Resumen de teórico
Arquitectura de Computadores

Agustin Curto, agucurto95@gmail.com

2016

Índice general

1. El procesador **2**
1.1. Descripción general de la segmentación 2
1.1.1. Diseño de repertorios de instrucciones para la segmentación 4
1.1.2. Riesgos del pipeline 4
1.1.3. Resumen de la visión general del *pipeline* 7
1.2. Camino de datos segmentados y control de la segmentación 7
1.3. Riesgos de datos: anticipación frente a bloqueos 7
1.4. Riesgos de control 7

2. Memoria **8**

Capítulo 1

El procesador

1.1. Descripción general de la segmentación

La **Segmentación (*pipelining*)** es una técnica de implementación que consiste en solapar la ejecución de múltiples instrucciones. Utilizaremos una analogía para describir los términos básicos y las características principales de la segmentación.

Cualquiera que hay tenido que lavar grandes cantidades de ropa ha usado de forma intuitiva la segmentación. El enfoque no segmentado de hacer una colada sería:

1. Introducir ropa sucia en la lavadora.
2. Cuando finaliza el lavado, intruducir la ropa mojada en la secadora.
3. Cuando finaliza el secado, poner la ropa seca sobre una mesa para ordenarla y doblarla.
4. Una vez que toda la ropa está doblada, la misma es guardada.

   Cuando ya se ha guardado, entonces se vuelve a comenzar con la siguiente colada.

El enfoque segmentado de lavado requiere mucho menos tiempo, tal y como lo muestra la figura 1.1.

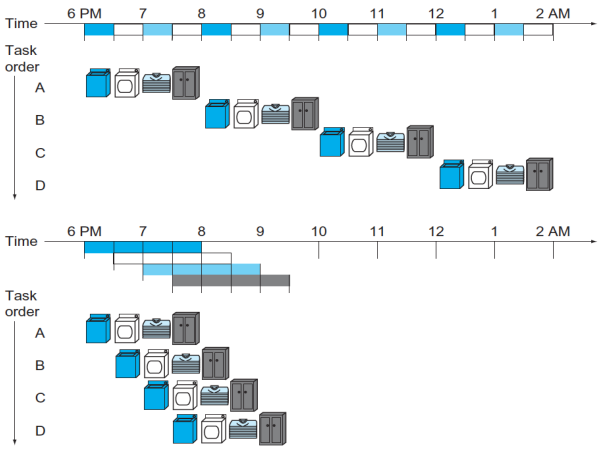

Figura 1.1: La analogía con la lavandería.

Todos los pasos, denominados **etapas de segmentación**, se llevan a cabo en forma concurrente. Se podrán segmentar las tareas siempre y cuando se disponga de recursos separados para cada etapa.

La segmentación es más rápida para varias coladas ya que todas las etapas se llevan a cabo en paralelo, y por lo tanto se completan más coladas por hora. La segmentación mejora la productividad de la lavandería sin mejorar el tiempo necesario para realizar una colada. La segmentación no reducirá el tiempo para completar una colada pero si la mejora de la productividad reducirá el tiempo total para completar todo el trabajo.

Se consideran cinco pasos para ejecutar las instrucciones MIPS:

1. Buscar una instrucción en memoria. ***fetch***
2. Leer los registros mientras se decondifica la instrucción. El formato de las instrucciones MIPS permite que la lectura y la decodificación ocurran en forma simultánea. ***decode***
3. Ejecutar una operación o calcular una dirección de memoria. ***execute***
4. Acceder a un operando en la memoria de datos. ***memory***
5. Escribir el resultado en un registro. ***write back***

Por lo tanto el cauce de segmentación, denominado *pipeline*, en MIPS tiene cinco etapas. El siguiente ejemplo muestra que la segmentación acelera la ejecución de instrucciones del mismo modo que lo hace para la colada.

Ejemplo

Limitaremos nuestra atención a sólo ocho instrucciones: cargar palabra ***lw***, almacenar palabra ***sw***, sumar ***add***, restar ***sub***, y-lógico ***and***, o-lógico ***or***, activar si menor ***set less than (slt)***, saltar si es igual textbf*brach equal (beq)*.

Compare el tiempo promedio entre la finalización de isntrucciones consecutivas en una implementación monociclo, en la que todas las instrucciones se ejecutan en un cíclo de reloj, con una implementación segmentada.

La figura 1.2 indica el tiempo requerido para cada una de las ocho instrucciones.

Instruction class	Instruction fetch	Register read	ALU operation	Data access	Register write	Total time
Load word (`lw`)	200 ps	100 ps	200 ps	200 ps	100 ps	800 ps
Store word (`sw`)	200 ps	100 ps	200 ps	200 ps		700 ps
R-format (`add`, `sub`, `AND`, `OR`, `slt`)	200 ps	100 ps	200 ps		100 ps	600 ps
Branch (`beq`)	200 ps	100 ps	200 ps			500 ps

Figura 1.2: Tiempo total para cada instrucción calculado a parir del tiempo de cada componente.

