

Winmips – RISC parte 2

Arquitectura de computadoras

Transferencia de memoria

- Las instrucciones LD y SD que transfieren de memoria/hacia memoria 8 bytes (64 bits)
- Winmips cuenta con instrucciones que permiten transferir 1,2, o 4 bytes.
- Las instrucciones de carga escriben el registro completo, por lo que deben completar los bits faltantes cuando el dato a transferir no los tiene.
- Es necesario indicar como completar los bits del registro:
 - Si se indica que la transferencia es de números sin signo, se completan con ceros.
 - Si se indica que la transferencia es de números con signo, se completan con el valor del bit mas significativo (1 si es negativo, 0 si es positivo)

Transferencia de memoria

Ejemplos:

- LD R1, dato(R2) ; Load Doble Word (64 bits)
- LB R1, dato(R2) ; Load Byte con signo (8 bits)
- LBU R1, dato(R2) ; Load Byte sin signo (8 bits)
- LH R1, dato(R2) ; Load Halfword con signo (16 bits)
- LHU R1, dato(R2) ; Load Halfword sin signo (16 bits)
- LW R1, dato(R2) ; Load Word con signo (32 bits)
- LWU R1, dato(R2) ; Load Word sin signo (32 bits)

Transferencia de memoria

- Las instrucciones de almacenamiento SD truncan (recortan) el registro.
- Solo transfieren los bits menos significativos. Si es un número con signo este queda en el bit mas significativo, por lo que no es necesario indicar si es con o sin signo.

Ejemplos:

- SD R1, dato(R2) ; Store Doubleword (64 bits)
- SB R1, dato(R2) ; Store Byte (8 bits)
- SH R1, dato(R2) ; Store Halfword (16 bits)
- SW R1, dato(R2) ; Store Word (32 bits)

Punto flotante

- Disponemos de 32 registros de doble precisión (64 bits) para trabajar con operaciones de punto flotante.
- Los denominamos $f\{i\}$ donde i esta entre 0 y 31.
- Disponemos de 3 unidades de ejecución paralelas para realizar las operaciones de:
 - Suma o Resta
 - Multiplicación
 - División

Punto flotante - Instrucciones

- L.D {reg. flot. origen}, {mem. destino}({reg. ent.})
 - Copiar un valor de memoria a un registro
- S.D {reg. flot. destino}, {mem. origen}({reg. ent.})
 - Copiar un valor de un registro en memoria
- ADD.D {reg. Destino}, {reg. Oper. 1}, {reg. Oper. 2}
 - Suma 2 registros y guarda en el registro destino
- SUB.D {reg. Destino}, {reg. Oper. 1}, {reg. Oper. 2}
 - Resta Oper. 2 a Oper. 1 y guarda en el registro destino
- MUL.D {reg. Destino}, {reg. Oper.1}, {reg. Oper. 2}
 - Multiplica 2 registros y guarda en el registro destino
- DIV.D {reg. Destino}, {reg. Oper.1}, {reg. Oper. 2}
 - Divide Oper. 1 por Oper. 2 y guarda en el registro destino

Punto flotante - Instrucciones

- MOV.D {reg. flot. destino}, {reg. flot. origen}
 - Copia el registro flotante origen en el destino
- MTC1 {reg. ent. origen}, {reg. flot. destino}
 - Copia los 64 bits del reg. ent. origen en los del reg. flot. Destino
- MFC1 {reg. ent. destino}, {reg. flot. origen}
 - Copia los 64 bits del reg. flot. origen en los del reg. ent. destino
- CVT.D.L {reg. flot. destino}, {reg. flot. origen}
 - Convierte el valor flotante del reg. origen a su representación entera de 64 bits y los almacena en el reg. destino
- CVT.L.D {reg. flot. destino}, {reg. flot. origen}
 - Interpreta los 64 bits del reg. flot. origen como un número entero , lo convierte su representación en punto flotante y lo guarda en reg. flot. destino

