Arquitectura superescalar

Temario

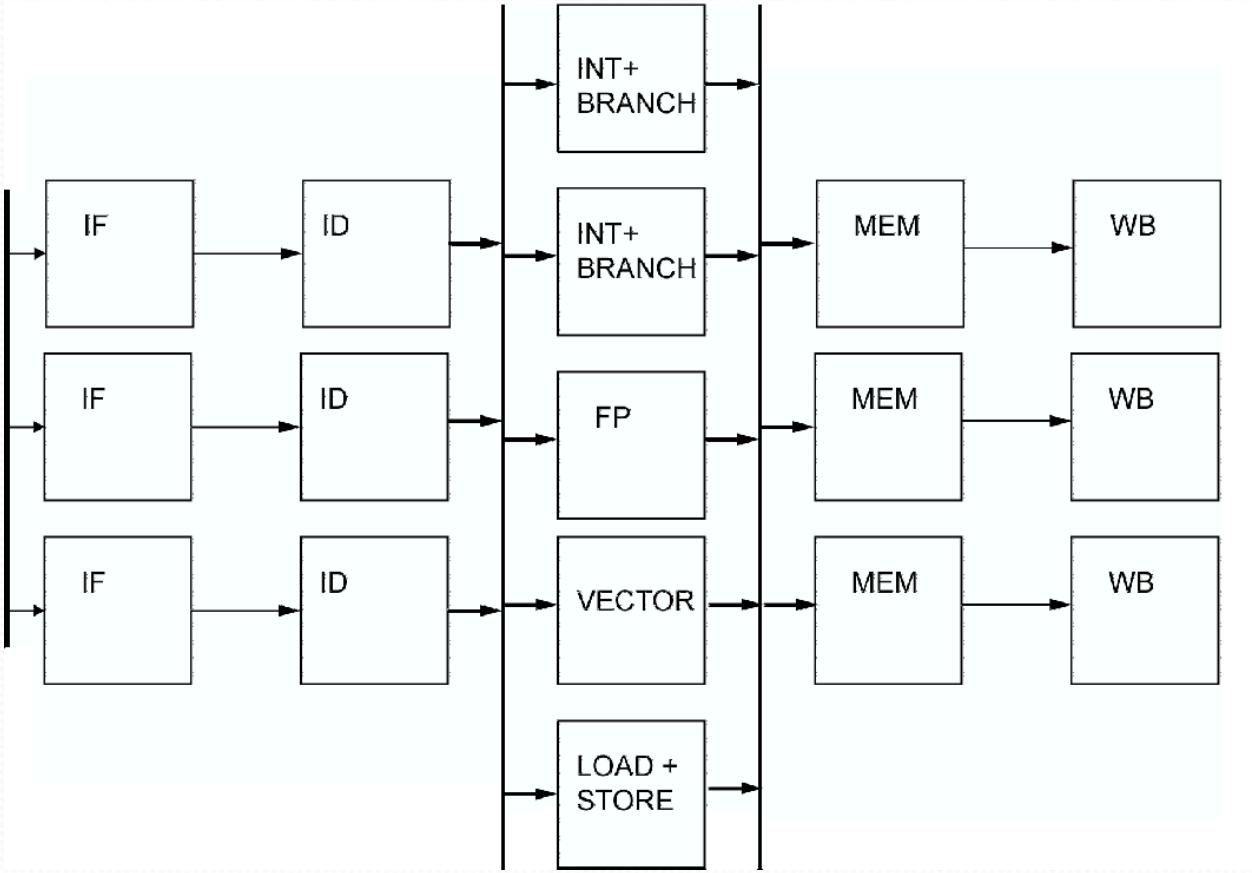
- Introducción.
- Peligros.
- Scheduling estático.
- Scheduling dinámico.

Introducción

- El término superescalar (superscalar) fue acuñado a fines de los 80s.
- Todas las CPUs modernas son superescalares.
- Es un desarrollo de la arquitectura con pipeline.
- El pipeline se replica y en cada ciclo de reloj se emiten varias instrucciones.

Introducción

- Superescalar de grado 3.



