Tema 3. Técnicas de procesamiento de saltos

ESTRUCTURA DE COMPUTADORES

Grupo de Arquitectura de Computadores (GAC)

Índice

Índice

Procesamiento de saltos

- Los **riesgos de control** se originan a partir de las instrucciones de salto condicional que determinan la secuencia de instrucciones que hay que procesar tras ellas
- Cuando se ejecuta un salto, éste puede cambiar o no el contador de programa (**PC**): salto **efectivo** / salto **no efectivo**
- El método más simple para tratar con saltos consiste en purgar el cauce y repetir la búsqueda de la instrucción siguiente al salto una vez que se detecta el salto.
- Pueden provocar una mayor pérdida de rendimiento para un cauce MIPS que un riesgo de datos
- Unos ciclos de parada en cada salto no parecen mucho, pero en la práctica puede suponer una pérdida del rendimiento de entre un 10 % y un 30 %, dependiendo de la frecuencia de saltos y de la latencia de los mismos (etapa en la que se decide el salto)

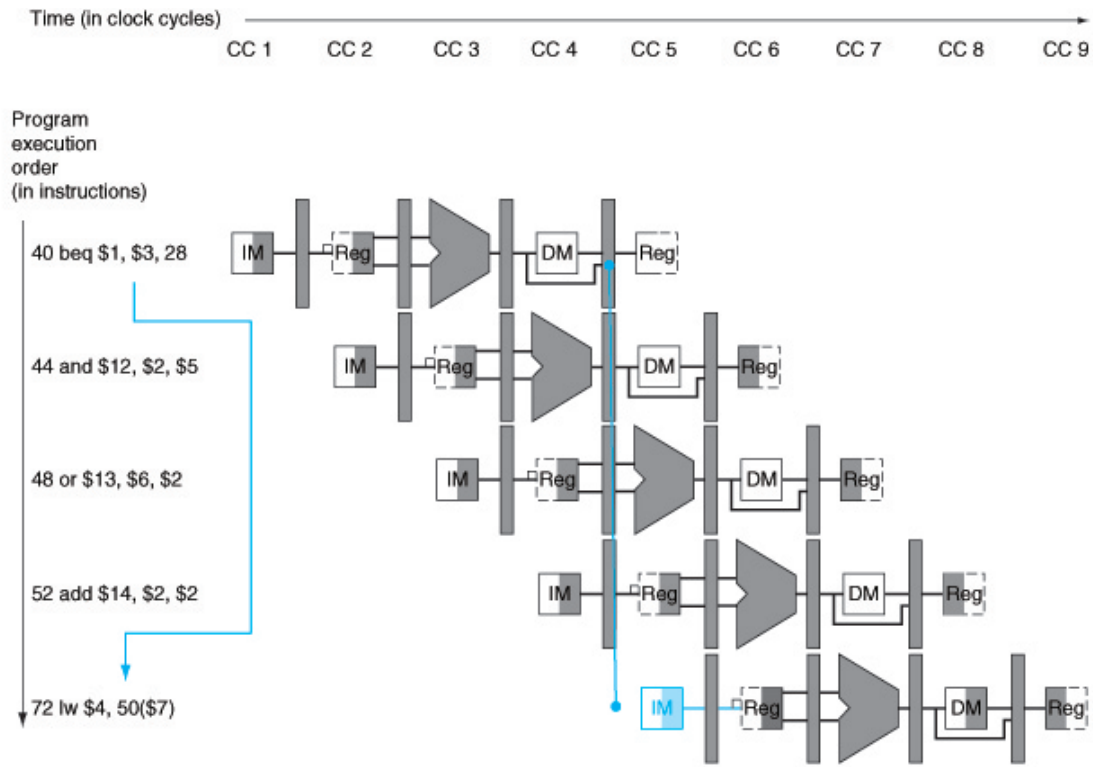
Procesamiento de saltos

Existen diferentes técnicas de procesamiento de salto, para minimizar las detenciones debido a las instrucciones de salto

