Pipeline o Segmentación Encausada

Material Elaborado por el Profesor Ricardo González
A partir de Materiales de las Profesoras
Angela Di Serio
Patterson David, Hennessy John “Organización y Diseño de Computadores”McGraw Hill. 1995

Pipeline

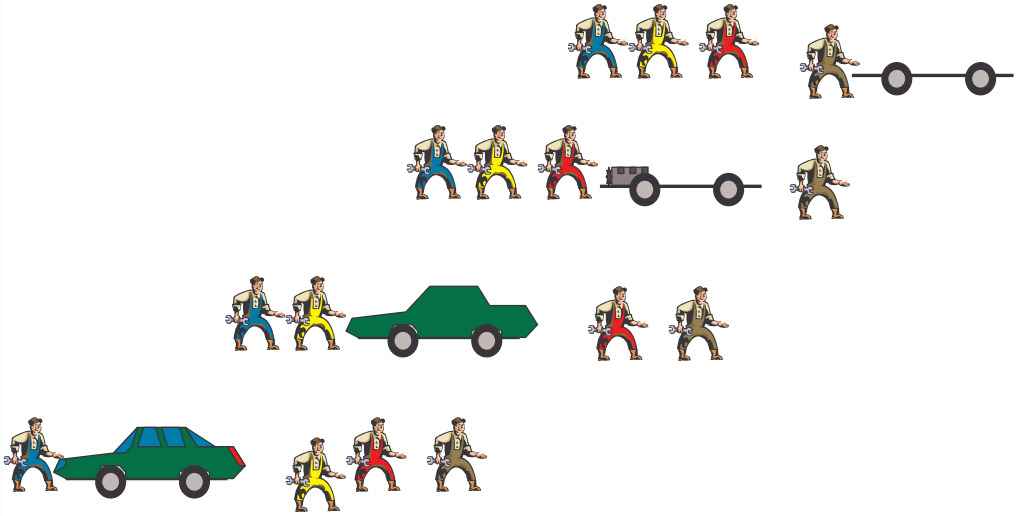
La segmentación de instrucciones es similar al uso de una cadena de montaje en una fábrica de manufactura.

- En las cadenas de montaje, el producto pasa a través de varias etapas de producción antes de tener el producto terminado.
- Cada etapa o segmento de la cadena está especializada en un área específica de la línea de producción y lleva a cabo siempre la misma actividad.

Esta tecnología es aplicada en el diseño de procesadores eficientes.

A estos procesadores se les conoce como *pipeline processors o procesadores con segmentación encausada.*

