

PRÁCTICA 6

Segmentación de cauce y atascos en procesadores RISC

Objetivos Comprender el funcionamiento de la segmentación de cauce del procesador MIPS de 64 bits.
Analizar las ventajas e inconvenientes de este tipo de arquitectura.

CPI

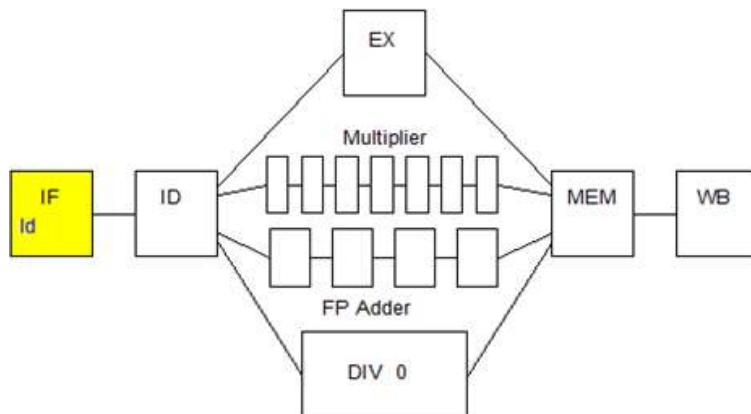
Los Ciclos Por Instrucción (CPI) son una medida de uso eficiente del procesador. Se definen como

$$\text{CPI} = \frac{\text{Ciclos}}{\text{Instrucciones}}$$

De esta forma, un procesador es más eficiente si para las mismas instrucciones, requiere menos ciclos. En ese caso, su CPI es más *bajo*.

Valor mínimo del CPI En arquitecturas tradicionales, el CPI no puede ser menor que 1, ya que en el mejor caso se termina una instrucción distinta en cada ciclo. En arquitecturas con múltiples unidades de ejecución si puede ser menor a 1.

La arquitectura del simulador WinMIPS tiene un pipeline de 5 etapas para las instrucciones más simples que usan la etapa EX, donde cada etapa requiere 1 ciclo para completarse si no hay atascos..



Por ese motivo, generalmente para un programa sin atascos el CPI puede calcularse como:

$$\text{CPI} = \frac{\text{Ciclos}}{\text{Instrucciones}} = \frac{\text{Instrucciones} + 4}{\text{Instrucciones}}$$

En este caso, el valor 4 proviene de que en los primeros 4 ciclos no se termina ninguna instrucción

Valor máximo del CPI En teoría, el CPI no tiene un límite superior, y podría ser tan grande como la ineficiencia del procesador. Idealmente, para un programa optimizado el CPI debería ser cercano a 1. Por ejemplo, un valor común para un programa optimizado sería 1.1 o 1.2. No obstante, esos valores de CPI generalmente solo se obtienen si un programa requiere ejecutar una gran cantidad de instrucciones que no se **atasquen** demasiado.

Atascos Las instrucciones pueden atascarse por diversos motivos, principalmente por falta de datos requeridos para ejecutarse. Cuando una instrucción se **atasca**, la misma no puede avanzar y por ende requiere más ciclos para ejecutarse, aumentando el CPI. En una arquitectura donde las etapas de las instrucciones duran 1 ciclo, entonces el número de ciclos requeridos para ejecutar la instrucción aumenta en 1 por cada atasco. En ese caso, para calcular el CPI podemos asumir:

$$\text{CPI} = \frac{\text{Ciclos}}{\text{Instrucciones}} = \frac{\text{Instrucciones} + 4}{\text{Instrucciones}} = \frac{\text{Instrucciones} + \text{Atascos} + 4}{\text{Instrucciones}}$$

Esta fórmula solo funciona cuando las instrucciones pasan todas por las mismas 5 unidades de ejecución básicas del WinMIPS64. A continuación, veremos los distintos tipos de atascos que pueden darse en la ejecución de los programas y cómo minimizarlos.

Parte 1: Estructura del Cauce

Para comprender cómo se puede atascar una instrucción, primero se debe comprender qué parte de la instrucción se ejecuta en cada etapa del cauce.

1. Funciones de las etapas del cauce

Indicar a qué etapa corresponde cada función que realiza el procesador al ejecutar una instrucción típica:

Función	Etapa
A. Decidir si se toma un salto o no B. Buscar la instrucción en memoria y llevarla a la CPU C. Calcular la dirección de un acceso a memoria D. Guardar el valor de un registro en memoria E. Traer de memoria un valor a un registro intermedio F. Almacenar el valor final de un registro G. Calcular la dirección de un salto H. Verificar si están disponibles los operandos necesarios para continuar con la ejecución de la instrucción	1. IF: Obtención de Instrucción 2. ID: Decodificación de Instrucción 3. EX: Ejecución principal de instrucción 4. MEM: Acceso a Memoria 5. WB: Escritura de resultado en registro destino

2. Etapas de una instrucción

Indicar, para las siguientes instrucciones, qué realizan en cada una de las etapas IF/ID/EX/MEM/WB.