Introducción

- En los superescalares estáticos las instrucciones corren en orden y pueden terminar en desorden.
- En los superescalares dinámicos las instrucciones pueden correr y terminar en desorden.
- Se generan nuevos peligros de datos por dependencias entre instrucciones.

Instrucciones por ciclo (IPC)

- Al emitir varias instrucciones por ciclo el CPI (ciclos por instrucción) es menor a 1.
- Una CPU de 6 GHz de grado 4 puede emitir hasta 4 instrucciones a la vez y ejecutar hasta 24 mil millones de instrucciones por segundo para un CPI de 0.25.
- En vez de CPI se usa IPC (instrucciones por ciclo de reloj) = 1 / CPI.
- Un CPI de 0.25 es igual a un IPC de 4.

Definiciones

- **Grado**. Número de instrucciones que el CPU *intenta* ejecutar (no hay garantía que lo pueda lograr).
- **Scheduling**. En que orden se ejecutan las instrucciones. Hay dos opciones:
  - Estático: la CPU no las puede reordenar.
  - Dinámico: la CPU si las puede reordenar.
- **Emisión (issue)**. Una instrucción se emite al obtenerse (fetched) de la memoria.
- **Paquete de emisión**. Conjunto de instrucciones que se emiten en cada ciclo.

Definiciones

- Hay dos estrategias de emisión para un procesador superescalar de grado n:
  - Dinámica: el paquete de emisión puede contener 0, 1 o hasta n instrucciones.
  - Estática: el paquete de emisión siempre contiene n instrucciones. Si es necesario, el paquete se completa con instrucciones nop.

Definiciones

- **Ejecución**. Una instrucción se ejecuta cuando sus operandos están listos. Hay dos opciones:
  - En orden del programa.
  - Fuera de orden.
- **Especulación**. Se intenta adivinar los brincos y el resultado de algunas otras instrucciones como por ejemplo que instrucciones consecutivas de acceso a memoria no se refieren a la misma dirección.

Definiciones

- **Escribir (write back)**. La instrucción escribe el resultado en un lugar temporal.
- **Compromiso (commit)**. Una instrucción se compromete cuando la CPU le permite actualizar el banco de registros o la memoria.
- **Retiro**. Después de escribir, la instrucción es retirada del pipeline. También se dice que la instrucción ha sido *completada*.

Clasificación

- Superescalares estáticos.
  - Scheduling estático. Las instrucciones se emiten y ejecutan en orden, pero pueden terminar en desorden.
  - Emisión dinámica. El paquete de emisión no necesariamente tiene n instrucciones.

Clasificación

- Superescalares dinámicos.
  - Scheduling dinámico. Las instrucciones se emiten en orden, pero pueden correr y terminar en desorden.
  - Emisión dinámica. El paquete de emisión no necesariamente tiene n instrucciones.
  - Capacidad para especular brincos.

Clasificación

Common name	Issue structure	Hazard detection	Scheduling	Distinguishing characteristic	Examples
Superscalar (static)	Dynamic	Hardware	Static	In-order execution	Mostly in the embedded space: MIPS and ARM, including the ARM Cortex-A8
Superscalar (dynamic)	Dynamic	Hardware	Dynamic	Some out-of-order execution, but no speculation	None at the present
Superscalar (speculative)	Dynamic	Hardware	Dynamic with speculation	Out-of-order execution with speculation	Intel Core i3, i5, i7; AMD Phenom; IBM Power 7
VLIW/LIW	Static	Primarily software	Static	All hazards determined and indicated by compiler (often implicitly)	Most examples are in signal processing, such as the TI C6x
EPIC	Primarily static	Primarily software	Mostly static	All hazards determined and indicated explicitly by the compiler	Itanium

Peligros (hazards)

- Los procesadores superescalares tienen tres clases de peligros: estructurales, de datos y de control.
- Los peligros estructurales son los mismos que en el pipeline y se resuelven de la misma forma.
- Los peligros de control son los mismos que en el pipeline y se resuelven de la misma forma.

Peligros (hazards)

- Los procesadores superescalares tienen tres peligros de datos: RAW (read-after-write), WAW (write-after-write) y WAR (write-after-read).
- Los peligros de datos se generan por dependencias entre instrucciones.
- Además de la dependencia de datos verdadera que puede producir un peligro RAW, hay dos dependencias de *nombre*.

