

Arquitectura de Computadoras 2019

Práctico N° 3: Procesador con pipeline

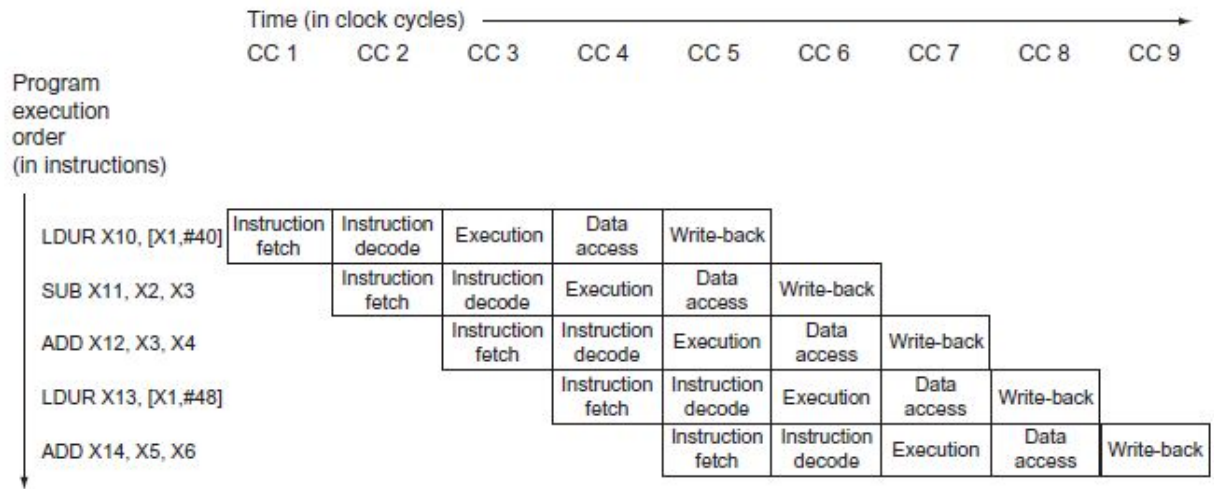


FIGURE 4.43 Traditional multiple-clock-cycle pipeline diagram of five instructions in Figure 4.42.

