Paralelismo a Nivel de Instrucción (PNI)

- Segmentación: ejecución de múltiples instrucciones en paralelo
- Para incrementar el PNI
  - Pipeline más profundo
    - Menos trabajo por etapa ⇒ Ciclo de reloj más chico
  - Despacho Múltiple
    - Replicando etapas del pipe ⇒ múltiples pipelines
    - Se inician múltiples instrucciones por ciclo
    - CPI < 1, por lo que usa instrucciones por ciclo (IPC)
    - E.g., 4GHz 4-way despacho múltiple
      - 16 X 10^9 IPS, máximo CPI = 0.25, máximo IPC = 4
    - En la práctica se reduce por las dependencias

Despacho Múltiple

- Estático
  - El compilador agrupa instrucciones para despacharlas juntas
  - Las empaqueta en “franjas de despacho”
  - El Compilador detecta y evita los riesgos
- Dinámico
  - La CPU examina la corriente de instrucciones y elige las instrucciones para despachar en cada ciclo
  - El Compilador puede colaborar reordenando las instrucciones
  - La CPU resuelve los riesgos en tiempo de ejecución usando técnicas avanzadas.

Especulación

- “Adivinar” que hacer con una instrucción
  - Comenzar la operación lo más pronto que sea posible
  - Controlar si la suposición es correcta
    - si lo es, completar la operación
    - sino, deshacer lo hecho y hacer lo correcto
- Presente en despacho múltiple estático y dinámico
  - Especular sobre el resultado de los saltos
    - deshacer si se tomó el camino equivocado
  - Especulación sobre los load
    - deshacer si la posición es actualizada

  } Ejemplos
- Se necesita verificar y poder deshacer.

Especulación Compilador/Hardware

- El Compilador puede reordenar instrucciones
  - ej. mover una asignación antes de un salto
  - Puede incluir instrucciones de “reparación” para recuperarse de una suposición incorrecta.
- Hardware puede mirar hacia adelante para ver las instrucciones para ejecutar.
  - Usar buffer para los resultados hasta determinar que son efctivamente necesarios
  - Vaciar los buffers si la especulación fue incorrecta

Especulación and Excepciones

- ¿Qué pasa si ocurre una excepción en una instrucción ejecutada especulativamente
  - ej un load especulativo a una dirección no válida
- Especulación estática
  - se agrega un soporte para demorar las excepciones
- Especulación dinámica
  - la excepción se retiene hasta que la instrucción casi termina

Despacho múltiple estático

- El compilador agrupa las instrucciones en "paquetes de despacho"
  - Un grupo de instrucciones que pueden ser despachadas en un mismo ciclo
  - se determina según los recursos requeridos del pipe
- Un paquete de despacho es como una instrucción grande
  - Especifica múltiples operaciones concurrentes
  - ⇒ Very Long Instruction Word (VLIW)