- Purga del cauce
- Reducir la latencia del salto
- Predicción de salto
- Salto retardado

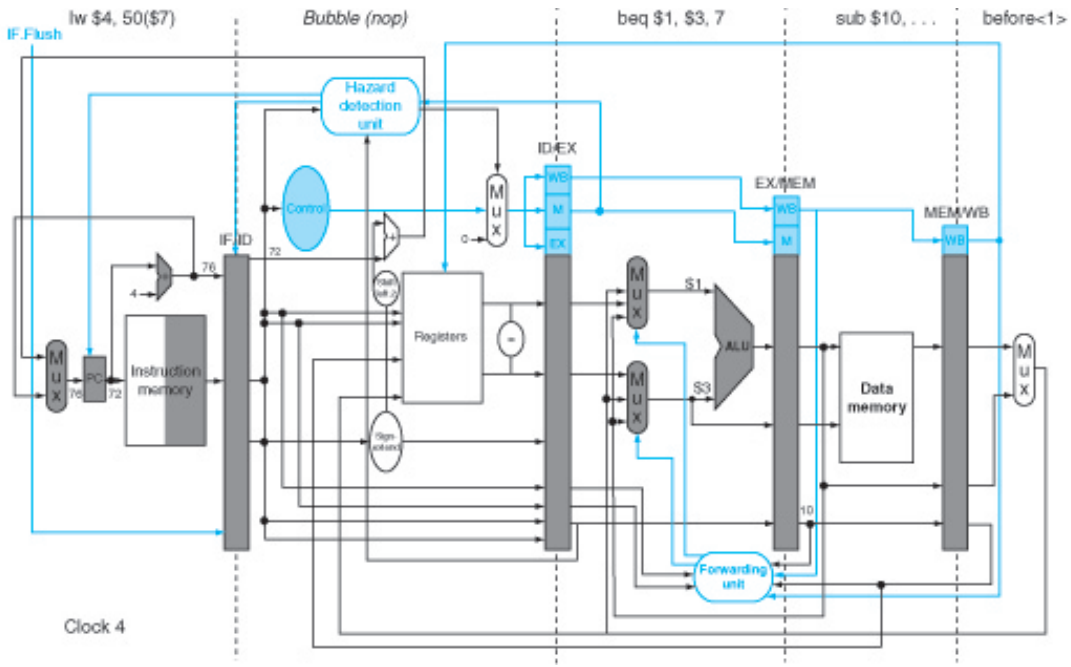
Purga del cauce

- Eliminar del cauce aquellas instrucciones que han entrado después de la instrucción de salto y antes de que este se resuelva
- Solución muy simple
- Altamente ineficiente



