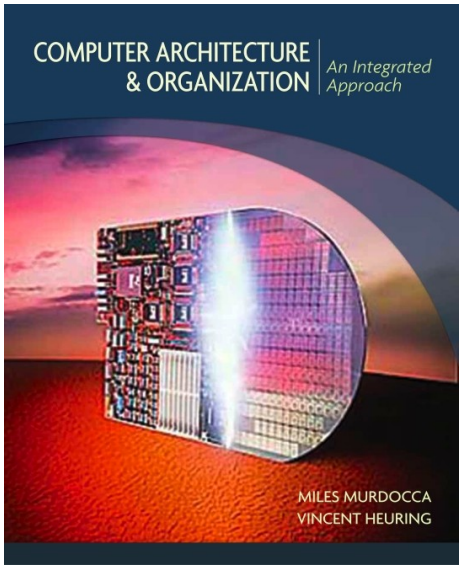
Organización de las Computadoras

Leonardo Giovanini



Microarquitecturas avanzadas

Contenidos

6.1 Ejecución paralela

6.2 Políticas de ejecución

6.3 Planificación dinámica

6.4 Implementación superescalar

Ejecución paralela

La microarquitectura de la CPU

Ejecución paralela

El ***paralelismo a nivel de instruction*** (ILP) es una medida de cuantas operaciones de un programa se pueden ejecutar simultáneamente.

El solapamiento en la ejecución de instrucciones se lo denomina como ***grado de paralelismo a nivel de instruction*.**

El ***paralelismo a nivel de máquina*** (MLP) es la capacidad de un procesador de tomar ventaja del ILP de un programa. Esto implica la ***disponibilidad de recursos*** para resolver los ***problemas generados por las dependencias***.

Para lograr un alto desempeño **necesitamos del ILP y MLP**.

Los compiladores y las microarquitecturas modernas ***identifican las oportunidades*** en las que se puede ***solapar la ejecución*** de instrucciones (0<ILP<1) para hacer mas eficiente su ejecución, lo cual está dado por la ***cantidad de dependencias en relación con la cantidad de instrucciones***.

La microarquitectura de la CPU

Ejecución paralela

Las técnicas utilizas para explotar el ILP de un programa

	Técnica	Reduce
Dinámicas	Planificación Dinámica	Paradas por riesgos de datos
	Predicción dinámica de saltos	Paradas por riesgos de control
	Lanzamiento múltiple Varias instrucciones por ciclo	CPI Ideal
	Especulación	Riesgos de datos y control
	Desambiguación dinámica de memoria	Paradas por riesgos de datos en memoria
Estáticas	Desenrollado de bucles	Paradas por riesgos de control
	Planificación por el compilador	Paradas por riesgos de datos
	Segmentación de software	CPI Ideal y Paradas por riesgos de datos
	Predicción estática y Especulación por el Compilador	CPI Ideal, paradas por riesgos de datos y control

La microarquitectura de la CPU

Ejecución paralela

PLANIFICACIÓN ESTÁTICA	PLANIFICACIÓN DINÁMICA
Menos Hardware	Complicación hardware
Compilador más difícil	Compilador no tiene que optimizar
Posibles problemas de herencia (compilador debe conocer endoarquitectura)	Transparente al usuario
Inconveniente: Dependencia compilación-rendimiento	El hardware extrae el rendimiento que puede en cada versión
Tamaño de código estático puede crecer ⇒ más fallos de caché	Tamaño de código estático no se toca
Ventaja: Ventana de instrucciones infinita (análisis global)	Defecto: Ventana de instrucciones limitada (fase IF) (análisis local)
El compilador no puede conocer: - valores de registros (dir. acceso) - predicción dinámica, etc.	En tiempo de ejecución se conoce más: - valores de registros (dir. acceso) - predicción dinámica, etc.
Puede eliminar instrucciones (de overhead u otras)	No puede eliminar instrucciones
Puede necesitar muchos registros de usuario	No necesita muchos registros de usuario (son internos, ocultos al usuario; ej. CISC)

La microarquitectura de la CPU

Ejecución paralela

Una forma de incrementar el ILP es dividiendo la ejecución de las instrucciones en tareas pequeñas y desacopladas (segmentacion).

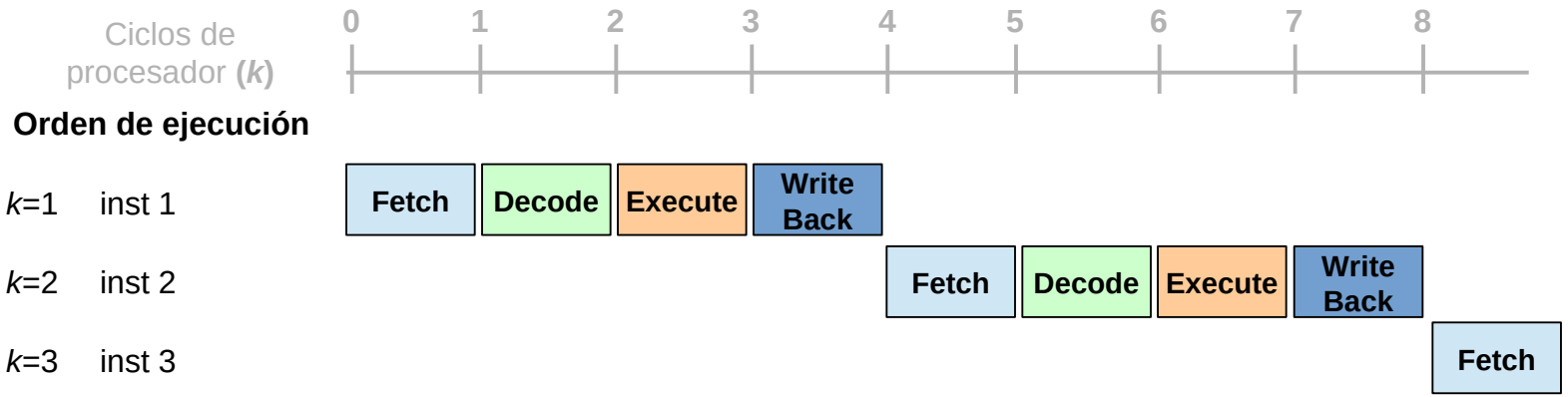
Durante la ejecucion de una instrucción hay cuatro actividades importantes:

- **Lectura de la instrucción** (Instruction *Fetch*) - una instrucción es leida de la memoria;
- **Lectura de los datos** (Data *Fetch*) – los datos necesarios para la ejecución de una instrucción son leidos de los registros y la memoria y se inicia su ejecución;
- **Ejecución de la instrucción** (*Execution*) – se completa la ejecución de las operaciones indicadas por la instrucción;
- **Escritura de la resultados** (*Commit*) – los resultados de la operación son almacenados en los registros o memoria.

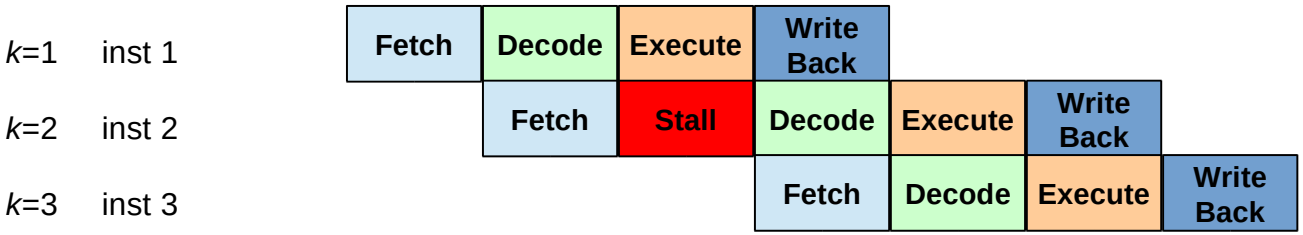
Una vez finalizadas estas actividades, el procesador procede a ejecutar una nueva instrucción

La microarquitectura de la CPU

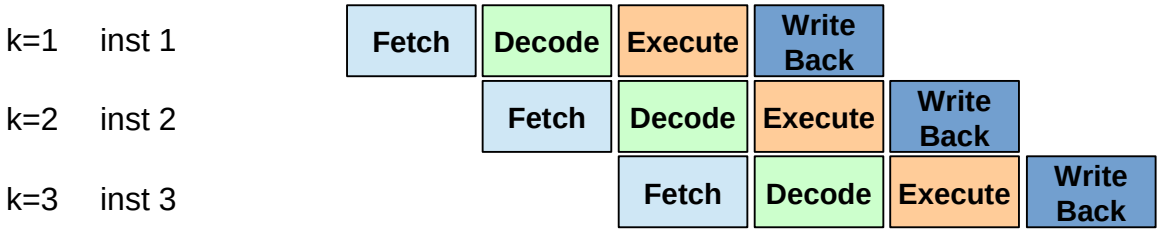
Ejecución paralela



Ejecución multiciclo (ILP = 0)



Ejecución segmentada real (0 < ILP < 1)



Ejecución segmentada ideal (ILP = 1)

La microarquitectura de la CPU

Ejecución paralela

Las dependencias son propias de los programas.

Determinar las dependencias es crítico para obtener el máximo paralelismo.

¿ Cuáles hay? ¿ A qué recursos afectan?

La **presencia de una dependencia** indica la posibilidad de **aparición de un riesgo**, pero la aparición de éste y la posible parada depende de las características de la microarquitectura.

La dependencias * Indican la posibilidad de un riesgo;

* Determinan el orden de cálculo de los resultados;

* Imponen un límite al paralelismo que es posible obtener.

- Los mecanismos de ejecución deben *preservar el orden del programa:* ***Mismo resultado que en ejecución secuencial;***
- ***Explotar todo el paralelismo posible sin afectar al resultado de la ejecución;***
- Para las dependencias de nombre ***eliminar la dependencia*** usando otros ***nombres***

La microarquitectura de la CPU

Ejecución paralela

Los tipos de dependencias que se pueden producir son:

Dependencias de datos

- Dependencia verdadera (Read-After-Write)

Dependencias de nombre

- Antidependencia (Write-After-Read);
- Dependencia de salida (Write-After-Write).

Fáciles de determinar para registros - Difíciles para direcciones de memoria

Debe conocer dependencias entre load y stores para permitir su reordenamiento

Dependencias de control

Cada instrucción depende de un conjunto de saltos y en general esta dependencia debe preservarse para **preservar el orden del programa**.

Las dependencias de control pueden violarse. Se pueden ejecutar instrucciones no debidas si **no afecta al resultado correcto del programa**

LO IMPORTANTE es preservar el comportamiento de las excepciones y el flujo de datos

Políticas de ejecución

La microarquitectura de la CPU

Politicas de ejecución

Para mejorar el ILP de un procesador se deben ***paralelizar estas actividades preservando el resultado final***.

Esto se puede lograr ***reduciendo la mayor cantidad posible*** de dependencias de datos y control de ejecución.

Esto se puede lograr ***cambiando el orden*** en que las instrucciones son leidas (Fetch), ejecutadas y los resultados actualizados (Writeback), pidiendo que no cambie el resultado.

Para implementar estas ideas, el hardware deberá:

- Mantener la verdadera dependencia (modo de flujo de datos);
- Mantener el comportamiento de excepción: y
- Encuentre el ILP dentro de una ventana de instrucciones (grupo) a partir de un predictor de saltos preciso.

La microarquitectura de la CPU

Politicas de ejecución

```
i1:  r2   = 4(r22)
i2:  r10 = 4(r25)
i3:  r10 = r2 + r10
i4:  4(r26) = r10
i5:  r14 = 8(r27)
i6:  r6   = (r22)
i7:  r5   = (r23)
i8:  r5   = r6 – r5
i9:  r4   = r14 * r5
i10: r15 = 12(r27)
i11: r7   = 4(r22)
i12: r8   = 4(r23)
i13: r8   = r7 – r8
i14: r8   = r15* r8
i15: r8   = r4 – r8
i16: (r28) = r8
```