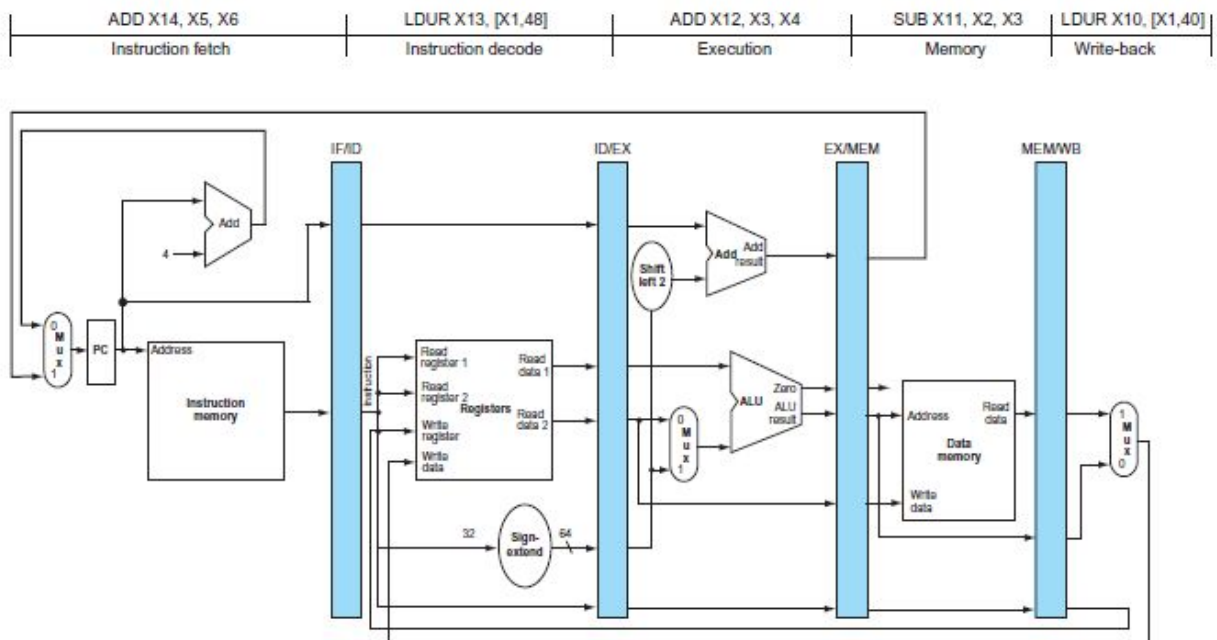
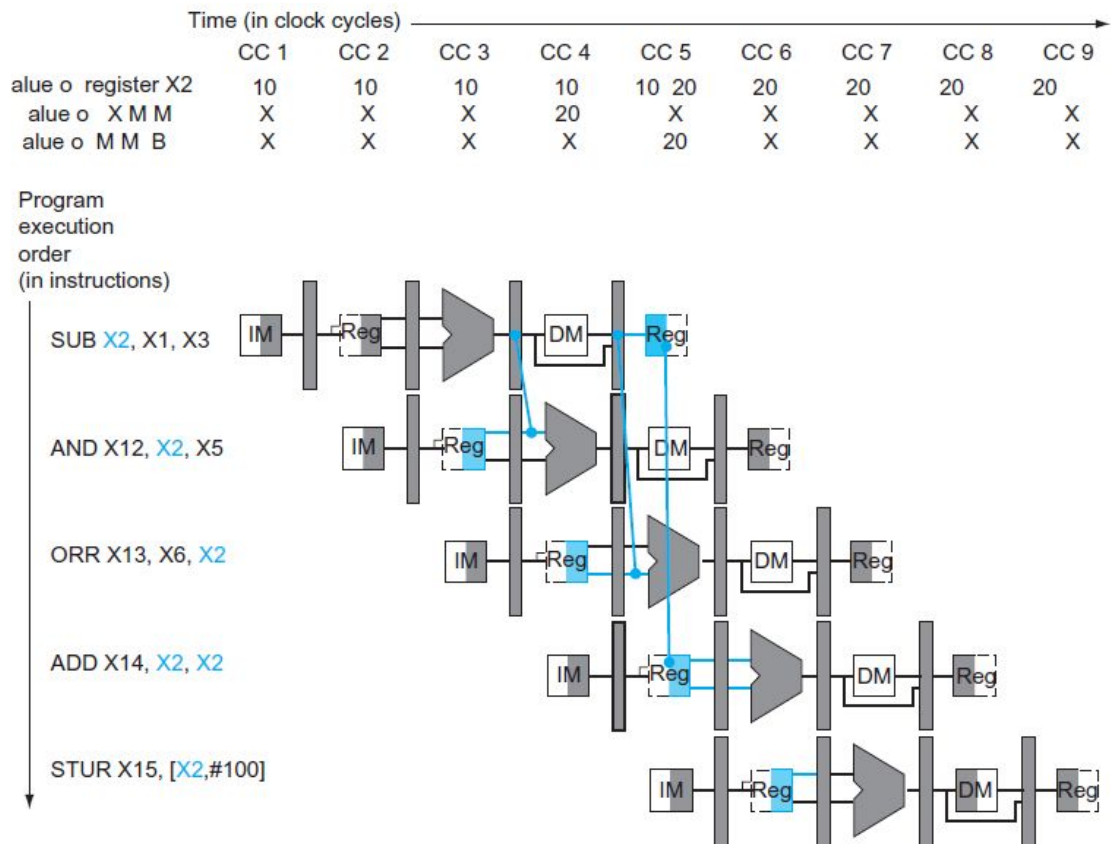
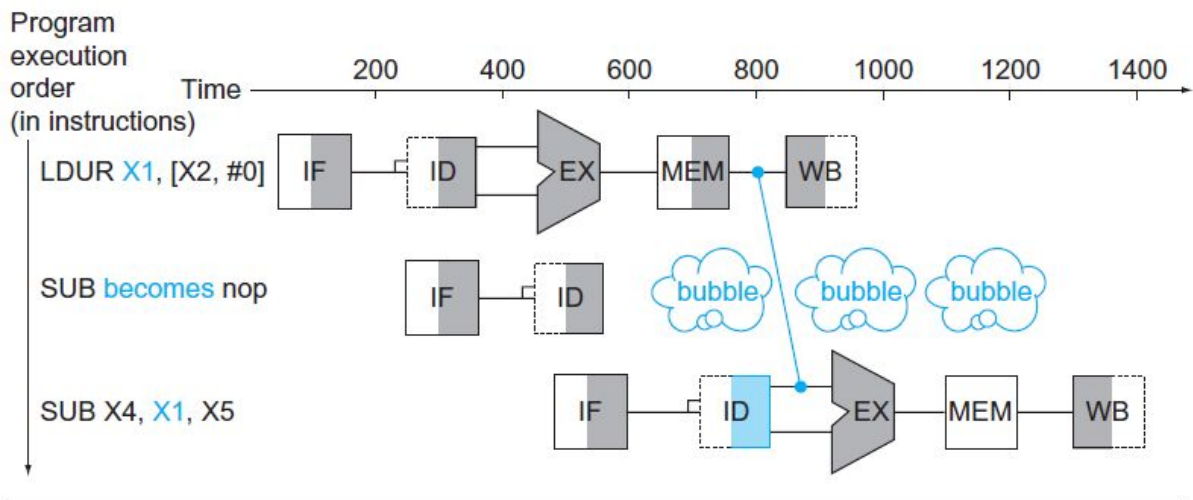