Manufactura artesanal



Cadena de Montaje

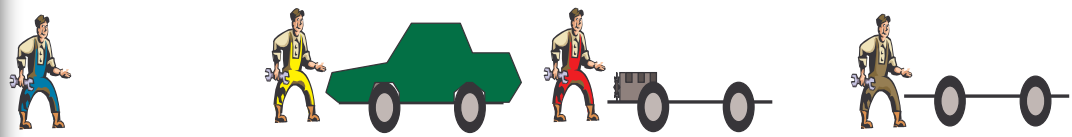
Tiempo T1

Cadena de Montaje

Tiempo T2

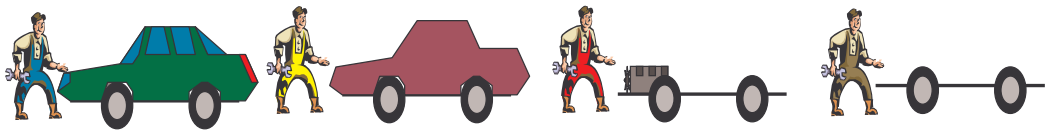
Cadena de Montaje

Tiempo T3

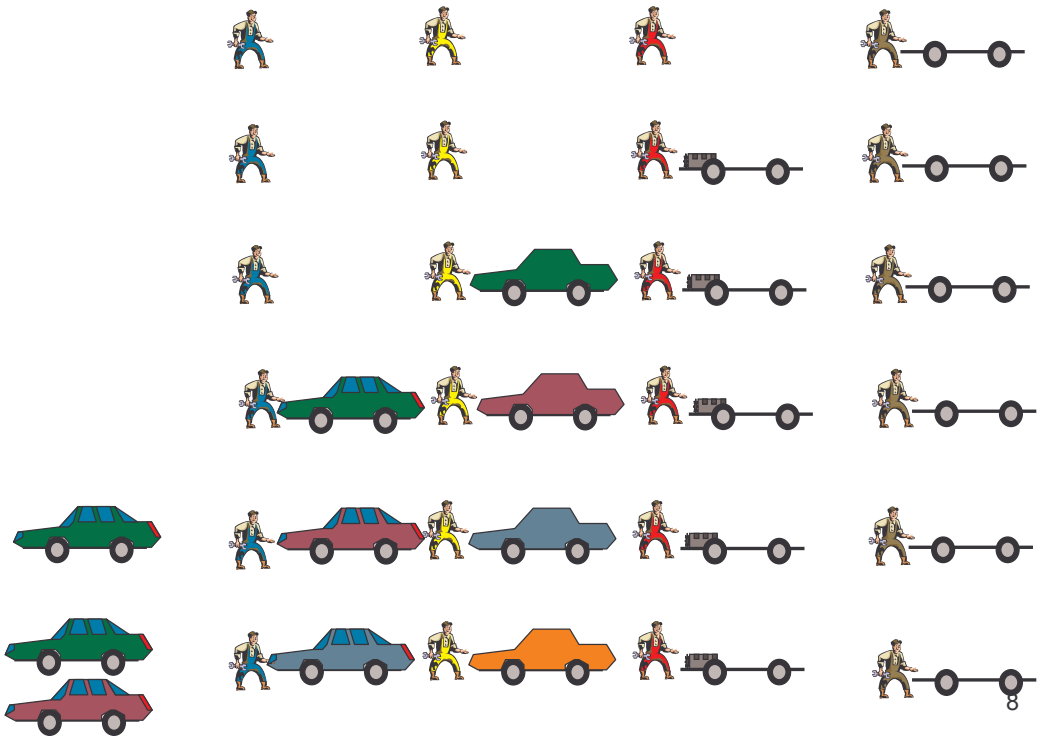


Cadena de Montaje

Tiempo T4



Cadena de Montaje



Pipeline

- Un *pipeline processor* está compuesto por una lista de segmentos lineales y secuenciales en donde cada segmento lleva a cabo una tarea o un grupo de tareas computacionales. Puede ser representado gráficamente en dos dimensiones, en donde en el eje vertical encontramos los segmentos que componen el pipeline y en el segmento horizontal representamos el tiempo

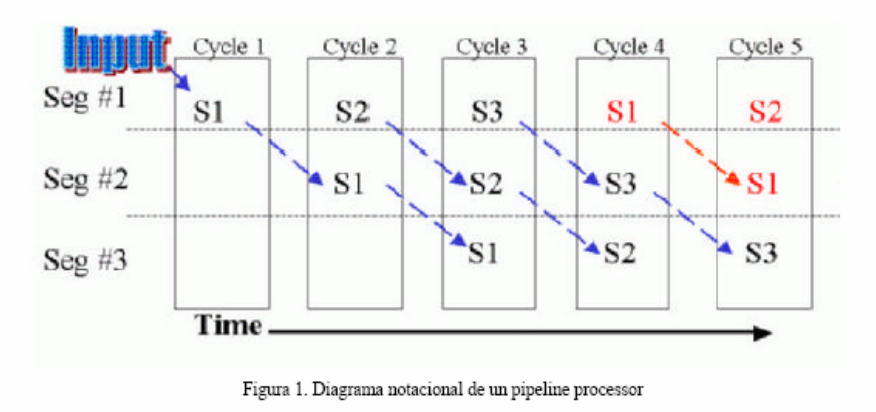


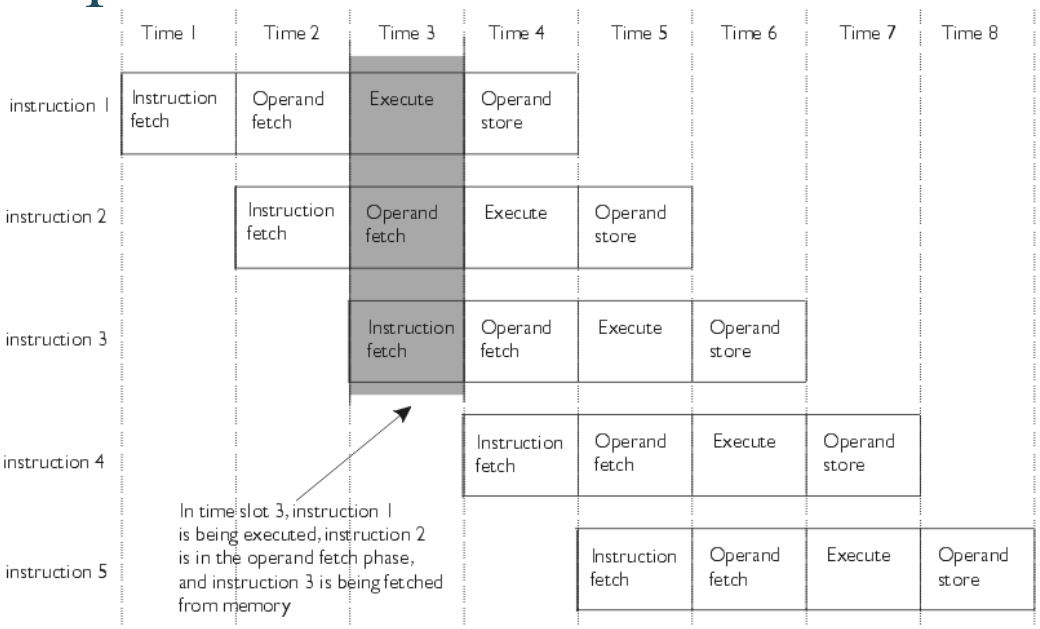
Figura 1. Diagrama notacional de un pipeline processor