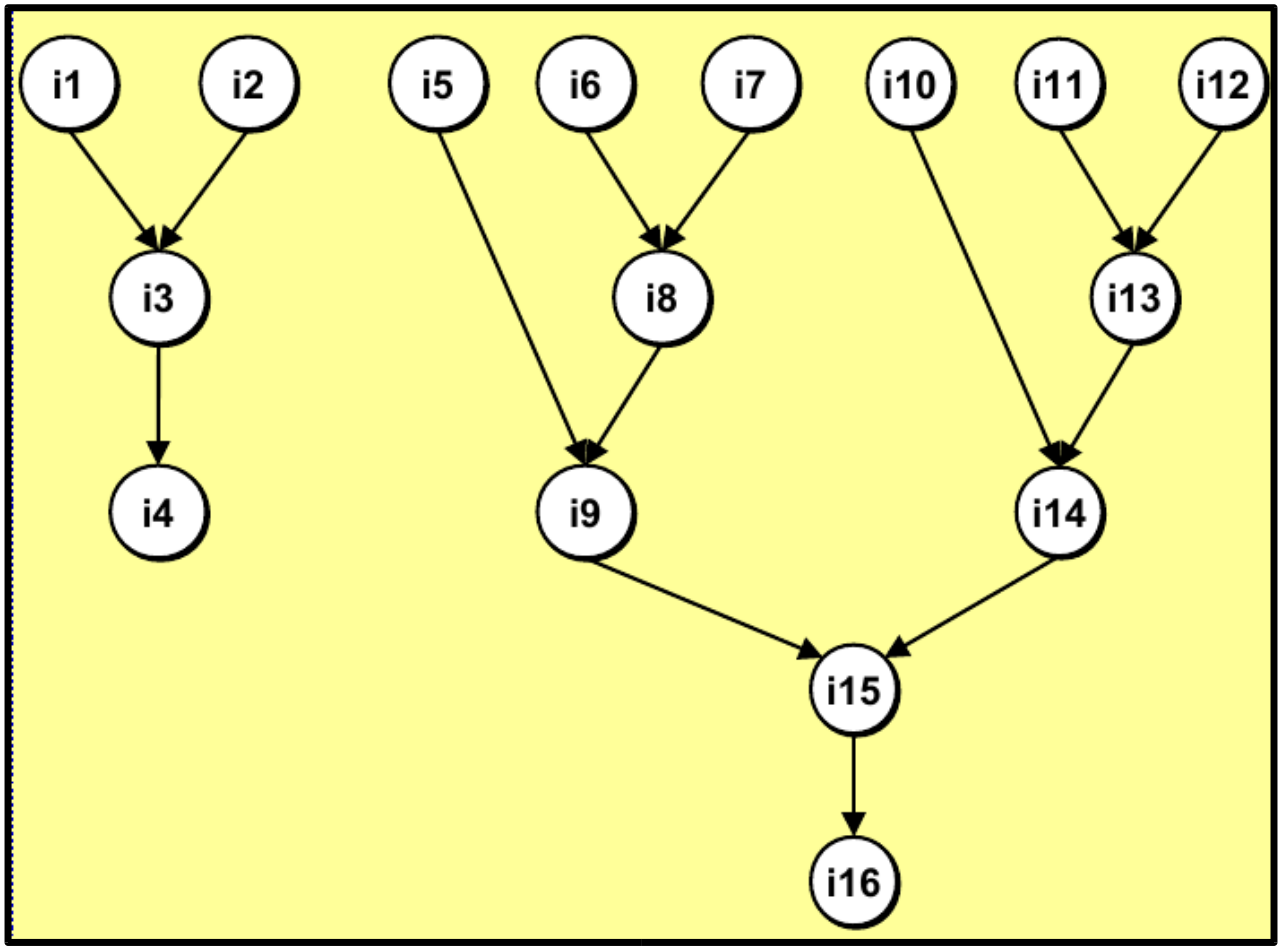


Gráfico de dependencias

La microarquitectura de la CPU

Politicas de ejecución

```
i1:  r2   = 4(r22)
i2:  r10 = 4(r25)
i3:  r10 = r2 + r10
i4:  4(r26) = r10
i5:  r14 = 8(r27)
i6:  r6   = (r22)
i7:  r5   = (r23)
i8:  r5   = r6 – r5
i9:  r4   = r14 * r5
i10: r15 = 12(r27)
i11: r7   = 4(r22)
i12: r8   = 4(r23)
i13: r8   = r7 – r8
i14: r8   = r15* r8
i15: r8   = r4 – r8
i16: (r28) = r8
```

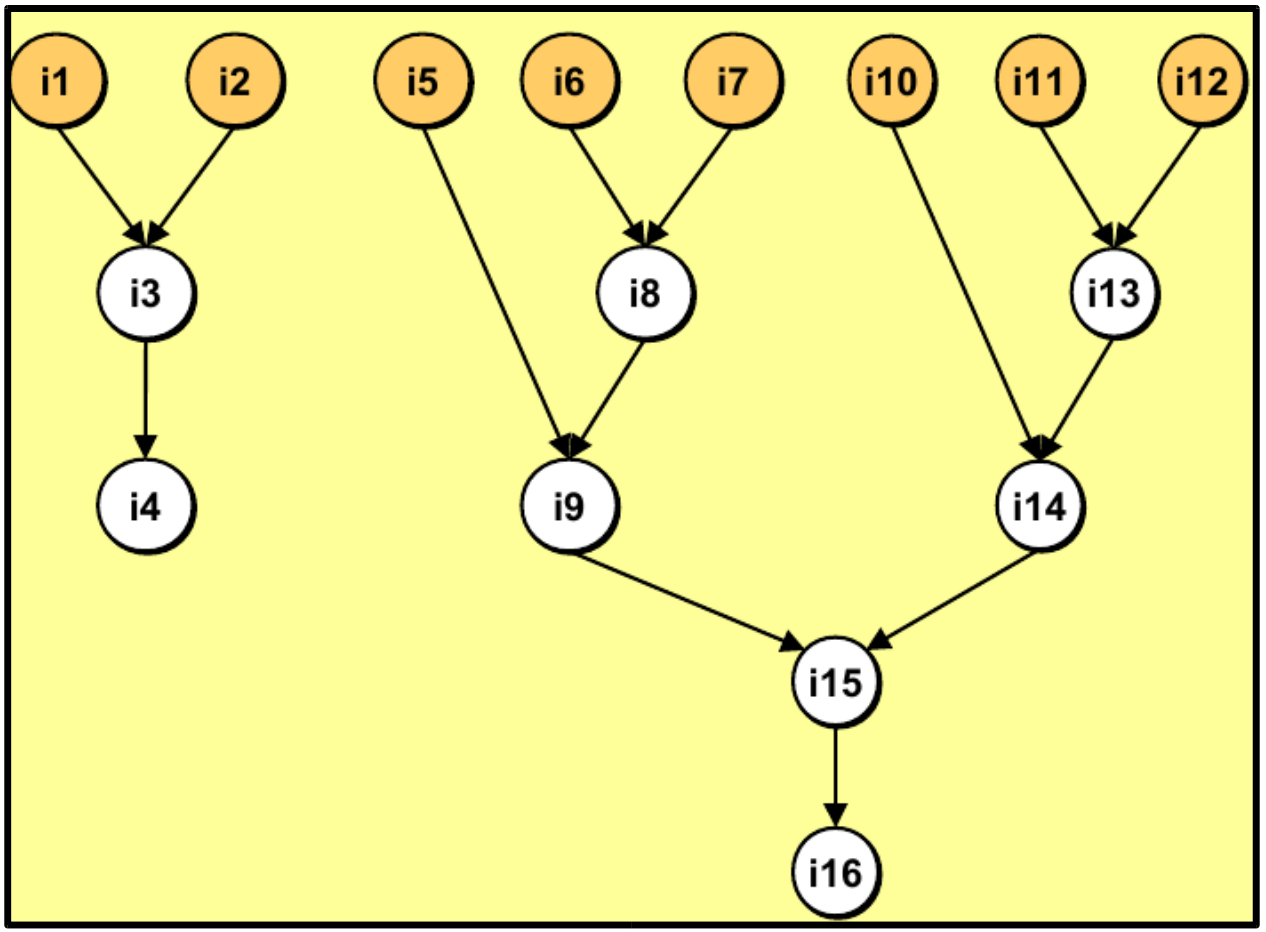


Gráfico de dependencias

La microarquitectura de la CPU

Politicas de ejecución

Las políticas de ejecución que se pueden implementar son

- **In-Order Issue – In-Order Completion** - Las instrucciones son leidas, ejecutadas y los resultados actualizados en *el orden en que son ejecutadas en el programa*:
  - Es muy ineficiente por la dependencias;
  - Las instrucciones deben ser detenidas (***stall***) y el sistema vaciado (***flush***).

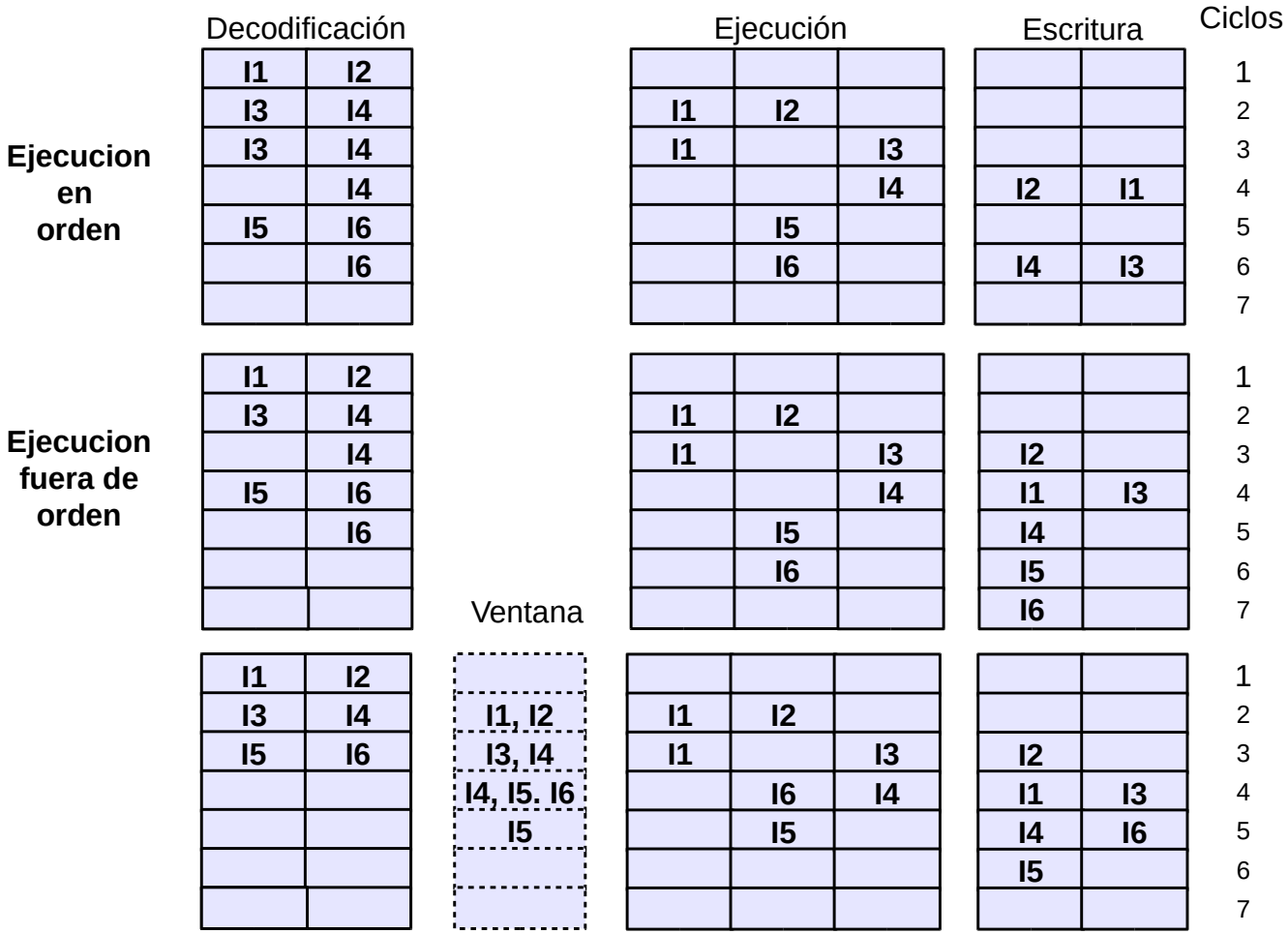
- **In-Order Issue – Out-of-Order Completion** - las instrucciones *son leidas en el orden en que son ejecutadas en el programa pero son ejecutadas y los resultados actualizados de acuerdo con la disponibilidad de recursos* y las dependencias existentes:
  - Es muy eficiente ya que evita las dependencias de datos;
  - Las instrucciones solo son detenidas (stall).

La microarquitectura de la CPU

Politicas de ejecución

Ejemplo: Consideremos el siguiente programa

I1 requiere de 2 ciclos para su ejecucion;
I3 & I4 compiten por la misma unidad funcional;
I5 depende del resultado de I4;
I5 & I6 compiten por la misma unidad funcional.



La microarquitectura de la CPU

Politicas de ejecución – Ejecucion fuera de orden

- **Out-of-Order Issue – Out-of-Order Completion** - Las instrucciones son leidas, ejecutadas y los resultados actualizados ***de acuerdo con la disponibilidad de recursos y las dependencias existentes***. Esta politica tiene como objetivo
  - ***Desacoplar*** la decodificacion de la ejecucion;
  - ***Continuar*** leyendo y decodificando hasta que la ventana este llena;
  - Ejecutar la instruccion ***cuando la unidad funcional esta disponible***.

La ejecución fuera de orden *rellena los huecos* de tiempo de las instrucciones que están listas para ejecutarse para después ***reordenar los resultados*** y ***aparentar*** que fueron procesadas de manera normal.

La forma en que las instrucciones son ordenadas en el código original a ejecutar se conoce como ***orden de programa***,

El orden en que el procesador maneja las instrucciones es el ***orden de datos***, siendo aquel en que los datos van quedando disponibles.

Planificación dinámica

La microarquitectura de la CPU

Planificación dinámica

Para implementar estas ideas se introduce el concepto de ***planificación dinámica*** de la ejecución de las instrucciones.

La ***planificación dinámica*** es un método de ejecución en el que el hardware determina qué instrucciones ejecutar, a diferencia de la planificación estática en la que el compilador determina el orden de ejecución.

En este método el procesador ejecuta las instrucciones fuera de orden teniendo en cuenta la disponibilidad de los operandos y recursos.

Los procesadores que utilizan planificación dinámica pueden aprovechar el paralelismo que no sería visible en el momento de la compilación. También son más versátiles, ya que el código no necesariamente tiene que volver a compilarse para ejecutarse de manera eficiente, ya que el ***hardware se ocupa de gran parte de la planificación***.

Este algoritmo permite

- ***Implementar*** la política de ejecución fuera de orden - finalización fuera de orden; y
- ***Verificar dependencias*** estructurales y de datos en la ***etapa de decodificación***.

La microarquitectura de la CPU

Planificación dinámica– Algoritmo de Scoreboard

El ***algoritmo de scoreboard*** es un método para implementar planificación dinámica que utiliza tablas (***scoreboard***) para

- Mantener un registro de las instrucciones que se recuperan, emiten y ejecutan;
- Determinar los recursos (unidades funcionales y operandos) que se utilizan y necesitan; y
- Seguir qué instrucción modifica qué registros.

