

PRÁCTICA 3: OPTIMIZACIONES EN EL MIPS

ARQUITECTURA DE COMPUTADORES. 2º CURSO

1. OBJETIVOS

Con esta práctica se pretende que el alumno refuerce y amplíe los conocimientos adquiridos en sesiones de teoría sobre las optimizaciones del procesador encaminadas a reducir los bloqueos y disminuir el tiempo de ejecución de los programas. Para ello, el alumno experimentará con las optimizaciones que el simulador proporciona y determinará la mejora que se obtiene en cada caso.

2. PREPARACIÓN

Antes de acudir a la sesión de laboratorio el alumno debe:

- Leer y asimilar los contenidos teóricos del apartado 3 de este boletín.
- Realizar el cuestionario que se encuentra en la última página de este boletín de forma voluntaria para afianzar los conocimientos y de cara a la siguiente sesión presencial.

3. INTRODUCCIÓN TEÓRICA.

En esta sesión el alumno trabajará con las siguientes optimizaciones del procesador:

- Reordenación de instrucciones.
- Salto retardado.

3.1. REORDENACIÓN DE INSTRUCCIONES

Si bien esta optimización se ha estudiado en las sesiones teóricas y se ha incidido levemente en ella durante la sesión práctica anterior, es interesante repasarla a nivel del simulador para poder completar satisfactoriamente los ejercicios propuestos en esta sesión.

Una forma de eliminar los riesgos por dependencia de datos RAW consiste en separar durante la ejecución las instrucciones dependientes hasta que el riesgo desaparezca. Como se ha visto en clase, la forma más inmediata de conseguirlo es bloquear la instrucción dependiente (la que entra en ejecución más tarde). La figura 1 muestra un ejemplo de ejecución sin *forwarding* en el que la instrucción *sub* se detiene mientras *add* continúa avanzando durante 2 ciclos para alcanzar la etapa WB que hace desaparecer el riesgo por dependencia.

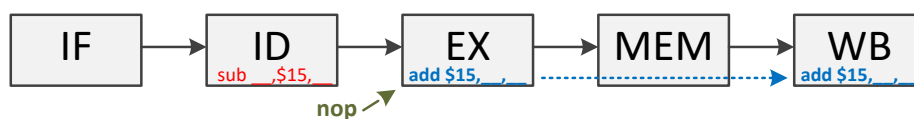


Figura 1. Ejecución de dos instrucciones en un procesador sin forwarding. La instrucción *sub* permanece bloqueada hasta que *add* alcance WB.

Los bloqueos de datos impiden la ejecución de otras instrucciones lo que provoca que el CPI aumente y por tanto que el rendimiento disminuya. Estos bloqueos provocan la inyección de una instrucción *nop* por cada ciclo de bloqueo en la etapa siguiente a la bloqueada (etapa EX en la figura 1). Las *nop* inyectadas están ocupando recursos del procesador y no realizan ninguna operación útil desde el punto de vista del programador.

	1	2	3	4	5	6
add \$15, \$16, \$17	IF	ID	EXi	MEM	WB	
sub \$5, \$15, \$7		IF	ID	ID	ID	EXi
sll \$21, \$21, 2			IF	IF	IF	ID
addiu \$25, \$25, 0x0004						IF

	1	2	3	4	5
add \$15, \$16, \$17	IF	ID	EXi	MEM	WB
sll \$21, \$21, 2		IF	ID	EXi	MEM
sub \$5, \$15, \$7			IF	ID	ID
addiu \$25, \$25, 0x0004				IF	IF

	1	2	3	4
add \$15, \$16, \$17	IF	ID	EXi	MEM
sll \$21, \$21, 2		IF	ID	EXi
addiu \$25, \$25, 0x0004			IF	ID
sub \$5, \$15, \$7				IF

a) Sin instrucciones entre *add* y *sub*. Provoca 2 ciclos de bloqueo

b) *sll* ha sido introducida entre *add* y *sub*. Reduce los ciclos de bloqueo a 1.

c) Dos instrucciones entre *add* y *sub*. Elimina todos los ciclos de bloqueo.

Figura 2. Ejemplo de reducción de ciclos de bloqueo entre *add* y *sub* en un procesador sin forwarding mediante la inserción de otras instrucciones del programa (reordenación de instrucciones).

