uc3m | Universidad Carlos III de Madrid

Paralelismo a nivel de instrucción

Arquitectura de Computadores

J. Daniel García Sánchez (coordinador)

Departamento de Informática
Universidad Carlos III de Madrid

Licencia Creative Commons

Este trabajo se distribuye bajo licencia **Reconocimiento-NoComercial-SinObraDerivada 4.0 Internacional (CC BY-NC-ND 4.0)**.

Usted es libre de:

Compartir – copiar y redistribuir el material en cualquier medio o formato.
El licenciador no puede revocar estas libertades mientras cumpla con los términos de la licencia.

Bajo las condiciones siguientes:

Reconocimiento – Debe **reconocer adecuadamente la autoría**, proporcionar un enlace a la licencia e **indicar si se han realizado cambios**. Puede hacerlo de cualquier manera razonable, pero no de una manera que sugiera que tiene el apoyo del licenciador o lo recibe por el uso que hace.
NoComercial – No puede utilizar el material para una **finalidad comercial**.
SinObraDerivada – Si **remezcla, transforma o crea a partir** del material, no puede difundir el material modificado.

No hay restricciones adicionales – No puede aplicar **términos legales** o **medidas tecnológicas** que legalmente restrinjan realizar aquello que la licencia permite.

Texto completo de la licencia disponible en
`https://creativecommons.org/licenses/by-nc-nd/4.0/legalcode`

1 Riesgos de control

2 Predicción de saltos

3 Operaciones multiciclo

4 Técnicas de compilación e ILP

5 Conclusión

Riesgos de control

- Un **riesgo de control** se produce en una instrucción de alteración del PC.
  - **Terminología**:
    - **Bifurcación tomada**: Si se modifica el PC.
    - **Bifurcación no tomada**: Si no se modifica el PC.
  - **Problema**:
    - La segmentación asume que lo bifurcación **no** se tomará.
  - Si se toma la bifurcación $\Rightarrow$ actualización de **PC** al final de **ID**.
    - **Cálculo de dirección efectiva**: Sumador en **ID**.
    - **Evaluación de condición**: Comparador en **ID**.
  - ¿Qué hacer si después de la etapa **ID** se determina que **SI** hay que tomar la bifurcación?

Alternativas ante riesgos de control

- **Tiempo de compilación**: Prefijadas durante toda la ejecución del programa.
  - El software puede intentar minimizar su impacto si conoce el comportamiento del hardware.
  - El compilador puede hacer este trabajo.

- **Alternativas en tiempo de ejecución**: Comportamiento variable durante la ejecución del programa.
  - Intentan predecir qué hará el software.

Riesgos de control: soluciones estáticas

1. Congelación del *pipeline*.
2. Predicción prefijada.
   - Siempre no tomado.
   - Siempre tomado.
3. Bifurcación con retraso.

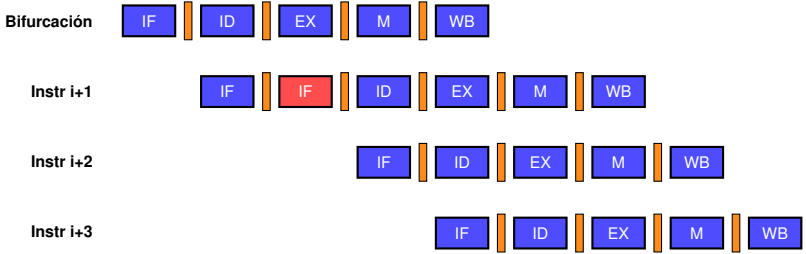
- En muchos casos el compilador necesita saber qué se va a hacer para reducir el impacto.

Congelación del pipeline

- **Idea**: Si la instrucción actual es una bifurcación → parar o eliminar del pipeline instrucciones posteriores hasta que se conozca el destino.
  - Penalización en tiempo de ejecución conocida.
  - El software (compilador) no puede hacer nada.

- El destino de la bifurcación se conoce en la etapa **ID**.
  - Repetir el **FETCH** de la siguiente instrucción.

Repetición de captación



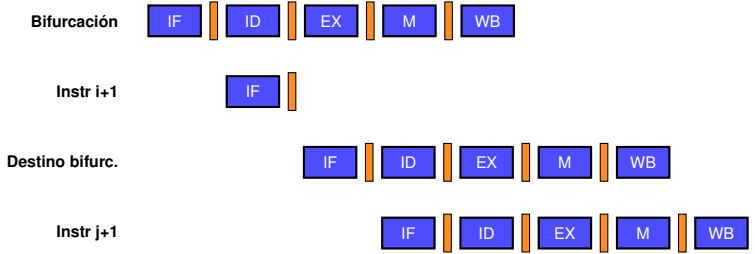
- La repetición de IF equivale a una detención.
- Una detención por bifurcación puede dar lugar a una **pérdida de rendimiento** de 10% a 30%.