Pipeline

superpipelining.mov

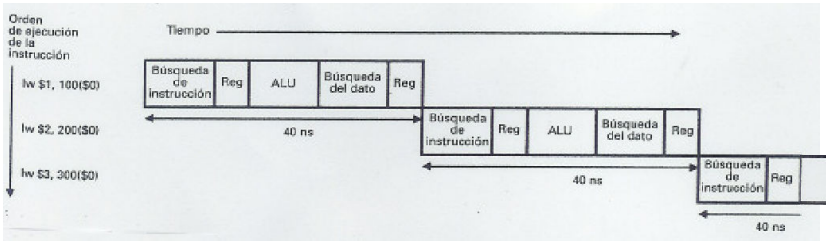
(Hacer docle click sobre el nombre del archivo para reproducirlo)

Pipeline

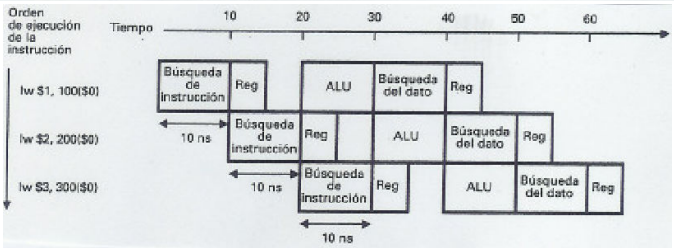


PipeLine

Tipo de instrucción	Memoria de instrucciones	Lectura de registro	Operación ALU	Memoria de datos	Escritura de registro	Tiempo total
Cargar palabra (lw)	10 ns	5 ns	10 ns	10 ns	5 ns	40 ns
Almacenar palabra(sw)	10 ns	5 ns	10 ns	10 ns		35 ns
Formato R (add, sub, and, or)	10 ns	5 ns	10 ns		5 ns	30 ns
Saltar (beq)	10 ns	5 ns	10 ns			25 ns



Tiempo de ejecución1 = 120ns



Tiempo de ejecución2 = 70ns

PipeLine

Tiempo de ejecución1 = 120ns
Tiempo de ejecución2 = 70ns
SpeedUp en tres instrucciones = $\frac{\text{Tiempo Secuencial}}{\text{Tiempo Paralelo}}$ = 1.71

Que ocurriría si procesamos 1.003 instrucciones en lugar de 3
Cada nueva instrucción añade 10ns al tiempo de ejecución

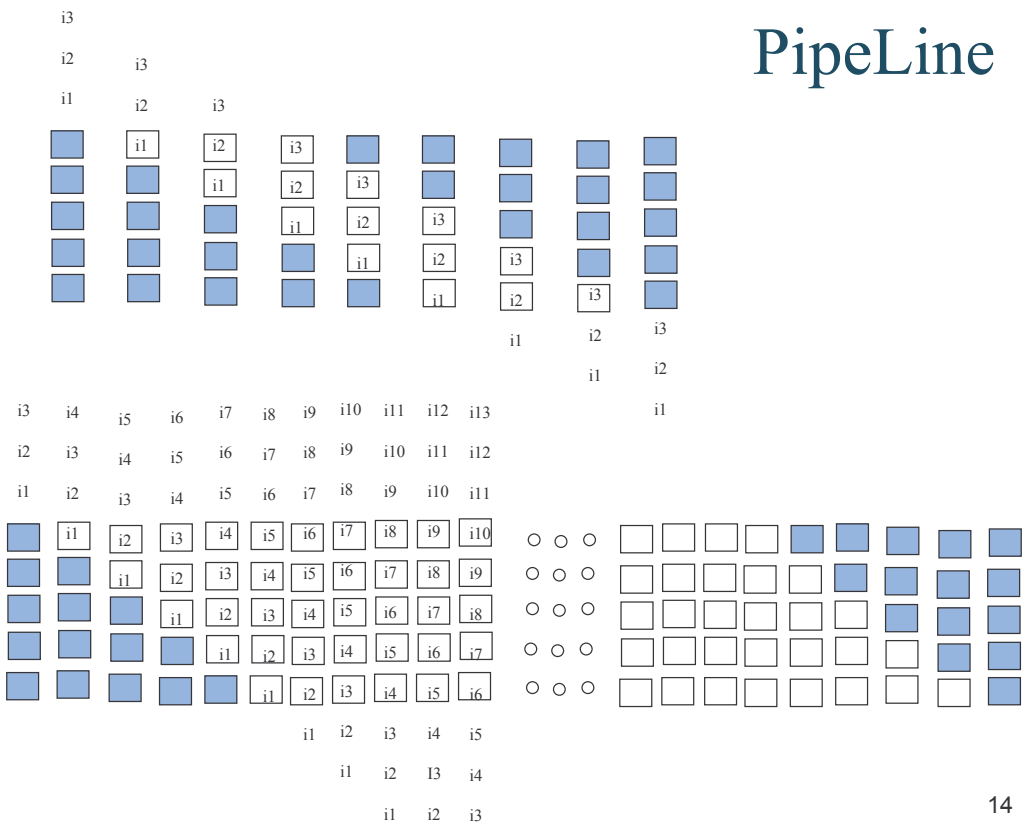
Tiempo de ejecución segmentado3 = 1000 x 10 ns + 70 ns = 10070 ns