El algoritmo usa esta información para planificar dinámicamente las instrucciones, determinando cuándo y dónde comienza y termina la ejecución de una instrucción, la cual pasa por cuatro etapas

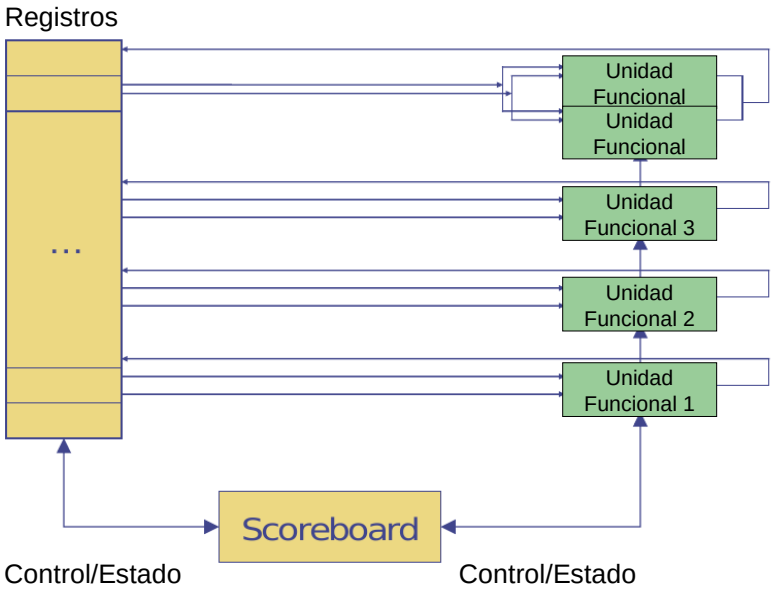
- **Issue** - en esta etapa se comprueba si hay una unidad funcional disponible y cualquier peligro WAW. Si detecta un peligro o no hay unidades disponibles se detiene la instrucción;
- **Operands fetch** - en esta etapa se verifica la disponibilidad de los operandos, indicando a la unidad funcional que lea los operandos de los registros y comience a ejecutarse. Se considera que un operando está disponible si no se va a escribir ninguna instrucción emitida en ese momento o si se está escribiendo en el registro;
- **Execution** - la unidad funcional notifica al algoritmo cuando finaliza la ejecución; y
- **Writeback** - una vez que se recibe la notificación de que una unidad funcional ha finalizado su ejecución comprueba si existe algún peligro potencial de WAR. Si encuentra un riesgo, el algoritmo le indica a la unidad funcional que se detenga hasta que desaparezca y la unidad funcional no está disponible para otras instrucciones.

La microarquitectura de la CPU

Planificación dinámica– Algoritmo de Scoreboard

Para controlar la ejecución de las instrucciones, el algoritmo mantiene tres tablas de estado

- **Estado de instrucción** - indica, para cada instrucción que se está ejecutando, en cuál de las etapas se encuentra;
- **Estado de unidad funcional** - indica el estado de cada unidad funcional y mantiene nueve campos
  - *Busy* - indica si la unidad se utiliza o no;
  - *Op* - indica la operación a realizar en la unidad;
  - *Fi* - indica el registro de destino de la operación;
  - *Fj*, *Fk* - indica los registros fuentes de la operación;
  - *Qj*, *Qk* - indica las unidades funcionales que producirán los registros Fj, Fk; y
  - *Rj*, Rk - indicadores que indican cuándo Fj, Fk están listos para y aún no se han leído.



- **Estado de registro** - indica, para cada registro, qué unidad de función escribirá los resultados.

La microarquitectura de la CPU

Planificación dinámica– Algoritmo de Scoreboard

Este algoritmo mejora la velocidad de ejecución en un factor de dos comparado con el compilador.

Sin embargo las limitaciones de este algoritmo son

- No se puede adelantar datos (primero escribe el registro y luego lo lee);
- Limitado a instrucciones en el bloque básico (ventana pequeña)
- Reducido número de unidades funcionales (riesgos estructurales)
- Detiene el procesador cuando hay riesgos WAW y WAR.

La microarquitectura de la CPU

Planificación dinámica

Para resolver estos inconvenientes se incorporan en la unidad de ejecución los siguientes elementos

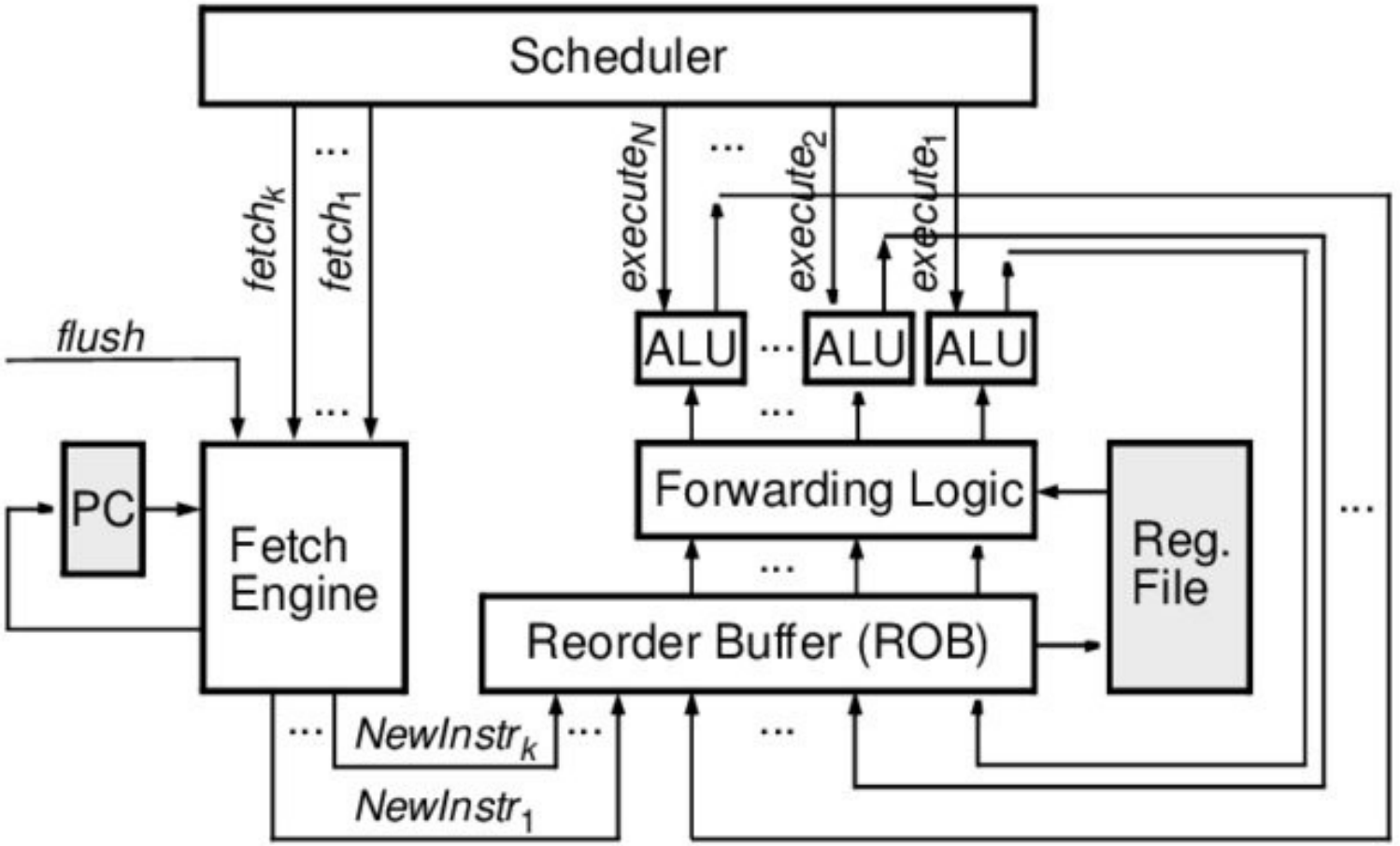
- **Buffer de Reordenamiento** – es una estructura indexada secuencialmente en base a instrucciones ejecutadas. Almacena los resultados de instrucciones cuya ejecución ha finalizado pero esperan de actualizar registros o son dependientes de un salto, permite el paso de operandos entre instrucciones especuladas con dependencia RAW;

- **Estaciones de reserva** – son registros de la unidad de ejecución que le permiten obtener y reutilizar datos tan pronto como se haya calculado en lugar de esperar a que se almacene en un registro o varias instrucciones deben escribir el mismo registro; y

- **Bus común de datos** – conecta las estaciones de reserva a las unidades funcionales de modo que puedan acceder al resultado sin involucrar un registro. Esto que permite que varias unidades que esperan un resultado continúen sin esperar a resolver el riesgo. Además, controlan cuándo se puede ejecutar una instrucción.

La microarquitectura de la CPU

Planificación dinámica



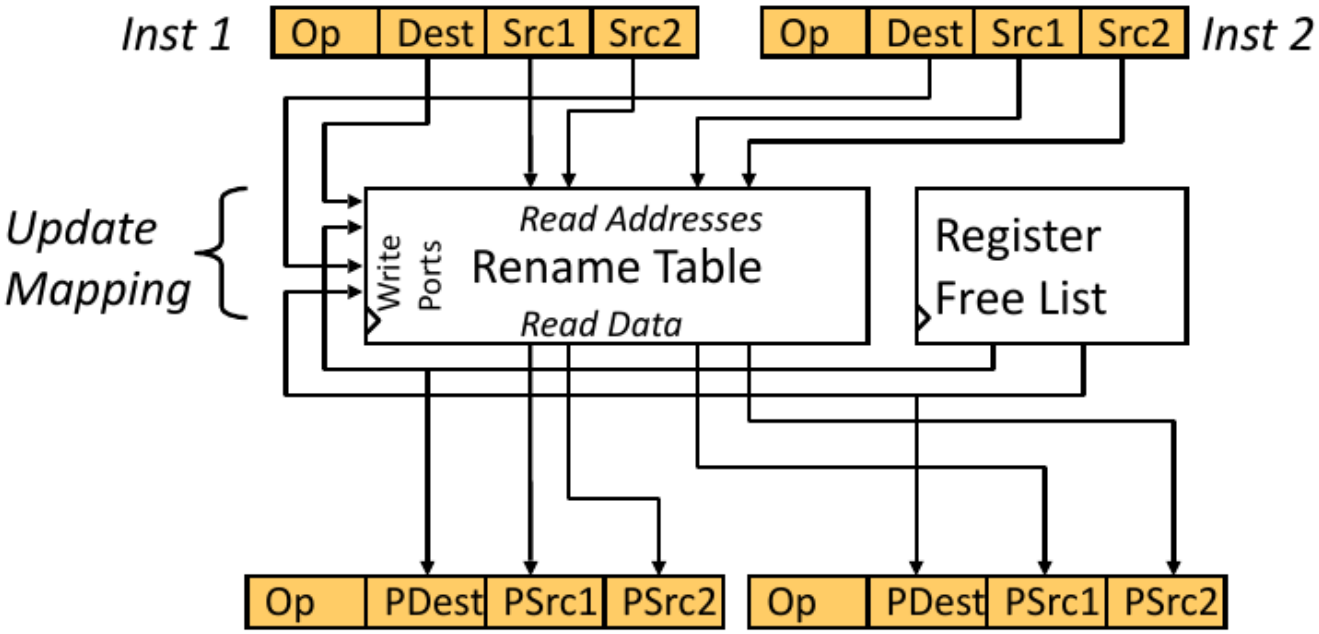
La microarquitectura de la CPU

Planificación dinámica – Algoritmo de Tomasulo

Renombrado de registro

El ***renombrado de registros*** es una técnica utilizada para eliminar las dependencias falsas de datos que surgen de la reutilización de registros mediante instrucciones sucesivas que no tienen dependencias reales de datos entre ellos.

Una técnica para implementar el renombrado de registros utiliza un ***archivo de renombrado*** que tiene un puerto de lectura para cada entrada y un puerto de escritura para cada salida de cada instrucción renombrada.

