

Arquitectura de Computadoras Curso 2020

Segmentación – MIPS 64

Prof. Jorge Runco

MIPS 64 - Características

- Bus de datos de 64 bits
- Registros de 64 bits
- 32 registros de propósito general (R0..R31)
- 32 registros de coma flotante (F0..F31)
- Instrucciones de longitud fija (32 bits)
- Código de operación de longitud fija
- Sólo 2 instrucciones acceden a memoria (Load/Store)
- Memoria separada para instrucciones y datos (Arquitectura Harvard)
- Cauce segmentado en 5 etapas de 1 ciclo cada una

¿Qué sucede en cada etapa?



Etapa IF

- Obtención de la instrucción
- $PC = PC + 4$ (instrucciones de 4 bytes)

Etapa ID

- Decodificación de la instrucción
- Lectura de los registros
- Si es un salto verificar la condición
- Actualización del PC en caso de salto

Etapa EX

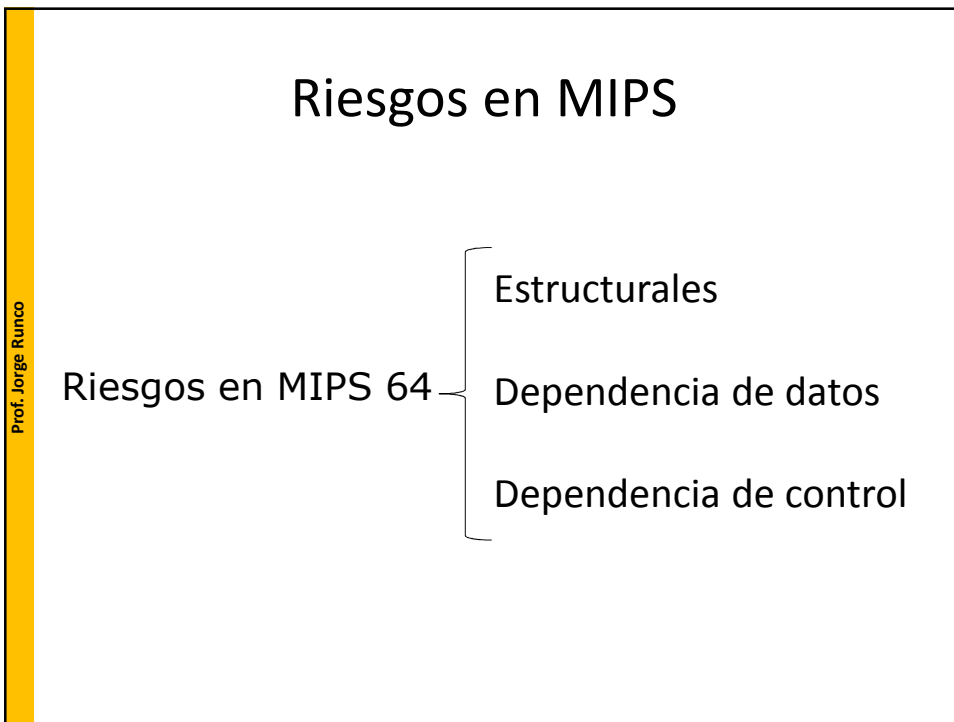
- Cálculos en la ALU
- Cálculo de dirección de operandos de memoria

Etapa MEM

- Acceso a memoria (Load/Store)

Etapa WB

- Escritura de los registros



- Prof. Jorge Runco
- ## Riesgos estructurales
- Cuando dos ó más instrucciones intentan usar el mismo recurso de hardware a la vez.



- Dependencia verdadera: cuando una instrucción depende de los resultados de otra instrucción, de manera que ambas no pueden ejecutarse de forma solapada.

4

Riesgos por dependencia de datos

- Otra dependencia se produce cuando instrucciones difieren en el número de ciclos para completarlas ó hay ejecución desordenada. Dos instrucciones usan el mismo registro, pero no hay intercambio de información entre las instrucciones. Distinguimos 2 clases:

Antidependencia

- Cuando una instrucción j escribe un registro que lee la instrucción i. Se debe mantener el orden original para asegurar que i lee el valor correcto
 - i) MUL.D F4, F5, F6
 - j) ADD.D F5, F7, F8
- La instrucción j tarda menos en ejecutarse que la i y podría cambiar el valor de F5 antes de que i lo lea.

Dependencia de salida

- Se produce cuando la instrucción i y la j escriben en el mismo registro. Se debe mantener el orden entre las instrucciones para que el valor final escrito corresponda a la instrucción j.
 - MUL.D **F4**, F5, F6
 - ADD.D **F4**, F7, F8
- Estas dos últimas dependencias, no son una dependencia verdadera, instrucciones involucradas no intercambian datos. Cambiando el nombre de los registros se soluciona el problema. (También llamada dependencia de nombre).