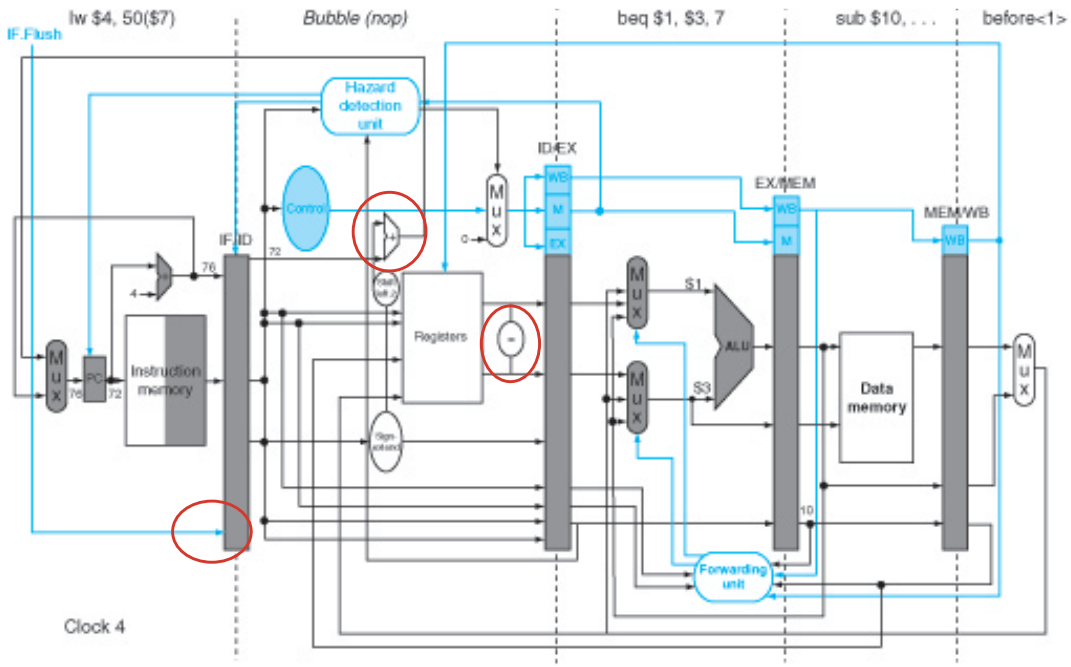
Reducir la latencia del salto

- Adelantar la ejecución del salto de la etapa MEM a la etapa ID
- Implica la modificación del hardware y del control del cauce



Reducir la latencia del salto

- Adelantar la ejecución del salto de la etapa MEM a la etapa ID
- Implica la modificación del hardware y del control del cauce



Índice

Predicción de salto

Existen dos tipos de predicción de saltos

- Predicción **fija**: la predicción no depende del comportamiento dinámico del salto, el hardware siempre realiza la misma predicción para cada salto
- Predicción **dinámica**: depende del comportamiento del salto en tiempo de ejecución y cambiará si el salto cambia su comportamiento durante la ejecución.

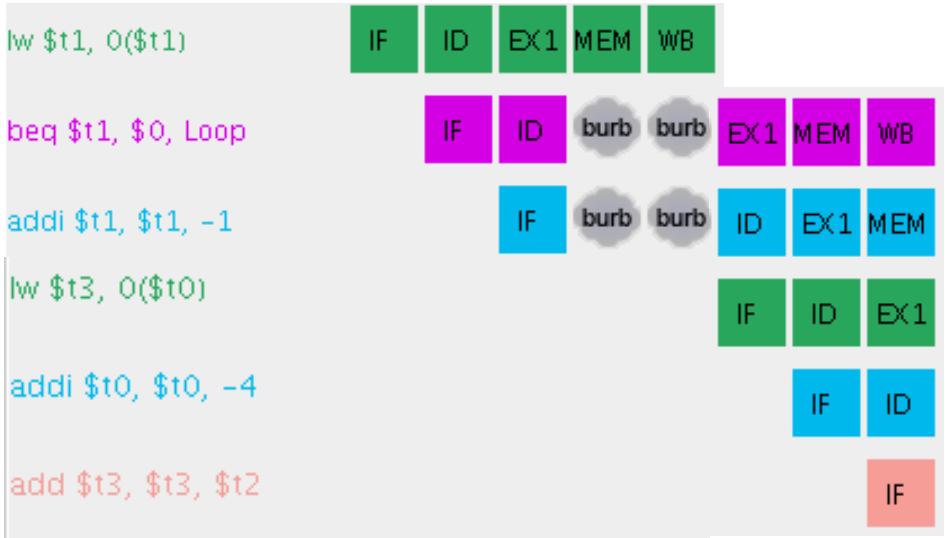
Predicción fija

- Predicción de salto **no efectivo**
  - Se continúa la búsqueda de instrucciones de forma normal
  - Si el salto finalmente no es efectivo, se continúa con la ejecución, sin penalización
  - Si el salto finalmente es efectivo, se purga el cauce, borrando las instrucciones que se habían introducido en él después del salto
- Predicción de salto **efectivo**
  - Se calcula la dirección de salto y se comienza la etapa IF a partir de ella
  - Si el salto finalmente es efectivo, se continúa con la ejecución, con una penalización mínima (la latencia del cálculo de la dirección de salto)
  - Si el salto finalmente no es efectivo, se purga el cauce
  - Está solución es sólo de aplicación en procesadores que tienen condiciones de salto complejas, donde el salto se decide en etapas posteriores al cálculo de la dirección destino. Inútil en el caso del procesador MIPS que ponemos como ejemplo.
- El compilador puede mejorar rendimiento reorganizando el código de forma que el camino frecuente coincida con la predicción