El diseño monocilco debe permitir acomodar a la instrucción más lenta, que en la figura 1.2 es *lw*, y por lo tanto la duración de todas las instrucciones será de 800 ps.

La figura 1.3 compara las ejecuciones segmentadas y no segmentadas de tres instrucciones *lw*.

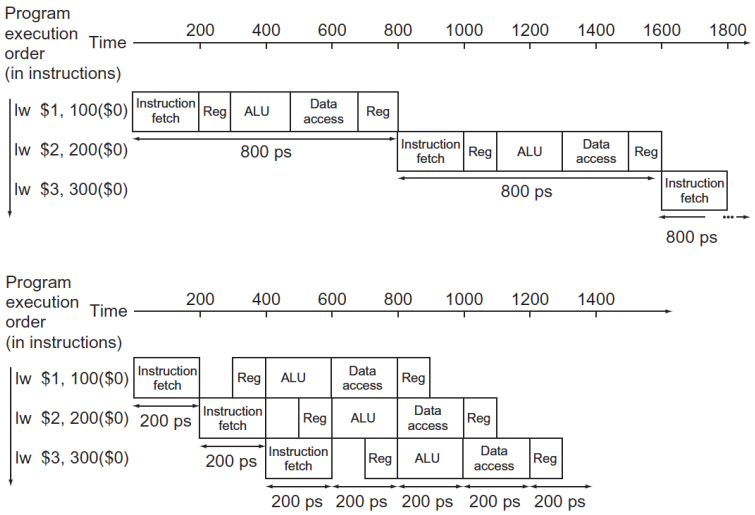

Figura 1.3: La analogía con la lavandería.

Es posible convertir en una fórmula la discusión anterior sobre la ganacia de la segmentación. Si las etapas están perfectamente equilibradas, entonces el tiempo entre las instrucciones en el procesador segmentado, suponiendo condiciones ideales, es igual a:

$$Tiempo\ entre\ Instrucciones_{segmentado} = \frac{Tiempo\ entre\ instrucciones_{no\ segmentado}}{Numero\ de\ etapas\ de\ segmentacion}$$

En condiciones ideales y con una gran número de isntrucciones, la ganancia debida a la segmentación es aproximadamente igual al número de etapas; un *pipeline* de cinco etapas es casi cinco veces más rápido.

1.1.1. Diseño de repertorios de instrucciones para la segmentación

1. Todas las instrucciones MIPS tiene la misma longitud
2. MIPS tiene solo unos pocos formatos de instrucciones, y además en todos ellos los campos de los registros fuentes están situados siempre en la misma posición de la instrucción.
3. En MIPS los operandos a memoria sólo aparecen en instrucciones de carga y almacenamiento.
4. Los operandos deben estar alineados en memoria.

1.1.2. Riesgos del pipeline

Hay situaciones de segmentación en las que la instrucción siguiente no se puede ejecutar en el siguiente ciclo. Estos sucesos se llaman *riesgos* (**hazards**) y existen tres tipos:

Riesgos estructurales

El hardware no admite una cierta combinación de instrucciones durante el mismo ciclo. En el ejemplo de la lavandería ocurriría si se usara una combinación lavadora-secadora en lugar de tener la lavadora y la secadora separadas.

El repertorio de instrucciones el MIPS fue diseñado para ser segmentado, por lo que facilita a los diseñadores evitar *riesgos estructurales*. Supongamos, sin embargo, que se tuviera una

sola memoria en lugar de dos. Si el *pipeline* de la Figura 1.3 tuviera una cuarta instrucción, se vería que en un mismo ciclo la primera instrucción está accediendo a datos de la memoria mientras que la cuarta está buscando una instrucción de la misma memoria. Sin disponer de dos memorias, nuestra segmentación podría tener riesgos estructurales.

Riesgos de datos

Ocurren cuando el *pipeline* se debe bloquear debido a que un paso de ejecución debe esperar a que otro paso sea completado.

Volviendo a la lavandería, supongamos que se esta doblando una colada y no se encuentra la pareja de un calcetín.

En un *pipeline* del computador, los riesgos de datos surgen de las dependencias entre una instrucción y otra anterior que se encuentra todavía en el *pipeline*. Por ejemplo, suponer que se tiene una instrucción *add* seguida inmediantamente por una instrucción *sub* que utiliza el resultado de la suma:

add	$s0, $t0, $t1
sub	$t2, $s0, $t3

Si no se interviene, un riesgo de datos puede bloquear severamente el procesador. la instrucción *add* no escribe su resultado hasta la quinta etapa, por lo que se tendrían que añadir tres burbujas en el procesador.