FIGURE 4.44 The single-clock-cycle diagram corresponding to clock cycle 5 of the pipeline in Figures 4.42 and 4.43. As you can see, a single-clock-cycle figure is a vertical slice through a multiple-clock-cycle diagram.

Data Hazards: Forwarding

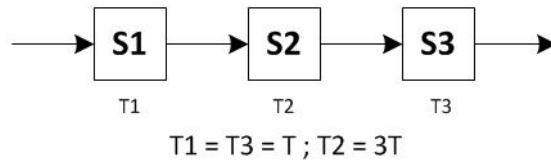


Data Hazards: Forwarding stall



Ejercicio 1:

En un microprocesador con tres etapas de pipeline: $S_1 \rightarrow S_2 \rightarrow S_3$, con tiempos de ejecución $T_1 = T_3 = T$ y $T_2 = 3T$.



- ¿Cuál de los tres segmentos o etapas causa la congestión? (el cuello de botella).
- Asumiendo que el segmento problemático se puede dividir en dos etapas consecutivas, ninguna de ellas con duración menor que T , ¿cuál sería la mejor partición posible? ¿Cuál sería el período de *clock* resultante para este nuevo pipeline de 4 etapas?
- Asumiendo que el segmento problemático se puede dividir en varias etapas consecutivas de duración T , ¿cuál es el período de *clock* del pipeline? ¿Cuál es el tiempo de ejecución entre instrucciones?

Ejercicio 2:

Asumiendo que las etapas de un procesador ARM tienen las siguientes latencias:

IF	ID	EX	MEM	WB
30ns	9ns	20ns	25ns	10ns

- ¿Cuánto tiempo se requiere en un microprocesador sin pipeline para completar la ejecución de una instrucción tipo R?
- ¿Cuánto tiempo se requiere en un microprocesador sin pipeline para completar la ejecución de la instrucción de mayor latencia, es decir, la latencia del procesador completo?
- Si se requiere ejecutar esa instrucción en un microprocesador con pipeline, ¿a qué velocidad debería trabajar el *clock*?
- ¿Cuál es el tiempo de ejecución de una instrucción en un microprocesador con pipeline? ¿Cada cuanto se ejecuta una nueva instrucción en este procesador?
- Si un microprocesador con pipeline ejecuta 3 instrucciones consecutivas ¿cuál es la ganancia de velocidad de un procesador con pipeline respecto de uno sin pipeline? ¿Y si se ejecutan 1000 instrucciones consecutivas?