Peligros (hazards)

- Las dependencias de datos verdaderas **no** se pueden eliminar.
- Las dependencias de datos por nombre se pueden eliminar.

Dependencias verdaderas

- La primera instrucción guarda su resultado en el mismo registro que una segunda instrucción va a leer.
- Ejemplo:

```
1. add $t1, $t2, $t3        # t1 = t2 + t3
2. sub $t4, $t5, $t1        # t4 = t5 – t1
```

- Si corren en desorden, sub lee el valor anterior de $t1.
- Puede producir un peligro RAW (read-after-write).

Dependencias de nombre

1. Dependencias de salida.
   - Dos instrucciones guardan sus resultados en la misma parte.
   - Ejemplo:

     1. add $t0, $t1, $t2 **# t0 = t1 + t2**

     2. add $t0, $t3, $t4 **# t0 = t3 + t4**
   - Si son ejecutadas fuera de orden, $t0 se queda con un valor incorrecto.
   - Puede producir un peligro WAW (write-after-write).

Dependencias de nombre

2. Antidependencias.
   - La primera instrucción necesita un valor que la segunda instrucción actualiza.
   - Ejemplo:

     1. add $t4, $t0, $t1 **# t4 = t0 + t1**

     2. sub $t0, $a0, $t2 **# t0 = a0 + t2**

   - Si son ejecutadas fuera de orden, add lee el valor actualizado de $t0.
   - Puede producir un peligro WAR (write-after-read).

Dependencias de nombre

- Dos métodos básicos para eliminar las dependencias de nombre:

1. Renombrar los registros.
2. Usar un buffer para reordenar (reorder buffer o ROB).

Renombrar registros

- La CPU mantiene una lista de registros virtuales.
- Los registros virtuales son invisibles al programador.
- Se asignan dinámicamente cuando un registro físico aparece como un registro destino.
- La CPU mantiene un mapeo entre registros virtuales y registros físicos.
- Varios registros virtuales pueden estar mapeados al mismo registro físico.

Renombrar registros

- Este mapeo funciona como una historia del registro físico.

Renombrar registros

- Ejemplo:

```
1. mul $r2, $r2, $r3    # r2 = r2 * r3
2. addi $r4, $r2, 1     # r4 = r2 + 1
3. addi $r2, $r3, 1     # r2 = r3 + 1
4. div $r5, $r2, $r4    # r5 = r2 / r4
```

- Dependencias de salida: 1 y 3 por $r2.
- Antidependencias: 2 y 3 por $r2.
- Dependencias verdaderas: 1 y 2 por $r2, 2 y 4 por $r4, 3 y 4 por $r2.
- El programa debe correr en orden.

Renombrar registros

- Los registros se pueden etiquetar:

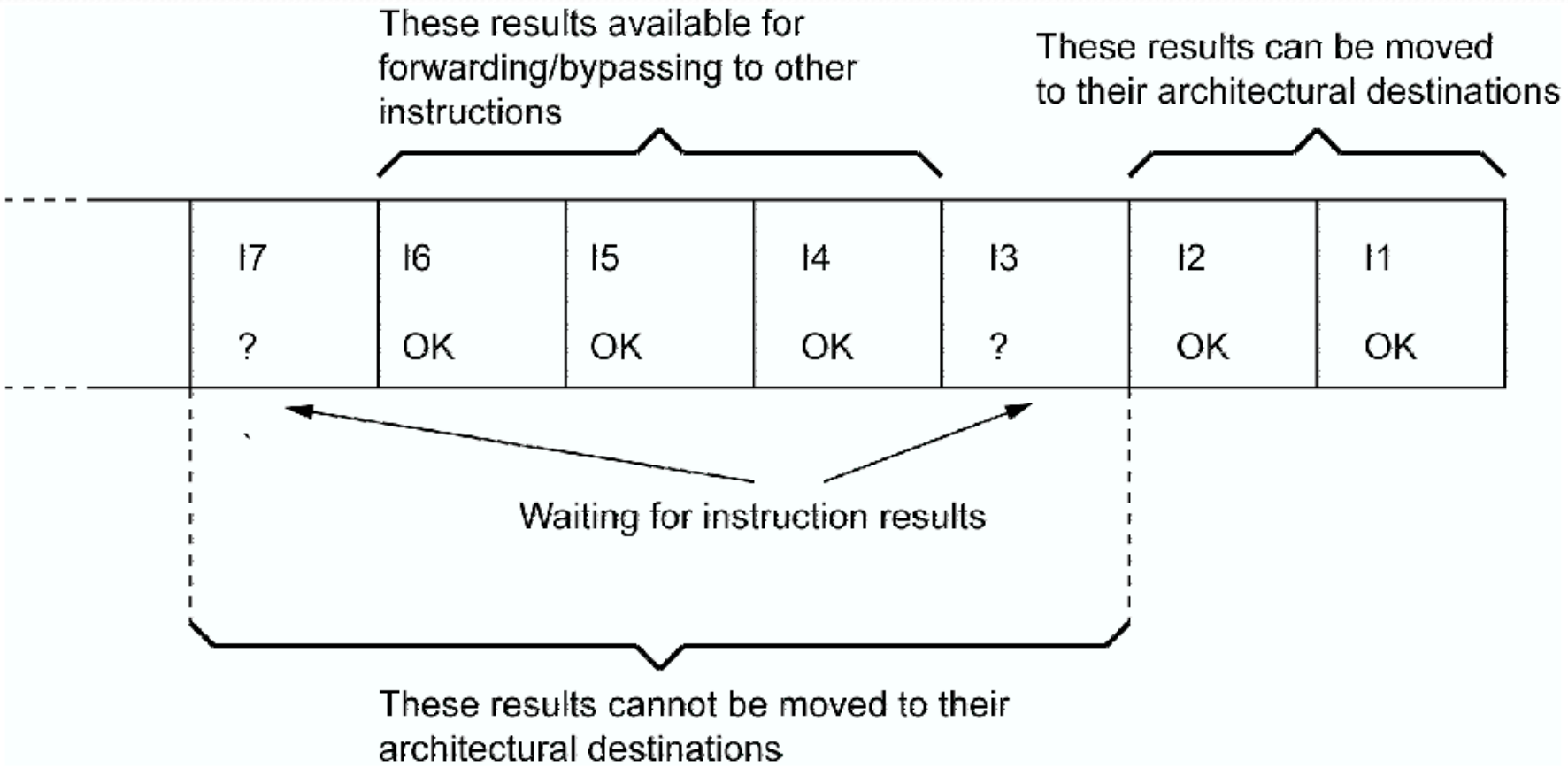
```
1. mul $r2_b, $r2_a, $r3_a      # r2 = r2 * r3
2. addi $r4_b, $r2_b, 1         # r4 = r2 + 1
3. addi $r2_c, $r3_a, 1         # r2 = r3 + 1
4. div $r5_b, $r2_c, $r4_b      # r5 = r2 / r4
```

- Ahora la instrucción 3 puede comenzar porque usa un registro $r2 "diferente" que 1 y 2.
- Las etiquetas funcionan como historia de cada registro: $r2_c es la versión más nueva de $r2, mientras que $r2_a es la versión más antigua.

ROB

- Provee registros adicionales.
- Mantiene el resultado de una instrucción desde que termina hasta que se compromete.
- Cada entrada en el ROB tiene 4 campos:
  - Tipo de instrucción: brinco, memoria o tipo R.
  - Registro destino si es carga o tipo R.
  - Valor de la instrucción.
  - Indicador si la instrucción terminó y el valor es válido.
- Las entradas están ordenados en cola (FIFO).

ROB


These results available for forwarding/bypassing to other instructions
These results can be moved to their architectural destinations
I7
?
I6
OK
I5
OK
I4
OK
I3
?
I2
OK
I1
OK
Waiting for instruction results
These results cannot be moved to their architectural destinations

ROB

- Si una instrucción necesita el valor de un registro, primero debe buscarlo en el ROB, de atrás hacia delante, y si no lo encuentra entonces lo toma del registro físico.

Resumen

- Los procesadores superescalares son una extensión de la arquitectura pipeline simple que emiten y pueden ejecutar varias instrucciones a la vez.
- Producen más peligros y requieren soluciones más sofisticadas que el pipeline simple.
- Pueden alcanzar CPIs menores a 1.

