Procesadores Superescalares

J. Smith and G. Sohi, *The Microarchitecture of Superscalar Processors. Proceedings IEEE, Vol. 83, No. 12, Diciembre 1995.*

William Stallings, *Organización y Arquitectura de Computadores, Capítulo 13: Paralelismo a nivel de instrucciones y procesadores superescalares.*

John Hennessy – David Patterson, *Arquitectura de Computadores – Un enfoque cuantitativo 3a Edición, Capítulo 3.*

Arquitecturas superescalares

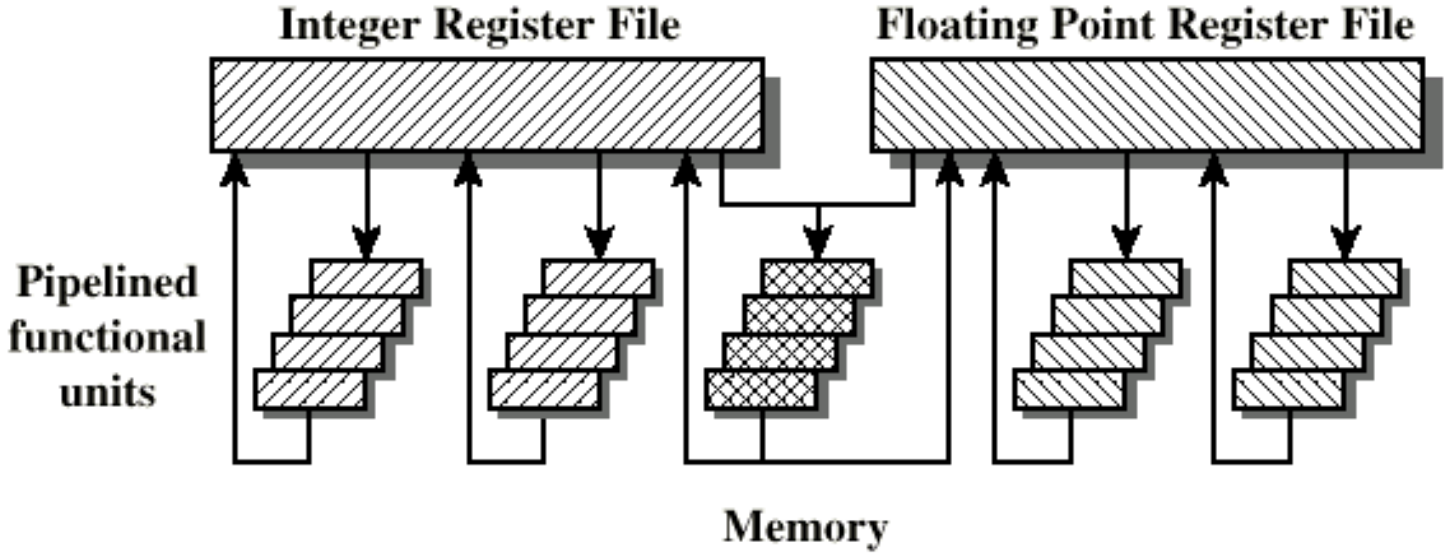
Introducción

- La ejecución segmentada permite instrucciones simultáneas, pero sólo una instrucción puede estar en cada etapa del cauce.
- En una arquitectura superescalar se dispone de **múltiples cauces de instrucciones independientes**, y a su vez segmentados, de modo que puede **iniciarse** la ejecución de varias instrucciones en forma simultánea (MIP: multiple issue precessors)
- Una arquitectura superescalar tiene las prestaciones de la segmentación, permitiendo además la existencia simultánea de varias instrucciones en la misma etapa. Para ello es necesaria la duplicación de recursos y la utilización de diversas técnicas que permitan optimizar su utilización.
- Como puede iniciarse la ejecución de varias instrucciones en el mismo ciclo, puede alcanzarse una productividad mayor que una instrucción por ciclo de reloj. En la práctica se consiguen aceleraciones cercanas a dos.

Arquitecturas superescalares

Introducción

- Ejemplo de organización superescalar. Múltiples unidades funcionales **segmentadas** (dos enteras, dos fp y una de carga-almacenamiento).