La principal solución se basa en la observación de que no es necesario esperar a que la instrucción se complete antes de intentar resolver el reigo de datos. Para la secuencia de código anterior, tan pronto como la ALU calcula la suma para la instrucción *add*, se puede suministrar el resultado como entrada para la resta. Se denomina **anticipación de resultados (forwarding)** o **retroalimentación (by passing)** al uso de hardware extra para anticipar antes el dato buscado usando los recursos internos del procesador.

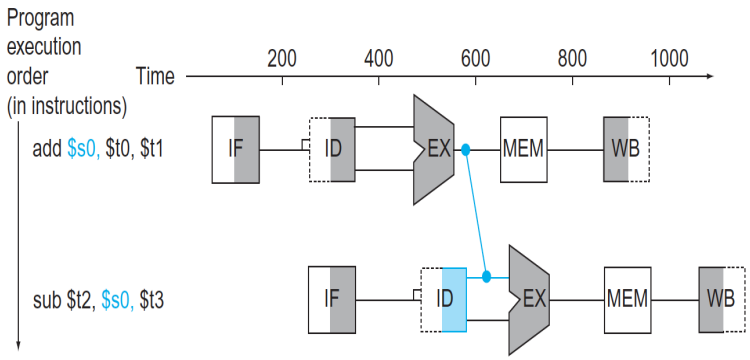

Figura 1.4: Representación gráfica de la anticipación de datos.

La anticipación funciona muy bien pero no es capaz de evitar todos los bloques. Por ejemplo, suponer que la primera instrucción fuera una carga en $s0, en lugar de una suma. Como se puede deducira partir de una mirada de la figura 1.4, el dato deseado sólo estará disponible después de la cuarta etapa de la primera instrucción, lo cual es demasiado tarde para la entrada de la etapa EX de la instrucción *sub*. Por lo tanto, incluso con la anticipación de resultados, habría que bloquear en una etapa en el caso del **riesgo del dato de carga (*load-use data hazard*)**, tal como se puede ver en la figura 1.5. Esta figura muestra un importante concepto de la segmentación, oficialmente denominado un **bloqueo del *pipeline* (*pipeline stall*)**, pero que frecuentemente recibe el apodo de **burbúja** (*bubble*).

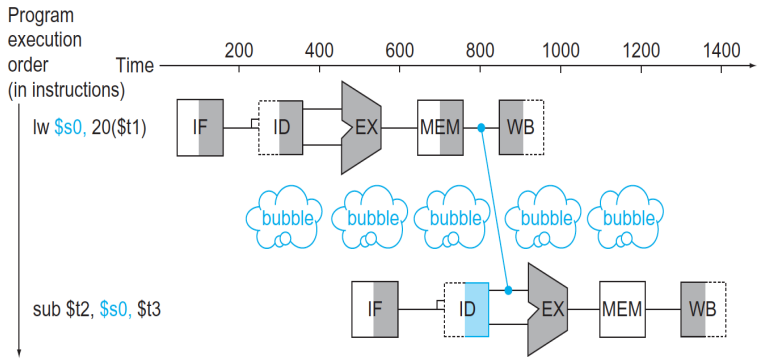

Figura 1.5: Representación gráfica de la anticipación de datos con burbújas.

La anticipación de resultado conlleva otra característica que hace comprender la arquitectura MIPS. Cada instrucción MIPS escribe como mucho un resultado y lo hace cerca del final del pipeline. La anticipación de resultado es más complicada si hay múltiples resultados que avanzar por cada instrucción, o si se necesita escribir el resultado antes del final de la ejecución de la instrucción.

Ejemplo: Supongamos que tenemos el siguiente código MIPS, suponiendo que todas las variables estáne ne memoria y son direccionables como desplazamientos a partes de $t0:

lw	$t1, 0($t0)
lw	$t2, 4($t0)
add	$t3, $t1, $t2
sw	$t3, 12($t0)
lw	$t4, 8($01)
add	$t5, $t1, $t4
sw	$t5, 16($t0)

Encuentre los riesgos que existen en el segmento de código y reordene las instrucciones para evitar todos los bloqueos del *pipeline*.

Respuesta: Las dos instrucciones add tienen un riesgo debido a sus respectivas dependencias con la instrucción lw que les precede inmediatamente. Observe que la anticipación de datos elimina muchos otros riesgos potenciales, incluidos la dependencia del primer add con el primer lw y los riesgos con las instrucciones store. Mover hacia a rriba la tercera instrucción lw elimina ambos riegos.

lw	$t1, 0($t0)
lw	$t2, 4($t0)
lw	$t4, 8($01)
add	$t3, $t1, $t2
sw	$t3, 12($t0)
add	$t5, $t1, $t4
sw	$t5, 16($t0)

En un procesador segmentado con anticipación de resultados, la secuencia reordenada se completará en dos cilcos menos que la versión original.

1.1.3. Resumen de la visión general del *pipeline*

1.2. Camino de datos segmentados y control de la segmentación

1.3. Riesgos de datos: anticipación frente a bloqueos

1.4. Riesgos de control

Capítulo 2

Memoria