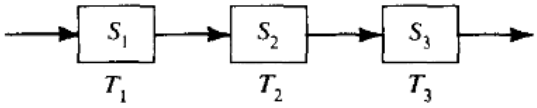
PRÁCTICO DE ARQUITECTURA DE COMPUTADORA 2012

Pipeline (Segmentación encauzada)

Problema 1:
Demostrar que un cauce lineal de *k*-etapas puede ser como máximo *k* veces más rápido que un procesador no encauzado.

Problema 2:
Se dispone del siguiente encauzamiento. Responda a las siguientes preguntas:



$T_1 = T_3 = T, T_2 = 3T$

a. Cuál de los tres segmentos o etapas es la que causa la congestión? (el cuello de botella)
b. Asuma que el segmento problemático se pueda dividir en sendas (dos) etapas consecutivas (ninguna de ellas con duración menor que *T*). Cuantas serian las particiones posibles? Cual es el período de clock de cada partición?
c. Asuma que el segmento problemático se pueda dividir en varias etapas consecutivas de duración *T*. Cual es el período de clock del pipeline?
d. Asuma que el segmento problemático NO se pueda dividir por la naturaleza propia del proceso. Sería todavía posible aumentar la velocidad del cauce? Cómo se lograría?

Problema 3:
Una instrucción requiere de cuatro etapas para su ejecución: La etapa 1 (**I**nstruction **F**etch) requiere 30 ns, la etapa 2 (**I**nstruction **D**ecode) = 9 ns, la etapa 3 (instruction **EX**ecute) = 20 ns y la etapa 4 (**W**rite **B**ack or store results) = 10 ns. Toda instrucción debe proceder a través de todas las etapas, en la secuencia dada.

a. Cuál es el tiempo asíncrono mínimo que se requiere para completar la ejecución de una sola instrucción?
b. Se requiere hacer esto como una operación encauzada (pipeline). ¿Cuántas etapas se deberían tener y a qué velocidad de clock debería trabajar el cauce?

Problema 4:
Para el cauce del problema 3:

a. Con que frecuencia se puede iniciar la ejecución de una nueva instrucción, y cuál es su latencia?
b. Cuál es la ganancia de velocidad del cauce?

Problema 5:
Se cuenta con un procesador RISC con instrucciones aritméticas registro-registro que tienen el formato R1 = R2 op R3. El encauzamiento para estas instrucciones se ejecuta con un clock de 100 MHz con las siguientes etapas: IF = 2 ciclos, ID = 1 ciclo, **O**perands **F**etch = 1 ciclo, EX = 2 ciclos, y store results = 1 ciclo.

a. A qué velocidad (en **M**illion **I**nstructions **P**er **S**econd), se pueden ejecutar las instrucciones registro-registro que no tengan dependencias de datos con otras instrucciones?

b. A qué velocidad se pueden ejecutar las instrucciones cuando cada instrucción depende de los resultados de la instrucción anterior?

c. Si se implementa forwarding interno. A qué velocidad se puede ahora ejecutar las instrucciones cuando cada instrucción depende de los resultados de la instrucción anterior?

Problema 6:

Se evaluará como el encauzamiento afecta al ciclo (o período) de clock del procesador. Asuma que las etapas individuales del pipeline tienen las siguientes latencias:

	IF	ID	EX	MEM	WB	Unidad
1	300	400	350	500	100	psec
2	200	150	120	190	140	psec

a. Cuál es el ciclo de clock para ambos, procesador encauzado y no encauzado?

b. Cuál es la latencia de la instrucción de transferencia lw para ambos, procesador encauzado y no encauzado?

c. Si se pudiera partir una etapa del encauzamiento en dos nuevas etapas, cada una con la mitad de la latencia de la etapa original, cual etapa elegiría y cual será el nuevo ciclo de clock?

Problema 7:

Su amigo le dice que su diseño del procesador es 10 veces mejor que el suyo, ya que cuenta con un cauce de 50 etapas en comparación a su diseño de 5 etapas. Tiene razón? Por qué si o por qué no? (pregunta intencionalmente sutil)

Problema 8: Hazards

En este ejercicio, se examinara los riesgos de dependencia estructural, de control, y como el diseño de las instrucciones puede afectar la ejecución en el cauce. Los problemas del ejercicio se refieren a los siguientes fragmentos de código MIPS:

A

```
        lw      $1, 40($6)
        beq     $2, $0, Label     ; Asuma $2 == $0
        sw      $6, 50($2)
Label:  add     $2, $3, $4
        sw      $3, 50($4)
```

B

```
Label:  lw      $5, -16($5)
        sw      $4, -16($4)
        lw      $3, -20($4)
        beq     $2, $0, Label     ; Asuma $2 != $0
        add     $5, $1, $4
```

a. Asuma de que los saltos condicionales son perfectamente predichos (elimina el riesgo de control) y que no se usan delay spots. Si se tiene una única unidad de memoria (para amos instrucciones y datos), existe un riesgo estructural cada vez que se necesita leer una instrucción en el mismo ciclo en que otra instrucción accede a un dato. Para garantizar el progreso del programa este riesgo se debe resolver siempre a favor de la instrucción que accede al dato. Cual es el tiempo total de

ejecución de la secuencia de instrucciones en un cauce de 5 etapas para este caso? Se vio que los riesgo de datos se pueden eliminar agregando instrucciones *nop* en el código. Se puede hacer lo mismo en este riesgo? Porque?

b. Asuma de que los saltos condicionales son perfectamente predichos (elimina el riesgo de control) y que no se usan delay spots.

c. Asumiendo estancamiento ante saltos (stall-on-branch) y que no se utilizan delay spots. Cual es la ganancia de velocidad que se logra en el código si las decisiones en los saltos condicionales se determinan en la fase ID, relativa a la ejecución donde los saltos se determinan en la fase EX?

Problema 9:

Un procesador cuenta con un encauzamiento de cinco etapas que consiste en: IF, ID, EX, MEM, y WB. Utiliza ranuras de retardo (delay slots) para manejar dependencias de control. La decisión y la dirección destino de las instrucciones de salto incondicionales (jump), y condicionales (branch) se resuelven en la fase EX del cauce.

a. Cuál es el número de ranuras de retardo necesarios para garantizar un funcionamiento correcto?

b. Qué instrucción(es) en los siguientes bloques assembly se colocan en la(s) ranura(s) de retardo, suponiendo que el número de ranuras de retardo es el de su respuesta en la parte (a)? Reescribir el código con la instrucción(es) apropiada(s) en la(s) ranura(s) de retardo.

	A			***B***			***C***	
	add	r5 <- r4, r3		add	r5 <- r4, r3		add	r2 <- r4, r3
	or	r3 <- r1, r2		or	r3 <- r1, r2		or	r5 <- r1, r2
	sub	r7 <- r5, r6		sub	r7 <- r5, r6		sub	r7 <- r5, r6
	j	addr		beq	r5 <- r7, addr		beq	r5 <- r7, addr
	[*DELAY SLOTS*]			[*DELAY SLOTS*]			[*DELAY SLOTS*]	
	lw	r10 <- (r7)		lw	r10 <- (r7)		lw	r10 <- (r7)
	add	r6 <- r1, r2		add	r6 <- r1, r2		add	r6 <- r1, r2
addr:	…		addr:	…		addr:	…	

c. Se puede modificar el encauzamiento para reducir el número de ranuras de retardo (sin introducir branch prediction)? Indique claramente su solución y explicar por qué.