Instrucción	IF	ID	EX	MEM	WB
daddi \$t1, \$t3, 4	Buscar la instrucción en memoria y llevarla a CPU	Decodificar la instrucción. Verificar si está disponible \$t3. Recuperar \$t3 del banco de registros	Realizar \$t3 + 4	-	Salvar en \$t1 la suma realizada en EX
dadd \$t2, \$t4, \$t3	Buscar la instrucción en memoria y llevarla a CPU	Decodificar la instrucción. Verificar si están disponibles \$t3 y \$t4. Recuperar \$t4 y \$t3 del banco de registros	Realizar \$t3 + \$t4	-	Salvar en \$t2 la suma realizada en EX
sd \$t3, tabla(\$t2)	Buscar la instrucción en memoria y llevarla a CPU	Decodificar la instrucción. Verificar si están disponibles \$t3 y \$t2. Recuperar \$t3 y \$t2 del banco de registros	Realiza la suma de "tabla" + \$t2.	Acceder a la posición de memoria calculada en EX. Guardar en dicha posición el valor de \$t3	-
ld \$t3, tabla(\$t2)	Buscar la instrucción en memoria y llevarla a CPU	Decodificar la instrucción. Verificar si está disponible \$t2. Recuperar \$t2 del banco de registros	Realizar la suma de "tabla" + \$t2	Acceder a la posición de memoria calculada en EX. Recuperar el valor	Guardar el valor recuperado en \$t3
bneq \$t1, \$t2, loop	Buscar la instrucción en memoria y llevarla a CPU	Decodificar la instrucción.	-	-	-

	memoria y llevarla a CPU	Verificar si están disponibles \$t1 y \$t2. Recuperar \$t1 y \$t2 del banco de registros. Si \$t1 y \$t2 son distintos, calcula el nuevo valor de PC			
j loop	Buscar la instrucción en memoria y llevarla a CPU	Decodificar la instrucción. Calcular el nuevo valor de PC	-.	-	---

3. Cálculo de CPI

Simular la ejecución de los siguientes programas **manualmente** (sin usar el simulador), dibujando el cauce como lo hace el simulador, y calculando la cantidad de instrucciones, ciclos, y CPIs.

Responder:

1. La fórmula de CPI presentada anteriormente ¿se cumple para los dos programas?
2. En el segundo programa, se utilizan las instrucciones ddiv y dmul ¿cuántos se requieren para su ejecución?
3. ¿Por qué la instrucción dmul tiene varias etapas llamadas M1, M2, etc, y ddiv no? Modificar el segundo programa, duplicando las instrucciones ddiv y dmul, de manera que cada una aparezca dos veces, y volver a simular. ¿Qué sucede en el caso de la división?
4. La instrucción HALT solo detiene la ejecución del programa ¿se cuenta en el cálculo del CPI?
5. El simulador considera a la instrucción NOP para calcular el CPI. No obstante, dicha instrucción no realiza ninguna tarea. Agregar 20 instrucciones NOP más en el primer programa, y calcular nuevamente el CPI. ¿Qué valor toma? En el caso de que sea menor, ¿eso quiere decir que el programa es más eficiente?

daddi \$t2,\$t3,5 dsub \$t4,\$t3,\$t5 xor \$t6,\$t3,\$t5 nop halt	ddiv \$t6,\$t3,\$t5 dmul \$t4,\$t3,\$t5 daddi \$t2,\$t3,5 Halt
---	---

Ciclo	1	2	3	4	5	6	7	8	9
daddi \$t2,\$t3,5	IF	ID	EX	MEM	WB				
dsub \$t4,\$t3,\$t5		IF	ID	EX	MEM	WB			
xor \$t6,\$t3,\$t5			IF	ID	EX	MEM	WB		
nop				IF	ID	EX	MEM	WB	
halt					IF	ID	EX	MEM	WB

$$CPI = \frac{9 \text{ ciclos}}{5 \text{ instrucciones}} = 1.8 \frac{\text{ciclos}}{\text{instrucción}}$$

Ciclo	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16
ddiv \$t6,\$t3,\$t5	IF	ID	DIV													
dmul \$t4,\$t3,\$t5	IF	ID	M0	M1	M2 ME	M3	M4	M5	M6	M7	M8	M ME	WB			
daddi \$t2,\$t3,5		IF	ID	EX	M	WB										
halt			IF	ID	EX	M	WB									

$$CPI = \frac{28 \text{ ciclos}}{4 \text{ instrucciones}} = 7 \frac{\text{ciclos}}{\text{instrucción}}$$

1. Se cumple solamente para el primer programa ya que no tiene atascos.
2. para dmul se utilizan 8 ciclos y para ddiv 24
3. Lo que sucede con la segunda instrucción ddiv es que se “atasca” a la espera de que la etapa de división termine para poder ingresar. Cuando un atasco ocurre porque una etapa que una instrucción necesita acceder no está disponible porque otra instrucción está pasando por esa etapa, se llama atasco estructural. En el caso de dmul no ocurre lo mismo porque la división ocurre en etapas diferentes, por lo tanto mientras la primera instrucción va para una etapa de la multiplicación, la segunda instrucción dmul puede utilizar cualquier etapa previa de la multiplicación.
4. Si, se cuenta

$$5. CPI = \frac{29 \text{ ciclos}}{25 \text{ instrucciones}} = 1.16 \frac{\text{ciclos}}{\text{instrucción}}$$

No es más eficiente ya que sin los NOP se puede realizar el mismo programa pero con 20 ciclos menos, es un resultado engañoso