Planificación estática del despacho dual

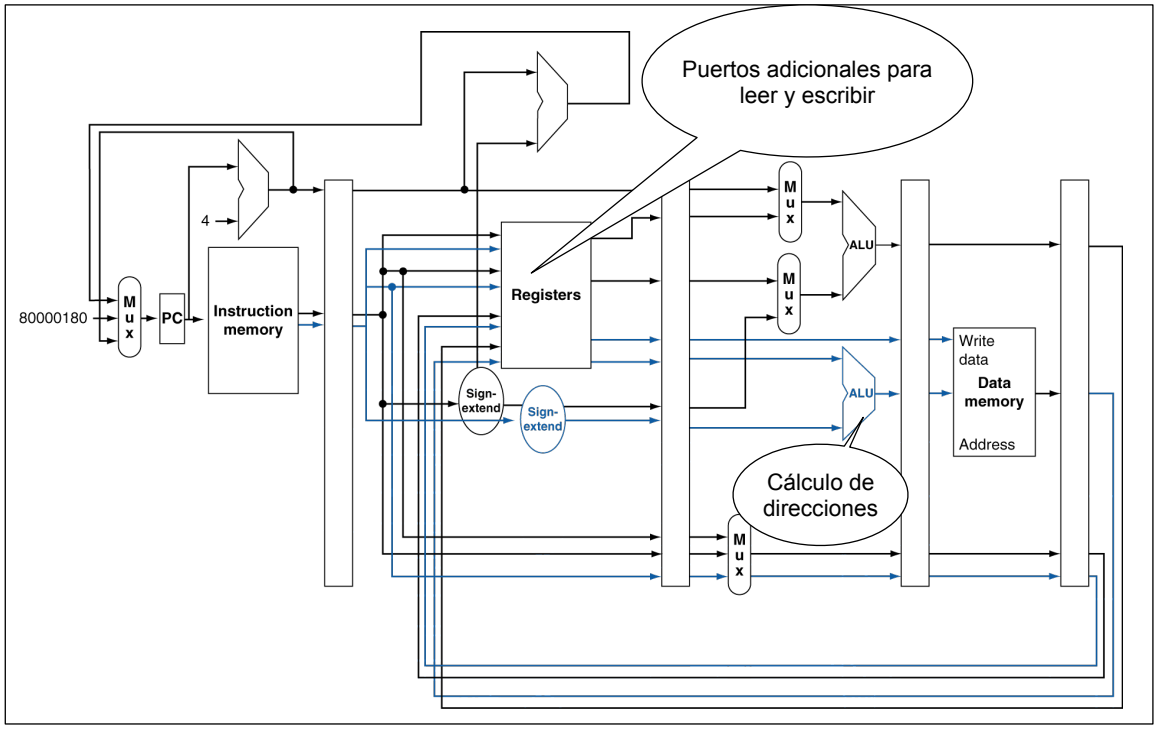
- El compilador debe eliminar algunos/todos los riesgos
  - Reordenar las instrucciones en paquetes de despacho
    - Una instrucción ALU o branch y una load/store
  - No debe haber dependencias dentro del paquete
  - Pueden haber dependencias entre paquetes
    - Depende de la ISA; el compilador la debe conocer
  - Completar con **nop** si es necesario

Despacho dual estático en MIPS

- Paquete de despacho doble
  - Una instrucción ALU/branch
  - Una instrucción load/store
  - Alineación de 64-bit
    - ALU/branch, seguido por load/store
    - Una instrucción aislada se acompaña con **nop**

Address	Instruction type	Pipeline Stages						
n	ALU/branch	IF	ID	EX	MEM	WB		
n + 4	Load/store	IF	ID	EX	MEM	WB		
n + 8	ALU/branch		IF	ID	EX	MEM	WB	
n + 12	Load/store		IF	ID	EX	MEM	WB	
n + 16	ALU/branch			IF	ID	EX	MEM	WB
n + 20	Load/store			IF	ID	EX	MEM	WB

Despacho dual estático en MIPS



Riesgos en el despacho dual estático

- Más instrucciones ejecutando en paralelo
- Riesgo de dato en EX
  - El adelantamiento evita atascos en el despacho simple
  - Un resultado de ALU no se puede usar en load/store en el mismo paquete
    - `add $t0, $s0, $s1`
      `load $s2, 0($t0)`
    - Dividir en dos paquetes, es realmente un atasco
- Riesgo Load-use
  - Sigue siendo de un ciclo, pero ahora de dos instrucciones
- El compilador debe hacer una buena planificación.

Ejemplo de planificación

- Planificación para el MIPS de despacho dual

```
Loop: lw   $t0, 0($s1)        # $t0=array element
      addu $t0, $t0, $s2      # add scalar in $s2
      sw   $t0, 0($s1)        # store result
      addi $s1, $s1,-4        # decrement pointer
      bne  $s1, $zero, Loop   # branch $s1!=0
```

	ALU/branch	Load/store	cycle
Loop:	nop	lw $t0, 0($s1)	1
	addi $s1, $s1,-4	nop	2
	addu $t0, $t0, $s2	nop	3
	bne $s1, $zero, Loop	sw $t0, 4($s1)	4

- IPC = 5/4 = 1.25 (c.f. peak IPC = 2)

Limitaciones del Paralelismo a Nivel de Instrucción (PNI)