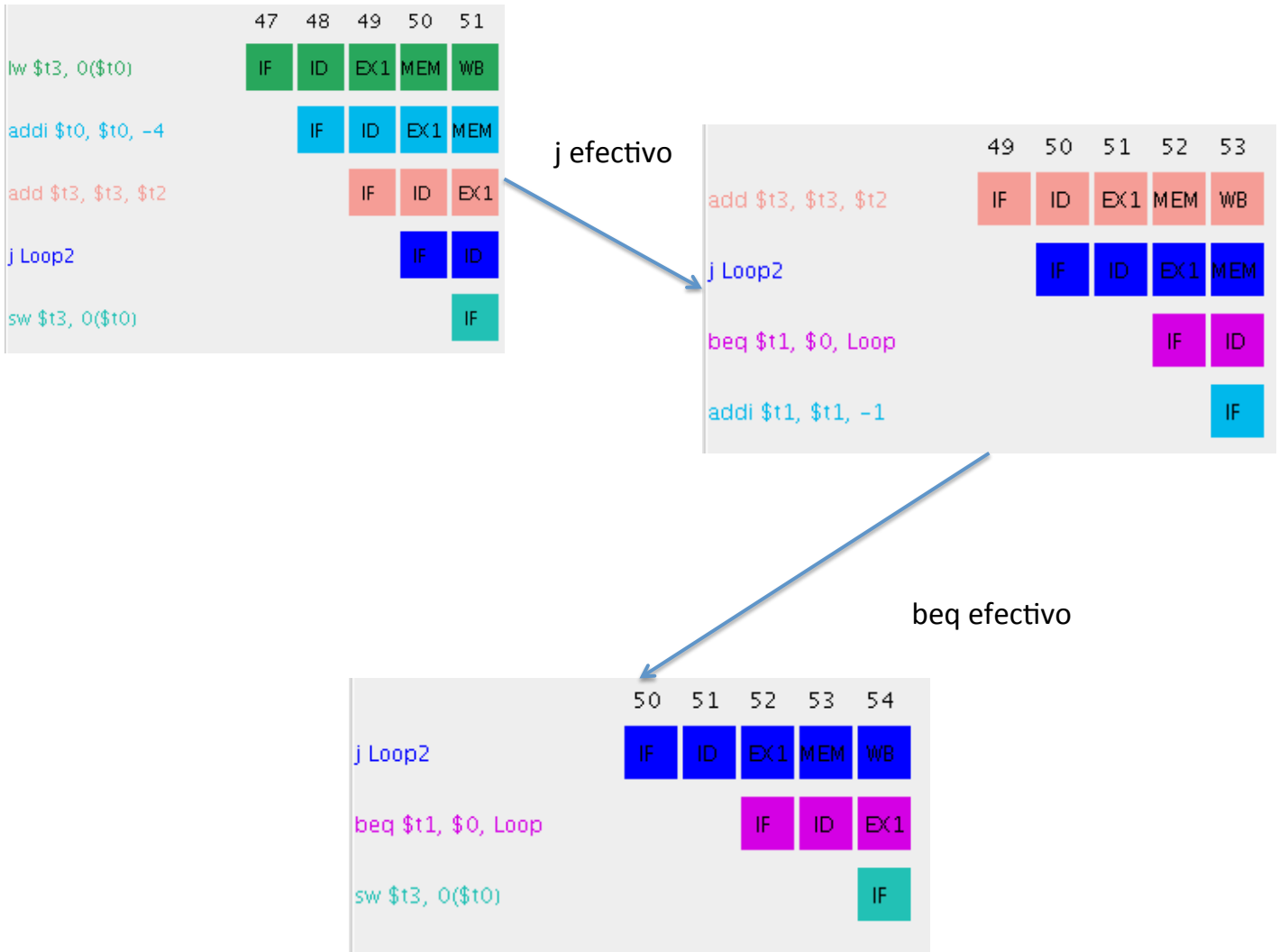
Ejemplo

```
            lw    $t1,   0($t1)
Loop2 :
            beq   $t1,   $0,    Loop
            addi  $t1,   $t1,   -1
            lw    $t3,   0($t0)
            addi  $t0,   $t0,   -4
            add   $t3,   $t3,   $t2
            j     Loop2
Loop :
            sw    $t3,   0($t0)
```