Una forma de reducir este impacto negativo en el rendimiento consiste en separarlas instrucciones que sufren riesgos por dependencia con otras instrucciones del programa. Al separarlas con estas otras instrucciones, se reducirá o incluso se eliminarán los ciclos de bloqueos antes producidos ya que las instrucciones *nop* inyectadas han sido sustituidas por otras que sí realizan operaciones útiles para el programa (véase el ejemplo de la figura 2)

Como es lógico, al reordenar las instrucciones la semántica del programa no debe cambiar (no debe alterarse el comportamiento del programa), para que esto no ocurra es muy importante que la instrucción que va a moverse **no tenga dependencia de datos con las instrucciones que va “encontrando a su paso”**. Véase el ejemplo de la figura 3, el cual se basa en el código de la figura 2b pero en el que *sll* y *addiu* poseen ahora dependencias RAW y WAW respectivamente con el registro \$5 de la instrucción *sub*.


<pre>add \$15,\$16,\$17 sub \$5,\$15,\$7 sll \$21,\$5,2 addiu \$5,\$25,4</pre>		<pre>add \$15,\$16,\$17 sll \$21,\$5,2 addiu \$5,\$25,4 sub \$5,\$15,\$7</pre>
--------------------------------------------------------------------------------	-----------------------------------------------------------------------------------	------------------------------------------------------------------------------------------------------

Figura 3. Ejemplo de una incorrecta reordenación al no tenerse en cuenta las dependencias.

Esta limitación es extensible también a las dependencias de control, esto es, la instrucción que va a ser movida no debe “atravesar” una instrucción de salto (ni etiquetas que sean destino de saltos) ya que la ejecución o no de la instrucción que ha sido movida dejaría de estar condicionada por el salto. Por tanto, los saltos y etiquetas van a actuar como “barreras” que delimitan bloques de instrucciones de forma que las instrucciones de un determinado bloque sólo podrán moverse dentro del bloque al que pertenece (figura 4).


<pre>add \$15,\$16,\$17 sub \$5,\$15,\$7 beq \$30,\$31, etiq sll \$21,\$21,2 addiu \$25,\$25,4</pre>		<pre>add \$15,\$16,\$17 sll \$21,\$21,2 sub \$5,\$15,\$7 beq \$30,\$31, etiq addiu \$25,\$25,4</pre>
------------------------------------------------------------------------------------------------------	-----------------------------------------------------------------------------------	-----------------------------------------------------------------------------------------------------------------

Figura 4. Ejemplo de una incorrecta reordenación al moverse más allá de una instrucción de salto

Además debe tenerse en cuenta que la reordenación de instrucciones puede provocar nuevos bloqueos pues, al separar instrucciones dependientes puede que se aproximen a otras dependientes que antes no se bloqueaban (véase el ejemplo de figura 5).

	1	2	3	4	5	6	7
addi \$25, \$0, 0x0001	IF	ID	EXi	MEM	WB		
add \$15, \$16, \$17		IF	ID	EXi	MEM	WB	
sub \$5, \$15, \$7			IF	ID	ID	ID	EXi
addiu \$25, \$25, 0x0004				IF	IF	IF	ID
sll \$21, \$21, 2							IF

	1	2	3	4	5	6
addi \$25, \$0, 0x0001	IF	ID	EXi	MEM	WB	
add \$15, \$16, \$17		IF	ID	EXi	MEM	WB
addiu \$25, \$25, 0x0004			IF	ID	ID	EXi
sll \$21, \$21, 2				IF	IF	ID
sub \$5, \$15, \$7						IF

Sin instrucciones entre *add* y *sub*. Provoca 2 ciclos de bloqueo (similar al caso de la figura 2a)

Al introducir dos instrucciones entre *add* y *sub* se elimina los bloqueos de *sub* pero aparece uno nuevo al aproximar *addiu* a *addi* (se soluciona intercambiando *addiu* con *sll*).

Figura 5. Ejemplo en un procesador sin forwarding de cómo la reordenación puede provocar que aparezcan nuevos bloqueos. Eliminar los bloqueos entre *sub* y *add* provoca un bloqueo entre *addiu* y *addi*.