Scheduling estático

- El procesador ejecuta las instrucciones en orden.
- El compilador:
  - Puede cambiar el orden de ejecución.
  - Genera el paquete de emisión.
  - Trata de prevenir o reducir los peligros de datos y de control.
- La mayoría de procesadores móviles son superescalares con scheduling estático.

Ejemplo

- Considerar un procesador MIPS superescalar con scheduling estático y doble emisión.
- Limitación: solo una instrucción puede ser de tipo R o brinco y la otra puede ser una carga o un store.
- El paquete de emisión tiene 8 bytes (2 instrucciones).
- La instrucción R o de brinco se pone primero y luego la instrucción de memoria.

Ejemplo

Instruction type	Pipe stages							
ALU or branch instruction	IF	ID	EX	MEM	WB			
Load or store instruction	IF	ID	EX	MEM	WB			
ALU or branch instruction		IF	ID	EX	MEM	WB		
Load or store instruction		IF	ID	EX	MEM	WB		
ALU or branch instruction			IF	ID	EX	MEM	WB	
Load or store instruction			IF	ID	EX	MEM	WB	
ALU or branch instruction				IF	ID	EX	MEM	WB
Load or store instruction				IF	ID	EX	MEM	WB

Fuente: COD-HSI 5, p. 335

Ejemplo

N	C	CPI = C / N
2	5	2.5
4	6	1.5
6	7	1.17
8	8	1.0
10	9	0.9
…	…	
n	(n / 2) + 4	((n / 2) + 4) / n

- $\mathrm{CPI} = \lim_{n\to\infty} \frac{\frac{n}{2}+4}{n} = 0.5 \qquad \mathrm{IPC} = \frac{1}{\mathrm{CPI}} = 2$

Peligros

- La forma de tratar los peligros depende de cada procesador.
- En algunos procesadores simples:
  - El compilador quita todos los peligros insertando instrucciones nop.
  - No hay detección de peligros por hardware ni detenciones (stalls).

Peligros

- Por ejemplo, en un procesador con doble emisión el compilador convierte el siguiente código:

```
add $t0, $t1, $t2          # paquete i
sub $t3, $t0, $t4          # paquete i
```

- en:

```
add $t0, $t1, $t2          # paquete i
nop                        # paquete i
sub $t3, $t0, $t4          # paquete i + 1
```

Peligros

- En otros procesadores más sofisticados:
  - El hardware detecta peligros de datos verdaderos y genera detenciones entre dos paquetes de emisión.
  - El compilador elimina las dependencias entre las instrucciones del mismo paquete de emisión.

Resumen

- Scheduling estático:
- Las instrucciones corren en el orden dictado por el compilador.
- Si los peligros no se detectan por hardware, el compilador debe hacerlo.
- Es más fácil de implementar que el scheduling dinámico.
- Se usa principalmente en procesadores móviles.

Scheduling dinámico

- El procesador puede reordenar las instrucciones.
- La mayoría de procesadores de escritorio son superescalares con scheduling dinámico.
- La mayoría emplean alguna variante de un método de scheduling dinámico conocido como algoritmo de Tomasulo.
- Desarrollado por Robert Tomasulo para el mainframe IBM 360/91.

Características

- No permite la ejecución de una instrucción mientras sus operandos no están listos.
- Así se eliminan los peligros RAW.
- Utiliza estaciones de reserva para renombrar registros y eliminar los peligros WAR y WAW.
- Permite especulación dinámica de brincos.
- Utiliza un ROB (reorder buffer) para guardar las instrucciones mientras se comprometen.