Predicción prefijada: no tomada

- **Idea**: Asumir que la bifurcación será **no tomada**.
  - Se evita modificar el estado del procesador hasta que se tiene la confirmación de que la bifurcación no se toma.
  - Si la bifurcación se toma, las instrucciones siguientes se retiran del pipeline y se capta la instrucción en el destino del salto.
    - Transformar instrucciones en **NOP**.

- **Tarea del compilador**:
  - Organizar código poniendo la opción más frecuente como no tomada e invirtiendo condición si es necesario.

Predicción prefijada: no tomada



- Cuando se sabe que el salto se tomará se capta la nueva instrucción.

Predicción prefijada: tomada

- **Idea**: Asumir que la bifurcación será **tomada**.
  - Tan pronto como se decodifica la bifurcación y se calcula el destino se comienza a captar instrucciones del destino.
  - En pipeline de 5 etapas no aporta ventajas.
    - No se conoce dirección destino antes que decisión de bifurcación.
    - Útil en procesadores con condiciones complejas y lentas.

- **Tarea del compilador**:
  - Organizar código poniendo la opción más frecuente como tomada e invirtiendo condición si es necesario.

Bifurcación retrasada

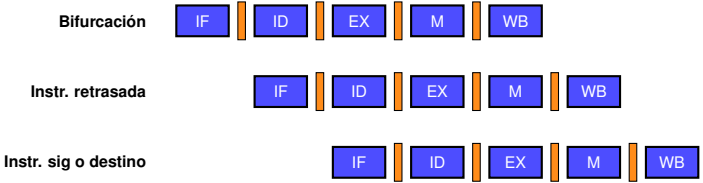
- **Idea**: La bifurcación se produce después de ejecutar las **n** instrucciones posteriores a la propia instrucción de bifurcación.
  - **En pipeline de 5 etapas** → 1 ranura de retraso (*delay slot*).

Ejemplo

```
I0:     bnez x1, etiq
I1:     addi x2, x2, 1
I2:     mul  x3, x2, x4
...
IN:     sub  x1, x1, 1
IN+1:   mul  x3, x3, x4
```

- Las instrucciones **I1**, **I2**, . . . , **IN** se ejecutan independientemente del sentido de la condición de bifurcación.
- La instrucción **IN+1** solamente se ejecuta si no se produce la bifurcación.

Bifrucación retrasada



- Caso de bifurcación retrasada con una ranura de retraso.
- Se espera siempre una instrucción antes de tomar la bifurcación.
- Es responsabilidad del programador poner código útil en la ranura.

Bifurcaciones retrasadas

- Efectividad del compilador para el caso de 1 ranura:
  - **Rellena** alrededor del 60% de ranuras.
  - En torno al 80% de instrucciones ejecutadas en ranuras **útiles** para computación.
  - En torno al 50% de ranuras **rellenados** de forma útil.

- Usada en primeros en pipelines cortos y simples:
  - **Razón**: Predicción hardware era muy costosa.
  - Complica la implementación en presencia de predicción dinámica de saltos.
  - No incluida en RISC-V.

Rendimiento y predicción prefijada

- El número de detenciones de bifurcaciones depende de:
  - Frecuencia de las bifurcaciones.
  - Penalización por bifurcación.

- Ciclos de penalización por bifurcación:

$$ciclos_{bifurc} = frecuencia_{bifurc} \times penaliz_{bifurc}$$

- Aceleración

$$S = \frac{profundidad_{pipeline}}{1 + frecuencia_{bifurc} \times penaliz_{bifurc}}$$

Caso práctico

- **MIPS R4000** (pipeline más profundo).
  - 3 etapas antes de conocer destino de bifurcación.
  - 1 etapa adicional para evaluar condición.
  - Asumiendo que no hay detenciones de datos en comparaciones.
  - **Frecuencia de bifurcaciones**:
    - **Bifurcación incondicional**: 4%.
    - **Bifurcación condicional, no-tomada**: 6%
    - **Bifurcación condicional, tomada**: 10%

Esquema bifurcación	Penalización		
	incondicional	no-tomada	tomada
Vaciar pipeline	2	3	3
Predecir tomada	2	3	2
Predecir no-tomada	2	0	3