Por último, hay situaciones en las que es posible reordenar instrucciones aunque tengan dependencias. El caso más habitual se encuentra en las instrucciones de carga y almacenamiento cuando las dependencias se producen con los registros dedicados al cálculo de la dirección. Puesto que en estas instrucciones la dirección de memoria se especifica mediante el modo de direccionamiento registro base más desplazamiento, ese desplazamiento puede modificarse para que la reordenación no altere la semántica del programa.

	1	2	3	4	5	6	7	8	9	10
addiu \$20, \$10, 40	IF	ID	EXi	MEM	WB					
s: lw \$5, 0(\$10)		IF	ID	EXi	MEM	WB				
addi \$5, \$5, 1			IF	ID	ID	ID	EXi	MEM	WB	
sw \$5, 0(\$10)				IF	IF	IF	ID	ID	ID	EXi
addiu \$10, \$10, 4							IF	IF	IF	ID
bne \$10, \$20, s										IF

	1	2	3	4	5	6	7	8	9
addiu \$20, \$10, 40	IF	ID	EXi	MEM	WB				
s: lw \$5, 0(\$10)		IF	ID	EXi	MEM	WB			
addiu \$10, \$10, 4			IF	ID	EXi	MEM	WB		
addi \$5, \$5, 1				IF	ID	ID	EXi	MEM	WB
sw \$5, -4(\$10)					IF	IF	ID	ID	ID
bne \$10, \$20, s							IF	IF	IF

Figura 6. Ejemplo de reordenación de una instrucción con dependencia

3.2. SALTOS RETARDADOS

Si bien este aspecto no se ha estudiado previamente (salvo en teoría), se hará una pequeña introducción a continuación con el fin de abordar algún ejercicio puntual. Tenga presente que, en la siguiente sesión práctica, se incidirá nuevamente en este concepto.

Los saltos retardados es una técnica orientada a reducir los bloqueos de control. Para llevar a cabo esta técnica es necesario que el procesador y el compilador (o el programador si fuese el caso) actúen conjuntamente. El procesador ejecutará siempre, se tome o no el salto, la instrucción posterior a la instrucción de salto en el código fuente por lo que no cancela su ejecución. Será tarea del compilador mover de forma adecuada una instrucción del código justo después de la instrucción de salto (hueco del salto) teniendo en cuenta que dicha instrucción nunca va a ser cancelada. Mediante esta técnica, en vez de provocar un ciclo de bloqueo de control en los saltos tomados por la cancelación de la instrucción posterior al salto, el procesador ejecutará dicha instrucción salto y que corresponderá cuando sea posible a la instrucción útil que el compilador haya ubicado en el hueco del salto.

Al igual que ocurría en la reordenación de instrucciones vista en la práctica anterior, para mover una instrucción al hueco del salto habrá que tener en cuenta las dependencias RAW, WAR y WAW con los registros del resto de instrucciones para no alterar el comportamiento del programa. En la elección de la instrucción a mover es preferible que la instrucción proceda del bloque de instrucciones anterior a la instrucción de salto pues las instrucciones de dicho bloque siempre van a ejecutarse. En caso de que no fuese posible optar por una anterior a la instrucción de salto, podrá seleccionarse una instrucción procedente de una de las ramas del salto (de la rama tomada o de la no tomada) siempre que su ejecución sea inocua para el comportamiento del programa en caso de que se ejecutara la rama contraria (véase la transparencia 63 del tema del procesador segmentado/paralelismo a nivel de instrucciones).

En un procesador MIPS con saltos retardados todos los saltos son retardados por lo que el hueco del salto debe ser rellenado siempre para asegurar que la ejecución del programa sea correcta. En caso de no encontrar una instrucción adecuada con la que rellenar el hueco del salto, podrá utilizarse una instrucción *nop* como última alternativa. Esta opción debe utilizarse como último recurso pues la ejecución de estas instrucciones *nop* actúan en realidad como un bloqueo (sino que introducido vía software) tanto para la rama tomada como no tomada. De hecho, en el cálculo del rendimiento, tales instrucciones *nop* no deberían contabilizar como instrucciones del programa.

```

1      addiu $20,$10,40
2  s:  lw $5,0($10)
3      addiu $10,$10,4
4      addi $5,$5,1
5      sw $5,-4($10)
6      bne $10,$20, s

```

Figura 9a. Código para un procesador *sin* salto retardado

```

      addiu $20,$10,40
s:    lw $5,0($10)
      addiu $10,$10,4
      addi $5,$5,1
      bne $10,$20, s
      sw $5,-4($10)