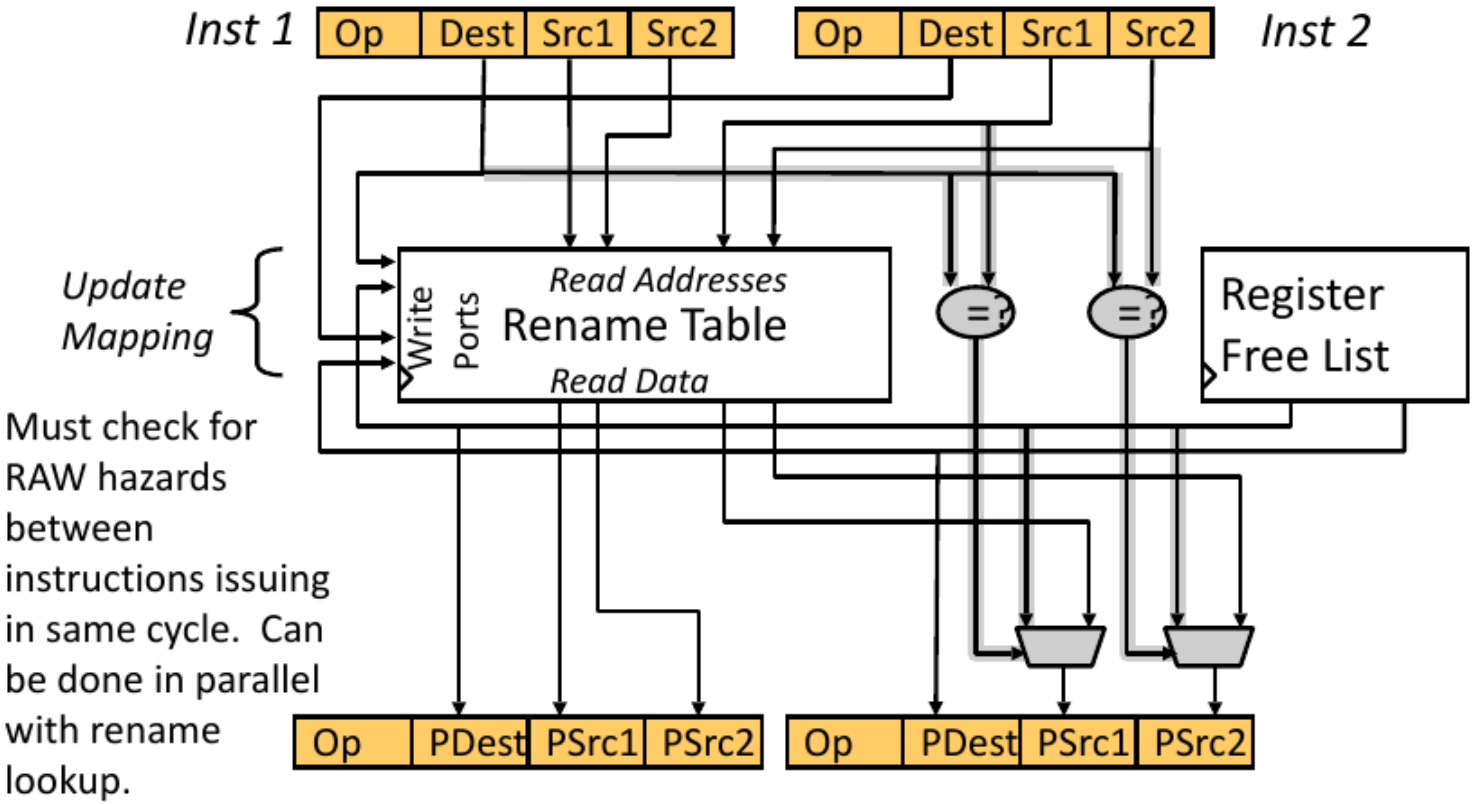


La microarquitectura de la CPU

Planificación dinámica – Algoritmo de Tomasulo

Renombrado de registro

Cuando se emite una instrucción a una unidad de ejecución, las etiquetas de los registros de origen se envían al archivo, donde los valores correspondientes a esas etiquetas se leen y envían a la unidad de ejecución.



La microarquitectura de la CPU

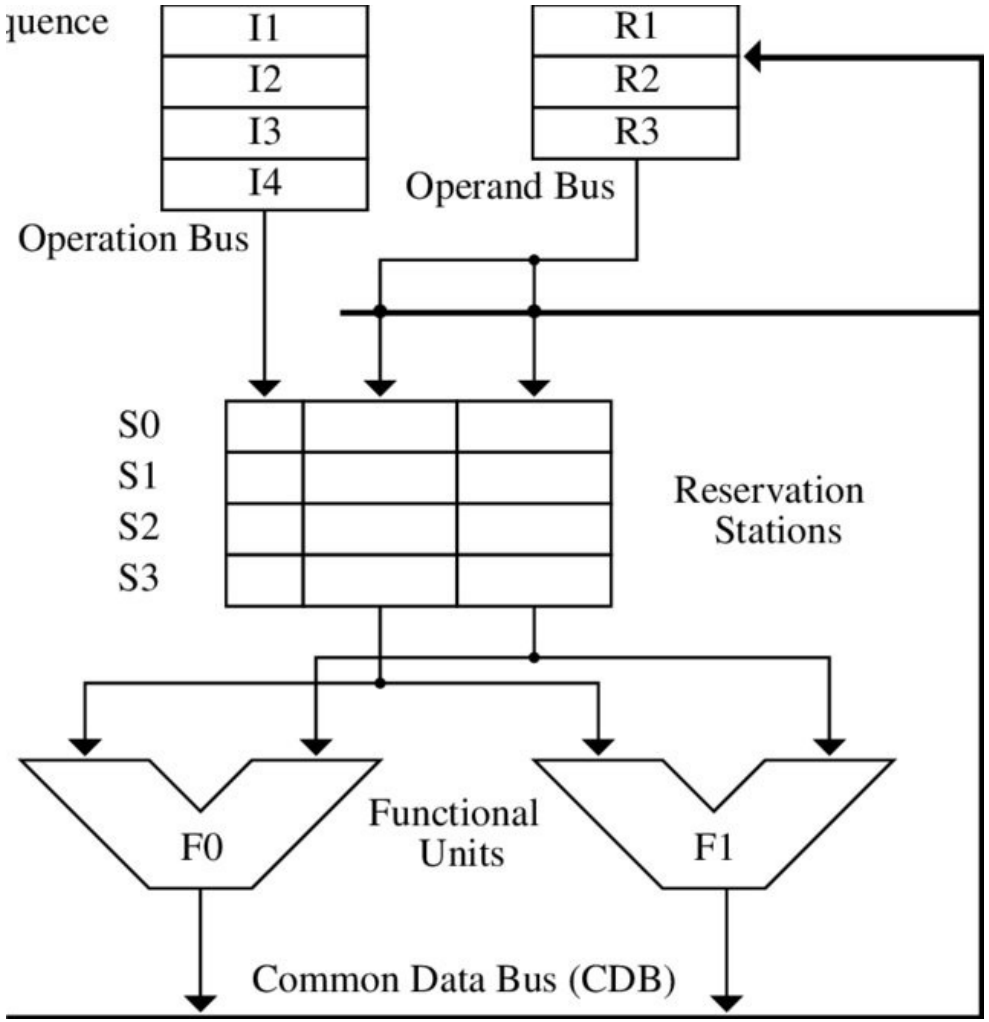
Planificación dinámica – Algoritmo de Tomasulo

Renombrado de registro

Otra técnica para implementar el renombrado de registros utiliza **estaciones de reserva**. Esta técnica se basa en el uso de registros asociativos, generalmente uno en las entradas de cada unidad de ejecución.

Cada operando de cada instrucción en una cola de emisión tiene un lugar para un valor en uno de estos registros.

Cuando se emite una instrucción a una unidad de ejecución, las entradas del archivo de registro correspondientes a la entrada de la cola de emisión se leen y envían a la unidad de ejecución.



La microarquitectura de la CPU

Planificación dinámica – Algoritmo de Tomasulo

El algoritmo de Tomasulo aprovecha el paralelismo a nivel de instruccióna partir de la ejecución de las siguiente tareas

Fetch la instrucción es leída desde la memoría;

Reserve la instrucción es enviada a la una cola de operaciones y los operandos a las estaciones de reserva;

Issue las instrucciones se emiten si todos los operandos y estaciones de reserva están listos o están detenidos. Los registros se renombran en este paso, eliminando los riesgos de WAR y WAW.

- Recupere la siguiente instrucción del tope de la cola de instrucciones. Si los operandos están en los registros y hay una unidad funcional disponible entonces
  - Emita la instrucción;
  - De lo contrario detenga la instrucción hasta que haya una estación libre;
- Si los operandos no están en los registros suponemos se utilizan valores virtuales determinados por la unidad funcional y se realiza un seguimiento de las unidades funcionales que producen el operando.

La microarquitectura de la CPU

Planificación dinámica – Algoritmo de Tomasulo

Execution las instrucciones se retrasan hasta que todos sus operandos estén disponibles, eliminando los riesgos de RAW. La correcta ejecución del programa se mantiene a través de un cálculo de dirección efectiva para evitar riesgos a través de la memoria.

- Si los operandos aún no están disponibles espere a que estén disponible en el CDB.
- Cuando todos los operandos están disponibles y si la instrucción es una carga o almacenamiento calcule la dirección efectiva cuando el registro base esté disponible y colóquelo en el búfer de carga / almacenamiento
  - Si la instrucción es de carga, ejecutela tan pronto como la unidad de memoria esté disponible;
  - Si la instrucción es de almacenamiento, entonces espere por el resultado que va ser almacendo antes de enviarlo a la unidad de memoria.
- Cuando todos los operandos están disponibles y si la instrucción es una operación de unidad lógica aritmética, ejecute la instrucción en la unidad funcional correspondiente

Writeback Commit los resultados de las operaciones de ALU se escriben en los registros y las operaciones de almacenamiento se escriben en la memoria.

- Si la instrucción fue una operación ALU y el resultado está disponible, entonces se escríbe en el CDB y de allí en los registros y en las estaciones de reserva que esperan este resultado;
- De lo contrario es una instrucción de almacenamiento, entonces escriba los datos en la memoria durante este periodo.

La microarquitectura de la CPU

Planificación dinámica – Algoritmo de Tomasulo

```
loop:   add   r4,   r4,   4
        ld    r2,   10(r4)        4 cycles lat
        add   r3,   r3,   r2
        sub   r1,   r1,   1
        bne   r1,   r0,   loop
```

RAV

r1	r2	r3	r4
1	1	1	1

Cycle 0

op	src1	src2	tgt	status

La microarquitectura de la CPU

Planificación dinámica – Algoritmo de Tomasulo

```
loop:   add   r4,   r4,    4
        ld    r2,   10(r4)        5 cycles lat
        add   r3,   r3,    r2
        sub   r1,   r1,    1
        bne   r1,   r0,    loop
```

RAV

r1	r2	r3	r4
1	1	1	**0**

Cycle 0

Ready to be executed

op	src1	src2	tgt	status
add	r4/1	NA/1	r4/0	Rdy

La microarquitectura de la CPU

Planificación dinámica – Algoritmo de Tomasulo

```
loop:   add   r4,   r4,   4
        ld    r2,   10(r4)
        add   r3,   r3,   r2
        sub   r1,   r1,   1
        bne   r1,   r0,   loop
```

Notify those waiting for R4

RAV

r1	r2	r3	r4
1	**0**	1	**1**

op	src1	src2	tgt	status
add	r4/1	NA/1	r4	Exec
ld	r4/1	NA/1	r2	Rdy

R4 gets produced now

La microarquitectura de la CPU

Planificación dinámica – Algoritmo de Tomasulo

```
loop:   add   r4,   r4,   4
        ld    r2,   10(r4)
        add   r3,   r3,   r2
        sub   r1,   r1,   1
        bne   r1,   r0,   loop
```

Result available @ cycle 6

RAV

r1	r2	r3	r4
1	0	**0**	1

Wait for r2

op	src1	src2	tgt	status
add	r4/1	NA/1	r4	Cmtd
ld	r4/1	NA/1	r2	Exec
add	r3/1	r2/0	r3	Wait

La microarquitectura de la CPU

Planificación dinámica – Algoritmo de Tomasulo

```
loop:   add   r4,   r4,   4
        ld    r2,   10(r4)
        add   r3,   r3,   r2
        sub   r1,   r1,   1
        bne   r1,   r0,   loop
```

Result available @ cycle 6

RAV

r1	r2	r3	r4
0	0	0	1

op	src1	src2	tgt	status
add	r4/1	NA/1	r4	Cmtd
ld	r4/1	NA/1	r2	Exec
add	r3/1	r2/0	r3	Wait
sub	r1/1	NA/1	r1	Rdy

Wait for r2

No dependences

La microarquitectura de la CPU

Planificación dinámica – Algoritmo de Tomasulo

```
loop:   add   r4,   r4,   4
        ld    r2,   10(r4)
        add   r3,   r3,   r2
        sub   r1,   r1,   1
        bne   r1,   r0,   loop
```

Result available @ cycle 6

RAV

r1	r2	r3	r4
1	0	0	1

op	src1	src2	tgt	status
add	r4/1	NA/1	r4	Cmtd
ld	r4/1	NA/1	r2	Exec
add	r3/1	r2/0	r3	Wait
sub	r1/1	NA/1	r1	Exec
bne	r1/1	r0/1	NA	Rdy

Wait for r2

r1 produced now
Notify consumers

r1 will be available next cycle

La microarquitectura de la CPU

Planificación dinámica – Algoritmo de Tomasulo

```
loop:   add   r4,   r4,   4
        ld    r2,   10(r4)
        add   r3,   r3,   r2
        sub   r1,   r1,   1
        bne   r1,   r0,   loop
```

RAV

r1	r2	r3	r4
1	0	0	1

Result available @ cycle 6

op	src1	src2	tgt	status
add	r4/1	NA/1	r4	Cmtd
ld	r4/1	NA/1	r2	Exec
add	r3/1	r2/0	r3	Wait
sub	r1/1	NA/1	r1	Compl
bne	r1/1	r0/1	NA	Exec

Wait for r2

Completed

executing

La microarquitectura de la CPU

Planificación dinámica – Algoritmo de Tomasulo

```
loop:   add   r4,   r4,   4
        ld    r2,   10(r4)
        add   r3,   r3,   r2
        sub   r1,   r1,   1
        bne   r1,   r0,   loop
```

RAV

r1	r2	r3	r4
1	**1**	0	1

Result available @ cycle 6
Notify consumers

op	src1	src2	tgt	status
add	r4/1	NA/1	r4	Cmtd
ld	r4/1	NA/1	r2	Exec
add	r3/1	r2/**1**	r3	Rdy
sub	r1/1	NA/1	r1	Compl
bne	r1/1	r0/1	NA	Exec

Wait for r2

Completed

executing

La microarquitectura de la CPU

Planificación dinámica – Algoritmo de Tomasulo

```
loop:   add   r4,   r4,   4
        ld    r2,   10(r4)
        add   r3,   r3,   r2
        sub   r1,   r1,   1
        bne   r1,   r0,   loop
```

RAV

r1	r2	r3	r4
1	1	1	1

Notify consumers

op	src1	src2	tgt	status
add	r4/1	NA/1	r4	Cmtd
ld	r4/1	NA/1	r2	Cmtd
add	r3/1	r2/1	r3	Exec
sub	r1/1	NA/1	r1	Compl
bne	r1/1	r0/1	NA	Compl

Executing

Completed

La microarquitectura de la CPU

Planificación dinámica – Algoritmo de Tomasulo

```
loop:   add   r4,   r4,   4
        ld    r2,   10(r4)
        add   r3,   r3,   r2
        sub   r1,   r1,   1
        bne   r1,   r0,   loop
```

RAV

r1	r2	r3	r4
1	1	1	1

op	src1	src2	tgt	status
add	r4/1	NA/1	r4	Cmtd
ld	r4/1	NA/1	r2	Cmtd
add	r3/1	r2/1	r3	Cmtd
sub	r1/1	NA/1	r1	Cmtd
bne	r1/1	r0/1	NA	Cmtd

La microarquitectura de la CPU

Planificación dinámica – Algoritmo de Tomasulo

Los conceptos de estaciones de reserva, renombrado de registro y el bus de datos común en el algoritmo de Tomasulo presentan avances significativos en el diseño de computadoras de alto rendimiento.