Tiempo de ejecución no segmentado3 = 1000 x 40 ns + 120 ns = 40120 ns

SpeedUp en 1003 instrucciones = $\frac{40120}{10070}$ = 3.98

Tiempo de ejecución2 = 70ns

PipeLine

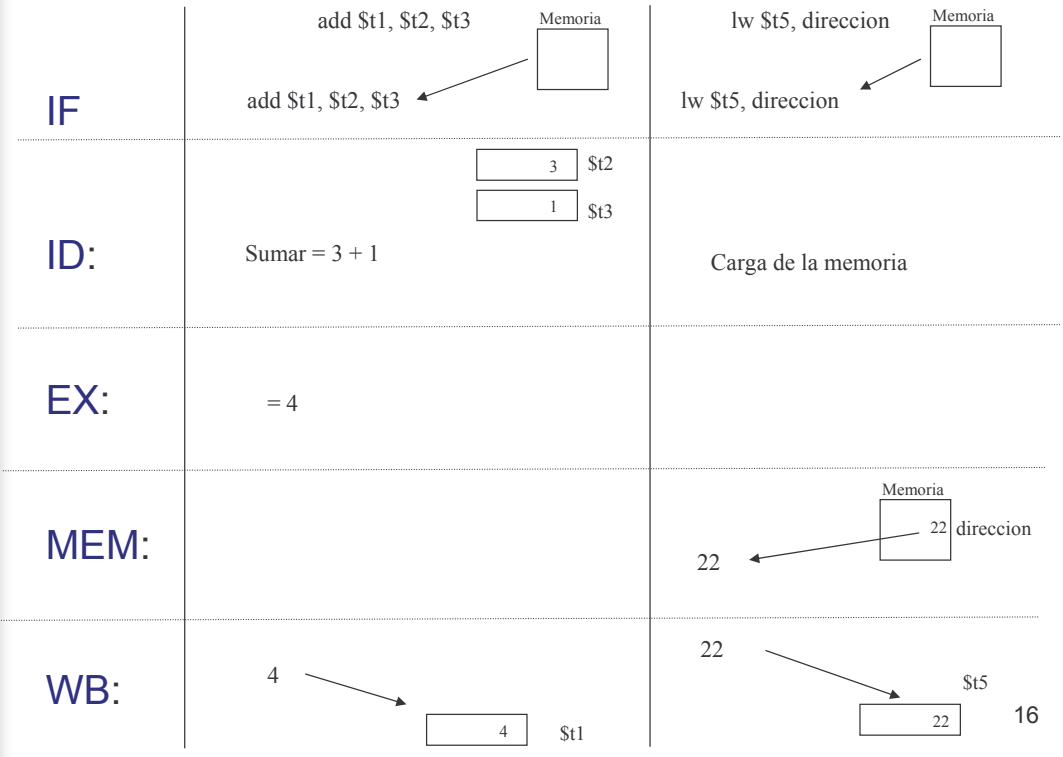


Etapas del Pipe para MIPS

Para poder tener una aceleración mayor, el *pipeline* debe tener más etapas. Consideremos la siguiente división en 5 etapas que presenta la arquitectura MIPS:

- IF: Instruction Fetch from memory.
- ID: Instruction Decode and read Registers.
- EX: Execute operation or calculate address.
- MEM: Access an operand in Data Memory.
- WB: Write Back result into a register.