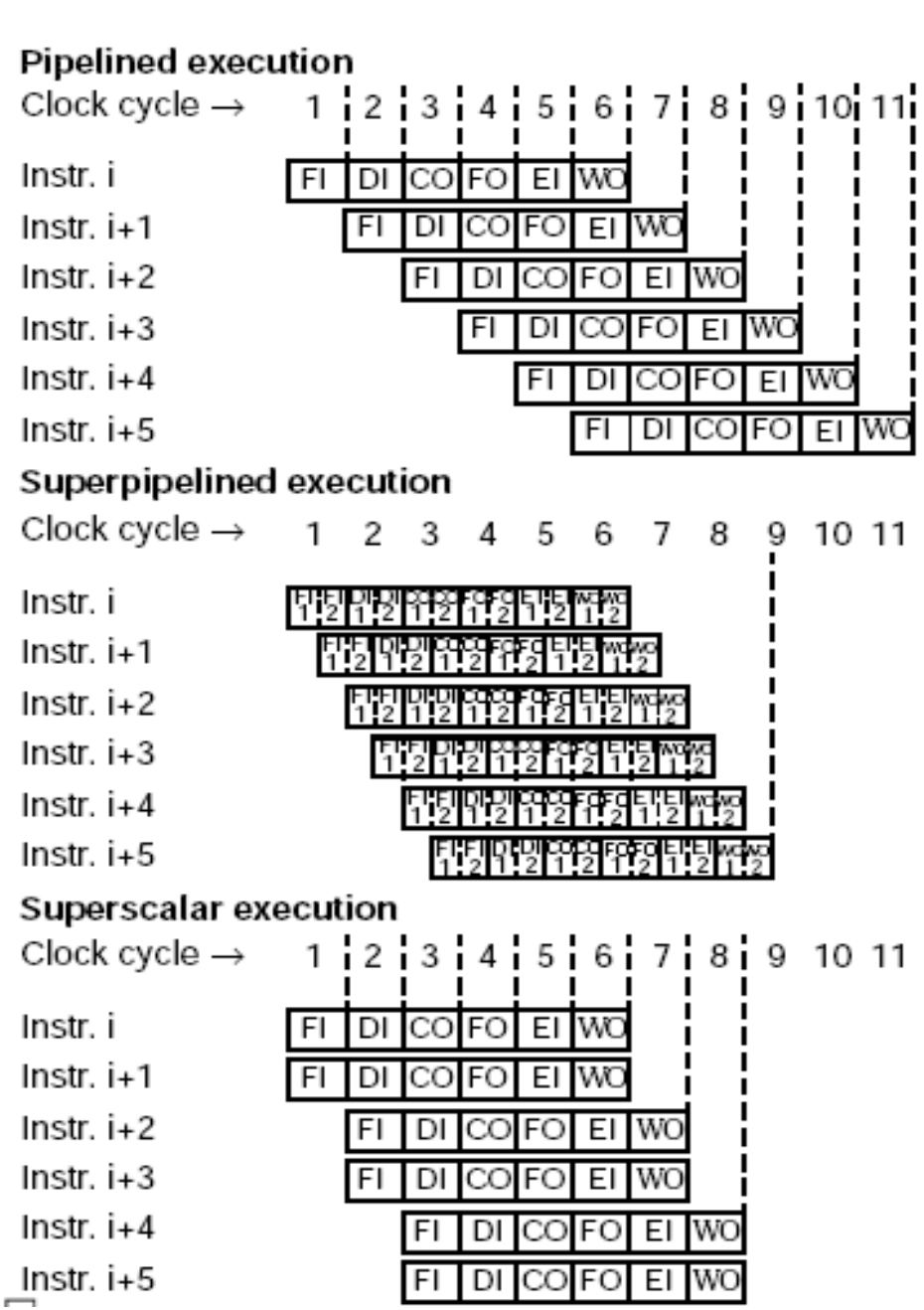


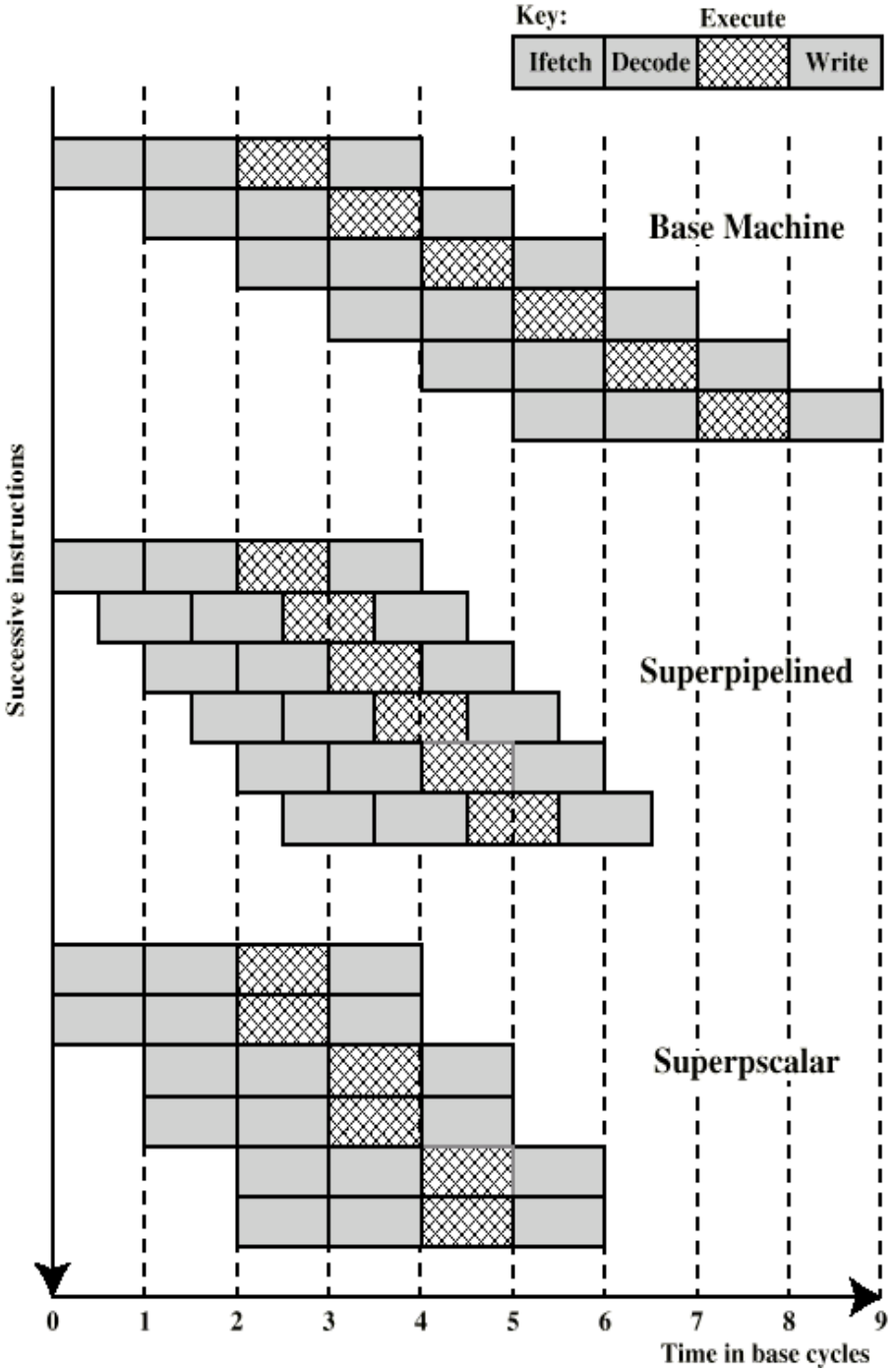
- Desde 1998 todos los procesadores comerciales son superescalares. Orígenes en RISC por ser de implementación más simple. Luego lo incorporan los CISC. INTEL Pentium Pro y Pentium II son procesadores CISC con microarquitectura RISC superescalar (2 ALU + 1 FP).
- Los más modernos, PowerPC970, 4 ALU + 1 MEM + 2 FP + 2 SIMD
- El caso del AMD K5 (1995) [ver paper IEEE]