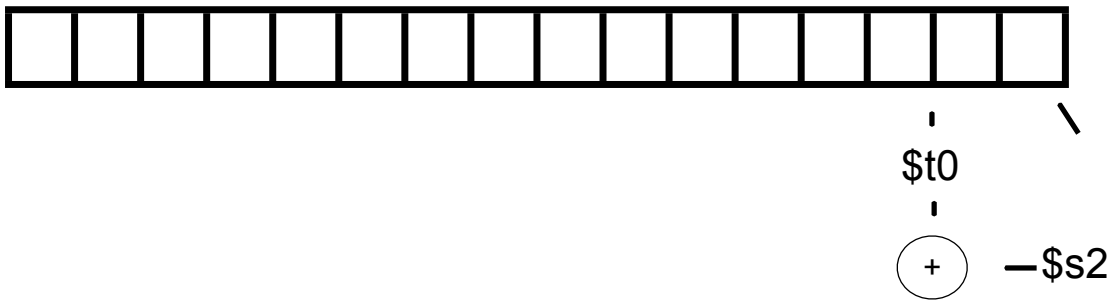
- El bloque básico es muy pequeño.
- Un bloque básico (BB) es una secuencia de código sin saltos. Puntos de entrada y de salida únicos.
- Sólo de 4 a 7 instrucciones.
- Fuertes dependencias entre ellas.
- El camino es explotar PNI entre varios BB.
- El caso más simple: paralelismo a nivel de loops

```
for (i=1000; i<0; i=i-1)
    x[i] = x[i] + s ;
```

Desenrollado de Loops

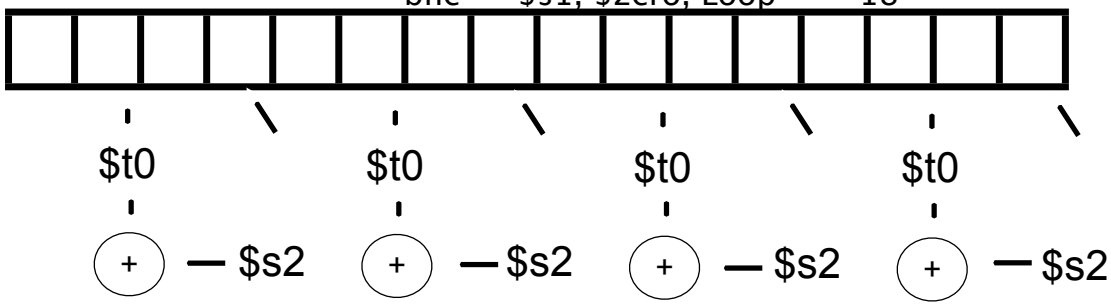
- El análisis a nivel de loop involucra determinar la existencia de dependencias entre los operandos de un loop en iteraciones **sucesivas** de ese loop.
- Ejemplo

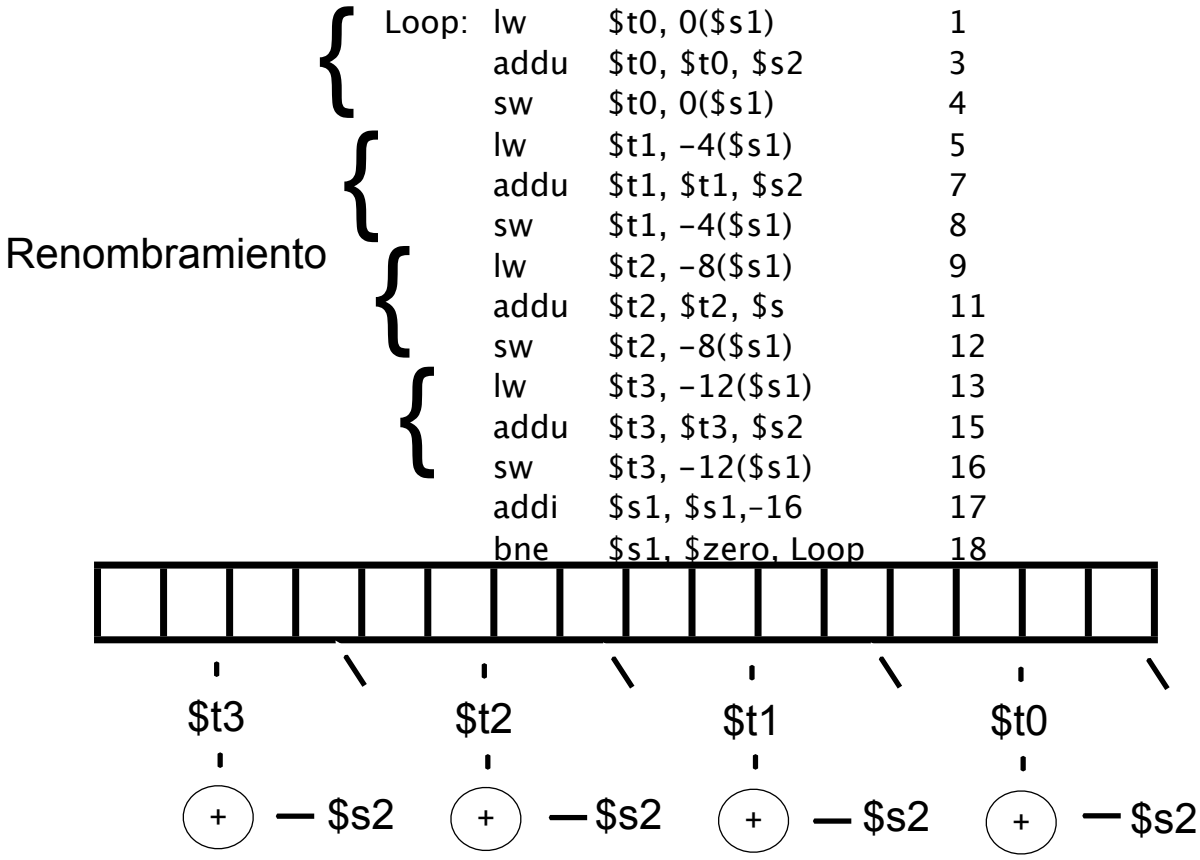
```
Loop: lw    $t0, 0($s1)      # $t0=array element
      addu  $t0, $t0, $s2    # add scalar in $s2
      sw    $t0, 0($s1)      # store result
      addi  $s1, $s1,-4      # decrement pointer
      bne   $s1, $zero, Loop # branch $s1!=0
```