Etapas del Pipe para MIPS



Riesgos en el *pipeline (Hazards)*

Los procesadores con *pipeline* presentan una serie de problemas conocidos como *hazards*, y que pueden ser de tres tipos:

- Riesgos Estructurales: ocurren cuando diversas instrucciones presentan conflictos al tratar de acceder a la misma pieza de hardware. Este tipo de problema puede ser aliviado teniendo hardware redundante, que evite estas colisiones. También se pueden agregar ciertas paradas (*stall*) en el *pipeline* o aplicar reordenamiento de instrucciones para evitar este tipo de riesgo.
- Riesgos de Datos: ocurren cuando una instrucción depende del resultado de una instrucción previa, que aún está en el *pipeline,* y cuyo resultado aún no ha sido calculado. La solución más fácil es introducir paradas en la secuencia de ejecución, pero esto reduce la eficiencia del *pipeline*.
- Riesgos de Control: son el producto de las instrucciones de salto que necesitan tomar una decisión, basada en un resultado de una instrucción, mientras se están ejecutando otras.

Riesgos Estructurales o *Structural Hazards*

Ocurre cuando no se dispone de suficiente hardware para soportar ciertos cómputos dentro de un segmento particular del *pipeline*. Por ejemplo en la figura 6 se puede observar que en el ciclo de reloj 5 (CC5) se debe llevar a cabo una escritura un registro y una lectura del mismo registros. Esto puede resolverse duplicando el registro, pero puede llevarnos a problemas de inconsistencia.

Otra alternativa es modificar el hardware existente para que pueda soportar operaciones de lectura y escritura concurrentes. Se puede modificar los registros de forma tal que la escritura de registros se lleva a cabo en la primera mitad del ciclo de reloj y la lectura de registros se efectúa en la segunda mitad del ciclo de reloj.

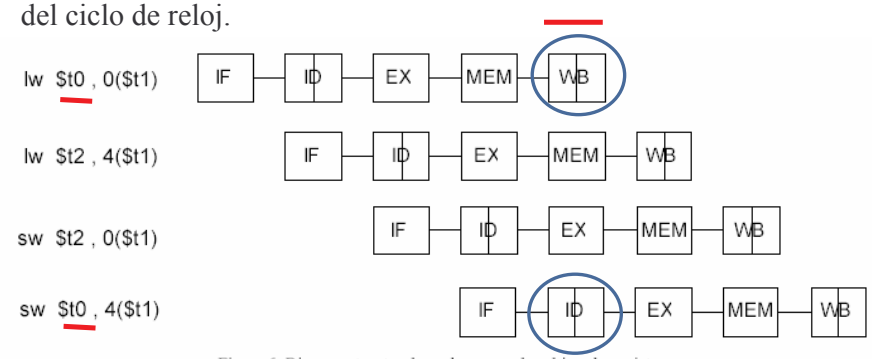


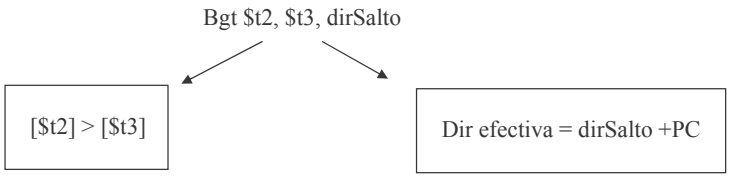
Figura 6. Riesgo estructural en el acceso al archivo de registros

Riesgos Estructurales o *Structural Hazards*

Otro riesgo estructural puede ocurrir durante una instrucción de salto condicional. Si no disponemos de dos ALU en el segmento EX.

Para este tipo de instrucción es necesario calcular la dirección de salto, pues en la instrucción se dispone del desplazamiento necesario para llegar al destino y también es necesario calcular la diferencia entre los operandos fuentes para determinar si se cumple o no la condición.

EX:



Riesgos de Datos o *Data Hazards*

Un *data hazard* ocurre cuando la instrucción requiere del resultado de una instrucción previa y el resultado aún no ha sido calculado y escrito en los registros. En la figura 7 se presenta un ejemplo de riesgo de datos.

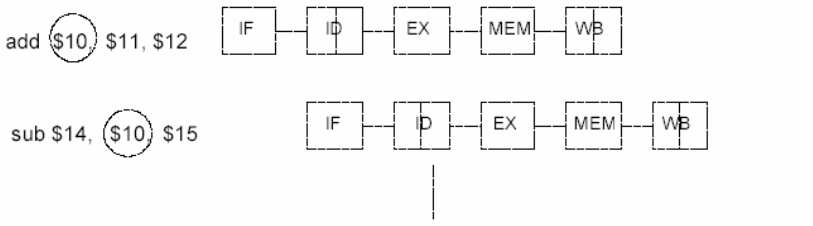


Figura 7. Riesgo de Datos en el registro $10

Riesgos de Datos o *Data Hazards*

Hay diversas formas para solventar este problema:

- *Forwarding*: se adicionan circuitos especiales en el *pipeline* de forma que el valor deseado es transmitido/adelantado al segmento del *pipeline* que lo necesita (Figura 8).