Arquitecturas superescalares

Límites de la Segmentación y Supersegmentación

- La supersegmentación consiste en dividir las etapas de un cauce un sub-etapas, aprovechando el hecho de que muchas tareas requieren menos de la mitad de un ciclo de reloj para completarse. Aumenta el número de instrucciones en el cauce en un determinado instante.
- Dividiendo cada etapa en dos y utilizando un reloj interno del doble de frecuencia (supersegmentación de grado dos) se pueden obtener dos instrucciones por ciclo de reloj.
- Esto tiene un límite en el número de etapas, como ya vimos. La solución superescalar es una forma de aumentar las prestaciones por encima de este límite.





Productividad: 1 o 2 inst/ciclo

Definición

Paralelismo

Discusión importante:

Paralelismo a nivel de las instrucciones (ILP)

vs.

Paralelismo a nivel de la máquina (MLP)

luego...

Paralelismo a nivel de threads (TLP)

Definición

Paralelismo a nivel de las instrucciones (ILP)

- Existe ILP cuando las instrucciones que componen un programa son **independientes**. En ese caso el código puede ser reordenado sin alterar el resultado. Por lo tanto existe la posibilidad de ejecutar las instrucciones en **paralelo**, rompiendo la **secuencialidad** implícita en un programa estático.
- Las **dependencias de datos** son críticas en este contexto.
- Ejemplo de operaciones independientes:

```
for(i=0;i<1000;i++) x[i]=x[i]+y[i];
```

- Ejemplo de operaciones dependientes:

```
for(i=1;i<1000;i++) x[i]=(x[i]+x[i-1])/2;
```

- El punto es **detectar** el ILP.

Discusión: El rol de la **segmentación**

Definición

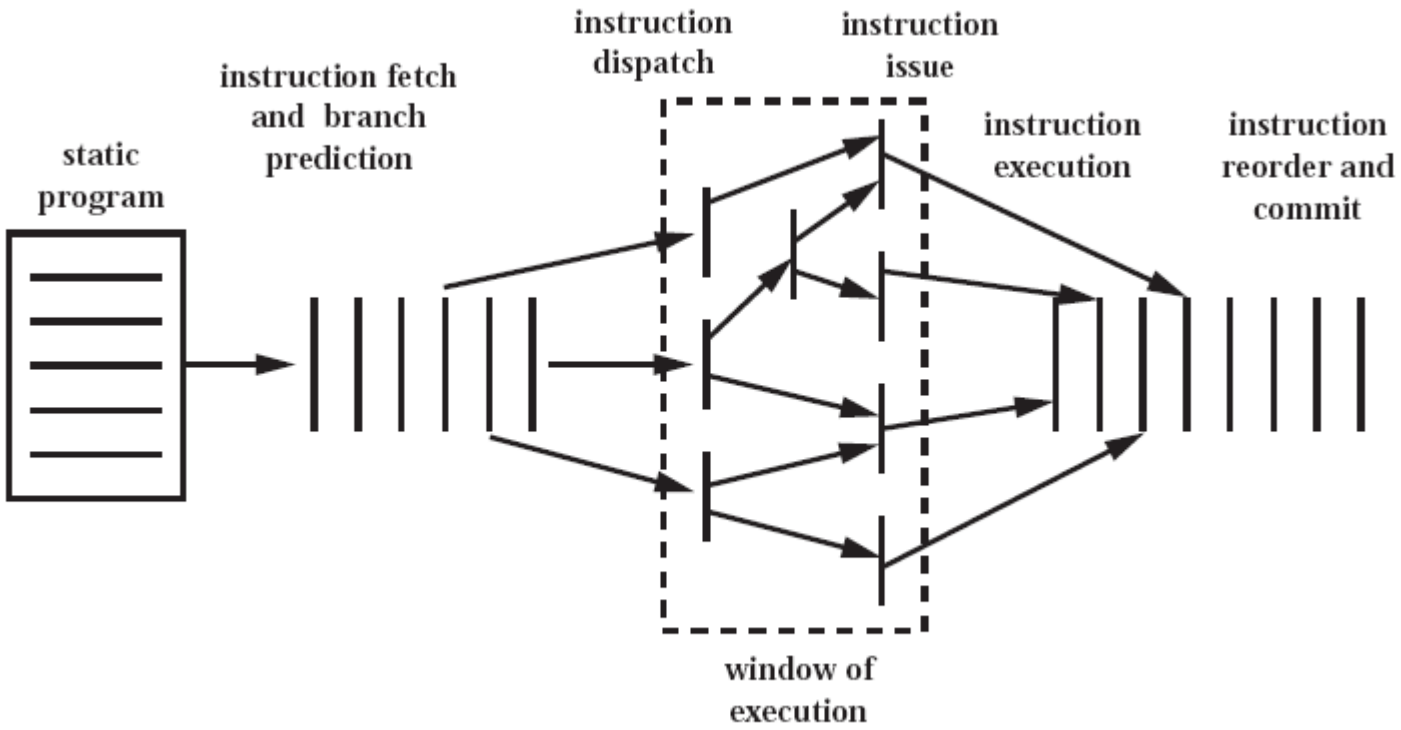
Paralelismo a nivel de la máquina (MLP)

- Es una medida de la capacidad del procesador para sacar partido del ILP.
- Depende del número de instrucciones que puedan captarse y ejecutarse simultáneamente (número de cauces paralelos).
- Dos aproximaciones:
  - **Scheduling Dinámico** (HW): P4, Athlon, UltraSPARCIII. Algoritmo de Tomasuolo: elimina WAW y WAR con renombrado de registros y disminuye RAW con tracking. **SUPERESCALARES**.
  - **Scheduling Estático** (SW): IA-64, Transmeta. **VLIW.**

Definición

Arquitectura Superescalar

Explotación **dinámica** (por hardware, via **MLP**) del **ILP** presente en el programa



FETCH	DECODE	DISPATCH	ISSUE	EXECUTE	COMMIT
Captar	Decodificar	Encolar	Emitir	Ejecutar //	Finalizar

VENTANA DE EJECUCION

Arquitecturas superescalares

El ciclo de instrucción

CAPTACION (fetch): múltiples instrucciones son captadas simultáneamente, utilizando técnicas de predicción de saltos y ejecución especulativa.