Las estaciones de reserva asumen la responsabilidad de esperar a los operandos en presencia de dependencias de datos y otras inconsistencias, como la variación del tiempo de acceso al almacenamiento y velocidades del circuito, liberando así las unidades funcionales. En particular, el algoritmo es más tolerante a los errores de caché. Además, los programadores se liberan de la implementación optimizadas del código. Esto es el resultado del trabajo coordinado del bus de datos común y la estación de reserva para preservar las dependencias y fomentar la concurrencia.

Al rastrear los operandos para obtener instrucciones en las estaciones de reserva y al renombrar registros, el algoritmo minimiza los riesgos RAW y elimina los riesgos WAW y WAR. Esto mejora el rendimiento al reducir el tiempo perdido que, de otro modo, se requeriría para las paradas.

El algoritmo no se limita a una microarquitectura específica, de modo que puede ser adoptado más ampliamente por diferentes microarquitecturas. Además, el algoritmo se extiende fácilmente para permitir la predicción de saltos.

La microarquitectura de la CPU

Planificación dinámica – Algoritmo de Tomasulo

Las ventajas e inconvenientes del algoritmo de Tomasulo pueden resumirse a

Resumen de ventajas e inconvenientes

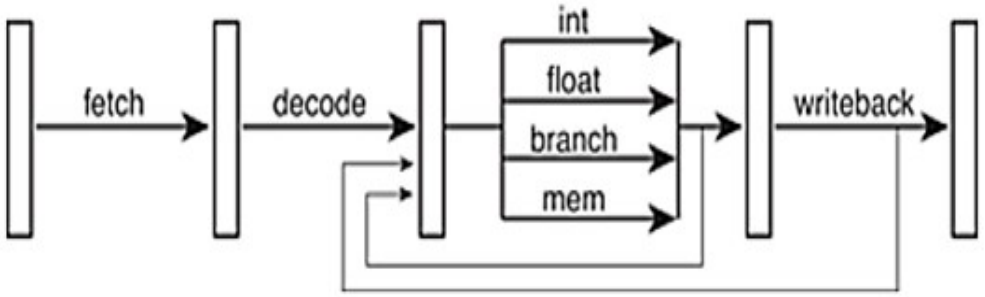
- Elimina el cuello de botella de los registros;
- Evita las dependencias de datos y nombres (WAR y WAW);
- Permite el desenrrollado del algoritmo en hardware;
- Introduce ejecución especulativa;
- Complejidad;
- Muchos cargas de registros asociativas por ciclo;
- Excepciones imprecisas.

Implementación superescalar

La microarquitectura avanzada

Implementación superescalar

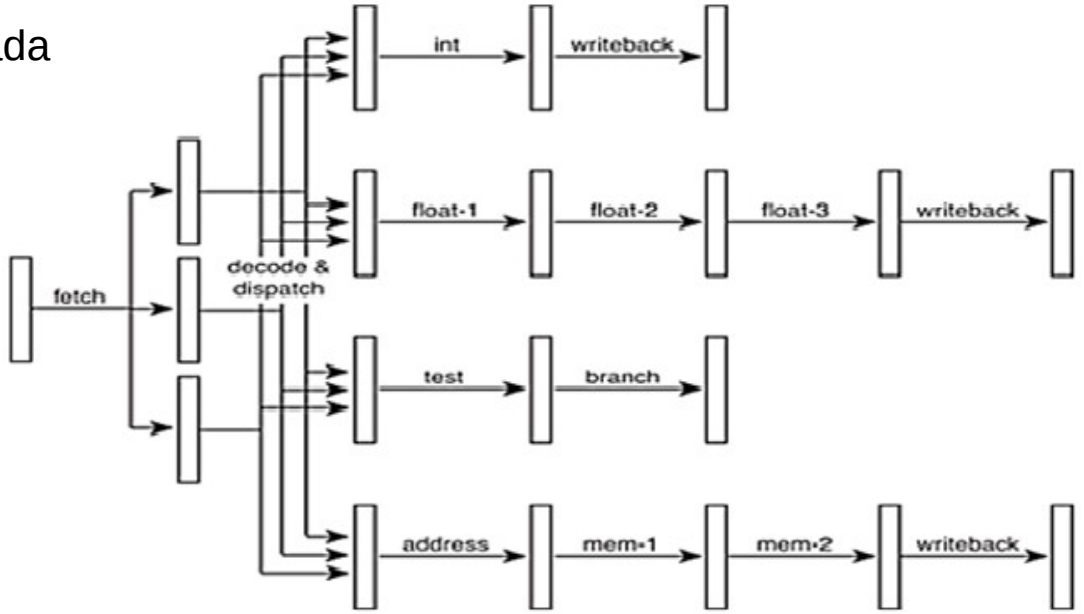
Las microarquitecturas y técnicas que permiten mejorar el ILP de los procesadores son:



Microarquitectura segmentada

Segmentación: descomponer las instrucciones en subtareas de manera que diferentes subtareas puedan ejecutarse al mismo tiempo;

Arquitectura superescalar: incrementar los recursos (unidades funcionales y registros) disponibles, de modo de la ejecución de las instrucciones individuales sea realizada en diferentes partes del procesador.

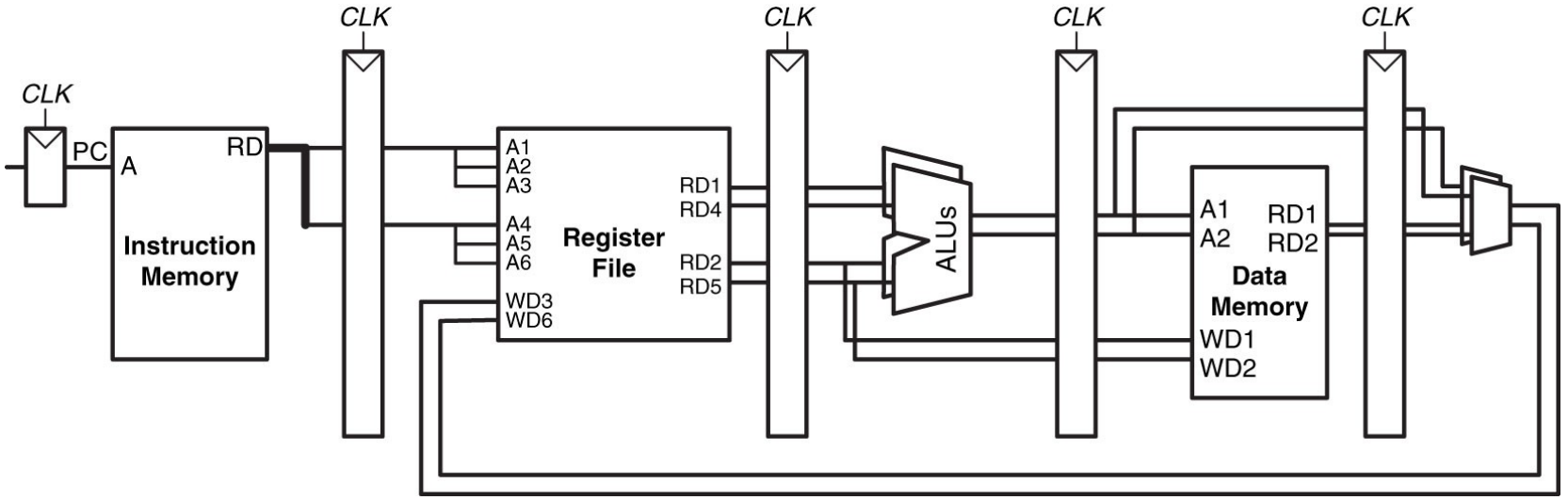


Microarquitectura superscalar

La microarquitectura avanzada

Implementación superescalar

El termino **superescalar** es una microarquitectura que mejora la performance de la ejecucion de instrucciones escalares a partir del aumento de recursos (registros y unidades de ejecucion) que operan de manera independiente, implementado paralelismo a nivel de maquina (MLP).

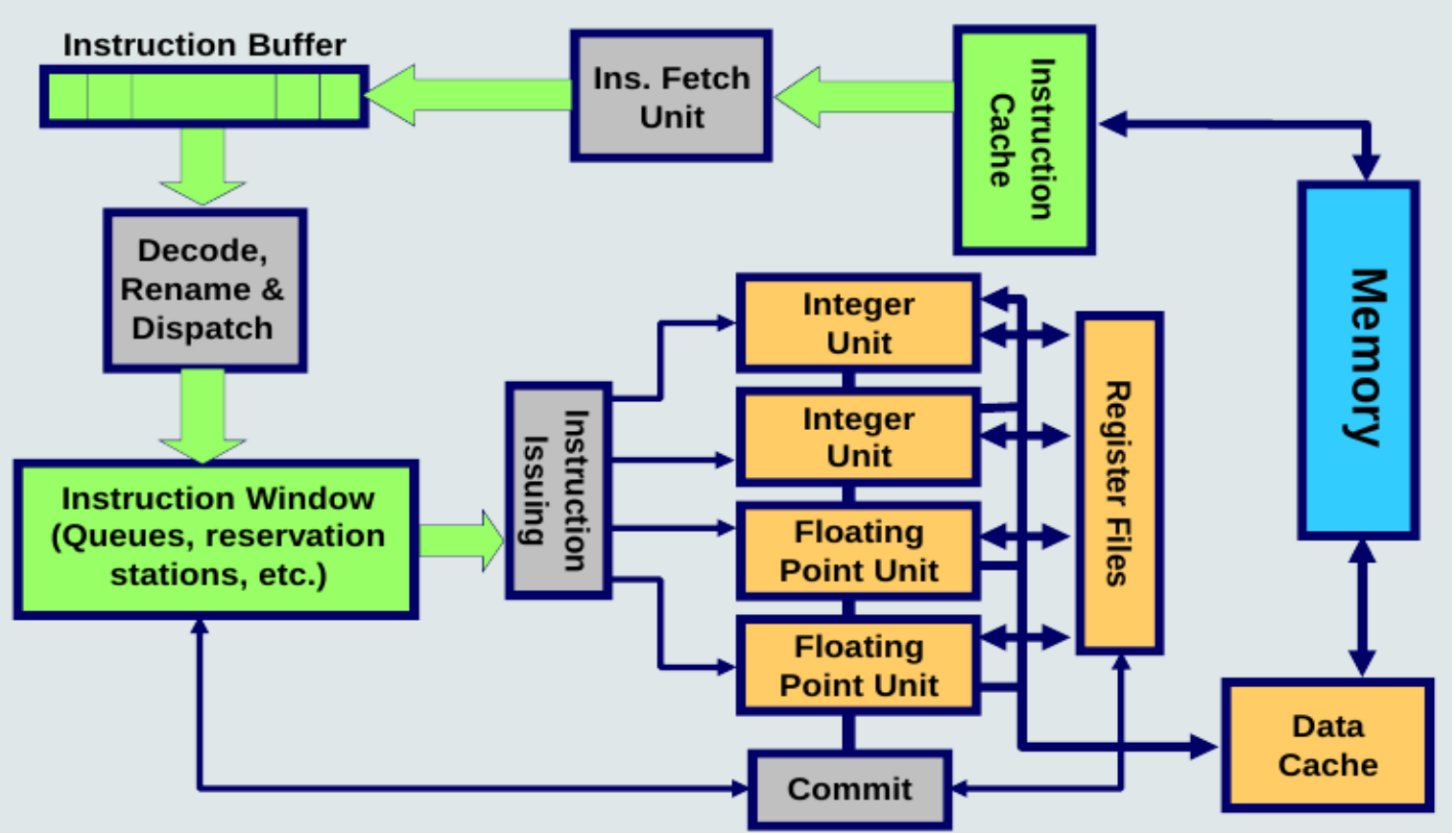


Cada pipeline consiste de multiples etapas que pueden ejecutar multiples instrucciones al mismo tiempo.

El uso de multiples pipelines introduce un nuevo nivel de paralelismo, permitiendo el procesamiento simultaneo de multiples grupos (streams) de instrucciones.