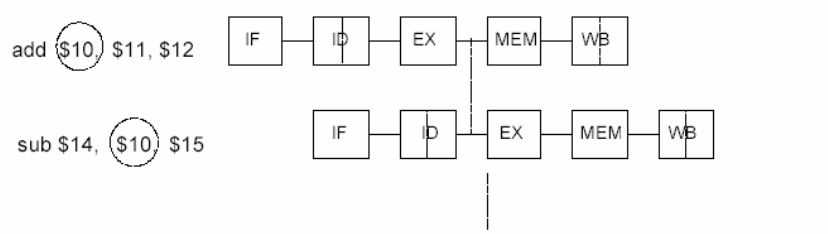


Figura 8. Forwarding para evitar riesgo de datos en el registro $10

Riesgos de Datos o *Data Hazards*

- Inserción de paradas (*Stall insertion*): es posible insertar una o más paradas en el *pipeline* (no-op) que retarden la ejecución de la instrucción actual, hasta que el dato requerido sea escrito en el archivo de registros (Figura 9).

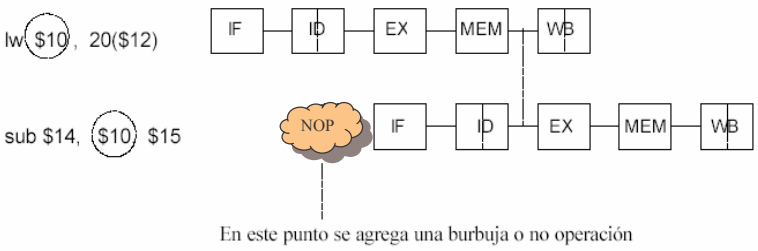


Figura 9. Inserción de no-op para evitar riesgo de datos en el registro $10

Riesgos de Datos o *Data Hazards*

Reordenamiento de código: en este caso, el compilador reordena instrucciones en el código fuente o el ensamblador reordena el código objeto, de forma que se coloquen una o más instrucciones entre las instrucciones que presentan dependencia de datos. Para esto se necesitan compiladores o ensambladores inteligentes.

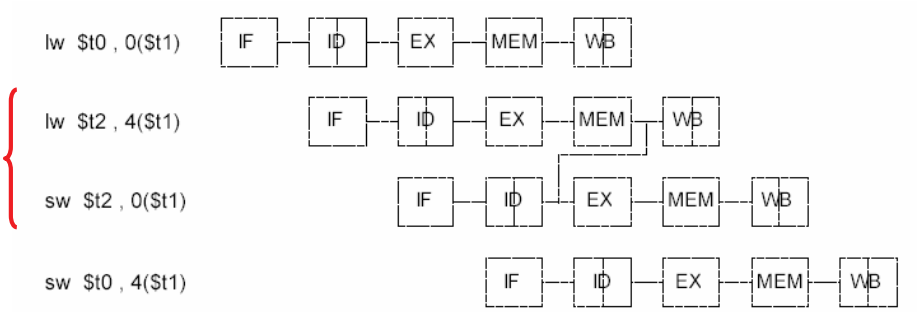


Figura 10. Ejemplo de riesgo de datos

Riesgos de Datos o *Data Hazards*

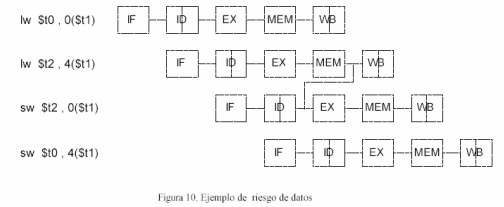


Figura 10. Ejemplo de riesgo de datos

En la tercera instrucción no se dispone del resultado de la segunda y no se puede aplicar adelantamiento del resultado (Figura 10) . Una alternativa es que se incluya una parada o burbuja en el *pipeline* (Figura 11).

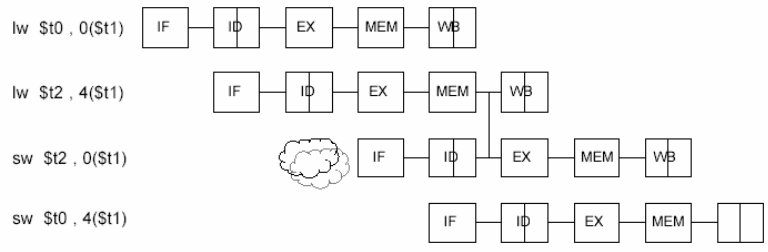


Figura 11. Ejemplo de riesgo de datos con inserción de no-op

Riesgos de Datos o *Data Hazards*

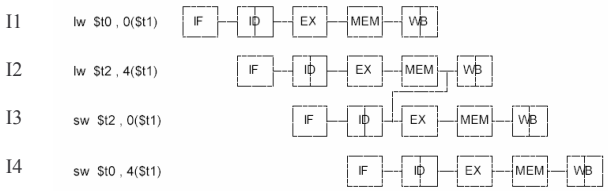


Figura 10. Ejemplo de riesgo de datos

Otra forma más eficiente es cambiando el orden de ciertas instrucciones. En este caso si intercambiamos la instrucción 3 y 4 resolvemos el problema de dependencia de datos (Figura 12).