DECODIFICACION (decode): en dos pasos, i) Predecodificación entre la memoria y el cache para identificación de saltos, y ii) Determinación de la operación, localización de operandos y localización del resultado.

VENTANA DE EJECUCION = ENCOLADO (dispatch) y EMISION (issue): identificación de las instrucciones de la cola que están listas para comenzar su ejecución, o sea que tienen sus dependencias satisfechas.

EJECUCION (execute): en paralelo, en diferentes unidades funcionales.

FINALIZACION (commit): El resultado es confirmado en su destino.

Arquitecturas superescalares

Limitaciones

Las limitaciones son las mismas que en todos los sistemas concurrentes (idem pipeline). Las consecuencias son más severas que en un cauce normal.

- **Conflictos en los recursos:** similares a los riesgos estructurales de los cauces. Duplicación de recursos.

- **Dependencia de control:** los saltos reducen la eficiencia. Predicción estática (PowerPC 601) o dinámica (PowerPC 620). El salto retardado es eficiente en RISC pero no en superescalares CISC, ya que hay que insertar varias instrucciones en la ventana. Loop unrolling (compilador).

- **Dependencia de datos:** existe cuando dos instrucciones utilizan el mismo registro. Impiden que las instrucciones puedan reordenarse. La verificación de la dependencia de datos hace crecer mucho la complejidad del dispatcher, limitando la implementación a un máximo de 4.

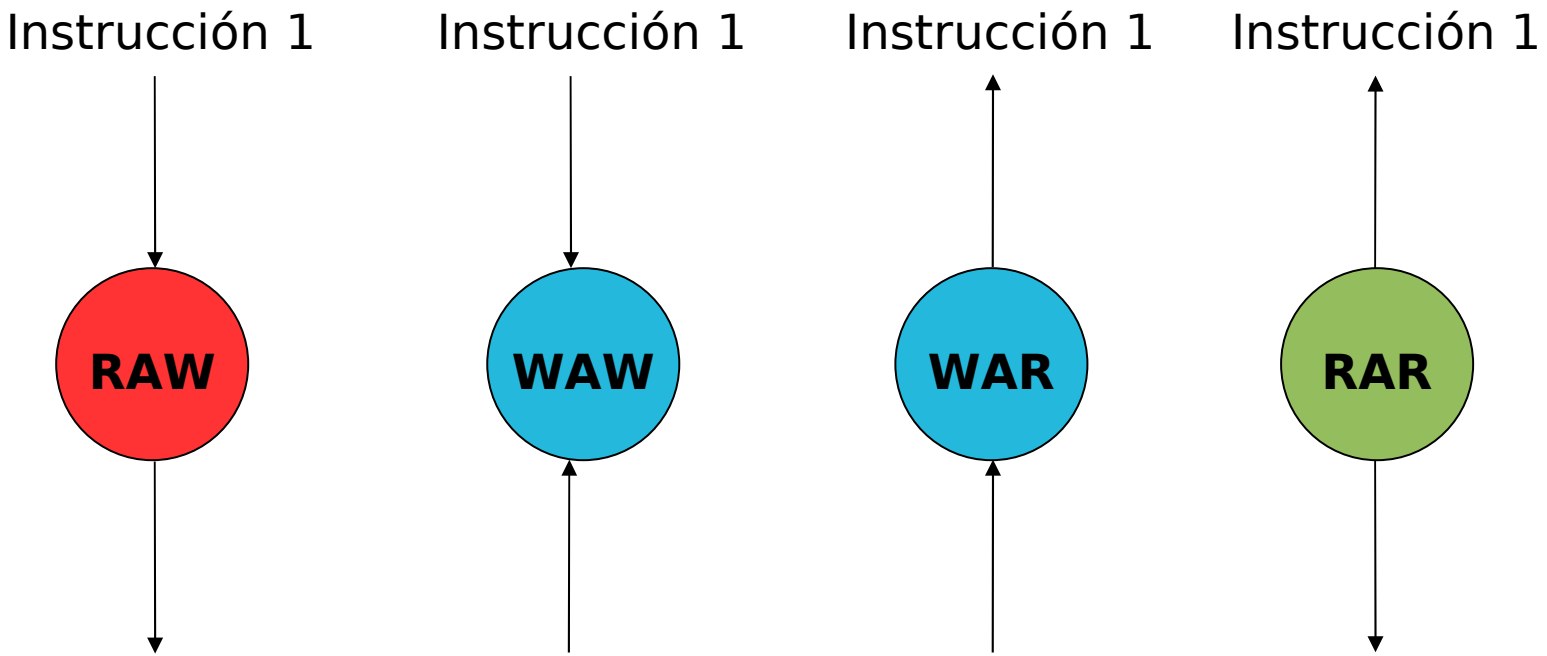
Arquitecturas superescalares

(Re)Clasificación de las dependencias de datos

- **Dependencias verdaderas:** necesito un operando generado por una instrucción anterior. Son intrínsecas del programa y no pueden eliminarse. **RAW**. Tracking.
- **Dependencias artificiales:** son conflictos de almacenamiento que pueden solucionarse con más registros (internos de la CPU, usualmente x2) (Register renaming):
    - **Dependencia de salida**: dos instrucciones escribiendo en el mismo destino. **WAW**.
    - **Antidependencia**: todavía estoy usando un registro y la instrucción siguiente lo reescribe. **WAR**.