$$\text{Ganancia de velocidad} = \frac{\text{Tiempo de ejecución sin pipeline}}{\text{Tiempo de ejecución con pipeline}}$$

Ejercicio 3:

Asumiendo que las etapas individuales del pipeline de dos procesadores ARM distintos tienen las siguientes latencias:

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

- a) ¿Cuál es el ciclo de *clock* para la versión con y sin pipeline para cada caso?
- b) ¿Cuál es la latencia de la instrucción **LDUR** para ambos, considerando las versiones de procesador con y sin pipeline?
- c) Si se pudiera partir una etapa del pipeline en dos nuevas etapas, cada una con la mitad de la latencia de la etapa original, ¿Que etapa elegiría y cuál será el nuevo ciclo de *clock*?

Ejercicio 4:

Dados los siguientes fragmentos de código de instrucciones LEGv8:

A	B
1> SUB X6, X1, X3 2> SUB X7, X6, X5	1> ADDI X10, X1, #8 2> LDUR X3, [X10, #0] 3> SUB X3, X2, X10

- a) Mostrar el orden de ejecución en un procesador con pipeline y determinar las dependencias de datos que provoquen *data hazard*.
- b) Mostrar el orden de ejecución en un procesador que utilice SOLO la técnica de *stall* para evitar los *data hazard*.
- c) Mostrar el orden de ejecución en un procesador que utilice la técnica de *forwarding stall*.
- d) ¿Cuántos ciclos toma la ejecución del código en cada caso?

Ejercicio 5:

Dado el siguiente fragmento de código de instrucciones LEGv8:

```

1> LDUR X1, [X10, #0]
2> LDUR X2, [X10, #8]
3> ADD X3, X1, X2
4> STUR X3, [X10, #24]
5> LDUR X4, [X10, #16]
6> ADD X5, X1, X4
7> STUR X5, [X10, #32]

```

- a) Analizar en el código las dependencias de datos y determinar cuales generan *data hazards*. En cada caso indicar: los números de las instrucciones involucradas, en qué etapa se encuentra c/u y el operando en conflicto.
- b) Mostrar el orden de ejecución de las instrucciones utilizando un procesador con *forwarding stall*.
- c) Reescribir la sección de código alterando el orden de las instrucciones para evitar *stalls* innecesarios. Mostrar el nuevo orden de ejecución.
- d) ¿Cuántos ciclos toma la ejecución del código en cada caso?

Ejercicio 6:

Dados los siguientes fragmentos de código de instrucciones LEGv8:

A	B
1> LDUR X1, [X2, 80]	1> ADD X1, X2, X3
2> ADD X2, X3, X3	2> STUR X2, [X1, 0]
3> ADD X1, X1, X2	3> LDUR X1, [X2, 8]
4> STUR X1, [X2, 40]	4> ADD X2, X2, X1

- Mostrar el orden de ejecución de las instrucciones utilizando un procesador con *stall*.
- Mostrar el orden de ejecución de las instrucciones utilizando un procesador con *forwarding stall*.
- Agregar instrucciones **nop** a las secuencias 'A' y 'B' para asegurar la correcta ejecución en un procesador sin soporte de *forwarding stall*. Mostrar gráficamente el orden de ejecución del programa en el pipeline.
- Considerando un clock de 1GHz, calcular el tiempo de ejecución de ambos programas para los tres casos anteriores.