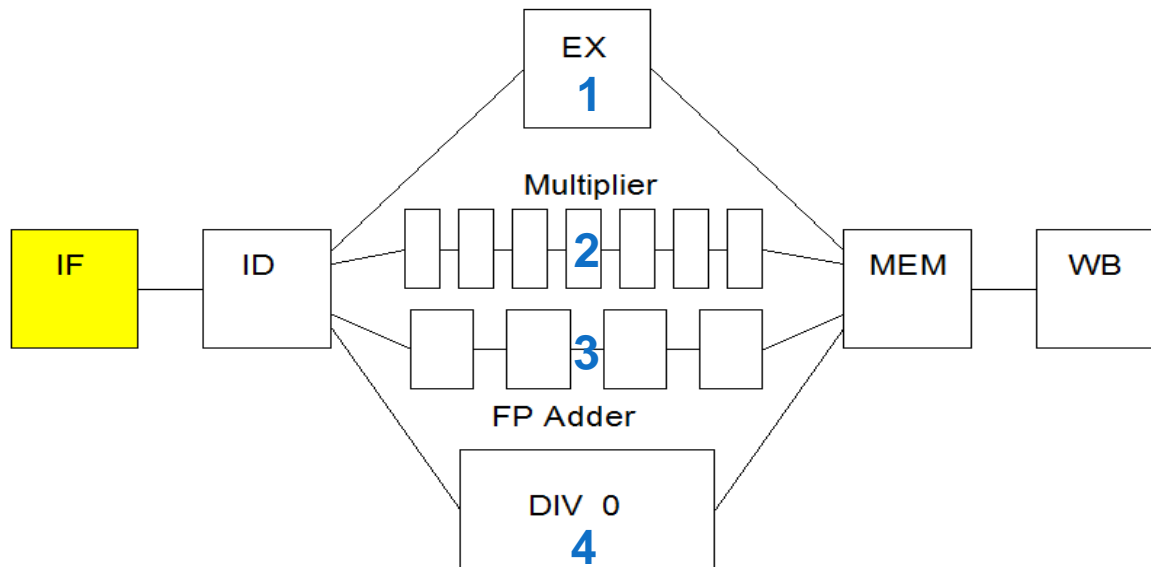
Punto flotante - Instrucciones

- Existe un flag de estado, denominado FP, asociado a instrucciones de comparaciones y saltos en punto flotante.
- C.LT.D {reg. flot.}, {reg. flot.}
 - Pone 1 en FP si el 1er reg. es menor que el 2do. Sino pone 0.
- C.LE.D {reg. flot.}, {reg. flot.}
 - Pone 1 en FP si el 1er reg. es menor o igual que el 2do . Sino pone en 0.
- C.EQ.D {reg. flot.}, {reg. flot.}
 - Pone en 1 FP si los registros son iguales. Sino pone en 0.
- BC1F {dir. destino}
 - Salta a la dir. destino si el flag FP esta en 0.
- BC1T {dir. destino}
 - Salta a la dir. destino si el flag FP esta en 1.

Winmips – Unidades de ejecución

Winmips tiene 4 unidades de ejecución:

1. Para instrucciones generales (1 etapa de 1 ciclo)
2. Para multiplicar (7 etapas de 1 ciclo c/u)
3. Para sumar punto flotante (4 etapas de 1 ciclo c/u)
4. Para dividir (1 etapa de 24 ciclos)



Winmips – Unidades de ejecución

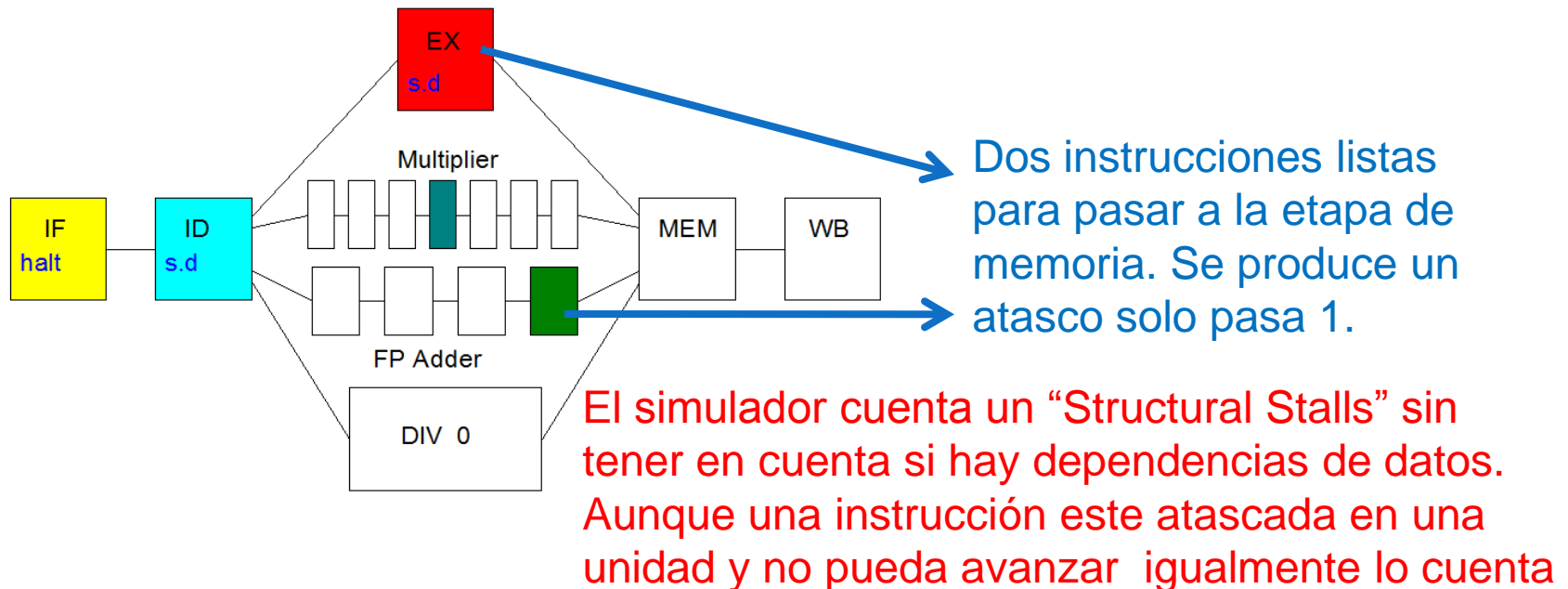
- Cada etapa de una unidad de ejecución puede tener una instrucción.
- Los operandos de una instrucción son requeridos en en la primer etapa de cada unidad de ejecución. Si no están disponibles se producirá un atasco RAW
- El resultado de una instrucción se producirá en la ultima etapa de cada unidad de ejecución. Al final de esta etapa pueden adelantarse los operandos si se utiliza “forwarding”