Arquitecturas superescalares

(Re)Clasificación de las dependencias de datos



Arquitecturas superescalares

Técnicas de optimización

Se utilizan básicamente tres **técnicas de hardware** para aumentar el paralelismo de la máquina (MLP) y por lo tanto las prestaciones del conjunto:

- **Duplicación de recursos**
- **Política de emisión desordenada de instrucciones con ventana de ejecución**
- **Renombrado de registros**

Se agregan, de ser posible, **técnicas de software** (compilador) para asistir al hardware en la detección.

IMPORTANTE: Si no existe ILP no hay posible MLP eficiente.

Técnicas de optimización

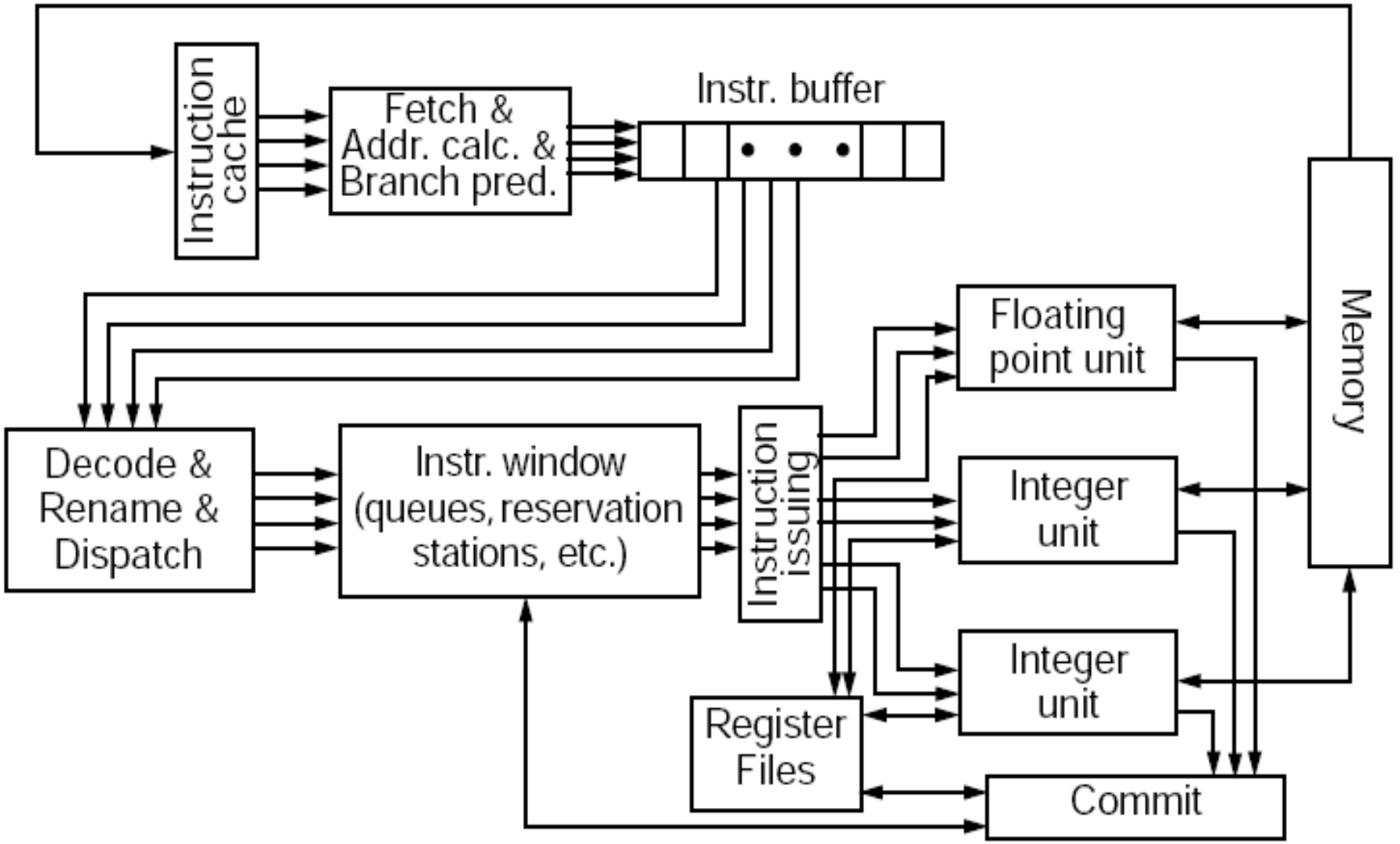
1. Duplicación de recursos

No sólo necesito recursos para **ejecutar** varias instrucciones en paralelo, sino que además necesito **captar** y **decodificar** varias instrucciones simultáneamente.

El **dispatcher** es crucial, ya que hay que mantener ocupadas las unidades funcionales (UF). Es el único elemento “inteligente”; el resto es fuerza bruta.

Tipos de Unidades Funcionales:

- Multiplicación/División en punto flotante
- Suma/Resta en punto flotante
- Operaciones con enteros
- Accesos a memoria: Load/Store
- Branch, etc.



FETCH	DECODE	DISPATCH	ISSUE	EXECUTE	COMMIT
Captar	Decodificar	Encolar	Emitir	Ejecutar //	Finalizar

VENTANA DE EJECUCION

Técnicas de optimización

2. Políticas de ejecución paralela

- Se clasifican según dos factores:
  - El orden según el cual las instrucciones son enviadas a ejecutar (emitidas, issued).
  - El orden en que las instrucciones finalizan su ejecución al escribir en registros o memoria (commit).