Antidependencia

```
Loop: lw    $t0, 0($s1)      1
      addu  $t0, $t0, $s2    3
      sw    $t0, 0($s1)      4
      lw    $t0, -4($s1)     5
      addu  $t0, $t0, $s2    7
      sw    $t0, -4($s1)     8
      lw    $t0, -8($s1)     9
      addu  $t0, $t0, $s     11
      sw    $t0, -8($s1)     12
      lw    $t0, -12($s1)    13
      addu  $t0, $t0, $s2    15
      sw    $t0, -12($s1)    16
      addi  $s1, $s1,-16     17
      bne   $s1, $zero, Loop 18
```





Ejemplo de desenrollado de Loop

	ALU/branch	Load/store	cycle
Loop:	addi $s1, $s1,-16	lw $t0, 0($s1)	1
	nop	lw $t1, 12($s1)	2
	addu $t0, $t0, $s2	lw $t2, 8($s1)	3
	addu $t1, $t1, $s2	lw $t3, 4($s1)	4
	addu $t2, $t2, $s2	sw $t0, 16($s1)	5
	addu $t3, $t4, $s2	sw $t1, 12($s1)	6
	nop	sw $t2, 8($s1)	7
	bne $s1, $zero, Loop	sw $t3, 4($s1)	8

- IPC = 14/8 = 1.75
  - cercano a 2, pero al costo de registros y tamaño de código

Despacho Múltiple dinámico

- Procesadores “Superscalar”
- La CPU decide si despachar 0, 1, 2, … en cada ciclo
  - Se evitan los riesgos de datos y estructurales
- No requiere la planificación del compilador
  - Aunque puede ayudar
  - La semántica del código es asegurada por CPU