Winmips – Unidades de ejecución

- Tener múltiples unidades de ejecución permite ejecutar mas instrucciones en el mismo tiempo.
- El uso de múltiples unidades tiene sus ventajas pero introduce nuevos problemas.
- Aparecen 3 tipos de atascos:
 - Dependencia Estructural
 - Dependencia de datos WAR
 - Dependencia de datos WAW

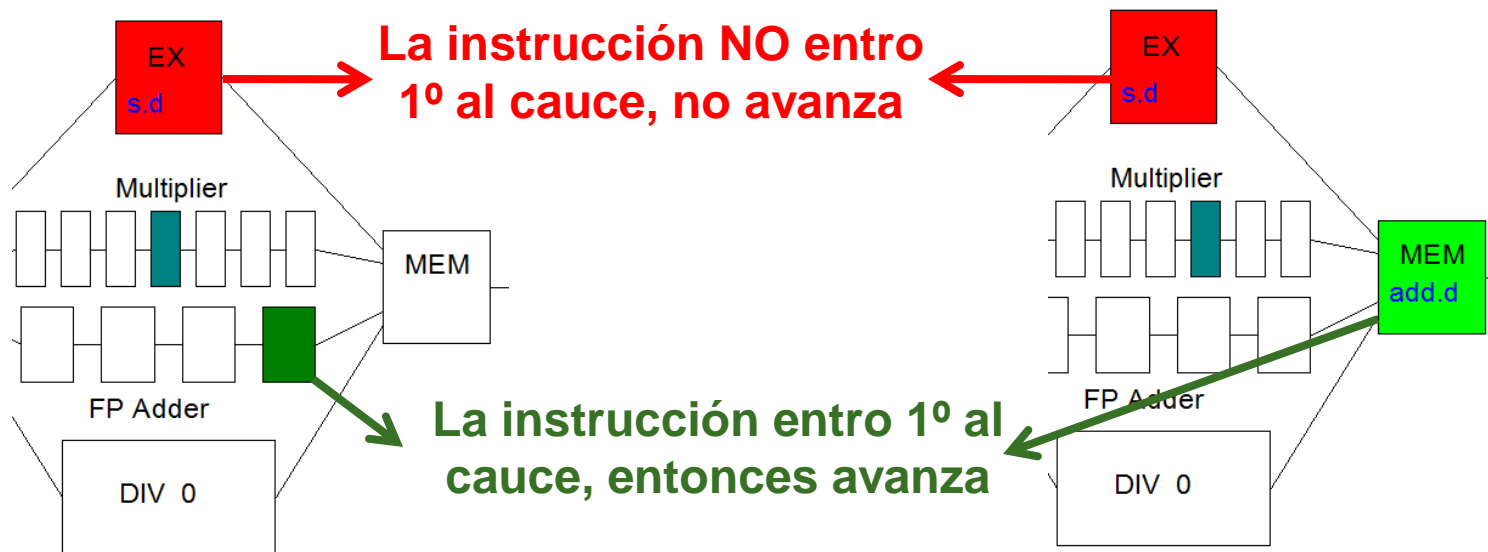
Atascos estructurales

- Los atascos estructurales son provocados por conflictos por los recursos.
- En el winmips solo puede suceder cuando dos instrucciones en unidades de ejecución distintas intentan acceder a la etapa memoria simultáneamente.