- La política más simple es **emitir y finalizar en orden secuencial**. Esta técnica tiene pocas probabilidades de encontrar instrucciones que puedan ejecutarse en paralelo y depende mucho de cómo está escrito y compilado el programa. La ventaja es que el procesador sólo debe preocuparse por las dependencias de datos reales ya que las dependencias de salida y las antidependencias no existen. Método usado en **segmentación** que genera detenciones del cauce (stall).

Técnicas de optimización

2. Políticas de ejecución (cont)

La performance mejora si se permite alterar el orden de emisión o finalización (o ambos), siempre que se mantenga la exactitud del resultado.

- **Finalización desordenada**: no espera a que terminen las instrucciones anteriores para enviar la actual. Aparecen las **dependencias de salida** (además de las verdaderas), por lo que se necesita una lógica de emisión más complicada. Peligroso ante interrupciones.

- **Emisión desordenada (implica la anterior)**: utilizando la **ventana de instrucciones** puede irse más allá de un punto de conflicto en la emisión para encontrar instrucciones sin dependencia de datos ni conflictos de recursos (anticipación). Aparecen las **antidependencias**.

Técnicas de optimización

2. Políticas de ejecución (cont)

Ventana de ejecución

- Es el conjunto de instrucciones que se consideran para su ejecución en un determinado momento. El hardware determina cuáles son las instrucciones de la ventana que pueden enviarse a ejecutar en paralelo, limitado por las dependencias de datos y disponibilidad de recursos.
- La ventana debe ser lo más grande posible. La limitan la capacidad de captar intrucciones a una gran tasa y el problema de los saltos.
- Se utiliza **predicción de saltos** y **ejecución especulativa**. La porción de código predicha se incorpora a la ventana de ejecución y se ejecutan tentativamente. Si la predicción fue correcta el resultado se hace visible y permanente (commit), si no se elimina.

Técnicas de optimización

3. Renombrado de registros

- Cuando se utilizan técnicas de desordenación los valores de los registros no pueden conocerse completamente en cada instante de tiempo. Las instrucciones entran en conflicto por el uso de registros y el procesador debe detener alguna etapa para resolverlo.
- Las técnicas de software de optimización de registros empeoran la situación.
- Los efectos de las dependencias artificailes pueden disminuirse por esta técnica, que consiste en disponer de registros adicionales (internos, ocultos al programador) y asignarlos (por hardware) a instrucciones en conflicto. Existen R3a y R3b, por ejemplo.
- Lo usual es tener duplicado el banco de registros. Ventajas y desventajas. EJEMPLO.

```
STORE  R3,(R1)
ADD    R1b,R6,R7
```

Técnicas de optimización

4. Técnicas de compilación

La implementación de técnicas de compilador está condicionada a que el programa pueda ser recompilado, lo cual no siempre es posible.

Algunas técnicas que pueden utilizarse:

- Reagrupación de instrucciones independientes (por ejemplo de a cuatro en el Pentium II).
- Adelanto de instrucciones que generen dependencia real de datos.
- Loop unrolling, etc. H-P 2a. ed. Más detalles la clase que viene.

Arquitecturas superescalares

Juntando todo

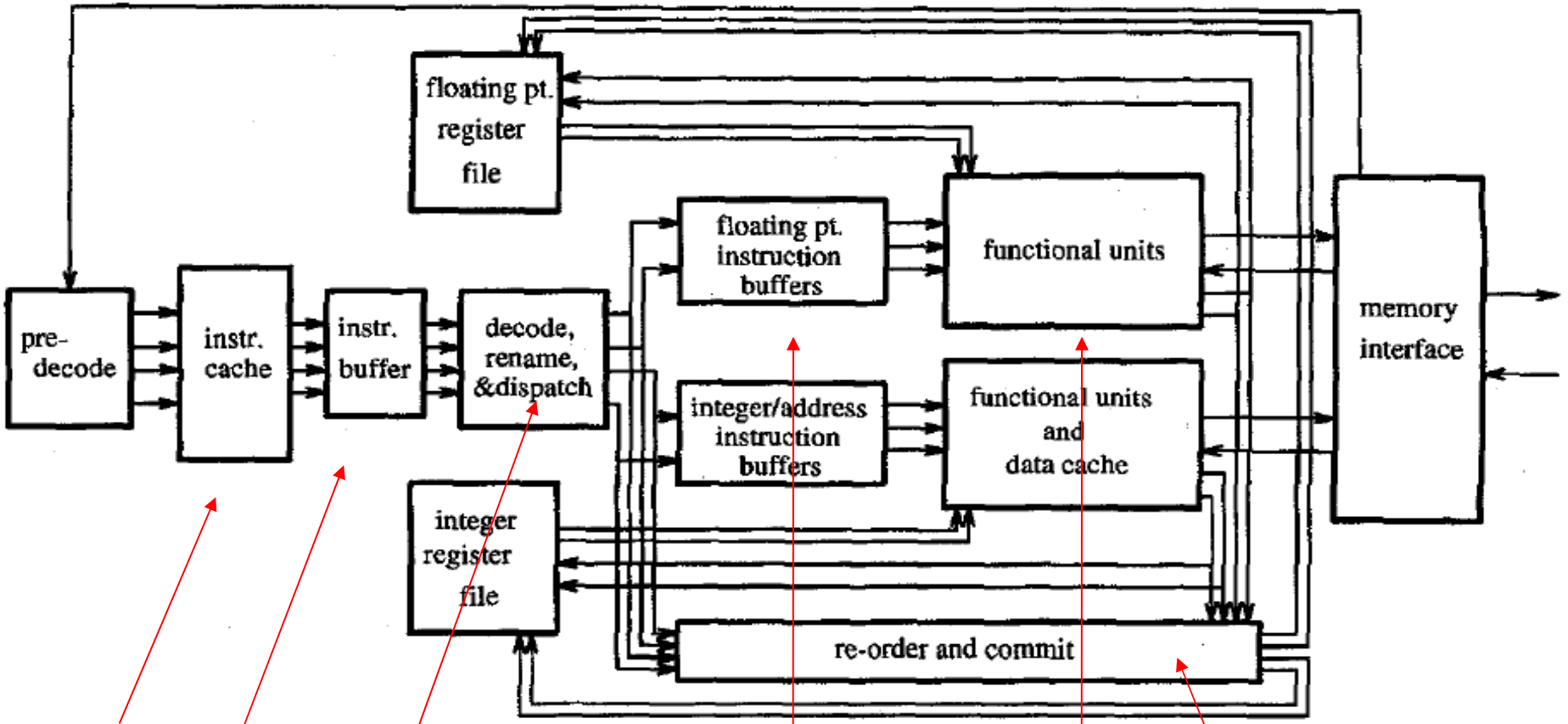
- Captación y decodificación simultánea de varias instrucciones.
- Varias unidades funcionales segmentadas en paralelo.
- Emisión y finalización desordenada con ventana.
- Renombrado de registros.
- Asistencia desde el compilador.