La microarquitectura avanzada

Implementación superescalar

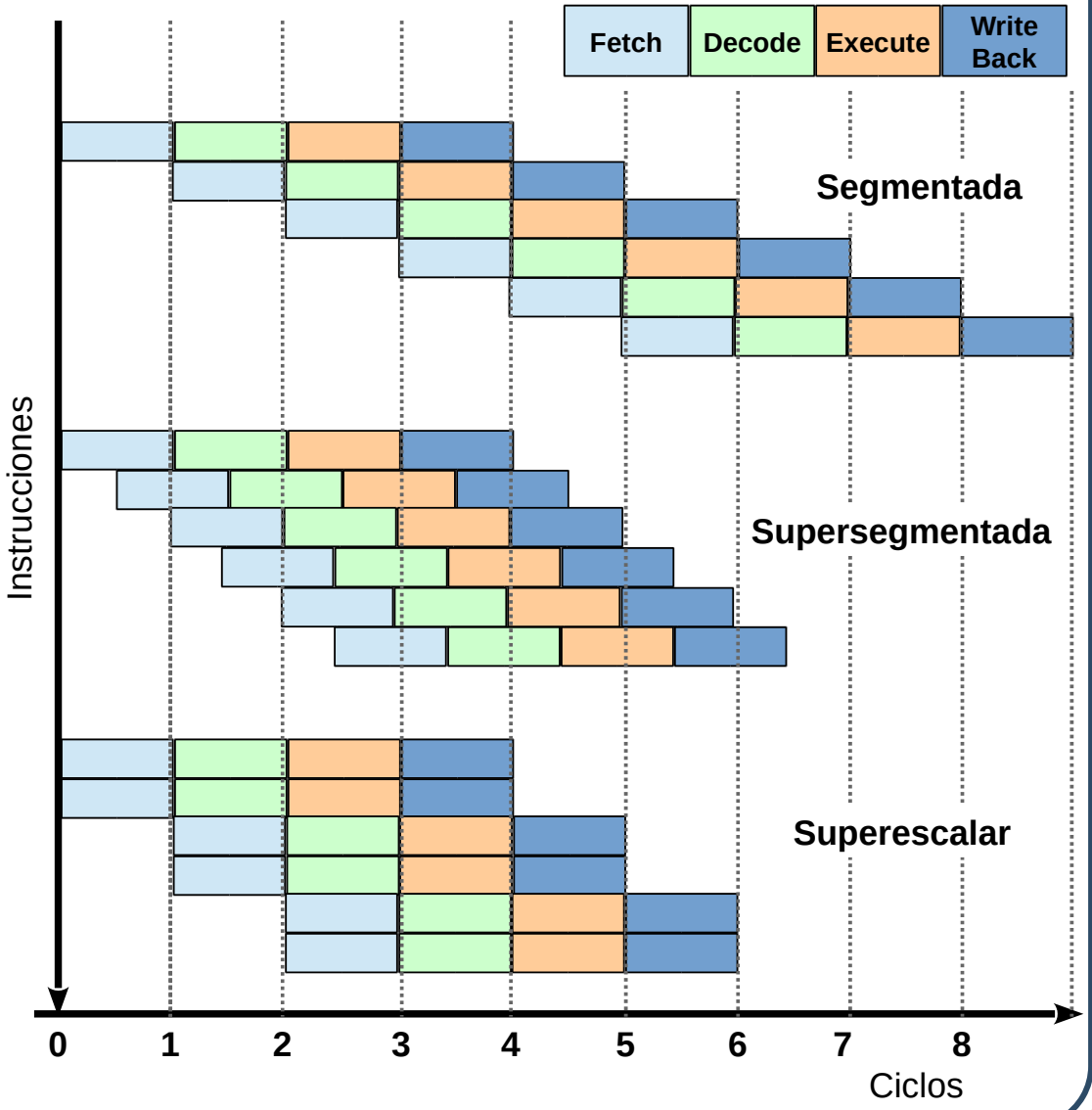


La microarquitectura avanzada

Implementación superescalar

Un procesador superescalar lee múltiples instrucciones al mismo tiempo. Intentando encontrar instrucciones que puedan ejecutarse de manera independiente.

La esencia del enfoque superescalar es la posibilidad de ejecutar instrucciones en paralelo en diferentes datapath.



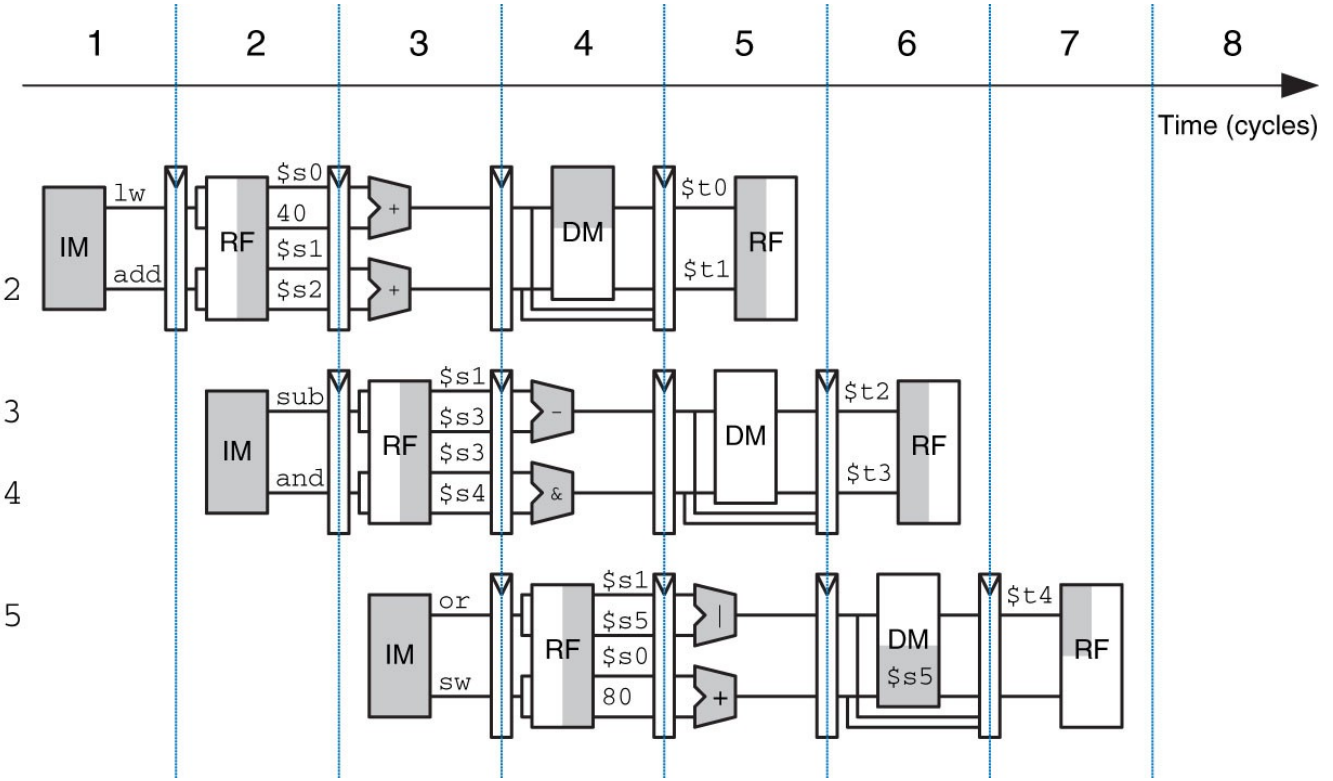
La microarquitectura avanzada

Implementación superescalar

```
lw  $t0, 40($s0)
add $t1, $s1, $s2

sub $t2, $s1, $s3
and $t3, $s3, $s4

or  $t4, $s1, $s5
sw  $s5, 80($s0)
```



La microarquitectura avanzada

Implementación superescalar

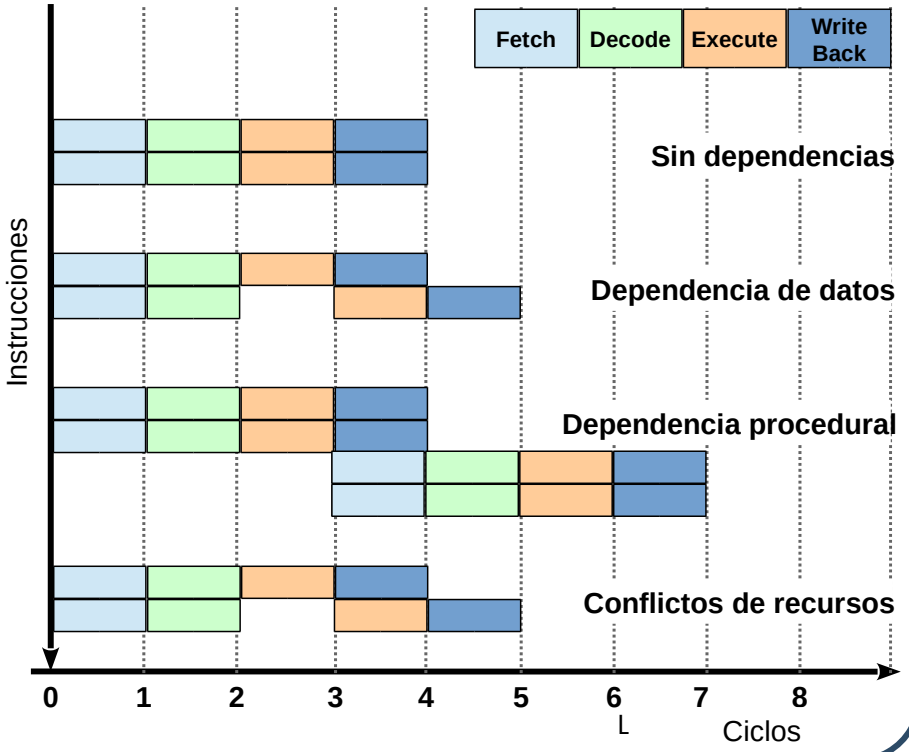
En esta microarquitectura el ILP se incrementa a partir de combinar

- **Optimización en el compilador:** los codigos de programa generados se optimizan para maximizar el ILP teniendo en cuenta las caracteristicas del procesador que ejecutara el codigo; y

Técnicas de hardware: los procesadores se diseñan par minimizar los efectos de las dependencias que quedan en el programa.

Esta mejora esta limitada por:

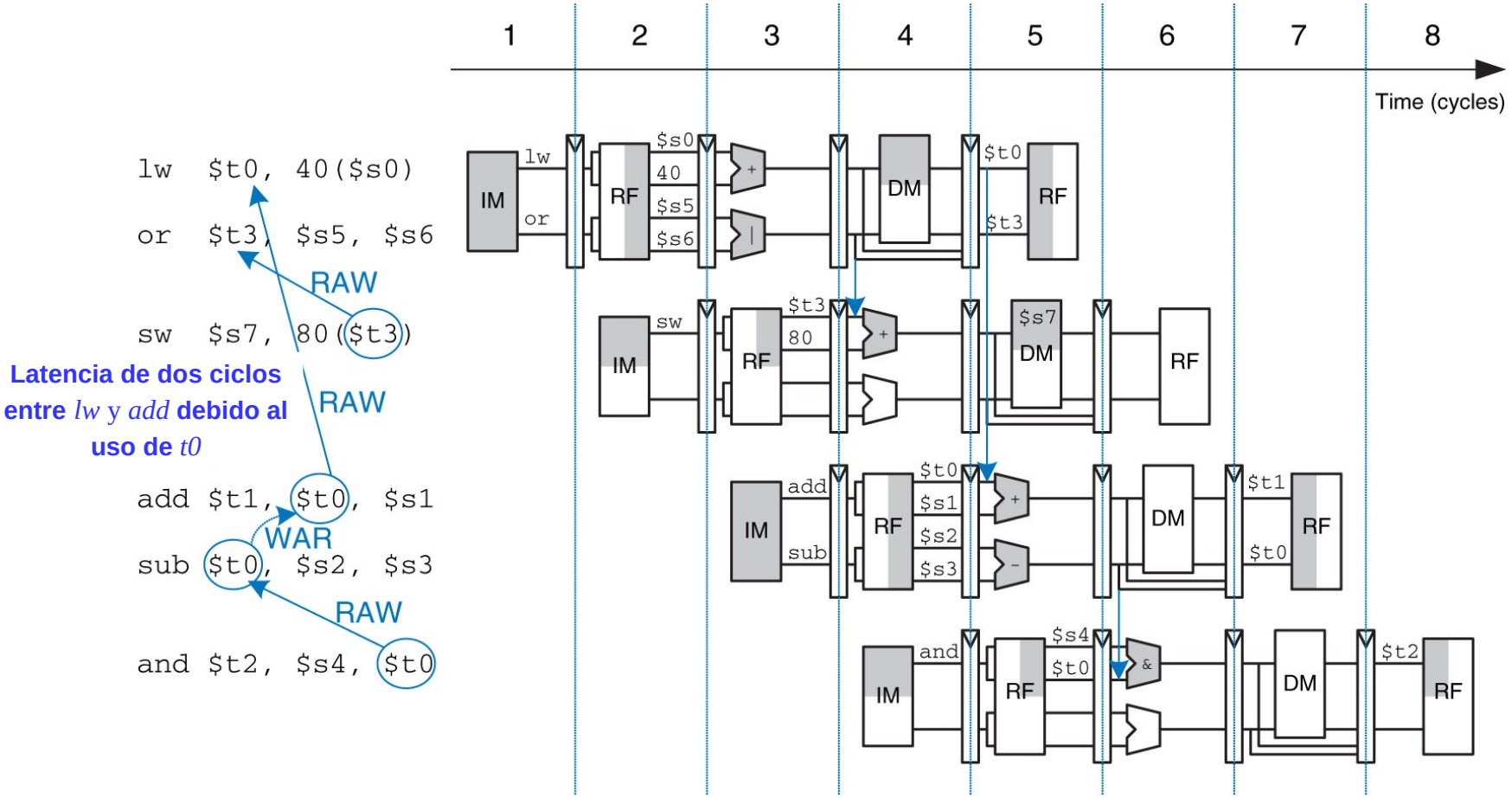
- **Dependencia de lectura-escritura:** no se puede ejecutar la siguiente instruccion por que depende del resultado anterior;
- **Dependencia de procedimiento:** no se puede ejecutar en paralelo instruciones antes y despues de un salto;
- **Conflictos de recursos:** dos o mas instruccions requieren el acceso al mismo recurso al mismo tiempo;