Salto fijo no efectivo. Salto no efectivo



Salto fijo no efectivo. Salto efectivo



Índice

Predicción dinámica

La eficiencia de un esquema de predicción de saltos depende de:

- la precisión con la que conseguimos predecir el salto
- el coste del salto cuando la predicción es incorrecta, pero también cuando la predicción es correcta

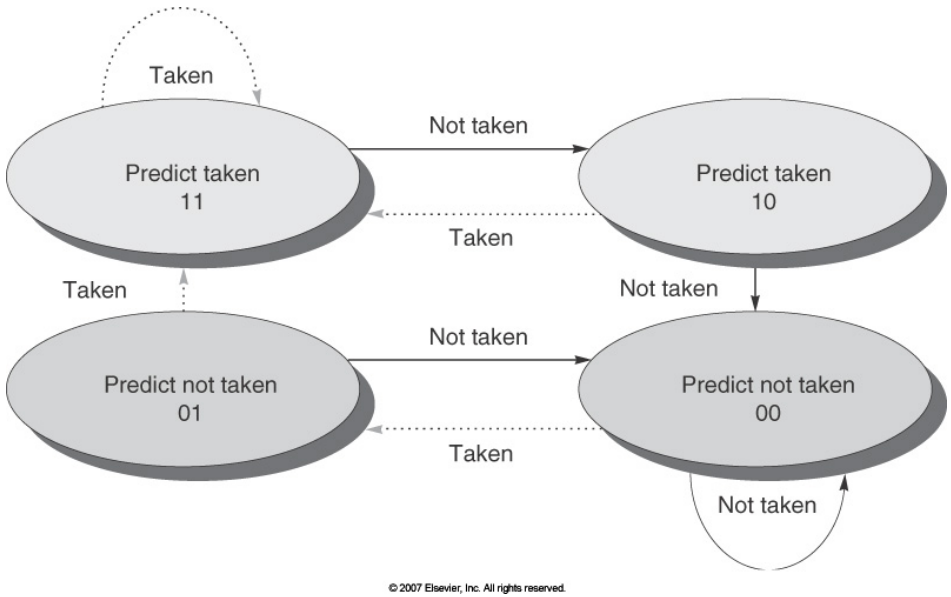
La penalización depende de la estructura del cauce, del tipo de predicción, y de las estrategias para recuperar el estado después de una predicción fallida.

Predicción dinámica

- El esquema más simple de predicción de saltos es un **buffer de predicción de saltos** o una tabla de históricos
  - pequeña memoria indexada por la porción inferior de la dirección de una instrucción de salto
  - contiene un bit que dice si el salto se ha tomado recientemente o no
- Si al final la predicción es incorrecta, el bit de predicción del buffer se invierte
- No se asegura que la información corresponda al salto en evaluación, podría estar puesta ahí por otro salto que tenga los mismos bits en las posiciones inferiores de su dirección
- Este esquema es útil sólo en los casos en que el retardo del salto es mayor que el tiempo para calcular el posible destino de salto