Ejercicio 7:

Dado el siguiente fragmento de código de instrucciones LEGv8:

```
1>      SUB X3, X1, X2
2>      CBZ X3, skip
3>      ADD X4, X3, XZR
4> skip: ADD X5, X3, XZR
```

- Analizar en el código las dependencias de datos y de control, determinar cuales generan *hazards*. En cada caso indicar: los números de las instrucciones involucradas, en qué etapa se encuentra c/u, el operando en conflicto y el tipo de hazard.
- Mostrar el orden de ejecución de las instrucciones utilizando un procesador con *forwarding stall* suponiendo que $X1 \neq X2$. ¿Cuál es la penalidad por el hazard de control?
- Mostrar el orden de ejecución de las instrucciones utilizando un procesador con *forwarding stall* suponiendo que $X1 = X2$. ¿Cuál es la penalidad por el hazard de control?

Ejercicio 8:

Para el siguiente fragmento de código de instrucciones LEGv8 que se ejecuta en un procesador con *forwarding stall*:

```
1>      ADD X1, X2, X3
2>      CBNZ X0, L
3>      SUB X1, X5, X6
4> L:    LDUR X0, [X3, #0]
5>      OR X7, X1, X8
```

- Analizar en el código las dependencias de datos y de control, determinar cuales generan *hazards*. En cada caso indicar: los números de las instrucciones involucradas, en qué etapa se encuentra c/u, el operando en conflicto y el tipo de hazard.

- b) Mostrar el orden de ejecución de las instrucciones suponiendo que $X0 = 0$.
- c) Mostrar el orden de ejecución de las instrucciones suponiendo que $X0 \neq 0$.

Ejercicio 9:

En la siguiente secuencia de código los registros X6 y X7 han sido inicializados con los valores 0 y 8N respectivamente.

```

loop:
    1> LDUR X2, [X6, #40]
    2> LDUR X3, [X6, #48]
    3> ADD X2, X2, X3
    4> STUR X2, [X6, #40]
    5> ADDI X6, X6, #8
    6> CMP X6, X7
    7> B.NE loop
...

```

- a) Determinar qué técnica de predicción de salto (no dinámica) generará la menor cantidad de demoras por flush instructions si $N > 1$.
- b) Analizar la ejecución del código considerando que el resultado y la dirección del salto se determinan en la etapa de decodificación (ID) y se aplican en la etapa de ejecución (EX) y que no hay hazards de datos (Figura 4.61 de "Computer organization and design - ARM edition" Patterson & Hennessy).

Ejercicio 10:

Asumiendo que la distribución de instrucciones dinámicas se divide en las siguientes categorías:

R-Type	CBZ/CBNZ	B	LDUR	STUR
40%	25%	5%	25%	5%

y las siguientes precisiones en los métodos de predicción de salto:

Always-Taken	Always-Not-Taken	2-Bit
45%	55%	85%

Considerando que el resultado y la dirección del salto se determinan en la etapa de decodificación (ID) y se aplican en la etapa de ejecución (EX) y que no hay hazards de datos.

- a) ¿Cuántos CPI (Ciclos por instrucción) extras se producen debido a los fallos de predicción del método *Always-Taken*?
- b) ¿Cuántos CPI (Ciclos por instrucción) extras se producen debido a los fallos de predicción del método *Always-Not-Taken*?
- c) ¿Cuántos CPI (Ciclos por instrucción) extras se producen debido a los fallos de predicción del método *2-Bit*?

Ejercicio 11:

Asumiendo que los saltos condicionales son perfectamente predichos (elimina el riesgo de *hazard* de control), si se tiene una única unidad de memoria para 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.

Dados el siguiente fragmento de código de instrucciones LEGv8:

```
1> STUR X16, [X6, #12]
2> LDUR X16, [X6, #8]
3> SUB X7, X5, X4
4> CBZ X7, Label
5> ADD X5, X1, X4
6> SUB X5, X15, X4
```

Label:

a) Asumiendo *forwarding stall* (el cual solo elimina los hazards de datos), mostrar gráficamente el orden de ejecución del programa en el pipeline. Identificar dónde se generan los *Hazards estructurales*.

b) Anteriormente, se vio que los riesgos de datos se pueden eliminar agregando instrucciones **nop** en el código. ¿Se puede hacer lo mismo con este hazard? ¿Por qué?