```

Figura 9b. Código para un procesador *con* salto retardado

Los cronogramas de la figura 10 muestran ejemplos de ejecución de los códigos de la figura 9 en su versión con y sin salto retardado. Según el código de la figura 9a, la instrucción 1 no puede utilizarse para rellenar el hueco de salto por encontrarse antes de la etiqueta 's' entre otros motivos, la instrucción 2 tampoco por su dependencia RAW y WAW con la instrucción 4, la instrucción 3 por su dependencia RAW con la instrucción de salto, la instrucción 4 por su dependencia RAW con la 5, en cambio *sw* sí puede moverse al hueco del salto por ser independiente de la instrucción de salto.

	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15
addiu \$20, \$10, 0x0028	IF	ID	EXi	MEM	WB										
lw \$5, 0x0000(\$10)		IF	ID	EXi	MEM	WB									
addiu \$10, \$10, 0x0004			IF	ID	EXi	MEM	WB								
addi \$5, \$5, 0x0001				IF	ID	ID	EXi	MEM	WB						
sw \$5, 0xFFFF(\$10)					IF	IF	ID	ID	ID	EXi	MEM	WB			
bne \$10, \$20, 0xFFFF							IF	IF	IF	ID	EXi	MEM	WB		
(sin decodificar)										IF					
lw \$5, 0x0000(\$10)											IF	ID	EXi	MEM	WB

Figura 10a. Cronograma del código de la figura 9a, suponiendo un procesador sin salto retardado

	1	2	3	4	5	6	7	8	9	10	11	12	13
addiu \$20, \$10, 0x0028	IF	ID	EXi	MEM	WB								
lw \$5, 0x0000(\$10)		IF	ID	EXi	MEM	WB							
addiu \$10, \$10, 0x0004			IF	ID	EXi	MEM	WB						
addi \$5, \$5, 0x0001				IF	ID	ID	EXi	MEM	WB				
bne \$10, \$20, 0xFFFF					IF	IF	ID	EXi	MEM	WB			
sw \$5, 0xFFFF(\$10)							IF	ID	ID	EXi	MEM	WB	
lw \$5, 0x0000(\$10)								IF	IF	ID	EXi	MEM	WB

Figura 10b. Cronograma del código de la figura 9b suponiendo un procesador con salto retardado

Con el código de la de la figura 1b para un procesador con saltos retardados se consigue eliminar los 9 bloqueos de control (el salto se toma 9 veces) mediante los saltos retardados y 10 ciclos de bloqueo de datos (uno por iteración) entre *addi* y *sw* ya que ambos se separan un ciclo al rellenar el hueco del salto con *sw* (véase el cronograma de la figura 2b)

3.3. CÁLCULO DE LA ACELERACIÓN

Recordando los conceptos estudiados en la primera sesión teórica, para calcular la mejora o aceleración de un determinado procesador o código respecto a la versión original se utiliza la expresión:

$$A = \frac{TiempoCPU_{Original}}{TiempoCPU_{Mejorado}} = \frac{N_{Original} \cdot CPI_{Original} \cdot \tau_{Original}}{N_{mejorado} \cdot CPI_{mejorado} \cdot \tau_{mejorado}}$$

Siendo N el número instrucciones ejecutadas y τ el periodo de reloj.

De esta expresión pueden eliminarse aquellos parámetros que permanecen constantes entre la versión original y mejorada. Por ejemplo, en el cálculo de la aceleración de un procesador con *forwarding* respecto a su versión sin *forwarding*, el número de instrucciones y la frecuencia de reloj no cambian, por lo que puede obtenerse como:

$$A = \frac{\cancel{N}_{Original} \cdot CPI_{Original} \cdot \cancel{\tau}_{Original}}{\cancel{N}_{bypass} \cdot CPI_{bypass} \cdot \cancel{\tau}_{bypass}} = \frac{CPI_{Original}}{CPI_{bypass}}$$

4. EJERCICIOS PROPUESTOS PARA REALIZAR:

Durante esta sesión, se recomienda encarecidamente que el alumno complete en la mayor medida posible la hoja de ejercicios de esta sesión. Esta hoja de ejercicios se aporta de forma separada a este documento.