Prof. Jorge Runco

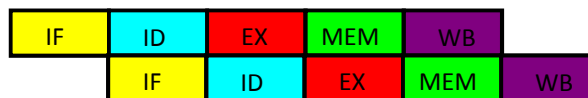
Dependencia de datos

RAR = Read After Read

No hay problemas

DADD R4,R5,**R6**

AND R4,R5,**R6**



RAW = Read After Write

R1 disponible recién aquí

DADD **R1**,R6,R7

AND R4,R5,**R1**



Lectura de R1 aquí

Aparecen los problemas

Prof. Jorge Runco

Dependencia de datos

WAR = Write After Read

DADD R4,R5,R6
AND R6,R7,R8

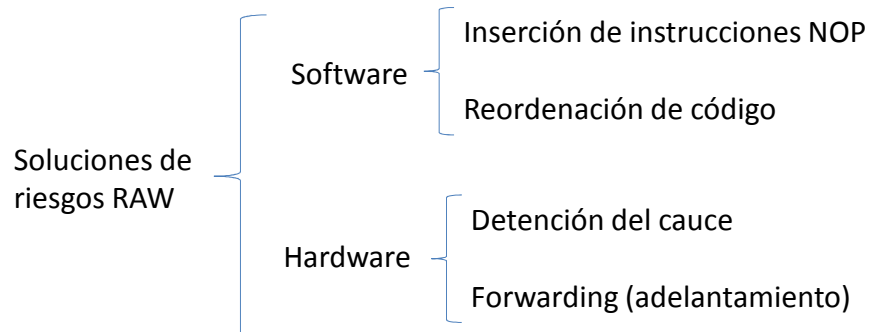
Si hay ejecución fuera de orden, la 2da instrucción puede terminar antes que la 1ra. Cambió R6 antes que lo use la 1ra.

WAW = Write After Write

DIV.D F4,F5,F6
ADD.D F4,F7,F8

Si hay ejecución fuera de orden ó cuando las instrucciones no “tardan” el mismo tiempo. DIV.D lleva más ciclos que ADD.D

Dependencias de datos



La dependencia de datos es detectada en la etapa ID

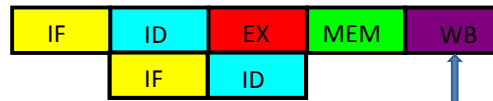
Vamos a plantear el problema

DADD R1,R2,R3

DSUB R6,R7,R1

OR R8,R9,R10

AND R11,R12,R13



Lectura de R1. No se encuentra disponible

Aquí está R1 disponible

Inserción de instrucciones NOP Solución de software

DADD R1,R2,R3

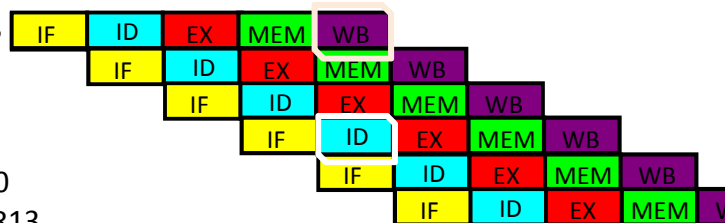
NOP

NOP

DSUB R6,R7,R1

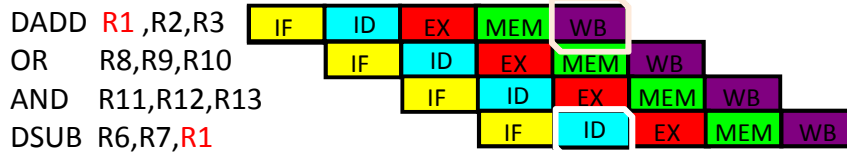
OR R8,R9,R10

AND R11,R12,R13



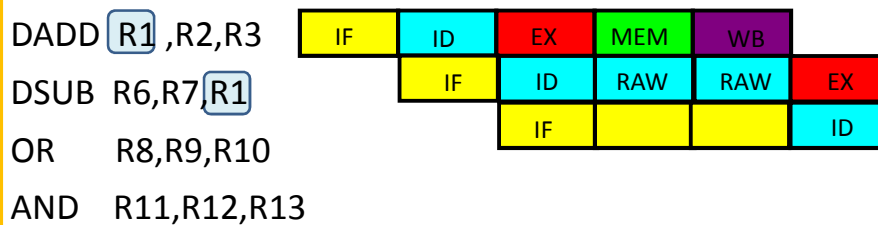
R1 está listo cuando lo necesite DSUB, pero se alarga el tiempo de ejecución.