Solución

Esquema bifurcación	Bifurcación			Total
	incondicional	no-tomada	tomada	
Frecuencia	4%	6%	10%	20%
Vaciar pipeline	$0.04 \times 2 = 0.08$	$0.06 \times 3 = 0.18$	$0.10 \times 3 = 0.30$	0.56
Predecir tomada	$0.04 \times 2 = 0.08$	$0.06 \times 3 = 0.18$	$0.10 \times 2 = 0.20$	0.46
Predecir no-tomada	$0.04 \times 2 = 0.08$	$0.06 \times 0 = 0.00$	$0.10 \times 3 = 0.30$	0.38

Contribución sobre CPI ideal

Speedup de predecir **tomada** sobre vaciar pipeline

$$S = \frac{1 + 0.56}{1 + 0.46} = 1.068$$

Speedup de predecir **no tomada** sobre vaciar pipeline

$$S = \frac{1 + 0.56}{1 + 0.38} = 1.130$$

1 Riesgos de control

2 Predicción de saltos

3 Operaciones multiciclo

4 Técnicas de compilación e ILP

5 Conclusión

Bifurcaciones y tiempo de ejecución

- Cada bifurcación condicional está **fuertemente sesgada**.
  - O se toma la mayoría de las veces,
  - O no se toma la mayoría de las veces.

- **Predicción basada en perfil de ejecución**:
  - Se ejecuta una vez para recoger estadísticas.
  - Se utiliza la información recogida para modificar el código y aprovechar la información.

Predicciones con perfil de ejecución

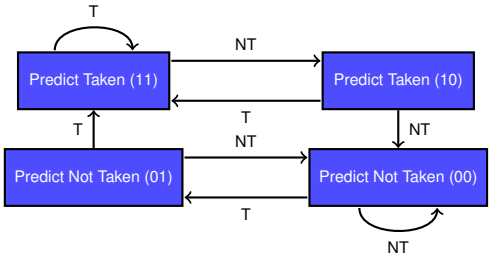
- SPEC92: Frecuencia de bifurcaciones 3% a 24%
- **Coma flotante**:
  - **Fallo de predicción**.
    - **Media**: 9%.
    - **Desviación estándar**: 4%.
- **Enteros**:
  - **Fallo de predicción**.
    - **Media**: 15%.
    - **Desviación estándar**: 5%.
- La predicción basada en perfil de ejecución suele funcionar mejor con aplicaciones intensivas en coma flotante.

Predicción dinámica: BHT

- Tabla histórica de saltos (Branch History Table)
  - **Índice**: Bits menos significativos de dirección (PC).
  - **Valor**: 1 bit (salto tomado o no la última vez).
- **Efectos**:
  - Se desconoce si la predicción es correcta.
    - Podría venir de otra instrucción situada en una dirección diferente con los mismos bits menos significativos.
  - Número de bits menos significativos implica tamaño de búfer.
    - 10 bits $\Rightarrow$ 1024 entradas.
  - Si la predicción falla se invierte bit.
  - **Desventaja**: Bifurcación de bucle falla dos veces.
    - Primera y última iteración.

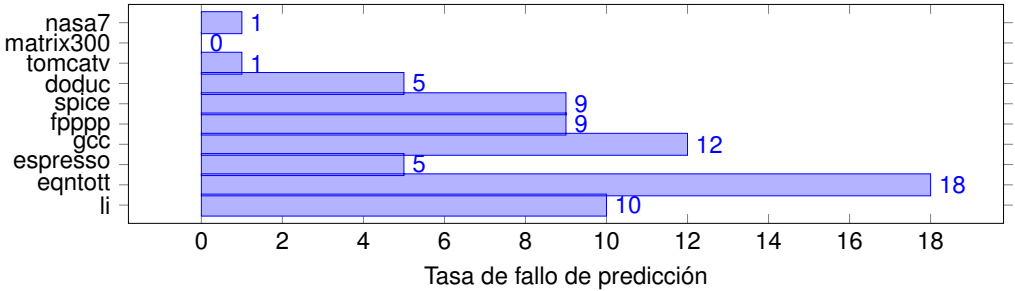
Predicción dinámica: BHT

- Tabla histórica de saltos (Branch History Table)
  - **Índice**: Bits menos significativos de dirección (PC).
  - **Valor**: 1 bit (salto tomado o no la última vez).
    - **00** y **01**: Predecir a no tomado.
    - **10** y **11**: Predecir a tomado.