La microarquitectura avanzada

Implementación superescalar

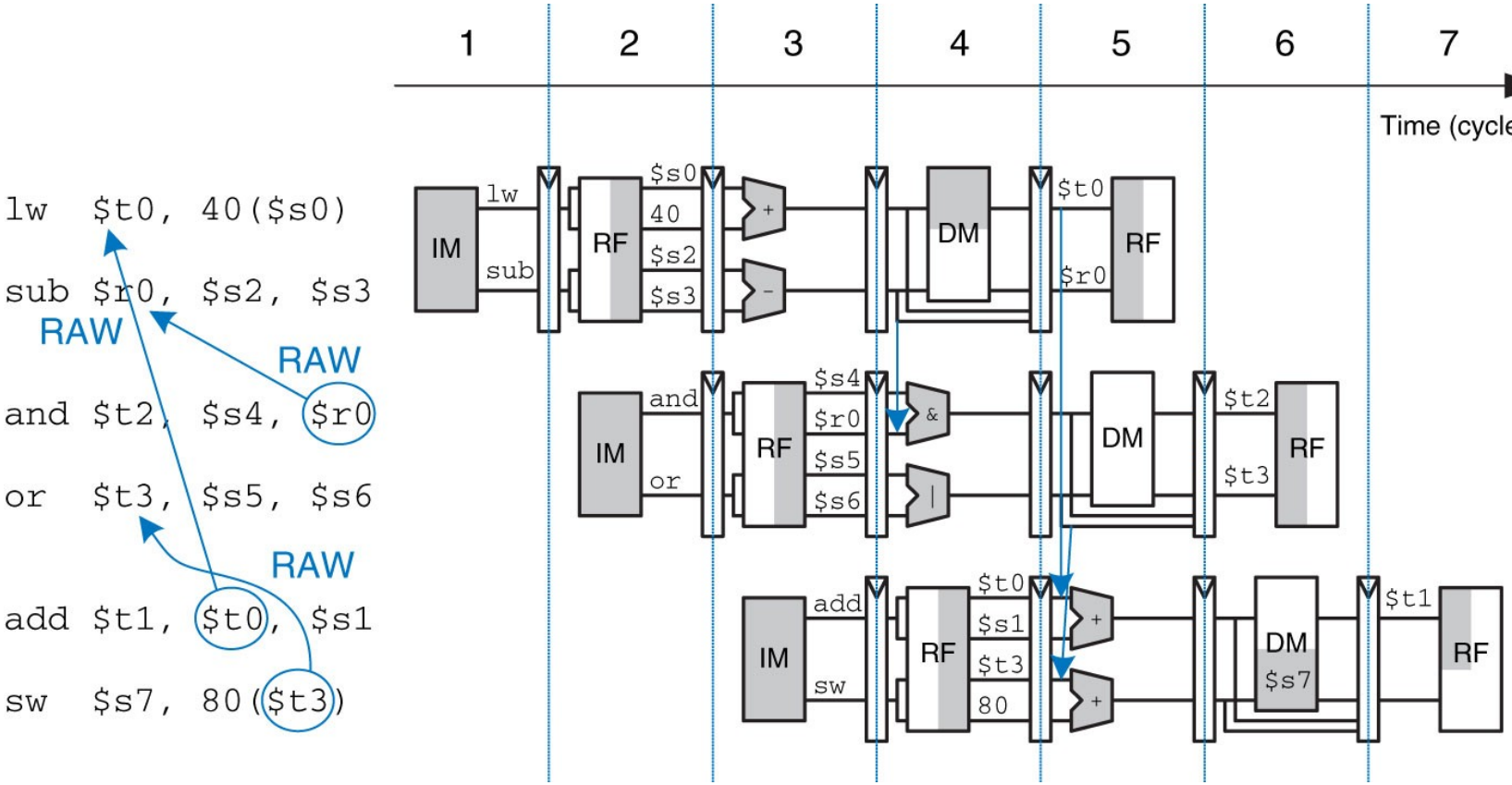
Ejecución **fuera-de-orden** de un programa con dependencias de datos



La microarquitectura avanzada

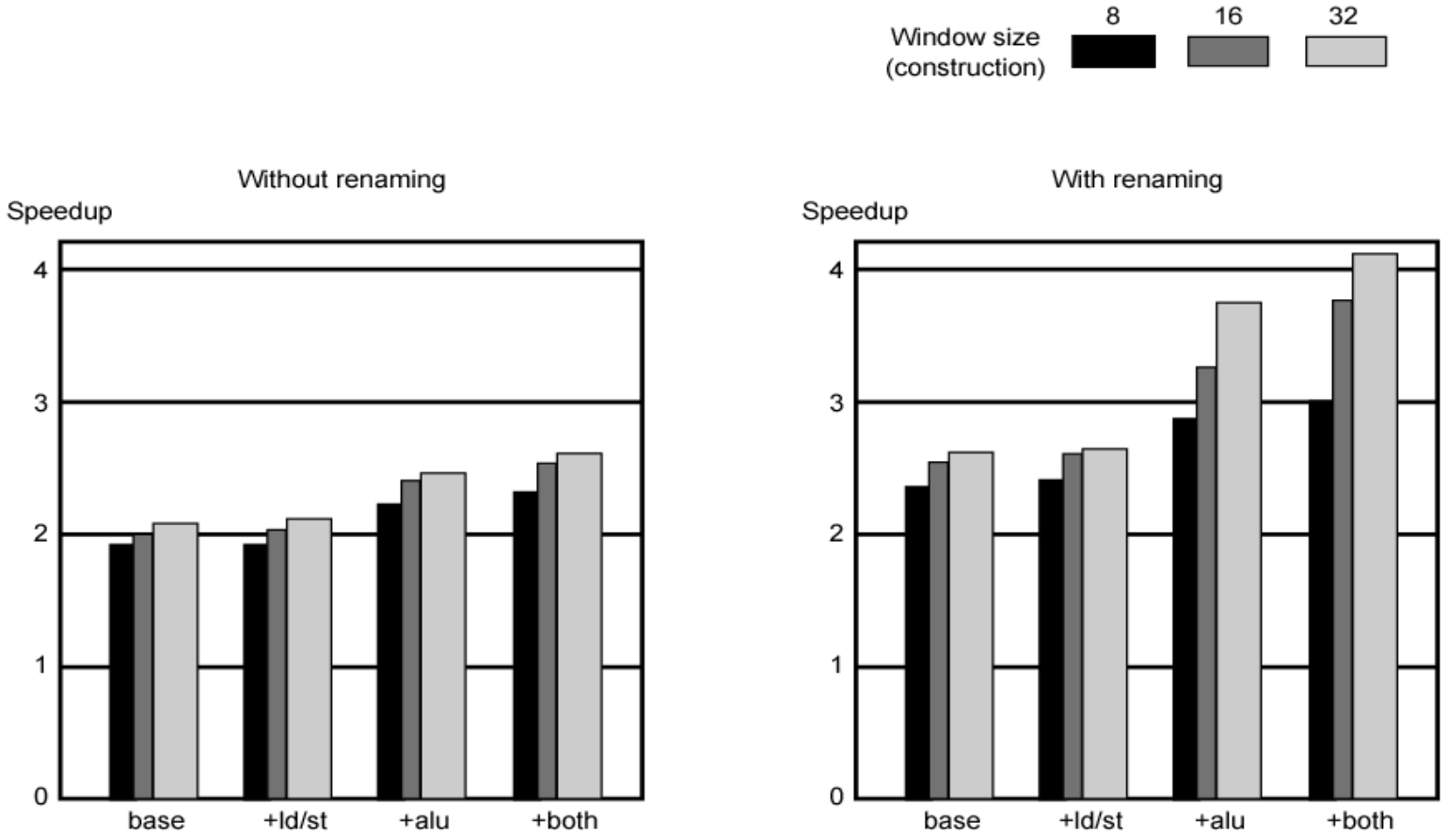
Implementación superescalar

Ejecución **fuera-de-orden** de un programa utilizando **renombrado de registros**