Reordenación de código Solución de software



R1 está listo cuando lo necesite DSUB, se reordenó el código con instrucciones que no usan R1. En lugar de NOP se “reordenan” dos instrucciones que había que ejecutar. Más eficiente que el caso anterior pues no aumenta el tiempo de ejecución.

Detención del cauce Solución de hardware

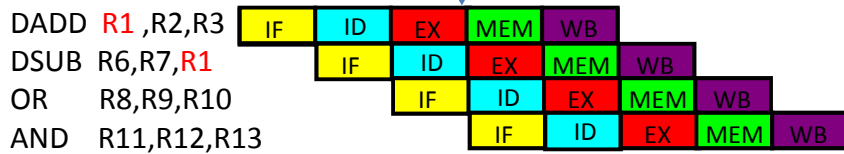


R1 no está listo cuando lo necesite DSUB, se detiene el pipeline y se alarga el tiempo de ejecución.

Adelantamiento (Forwarding)

Solución de hardware

Dato disponible para forwarding. Como es DADD después de la etapa EX.

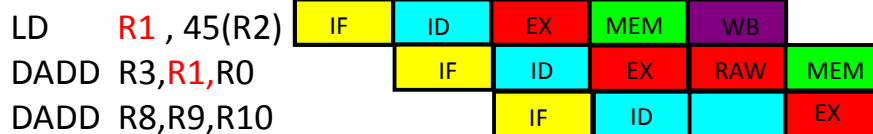


R1 está listo cuando lo necesite DSUB, no se alarga el tiempo de ejecución. El dato está disponible para forwarding después de la etapa EX. (DADD)

Se deben "habilitar" caminos de "realimentación" de la salida de la ALU a la entrada, de MEM a la ALU.

Forwarding (adelantamiento)

Solución de hardware (LD)



Se alarga la ejecución un ciclo

Tener en cuenta que es una instrucción de LD, el dato está disponible en la etapa MEM

Riesgos de control

Soluciones
de riesgos
de control

Software

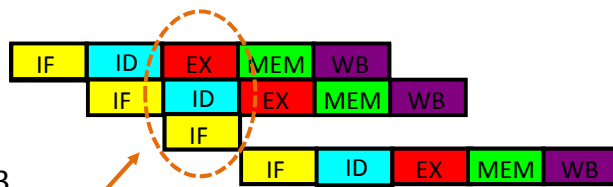
Inserción de instrucciones NOP
ó reordenar el código con salto
retardado

Hardware

Detención del cauce
Predicción del salto

Riesgos de control

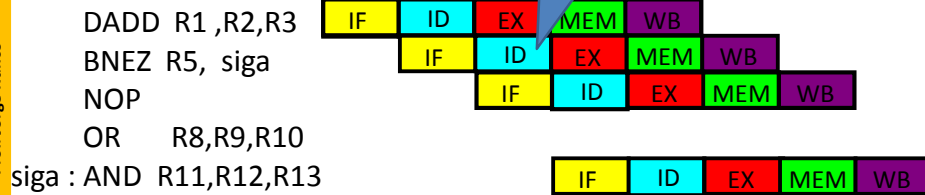
DADD R1 ,R2,R3
BNEZ R5, siga
OR R8,R9,R10
siga : AND R11,R12,R13



Hay que saltar y tirar lo que viene por el pipeline.
Empezó a “captar” la instrucción OR (siguiente)
pero no era lo correcta (había que saltar)

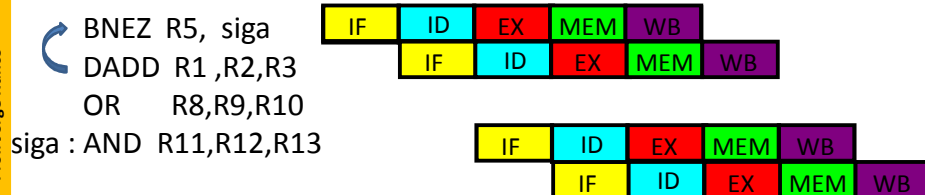
NOP y salto retardado

Recién acá se conoce la
dirección del salto



Con salto retardado, siempre se ejecuta la instrucción debajo del salto (NOP) y luego toma lugar el salto. Aumenta el tiempo de ejecución.