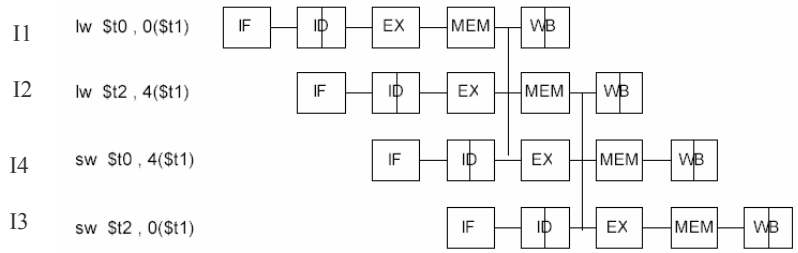
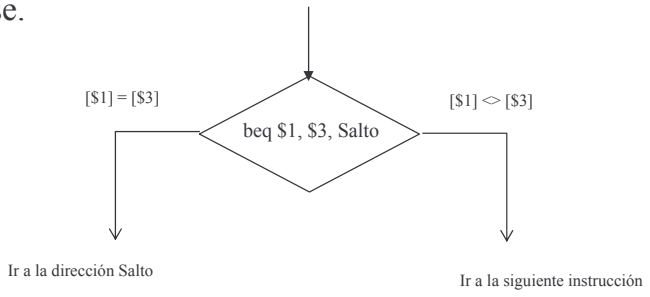


Figura 12. Reordenamiento de instrucciones para evitar un riesgo de datos

Riesgos de Control o Control Hazards

Un riesgo de control se produce cuando es necesario llevar a cabo una decisión basada en el resultado de una instrucción mientras se están ejecutando otras instrucciones. Cuando se ejecuta un salto no se conoce de antemano cuál será la siguiente instrucción que deberá ser ejecutada. Si la condición del salto falla, entonces se debe ejecutar la instrucción inmediata, si la condición del salto se cumple, se debe actualizar el PC con la dirección de la siguiente instrucción que debe ejecutarse.



Riesgos de Control o Control Hazards

Una estrategia para atacar este tipo de problema es asumir que la condición del salto no se cumplirá y por lo tanto la ejecución continuará con la instrucción que se encuentra inmediatamente después de la instrucción de salto. Si la condición del salto se cumplió entonces se deberá desechar del *pipeline* las instrucciones que fueron captadas y la ejecución continua con la instrucción que se encuentra en la dirección de salto (Figura 13).

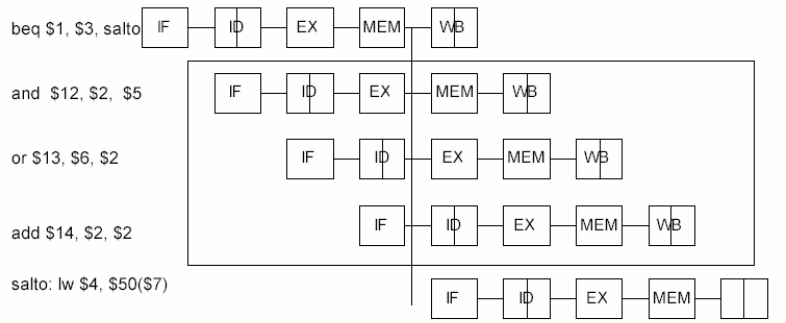


Figura 13. Riesgo de Control con decisión del salto en MEM

En el ejemplo de la figura 13, si la condición del salto se cumple, se deben desechar del *pipeline* las tres instrucciones que entraron en el *pipeline* pues se había asumido que dicha condición no se iba a cumplir.

Riesgos de Control o Control Hazards

Una forma para mejorar el rendimiento de los saltos es reduciendo el costo de no haber tomado el salto. La decisión del salto se toma en el segmento MEM del *pipeline*. En este segmento se tiene un sumador que le adiciona al PC el desplazamiento para llegar a la instrucción destino. Si esta decisión se puede mover a una etapa o segmento anterior entonces sólo una instrucción tendrá que ser sacada del *pipeline* cuando se tome una decisión incorrecta en el caso de los saltos. Muchas implementaciones MIPS mueven la ejecución del *branch* a la etapa ID. Para ello mueven el sumador que usa el *branch* para la etapa ID (Figura 14).