“Nomenclatura” en Intel Core 2 datasheet

- *Data Flow Analysis*
- *Speculative and Out of Order Execution*
- *Superscalar*
- *More accurate Branch Prediction*
- *Deeper Buffers*
- *Incluye MACRO FUSION (varias instrucciones x86 en una)*

Arquitecturas superescalares

MIPS R10000



8K

El doble de registros

Fetch X4

3 Colas

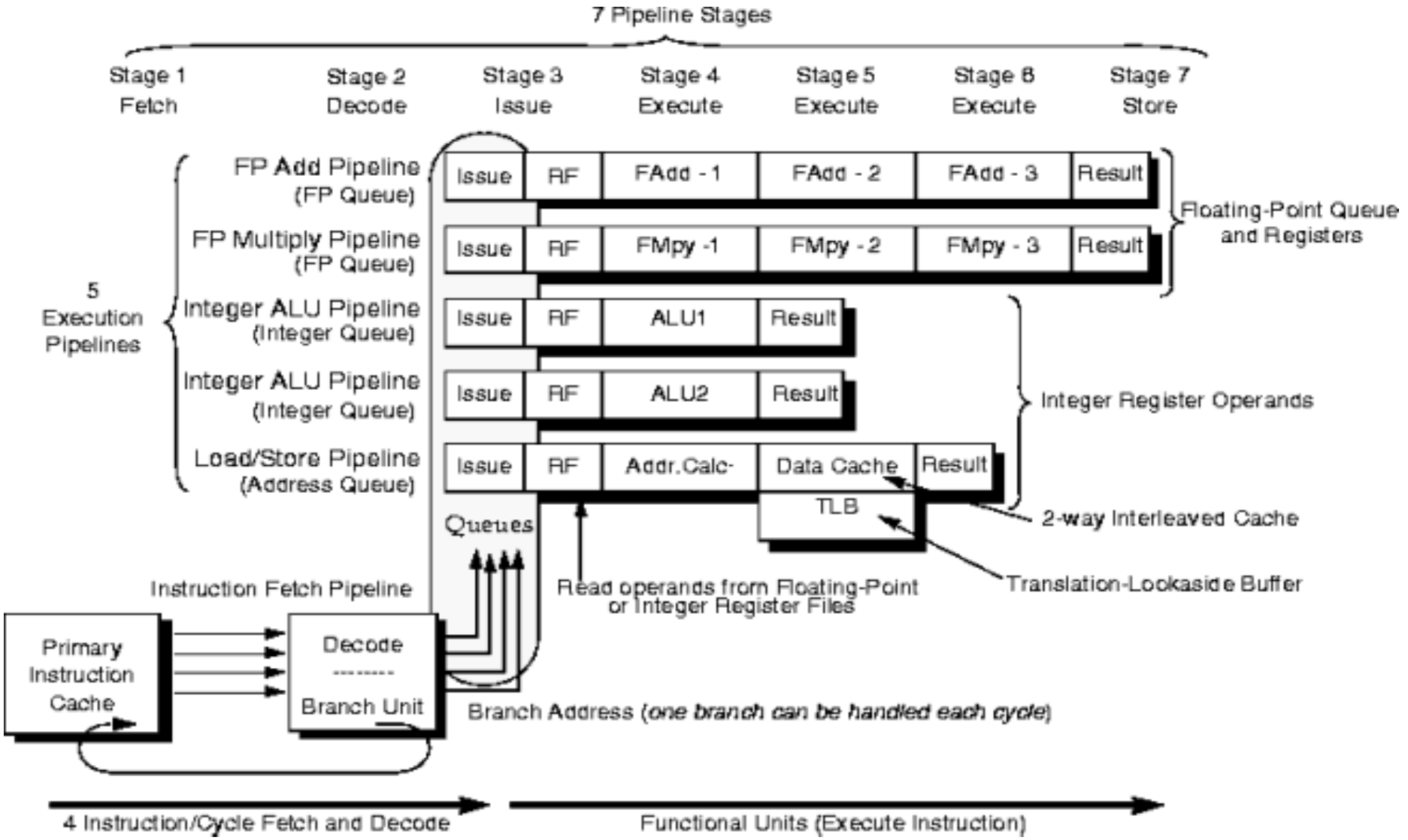
5 UF

Commit X4

Arquitecturas superescalares

MIPS R10000: Pipeline

MIPS R10000: 5 pipelines
Pentium: 2 pipelines



Arquitecturas superescalares

Conclusiones

- Solamente agregando unidades funcionales, sin otras técnicas, no se obtienen buenos resultados.
- La emisión desordenada es importante porque permite buscar instrucciones independientes hacia adelante.
- El renombrado de registros puede mejorar la performance hasta un 30%, con la única limitación de la dependencia real de datos.
- Es importante disponer de una buena capacidad de captación y decodificación para poder utilizar una ventana de instrucciones grandes.