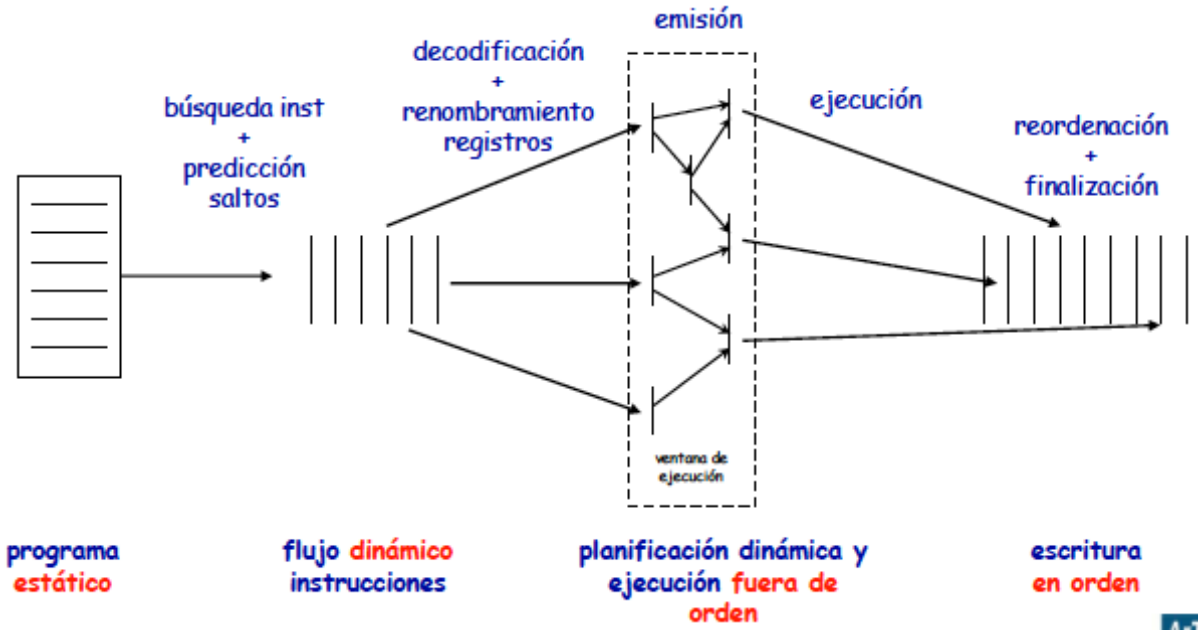
Pipeline con Planificación Dinámica

- Permite que la CPU ejecute instrucciones fuera de orden para evitar atascos.
  - Pero finalmente el resultado llega al registro en orden
- Example

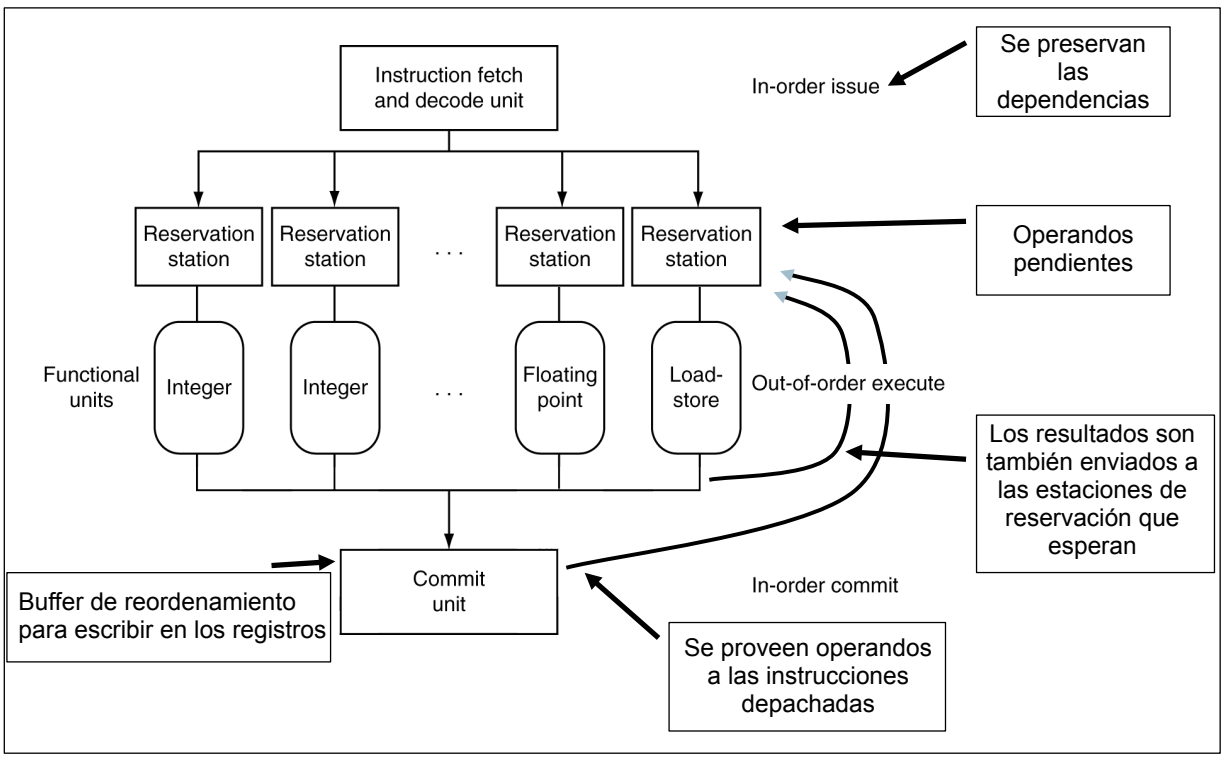
```
lw    $t0, 20($s2)
addu  $t1, $t0, $t2
sub   $s4, $s4, $t3
slti  $t5, $s4, 20
```