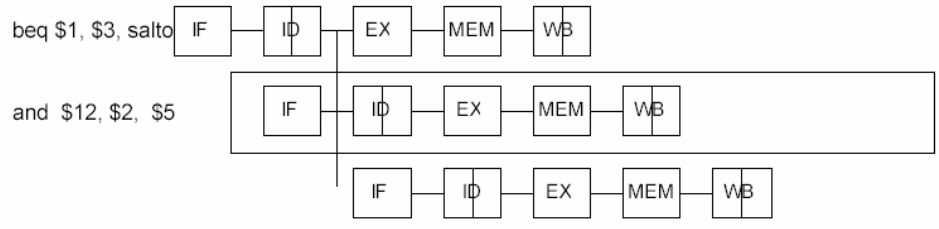


Figura 14. Riesgo de Control con decisión del salto en la etapa ID

Riesgos de Control o Control Hazards

Ejercicio.

Muestre qué sucede en el *pipeline* cuando la evaluación de la condición en el salto indica que debe tomarse el mismo. Asuma que el *pipeline* está optimizado para tomar la decisión en la etapa ID y que además simpre asume que las condiciones de los saltos no se cumplen.

```
        sub $10, $4, $8
        beq $1, $3, X
        and $12, $2, $5
        or $13, $2, $6
        add $14, $4, $2
        slt $15, $6, $7
X:      lw $4, 60($7)
```


Riesgos de Control o Control Hazards

En la figura 15 podemos observar el comportamiento del *pipeline* para el ejercicio. En el ciclo de reloj 4, la etapa IF capta la instrucción localizada en la dirección de memoria X y se desechan las instrucciones que fueron captadas antes de conocer el resultado de la condición del salto.

```
sub $10, $4, $8
beq $1, $3, X
and $12, $2, $5
or $13, $2, $6
add $14, $4, $2
slt $15, $6, $7
X: lw $4, 60($7)
```

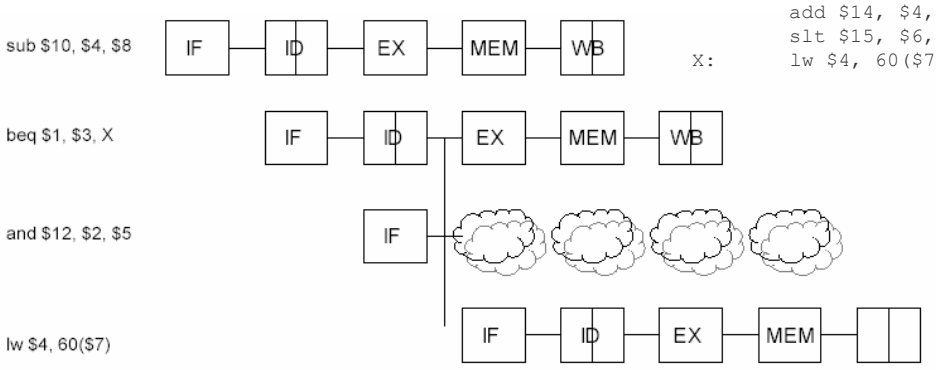


Figura 15. Ejemplo

Riesgos de Control o Control Hazards

Otra solución para atacar los riesgos de control es conocido como *delayed-branch.* En este caso los compiladores y ensambladores tratan de colocar una instrucción que siempre será ejecutada después del *branch*. Lo que buscan es que la instrucción que se ejecute después del *branch* sea válida y útil y no tenga que ser desechada después de ser captada.

Riesgos de Control o Control Hazards

Otra forma de riesgos de control incluyen a las excepciones. Supongamos una instrucción aritmética que produce un desbordamiento u *overflow*. Necesitamos transferir el control a la rutina de servicio de la excepción inmediatamente después de la instrucción porque no queremos que el resultado inválido contamine otros registros o localidades de memoria. En este caso, también se debe limpiar el pipeline y empezar a buscar instrucciones a partir de la nueva dirección (rutina de servicio de interrupciones).

¿Cuándo aplicar las transformaciones al código?

```
i1:     sub $10, $4, $8
i2:     beq $1, $3, X
i3:     and $12, $2, $5
i4:     or $13, $2, $6
X:      lw $4, 60($7)
i6:     lw $7, 128($12)
i7:     add $14, $4, $2
i8:     slt $15, $6, $7
```

```
i1:     sub $10, $4, $8
i2:     beq $1, $3, X
i6:     lw $7, 128($12)
i7:     slt $15, $6, $7}
i8:     add $14, $4, $2
        nop
i3:     and $12, $2, $5
i4:     or $13, $2, $6
X:      lw $4, 60($7)
```

1) Hacerlas a mano al intentar optimizar el código del programa

2) Dejar que los compiladores con optimización de código realicen estas tareas por su cuenta.