BHT: Precisión

- Fallos de predicción:
  - Predicción errónea para el salto.
  - Historia de otro salto en la entrada de la tabla.
- Resultados BHT de 2 bits y 4K entradas:



Predicción dinámica de bifurcación

- ¿Por qué funciona la predicción de saltos?
  - El algoritmo presenta regularidades.
  - Las estructuras de datos presenta regularidades.

- ¿Es mejor la predicción dinámica que la predicción estática?
  - Parecer serlo.
  - Hay un pequeño número de bifurcaciones importantes en programas con comportamiento dinámico.

1 Riesgos de control

2 Predicción de saltos

3 Operaciones multiciclo

4 Técnicas de compilación e ILP

5 Conclusión

Operaciones multi-ciclo

- **Actividad**
  1. *Lea* la **Sección C.5** – Extending the RISC-V Integer Pipeline to handle multicycle operations (pág C-45 – C-55).
  2. **Computer Architecture: A Quantitative Approach**. 6th Ed. Hennessy and Patterson. Morgan Kaufmann. 2017.

  3. **Aspectos clave**:
     - ¿Por qué la coma flotante requiere operaciones multiciclo?
     - ¿Por qué no todas las operaciones de coma flotante tienen la misma longitud?
     - ¿Qué son la latencia y el intervalo de iniciación?
     - ¿Cómo se ven afectados los riesgos y el envío adelantado?

1 Riesgos de control

2 Predicción de saltos

3 Operaciones multiciclo

4 Técnicas de compilación e ILP

5 Conclusión

Aprovechamiento de ILP

- ILP directamente aplicables a bloques básicos.
  - **Bloque básico**: secuencia de instrucciones sin saltos.
  - Programa **típico** en procesadores RISC:
    - Tamaño medio de bloque básico de 3 a 6 instrucciones.
    - Poco aprovechamiento de ILP dentro del bloque.
  - Es necesario explotar ILP entre bloques básicos.

Ejemplo

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

- **Paralelismo a nivel de bucle**.
  - Transformable a ILP.
  - Por compilador o hardware.
- **Alternativa**:
  - Instrucciones vectoriales.
  - Instrucciones SIMD en procesador.

Planificación y desenrollamiento de bucles

- **Aprovechamiento de paralelismo**:
  - Entrelazar ejecución de instrucciones no relacionadas.
  - Rellenar detenciones con instrucciones.
  - No alterar los efectos del programa original.

- El compilador puede usar el conocimiento detallado de la arquitectura.

Aprovechamiento de ILP

Ejemplo

```
for (i=999;i>=0;i--) {
  x[i] = x[i] + s;
}
```

- Cuerpo de cada iteración es independiente.

Latencias entre instrucciones

Instrucción que produce resultado	Instrucción que usa resultado	Latencia (ciclos)
Operación ALU FP	Operación ALU FP	3
Operación ALU FP	Almacenar doble	2
Cargar doble	Operación ALU FP	1
Cargar doble	Almacenar doble	0

Código compilado

- **x1** → Último elemento de array.
- **f2** → Escalar **s**.
- **x2** → Precalculado para que **8(x2)** sea primer elemento de array.

Código ensamblador

```
loop:   fld f0, 0(x1)          // f0 <- x[i]
        fadd.d f4, f0, f2      // f4 <- f0 + s
        fsd f4, 0(x1)          // x[i] <- f4
        addi x1, x1, -8        // --i
        bne x1, x2, loop       // x1!=x2 => loop
```

Detenciones en la ejecución

Original

```
loop:   fld f0, 0(x1)
        fadd.d f4, f0, f2
        fsd f4, 0(x1)
        addi x1, x1, -8
        bne x1, x2, loop
```

Detenciones

```
loop:   fld f0, 0(x1)
        <stall>
        fadd.d f4, f0, f2
        <stall>
        <stall>
        fsd f4, 0(x1)
        addi x1, x1, -8
        bne x1, x2, loop
```

Planificación de bucle

Original

```
loop:   fld f0, 0(x1)
        <stall>
        fadd.d f4, f0, f2
        <stall>
        <stall>
        fsd f4, 0(x1)
        addi x1, x1, -8
        bne x1, x2, loop
```

- 8 ciclos por elemento.

Planificado

```
loop:   fld f0, 0(x1)
        addi x1, x1, -8
        fadd.d f4, f0, f2
        <stall>
        <stall>
        fsd f4, 8(x1)
        bne x1, x2, loop
```

- 7 ciclos por elemento.