  - Puede empezar sub mientras addu espera por lw

Planificación Dinámica y especulación



CPU planificada dinámicamente



Renombramiento de Registros

- Las estaciones de reservación y los buffer de reordenamiento proveen renombramiento de registros
- Despacho de instrucciones a las estaciones de reservación
  - Si el operando está disponible en registro o en el buffer de reordenamiento
    - Se copia a la estación de reservación
    - Ya no se requiere en el registro; puede ser sobrescrito
  - Si el operando aun no está disponible
    - Será provisto a la estación de reservación por una unidad funcional
    - La actualización de registro puede no ser necesaria

Especulación

- Predecir el salto y continuar despachando
  - No finalizar (commit) en espera del resultado del salto
- Especulación del load
  - Se evitan demoras del load y fallos de cache
    - Predicción de la dirección efectiva
    - Predicción del valor del load
    - Completar los **loads** antes que terminen los **stores**
    - Adelantar los valores de stores a la unidad de load.
  - No finalizar los loads hasta clarificar la especulación

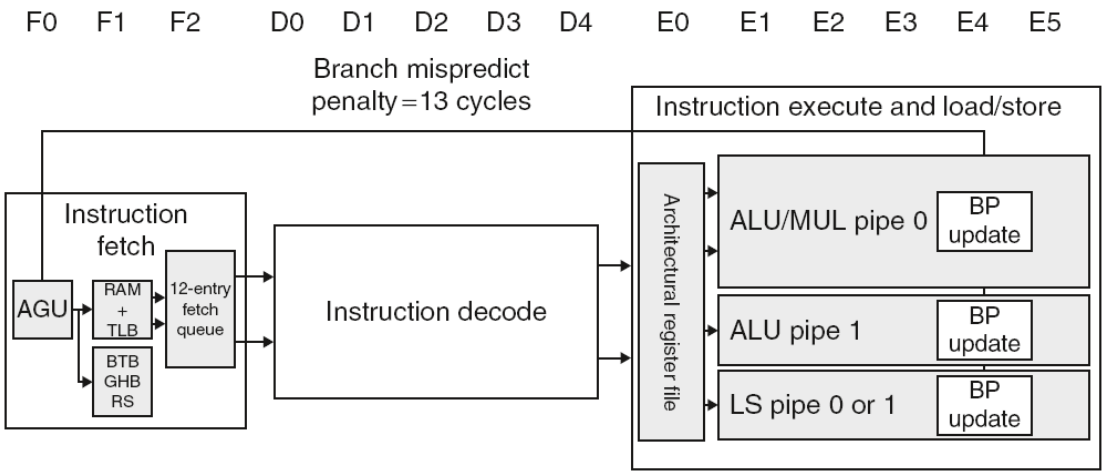
¿Por qué hacer planificación dinámica?

- ¿Por qué no simplemente dejar que el compilador planifique el código?
- No todos los atascos son predecibles
  - ej., fallos de cache
- Los saltos no siempre se pueden planificar.
  - El resultado del salto se determina dinámicamente
- Diferentes implementaciones de ACI tienen diferentes riesgos y latencias.

¿Funciona el despacho múltiple?

- Si, pero no tanto como pretendemos
- Los programas tiene dependencias reales que limitan el PNI (paralelismo a nivel de instrucción)
- Algunas dependencias son dificiles de eliminar
  - ej. punteros
- A veces el paralelismo es dificil de determinar
  - Durante el despacho de instrucciones se limita el tamaño de la ventana.
- Demoras de la memoria y **ancho de banda** limitado.
  - Es dificil hacer que el pipe funcione al máximo rendimiento
- La especulación puede ayudar si está bien hecha.

ARM Cortex-A8 Pipeline



¿Qué vimos?

-Aumentar el PNI con despacho múltiple.
-Planificación múltiple estática.
  ~desenrollado de loop
  ~renombramiento de registros
-Planificación múltiple dinámica.
  ~despacho
  ~ejecución (estaciones de reservación)
  ~finalización (commit)(buffer de reordenamiento)