La microarquitectura avanzada

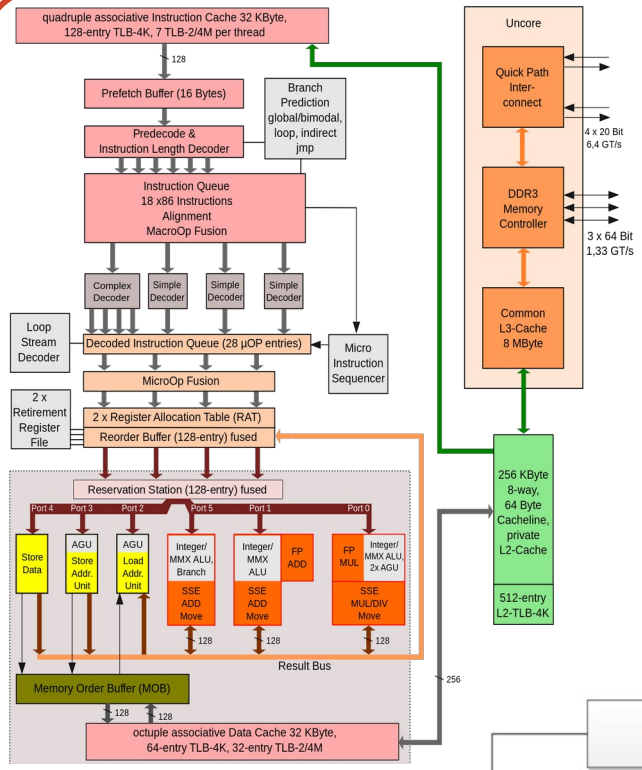
Implementación superescalar



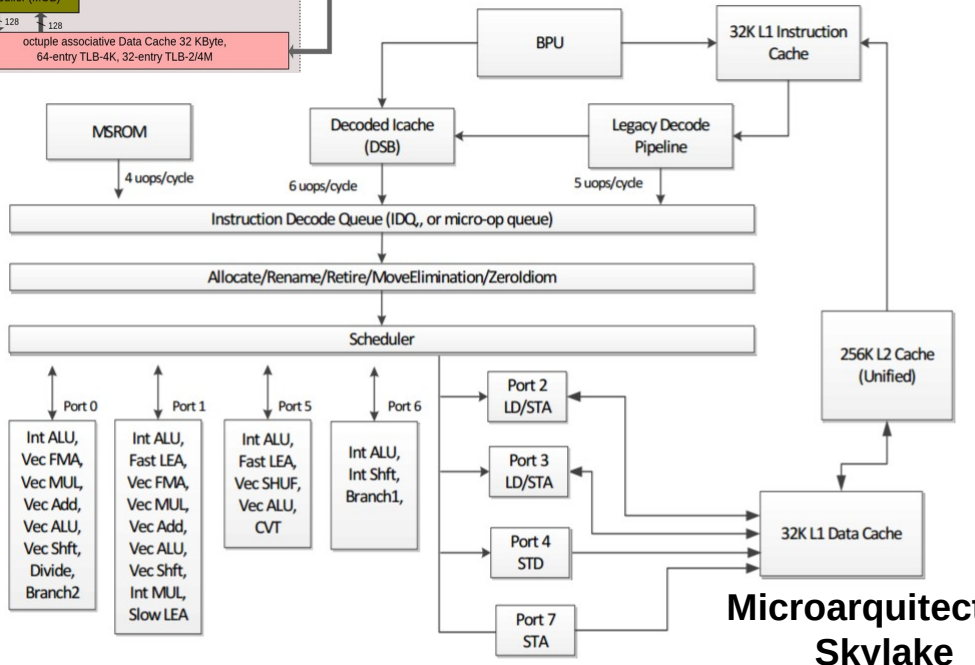
La microarquitectura avanzada

Procesadores superescalar comerciales

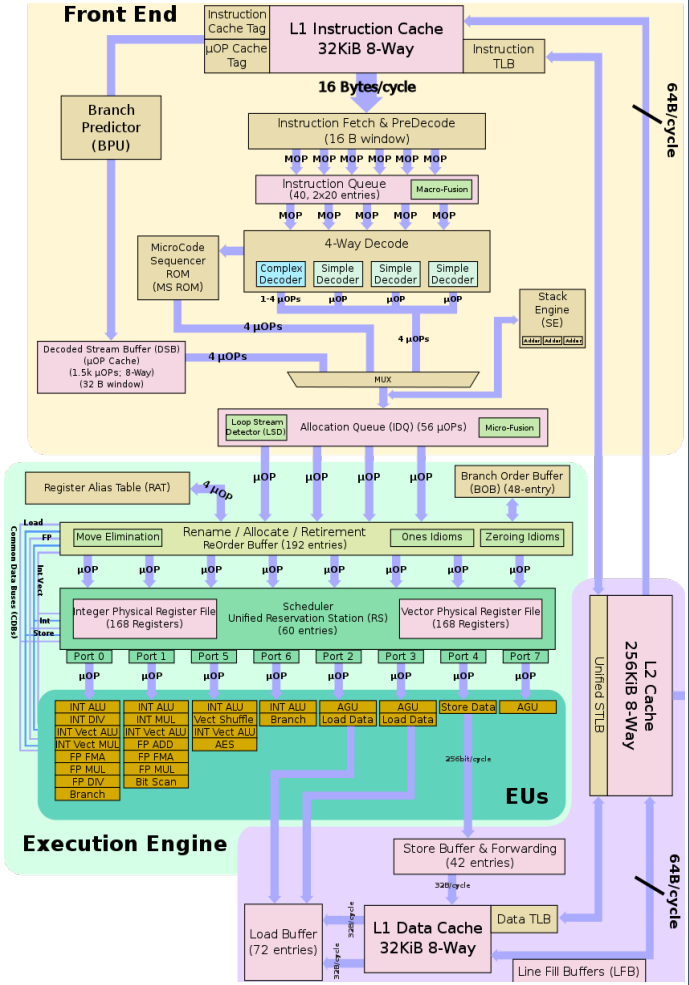
Intel



Microarquitectura Nehalen



Microarquitectura Skylake

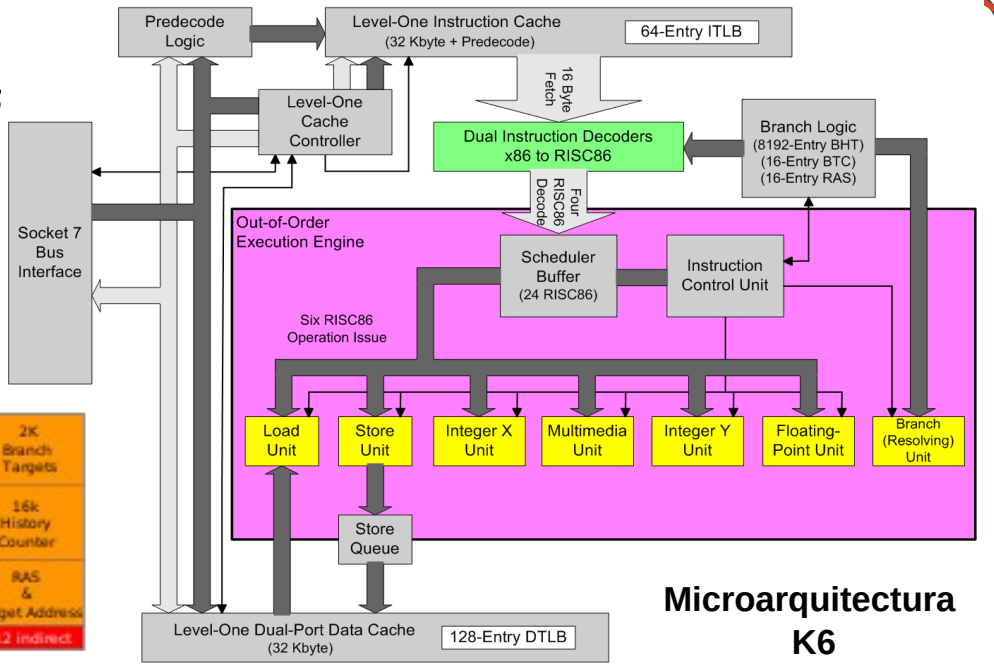


Microarquitectura Haswell

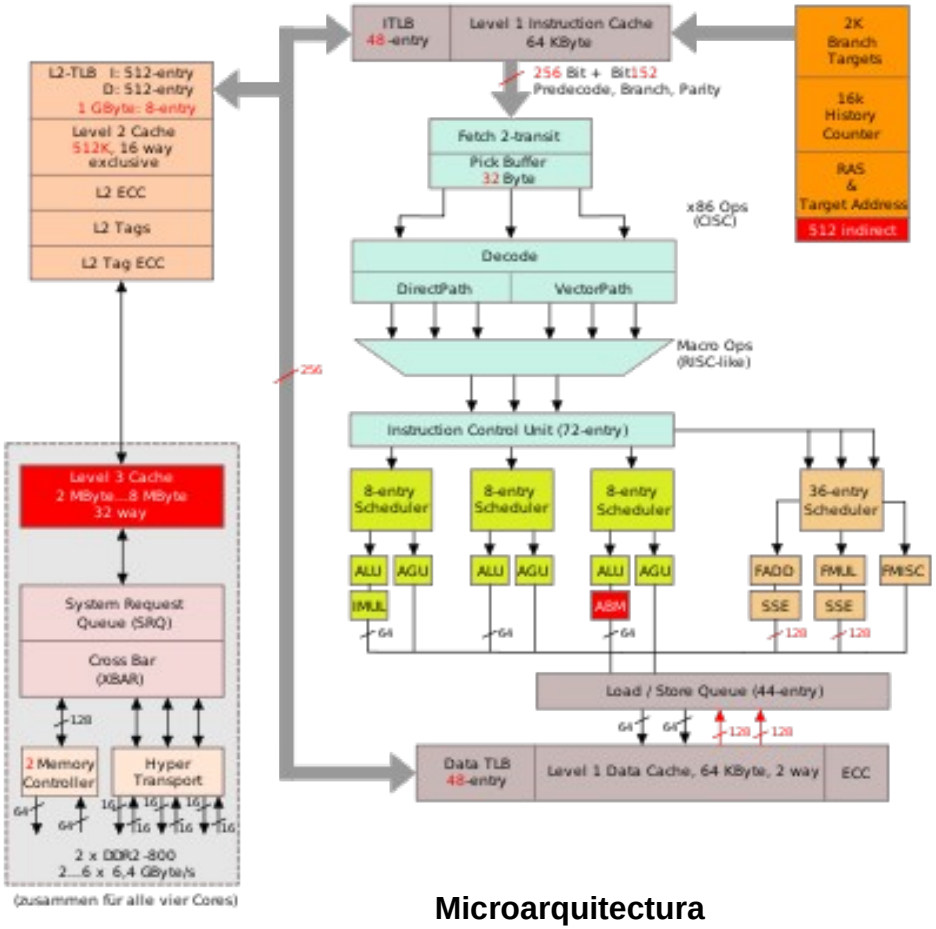
La microarquitectura avanzada

Procesadores superescalar comerciales

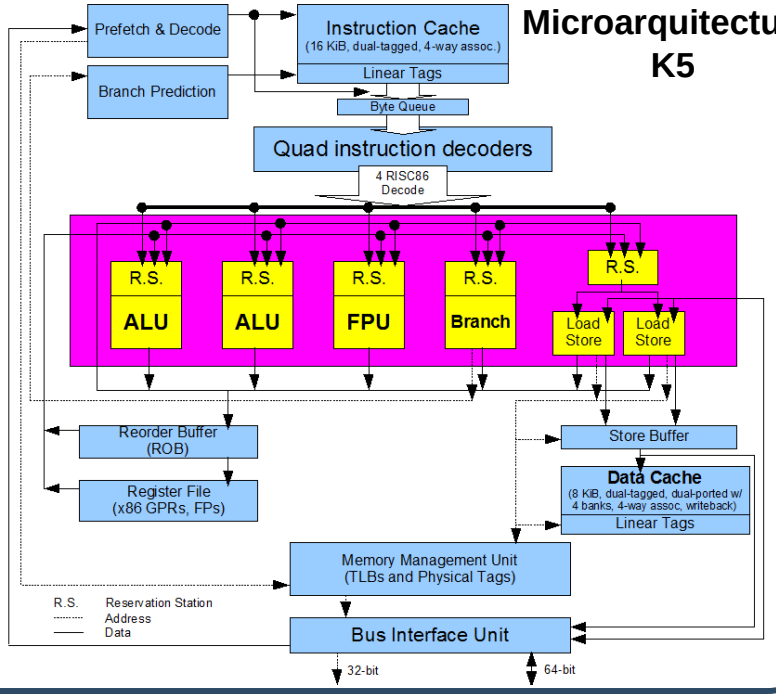
AMD



Microarquitectura K6



Microarquitectura K10

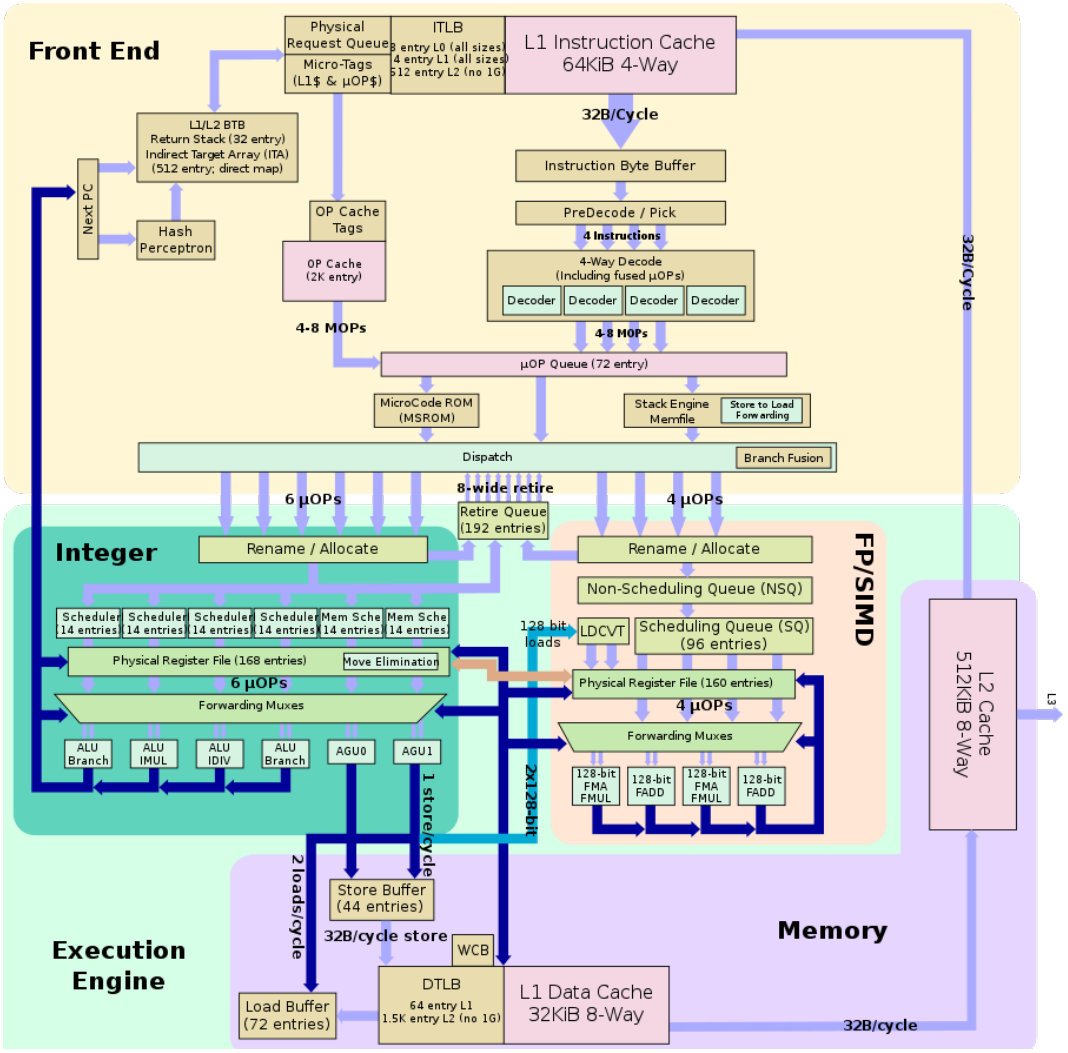


Microarquitectura K5

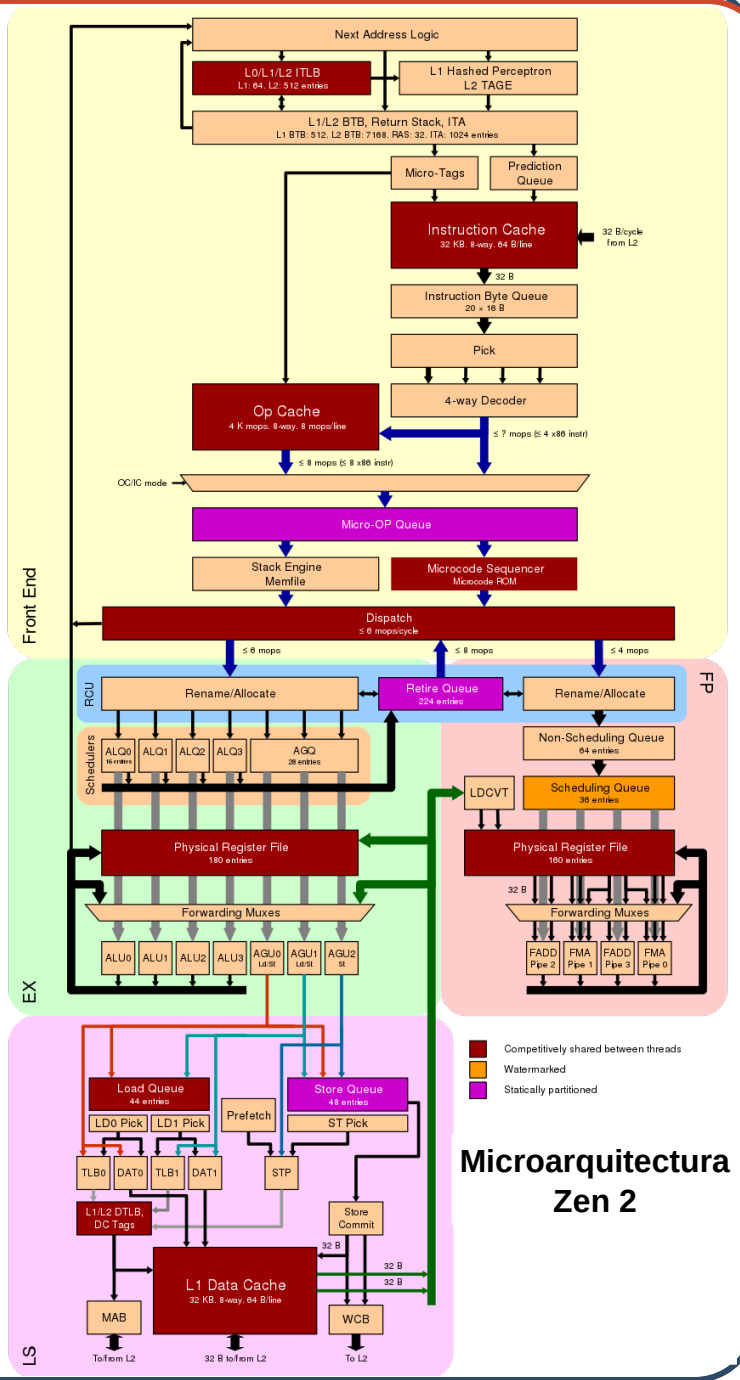
La microarquitectura avanzada

Procesadores superescalar comerciales

AMD



Microarquitectura Zen+



Microarquitectura Zen 2