Desenrollamiento de bucles

- **Idea**:
  - Replicar varias veces el cuerpo del bucle.
  - Ajustar el código de terminación.
  - Usa registros distintos para cada réplica para reducir dependencias.

- **Efecto**:
  - Aumenta la longitud del bloque básico.
  - Incrementa el aprovechamiento ILP disponible.

Desenrollamiento

Desenrollado (x4)

```
loop:   fld f0, 0(x1)
        fadd.d f4, f0, f2
        fsd f4, 0(x1)
        fld f6, -8(x1)
        fadd.d f8, f6, f2
        fsd f8, -8(x1)
        fld f10, -16(x1)
```

Desenrollado (x4)

```
        fadd.d f12, f10, f2
        fsd f12, -16(x1)
        fld f14, -24(x1)
        fadd.d f16, f14, f2
        fsd f16, -24(x1)
        addi x1, x1, -32
        bne x1, x2, loop
```

- 4 iteraciones requieren más registros.
- Este ejemplo asume tamaño de array múltiplo de 4.

Detenciones y desenrollamiento

Desenrollado (x4)

```
loop:   fld f0, 0(x1)
        <stall>
        fadd.d f4, f0, f2
        <stall>
        <stall>
        fsd f4, 0(x1)
        fld f6, -8(x1)
        <stall>
        fadd.d f8, f6, f2
        <stall>
        <stall>
        fsd f8, -8(x1)
        fld f10, -16(x1)
        <stall>
```

Desenrollado (x4)

```
        fadd.d f12, f10, f2
        <stall>
        <stall>
        fsd f12, -16(x1)
        fld f14, -24(x1)
        <stall>
        fadd.d f16, f14, f2
        <stall>
        <stall>
        fsd f16, -24(x1)
        addi x1, x1, -32
        <stall>
        bne x1, x2, loop
```

- 27 ciclos por 4 iteraciones $\rightarrow$ 6.75 ciclos por elemento.

Planificación y desenrollamiento

Desenrollado (x4)

```
loop:   fld f0, 0(x1)
        fld f6, -8(x1)
        fld f10, -16(x1)
        fld f14, -24(x1)
        fadd.d f4, f0, f2
        fadd.d f8, f6, f2
        fadd.d f12, f10, f2
        fadd.d f16, f14, f2
        fsd f4, 0(x1)
        fsd f8, -8(x1)
        fsd f12, -16(x1)
        addi x1, x1, -32
        fsd f16, 8(x1)
        bne x1, x2, loop
```

- Reorganización de código.
  - Respetando dependencias.
  - Semánticamente equivalente.
  - **Objetivo**: Aprovechar *stalls*.
- Actualización de **x1** suficientemente separada de **BNE**.
- 14 ciclos por 4 iteraciones → 3.5 ciclos por elemento.

Límites del desenrollamiento de bucles

- **Decremento** de la **ganancia** con cada desenrollamiento.
  - Ganancia limitada a eliminación de detenciones.
  - Sobrecoste se reparte entre iteraciones.
- **Crecimiento** en **tamaño** de código.
  - Puede afectar a la tasa de fallos de la caché de instrucciones.
- **Presión** sobre **banco registros**.
  - Puede generar falta de registros.
  - Si no hay registros suficientes se pierden las ventajas.

1 Riesgos de control

2 Predicción de saltos

3 Operaciones multiciclo

4 Técnicas de compilación e ILP

5 Conclusión

Resumen

- Riesgos de control:
  - Alternativas en tiempo de compilación.
  - Alternativas en tiempo de ejecución.
- Las operaciones multi-ciclo permiten mantener ciclos de reloj cortos.
- El desenrollamiento de bucles permite ocultar las latencias de detenciones, pero da una ganancia limitada.

Referencias

- **Computer Architecture: A Quantitative Approach**. 6th Ed.
  Hennessy and Patterson. Morgan Kaufmann. 2017.
  - C.2: The Major Hurdle of Pipelining – Pipeline Hazards (pág C-18 – C-26).
  - C.5: Extending the RISC V Integer Pipeline to Handle Multicycle Operations (pág C-45 – C-55).
  - 3.1: Instruction-Level Parallelism: Concepts and Challenges (pág 168 – 176).
  - 3.2: Basic Compiler Techniques for Exposing ILP (pág 176 –182).

uc3m | Universidad Carlos III de Madrid

Paralelismo a nivel de instrucción

Arquitectura de Computadores

J. Daniel García Sánchez (coordinador)

Departamento de Informática
Universidad Carlos III de Madrid