Arquitecturas superescalares

IBM RISC System RS/6000

- A good example of a superscalar processor is the IBM RS/6000. There are three major subsystems in this processor: the **instruction fetch unit**, an **integer processor**, and a **floating point processor**. The instruction fetch unit is a 2- stage pipeline; during the first stage a packet of four instructions is fetched from an instruction cache, and in the second stage instructions are routed to the integer processor and/or floating point processor. An interesting feature of this instruction unit is that it executes branch instructions itself so that in a tight loop there is effectively no overhead from branching since the instruction unit executes branches while the data units are computing values. The integer unit is a four-stage pipeline. In addition to executing data processing instructions this unit does some preprocessing for the floating point unit. The floating point unit itself is six stages deep.

- A 62.5 MHz RS/6000 system ran the LINPACK benchmark (Gaussian elimination) at a rate of 104 MFLOPS and has a theoretical peak performance of 125 MFLOPS. The HP 9000/735 workstation has a 99 MHz superscalar HP-PA processor. This machine executes the LINPACK benchmark at 107 MFLOPS, with a theoretical peak performance of 198 MFLOPS.

Otros ejemplos superescalares

- **PowerPC 601:** ver Stallings.
- **PowerPC 604:** 6 unidades de ejecución independientes (1 unidad de procesamiento de saltos, 1 unidad de carga almacenamiento, 3 unidades de enteros, 1 unidad de punto flotante), emisión en orden, renombrado de registros.
- **PowerPC 620:** idem con emisión desordenada.
- **Pentium:** Emisión en orden y 3 unidades de ejecución independientes (2 unidades de enteros, 1 unidad de punto flotante).
- **Pentium II:** ver Stallings. 6ta. ed. Pentium IV.
- **MIPS R10000** y **UltraSPARC-II**: ver Stallings. Eliminados en la 6ta. ed.
- Paper IEEE: **MIPS R10000**, **Alpha 21164**, **AMD K5** (x86).

Arquitecturas superescalares

Multi-Threading

Threads:

- Extensión de los lenguajes de programación que permite dividirlos en tareas pseudo-simultáneas.
- Históricamente ayudaban al scheduler del SO a realizar el time slice (en el caso de un monoprocesador) o a distribuir la carga (multiprocesadores).
- Es diferente al concepto de proceso, pues entre threads se comparte el estado y el espacio de memoria.
- Es más “liviano”. Los context switch son más rápidos a expensas de un cierto nivel de dependencia.

Arquitecturas superescalares

Multi-Threading (cont)

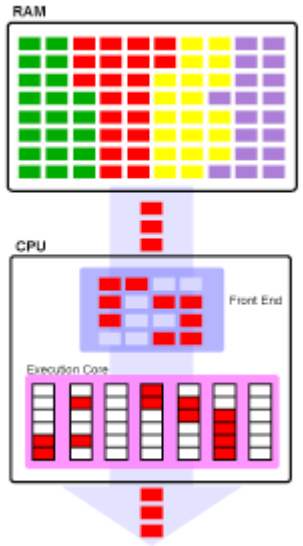
- El paralelismo a nivel de threads (TLP) es una medida de esta propiedad en los programas. Puede ser explotado por los procesadores superescalares, ya que es más explícito que el ILP. Se pueden asignar diferentes UFs a los threads.
- Muy útil para evitar que el procesador quede detenido ante un cache-miss: continúa ejecutando el otro thread.
- Son necesarios más registros y la captación simultánea desde diferentes threads (al menos dos).
- Si el SO está preparado, puede ver el HW como dos procesadores lógicos.

Ejemplos:

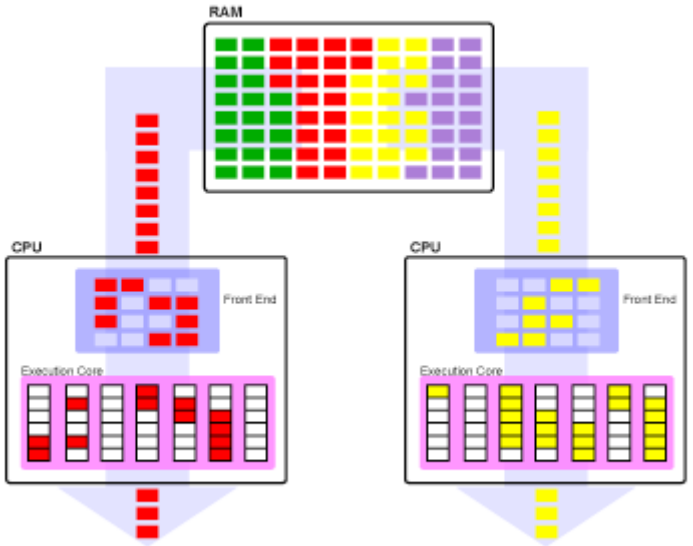
- INTEL HyperThreading (Pentium 4 HTT). Según Intel mejora 30%.
- Simultaneus Multithreading (SMT) Power, SPARC.
- P4x2 (S=1.3), Power5 x2, SPARC x4 (x8).

Arquitecturas superescalares

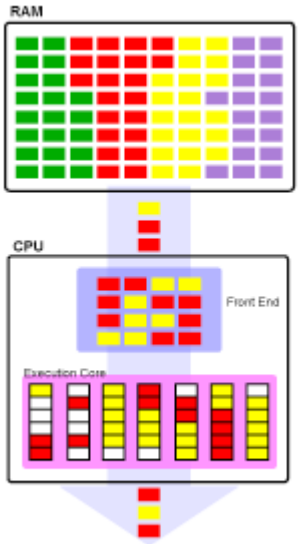
Multi-Threading (cont)



1 CPU/1 Thread



2 CPU/1 Thread



1 CPU/2 Thread

Arquitecturas superescalares

Práctica

Incorporar la siguiente información a los procesadores involucrados en la práctica de repaso:

- Detalles acerca de la segmentación del cauce de instrucciones.
- Características RISC de la arquitectura.
- Unidades funcionales disponibles e implementación superescalar.