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 MSX88 *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.

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

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.

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

WAR

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

```
.text  ; Activar Forwarding

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

WAW

Una instrucción escribe un dato después que otra posterior

```
.text

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

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

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

Soluciones a riesgos estructurales

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

- Duplicación de recursos hardware
  - Sumadores o restadores 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
  - Hardware
    - Adelantamiento de Operandos (Forwarding)
  - Software
    - Podemos agregar instrucciones *NOP*, o reordenar las instrucciones

Agregar Instrucciones NOP

Cambiemos el ejemplo visto anteriormente

```
.text
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
.text
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
.text
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 (X_{i+1}) sin esperar a la escritura (M_i o W_i).
- 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.

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. 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.
- La predicción puede fallar.
- Que se hace la primera vez que se evalua un salto depende de la arquitectura.
- Cada instrucción de salto tiene su propio control.

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.

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
        .text
        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
        .text
        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
        .text
        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