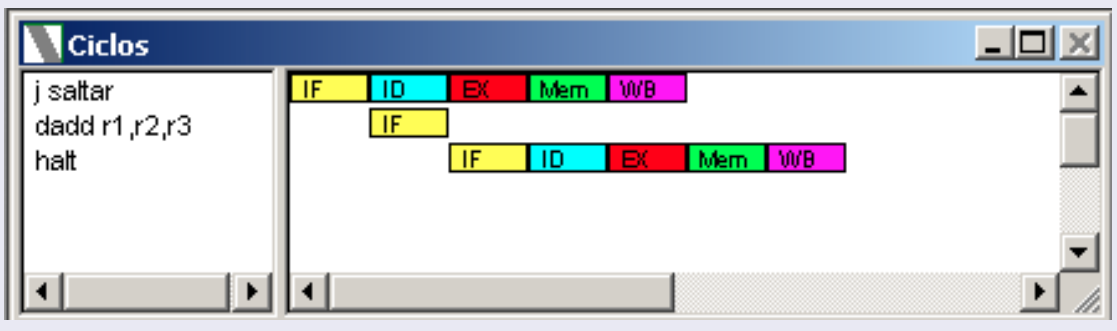
General

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



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