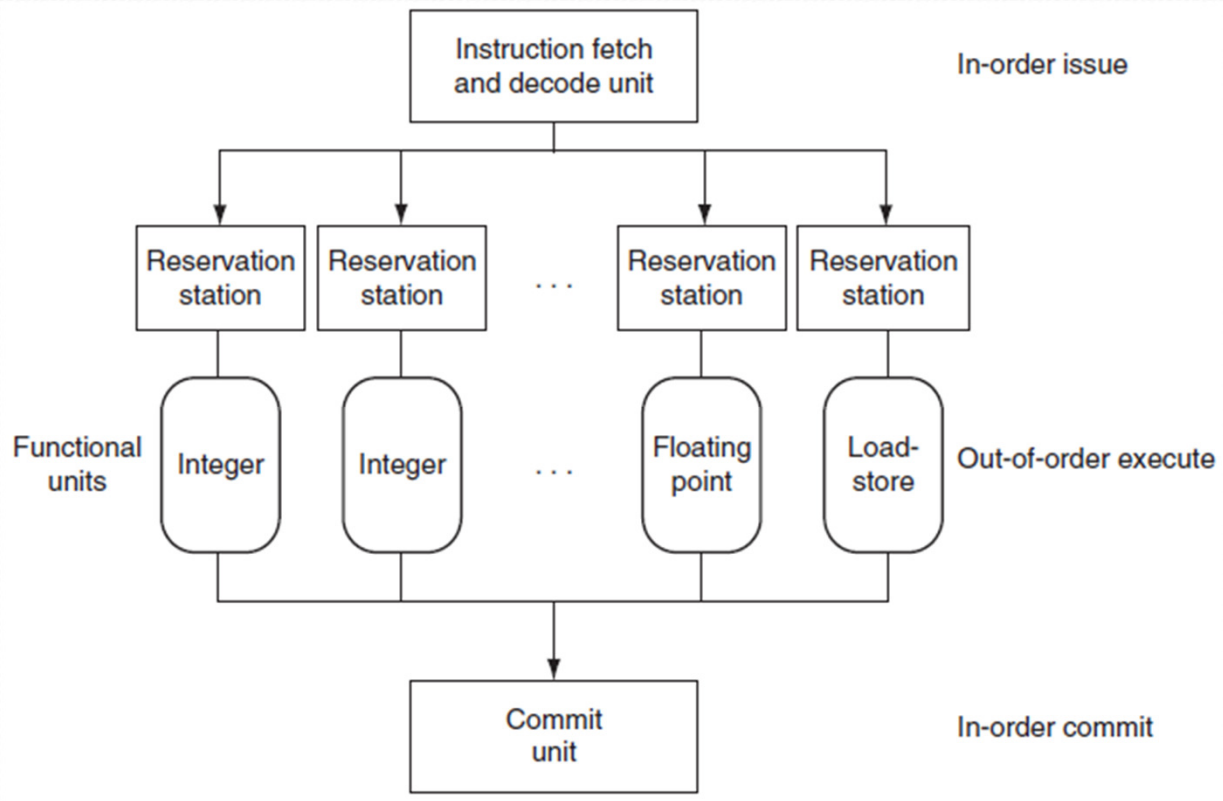
Características

- Cuando una instrucción se compromete, se le permite actualizar el banco de registros o la memoria de datos.
- Permite que las instrucciones se ejecuten en desorden.
- La emisión de instrucciones y el compromiso se hacen en orden.

Estaciones de reserva

- Reservation stations (RS).
- Guardan los operandos tan pronto están disponibles.
- Las instrucciones con dependencias apuntan a la RS de donde obtendrán sus operandos.
- Generalmente, hay más RS que registros físicos.

Estaciones de reserva



Fuente: COD-HSI 5, p. 340

Estaciones de reserva

- Cada RS tiene 7 campos:
  - Op – La operación a realizar.
  - V_j, V_k – El valor de los operandos libres.
  - Q_j, Q_k – Las RS de donde saldrán los operandos. Un 0 indica que los operandos ya están en V_j o V_k, o que son innecesarios.
  - A – Si la operación es una carga o un store, aquí se guarda la dirección efectiva.
  - Busy – Indica que la RS y la correspondiente unidad funcional están ocupadas.

Estaciones de reserva

- Además, el banco de registros tiene 1 campo extra:
  - Q_i – El número de RS que contiene la operación cuyo resultado va a ser guardado en este registro. Si Q_i es 0, no hay ninguna instrucción activa que vaya a guardar algo en este registro.

Bus de datos común

- CDB – Common Data Bus.
- Permite pasar el resultado de una instrucción, antes de que se comprometa, a otras instrucciones que necesiten ese valor.