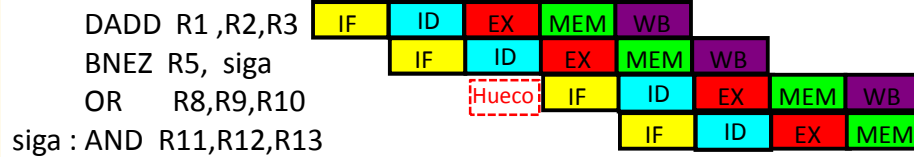
Reordenación de código y salto retardado



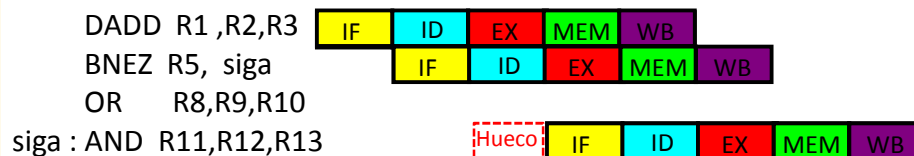
Se reordena el código, no hay instrucciones adicionales.

Detención del cauce: solución de hardware

El salto no se lleva a cabo



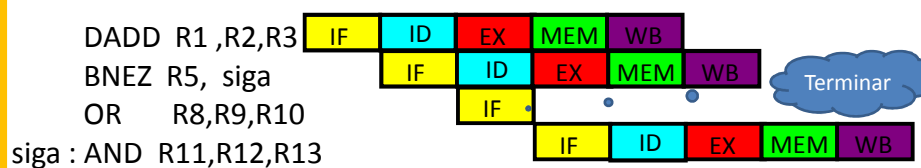
El salto se lleva a cabo



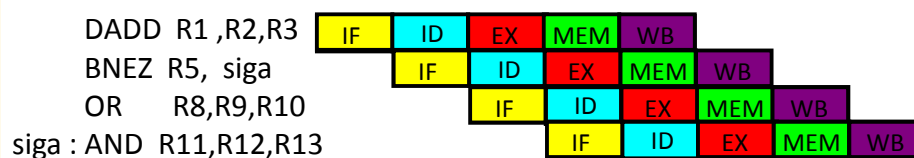
El cauce se detiene hasta comprobar la condición del salto
(si se ejecuta ó no) y saber donde saltar.

Predicción del salto: solución de hardware

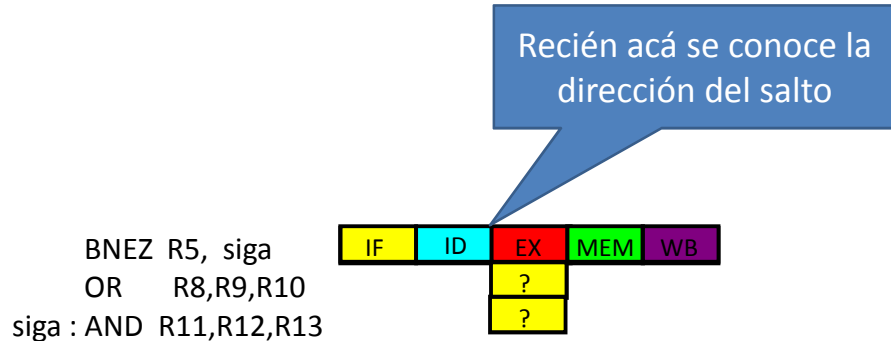
Se predice no saltar. Si falla penalización un ciclo



Se predice no saltar. Si acierta penalización cero



Predicción del salto: solución de hardware



Se predice sí saltar. La penalización siempre es un ciclo se lleva a cabo el salto ó no. No se usa en MIPS

BTB: Branch Target Buffer

- En la etapa ID se comprueba si el salto se lleva a cabo ó no y donde saltar.
- No se puede comprobar antes porque primero hay que decodificar la instrucción.
- Si en la etapa IF se pudiera saber si hay que saltar ó no, el pipe tomaría el camino correcto.
- ¿Pero cómo saber si la instrucción es un salto si todavía no fue decodificada?

BTB: Branch Target Buffer

- Se “agrega” en la etapa IF una memoria “muy rápida” (BTB) donde se almacenan las direcciones (IP) de memoria donde están los saltos solamente. Si saltó ó no la última vez y la dirección donde saltó.
- Cada vez que se busca una instrucción se comprueba si esta dirección está en la tabla (BTB). Si está es porque es un salto.
- Así sabemos que es un salto, sin decodificar, y si saltó y donde la última vez.

BTB: Branch Target Buffer

- En la tabla se anotan la dirección del salto (PC), donde saltar y la “predicción” del salto.
- La tabla es analizada en la etapa IF.
- Es como una memoria caché de saltos.