Predicción dinámica

- El esquema simple de predicción con 1 bit presenta desventajas desde el punto de vista del rendimiento
  - Supongamos que un salto se toma casi siempre (lazo **for**), cuando el salto no se tome haremos una predicción incorrecta dos veces
- Para remediarlo existen esquemas de predicción que usan 2 bits
  - una predicción debe fallar dos veces consecutivas antes de cambiar



Índice

Salto retardado

- Las instrucciones que se captan después de una instrucción de salto y antes de la modificación del PC se ejecutan en su totalidad
  - Se dice que estas instrucciones ocupan el **hueco de retardo**
- El compilador debe ubicar después de la instrucción de salto:
  - Una instrucción que no modifique la semántica del programa
  - Una **NOP**
- La tarea del compilador es hacer las instrucciones sucesoras válidas y útiles

Salto retardado

Existen varias técnicas que se pueden usar para planificar el hueco de retardo y sacar rendimiento:

- Mover una instrucción **desde antes**: el hueco de retardo se rellena con una instrucción independiente del salto que se tenía que ejecutar antes que este. Es la mejor opción.
- Mover una instrucción **desde destino**: el hueco de retardo se rellena con una instrucción que se tiene que ejecutar si este es tomado.
- Mover una instrucción **desde instrucciones siguientes**: el hueco de retardo se rellena con una instrucción que se tiene que ejecutar si este no es tomado.

Una precaución que hay que tener es que esta instrucción ha de ser tal que no pueda modificar la semántica del código en caso de que el salto sea tomado.

Desde antes

```
        add   $t0, $t1, $t2
        sub   $t3, $t0, $t4
        beq   $t5, $t6, Salto
        ???
        sw    $t3, 0($s0)
        ...
Salto:
        slt   $t3, $t3, $t5
```

```
        add   $t0, $t1, $t2
        beq   $t5, $t6, Salto
        sub   $t3, $t0, $t4
        sw    $t3, 0($s0)
        ...
Salto:
        slt   $t3, $t3, $t5
```

$$CPI = 1$$

Desde destino

```
Salto:
        add   $t1, $t2, $t2
        add   $t0, $t1, $t2
        beq   $t5, $t0, Salto
        ???
        lw    $t1, 0($s0)
```

```
        add   $t1, $t2, $t2
Salto:
        add   $t0, $t1, $t2
        beq   $t5, $t0, Salto
        add   $t1, $t2, $t2
        lw    $t1, 0($s0)
```

$$\begin{aligned} \textit{Salto efectivo} &\Longrightarrow CPI = 1 \\ \textit{Salto no efectivo} &\Longrightarrow CPI = 2 \end{aligned}$$

Desde Instrucciones siguientes

```
        add $t0 , $t1, $t2
        beq $t5, $t0 , Salto
        ???
        slt   $t3, $t4, $t5
        sw    $t3, 0($s0)
        ...
Salto:
        and $t3, $t4, $t5
```

```
        add $t0, $t1, $t2
        beq $t5, $t0, Salto
        slt   $t3, $t4, $t5
        sw    $t3, 0($s0)
        ...
Salto:
        and $t3, $t4, $t5
```

$$\begin{aligned} Salto\ \ efectivo &\Longrightarrow CPI = 2 \\ Salto\ \ no\ \ efectivo &\Longrightarrow CPI = 1 \end{aligned}$$

No se puede usar ninguna técnica

```
        add $t0 , $t1, $t2
        beq $t5, $t0 , Salto
        ???
        slt   $t3, $t4, $t5
        ...
Salto:
        and $t4, $t3, $t5
```

```
        add $t0 , $t1, $t2
        beq $t5, $t0 , Salto
        nop
        slt   $t3, $t4, $t5
        ...
Salto:
        and $t4, $t3, $t5
```

$$CPI = 2$$

Bibliografía

- John L. Hennessy y David A. Patterson. *Computer Arquitecture. A Quantitative Approach*. Morgan Kaufmann 2007