Compromiso

- Las instrucciones pasan por una etapa llamada *compromiso* (commit).
- Para poder comprometerse, una instrucción debe ser no especulada y estar al frente del ROB.
- Sólo entonces se le permite escribir al banco de registros o a la memoria.
- Las instrucciones de ejecutan fuera de orden pero se comprometen en orden.

ROB

- El ROB (reorder buffer) es una cola FIFO.
- El ROB mantiene el resultado de una instrucción desde que termina hasta que se compromete.
- Como ni la memoria ni el banco de registros se actualizan hasta que cada instrucción se compromete, el ROB provee operandos a las instrucciones que dependendan de instrucciones que están dentro del ROB.

Instrucciones de memoria

- Las cargas y los stores se hacen en dos pasos:

1. La dirección efectiva se calcula cuando el registro base está listo y luego se guarda en la entrada del ROB correspondiente.
2. Las cargas accesan la memoria tan pronto la unidad de memoria esté lista. Los stores pueden tener que esperar al registro fuente.

- Para evitar peligros los loads y stores se hacen en orden hasta el paso 1.

Instrucciones de memoria

- Las cargas actualizan el registro destino al comprometerse.
- Los stores actualizan la memoria al comprometerse.

Algoritmo de Tomasulo

- Las instrucciones pasan por 4 etapas:
  1. Emisión (issue).
  2. Ejecución.
  3. Escribir resultados en el ROB.
  4. Compromiso (escribir en los registros / memoria).
- Cada etapa tarda un número arbitrario de ciclos.

Algoritmo de Tomasulo

1. Emisión.
   - Obtiene una instrucción de la cola de instrucciones.
   - La instrucción se emite si hay una RS libre y espacio en el ROB donde poner el resultado.
   - Si uno o los dos operandos están disponibles en el banco de registros o en el ROB se copian los valores en los campos correspondientes.
   - Si uno o los dos operandos dependen de otra u otras instrucciones se marcan las dependencias en los campos correspondientes.
   - Si todas las RS están ocupadas o el ROB está lleno, la instrucción se detiene (stall).

Algoritmo de Tomasulo

2. Ejecución.
   - La instrucción se puede ejecutar si sus operandos están disponibles en la RS.
   - En otro caso, el CDB se monitorea esperando que los operandos sean calculados.
   - Este paso elimina los peligros RAW.
   - Las cargas se hacen dos pasos: 1) cálculo de la dirección; 2) lectura de la memoria de datos.

Algoritmo de Tomasulo

- Los stores solo calculan la dirección efectiva en esta etapa.

Algoritmo de Tomasulo

3. Escribir resultados.
   - Al terminar la ejecución, el resultado se escribe en el ROB y en el CDB para uso de otras RS que requieran el valor.
   - La RS se marca como libre.
   - Los stores necesitan atención especial:
     - Si el valor a guardar está libre, se escribe en el ROB.
     - Si no está disponible, el CDB se monitorea hasta que el valor aparezca y entonces se escribe en el ROB.

Algoritmo de Tomasulo

4. Compromiso.
   - Una instrucción puede comprometerse cuando llega al frente del ROB.
   - Las acciones dependen del tipo de instrucción.
   - Si es un store, la memoria se actualiza.
   - Si es otra instrucción (excepto un brinco), el banco de registros se actualiza.
   - Si es un brinco mal adivinado, el ROB se limpia y la ejecución comienza en el lugar correcto.
   - Si es un brinco bien adivinado no pasa nada.
   - En cualquier caso, la entrada del ROB se libera.

Ejemplo

- Mostrar el status de las tablas cuando el mul.d está listo para comprometerse.