Atascos estructurales

- Cuando sucede un atasco estructural solo una de las instrucciones avanza.
- Tiene prioridad la primera instrucción que entro en el cauce.



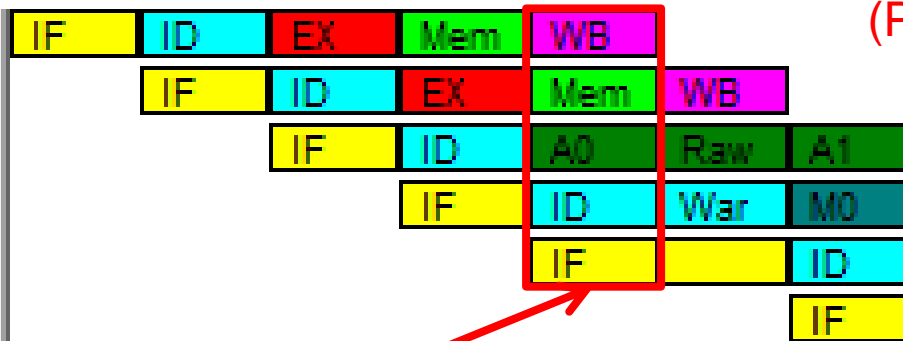
Atascos WAR y WAW

Los atascos WAR y WAW suceden cuando:

- **hay dependencia de datos** entre dos instrucciones (igual que RAW).
- **hay mas de una unidad** de ejecución.
- una instrucción que entra al cauce puede **sobrepasar** a una instrucción anterior, escribiendo un registro pendiente de lectura (WAR) o escritura (WAW).
- El simulador produce atascos cuando detecta una situación **potencial** de dependencia WAR o WAW.
- Muchas veces si la instrucción continúa el atasco WAR o WAW no sucede. **Bloquea por las dudas** ya que no tiene hardware que determine cuanto tarda una instrucción en cada unidad de ejecución.

Atascos WAR y WAW

- 1 L.D F1, n1(R0)
- 2 L.D F2, n2(R0)
- 3 ADD.D F3, F2, F1
- 4 MUL.D F1, F4, F0
- 5 MUL.D F4, F2, F1
- 6 S.D F3, res1(R0)
- 7 S.D F4, res2(R0)
- 8 HALT



(Programa ejecutado con forwarding habilitado)

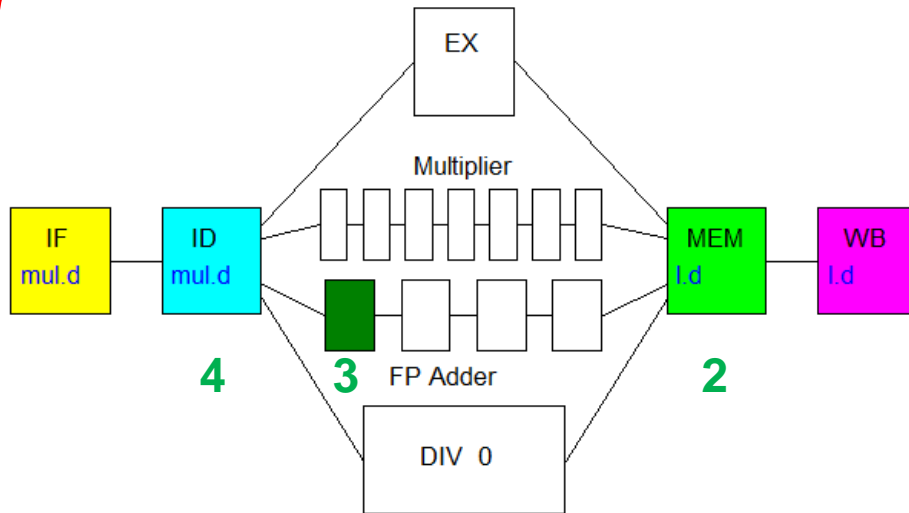
Situación 1:

- ADD.D (3) depende de F2
- ADD.D necesita F2 al inicio de A0
- L.D F2 (2) carga F2 al final de Mem

Resultado: RAW por F2 de ADD.D

Situación 2:

- MUL.D (4) escribirá F1
 - ADD.D generará RAW por F2 de L.D
 - El simulador no tiene forma de saber cuantos ciclos RAW provocará ADD.D
- Resultado:** WAR (potencial) por F1 de ADD.D por prevención



Observar que:

ADD.D depende de L.D y provoca 1 raw, pero si dependiera de DIV provocaría 24 raw