1. l.d f6, 32(r2) **# f6 = Mem[r2 + 32]**
2. l.d f2, 44(r3) **# f2 = Mem[r3 + 44]**
3. mul.d f0, f2, f4 **# f0 = f2 * f4**
4. sub.d f8, f2, f6 **# f8 = f2 – f6**
5. div.d f10, f0, f6 **# f10 = f0 / f6**
6. add.d f6, f8, f2 **# f6 = f8 + f2**

ROB

	Reorder buffer					
Entry	Busy	Instruction		State	Destination	Value
1	no	L.D	F6,32(R2)	Commit	F6	Mem[32 + Regs[R2]]
2	no	L.D	F2,44(R3)	Commit	F2	Mem[44 + Regs[R3]]
3	yes	MUL.D	F0,F2,F4	Write result	F0	#2 × Regs[F4]
4	yes	SUB.D	F8,F2,F6	Write result	F8	#2 – #1
5	yes	DIV.D	F10,F0,F6	Execute	F10	
6	yes	ADD.D	F6,F8,F2	Write result	F6	#4 + #2

Estaciones de reserva

	Reservation stations							
Name	Busy	Op	Vj	Vk	Qj	Qk	Dest	A
Load1	no							
Load2	no							
Add1	no							
Add2	no							
Add3	no							
Mult1	no	MUL.D	Mem[44 + Regs[R3]]	Regs[F4]			#3	
Mult2	yes	DIV.D		Mem[32 + Regs[R2]]	#3		#5	

Banco de registros

	FP register status									
Field	F0	F1	F2	F3	F4	F5	F6	F7	F8	F10
Reorder #	3						6		4	5
Busy	yes	no	no	no	no	no	yes	. . .	yes	yes

Ejemplo

- Mostrar la ejecución del siguiente código:

```
1. loop: l.d f0, 0(r1)          # f0 = Mem[r1]
2.       mul.d f4, f0, f2       # f4 = f0 * f2
3.       s.d f4, 0(r1)          # Mem[r1] = f4
4.       daddiu r1, r1, #-8     # r1 = r1 – 8
5.       bne r1, r2, loop       # if r1 ≠ r2 goto loop
```

Ejemplo

- Suponiendo que:
  - Se especula que el ciclo se toma.
  - Se han emitido las instrucciones en el ciclo 2 veces.
  - El `l.d` y `mul.d` de la primera iteración ya se comprometieron y las demás instrucciones terminaron de ejecutarse.

ROB

	Reorder buffer				
Entry	Busy	Instruction	State	Destination	Value
1	no	L.D F0,0(R1)	Commit	F0	Mem[0 + Regs[R1]]
2	no	MUL.D F4,F0,F2	Commit	F4	#1 × Regs[F2]
3	yes	S.D F4,0(R1)	Write result	0 + Regs[R1]	#2
4	yes	DADDIU R1,R1,#-8	Write result	R1	Regs[R1] – 8
5	yes	BNE R1,R2,Loop	Write result		
6	yes	L.D F0,0(R1)	Write result	F0	Mem[#4]
7	yes	MUL.D F4,F0,F2	Write result	F4	#6 × Regs[F2]
8	yes	S.D F4,0(R1)	Write result	0 + #4	#7
9	yes	DADDIU R1,R1,#-8	Write result	R1	#4 – 8
10	yes	BNE R1,R2,Loop	Write result		

Banco de registros

- Las RS no se muestran porque no hay instrucciones ejecutando.

	FP register status								
Field	F0	F1	F2	F3	F4	F5	F6	F7	F8
Reorder #	6				7				
Busy	yes	no	no	no	yes	no	no	...	no

Ejemplo

- Suponer que el primer brinco no debió ser tomado.
- Las instrucciones anteriores al brinco se comprometen.
- Cuando el brinco llega al frente, el ROB se limpia y se comienzan a sacar instrucciones del camino correcto.
- En la práctica, al momento de saber que se adivinó mal se limpia la parte del ROB que corresponde al camino equivocado.

Resumen

- El algoritmo de Tomasulo se inventó para la IBM 360/91 (1960s), pero solo fue práctico de implementar en microprocesadores hasta principios de los 90s.
- Ahora ha sido ampliamente adoptado en los procesadores de emisión múltiple.
- Consigue un gran rendimiento sin requerir que el compilador sea adaptado para cada arquitectura.
- Los peligros WAW y WAR se eliminan porque las instrucciones se comprometen en orden.

Resumen

- Los peligros RAW se eliminan al no permitir que una instrucción continúe hasta que sus operandos están listos.
- Los peligros RAW en la memoria se eliminan ejecutando las cargas y los stores en orden.
- El scheduling dinámico es la forma preferida de scheduling usada actualmente en los procesadores de escritorio.