Argomenti avanzati

Pipeline

Sandro Savino (sandro.savino@dei.unipd.it)
Department of Information Engineering, University of Padova

Argomenti:

- Pipeline

Materiale:

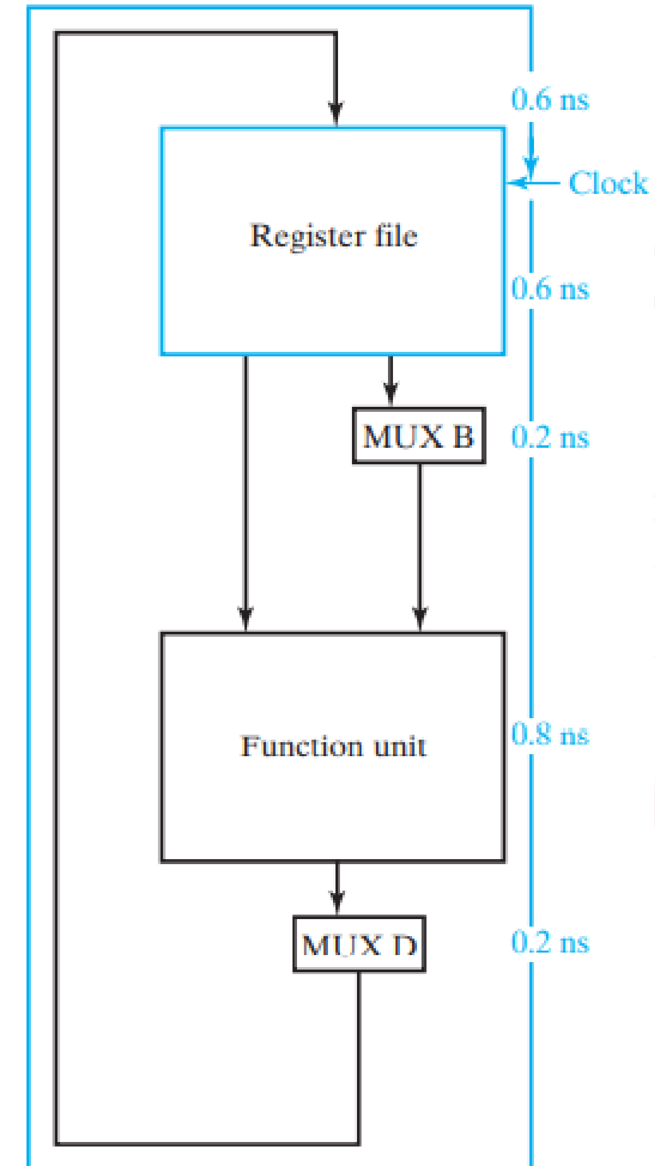
- Capitolo 10

DIPARTIMENTO
DI INGEGNERIA
DELL'INFORMAZIONE

Come velocizzare il computer a ciclo singolo

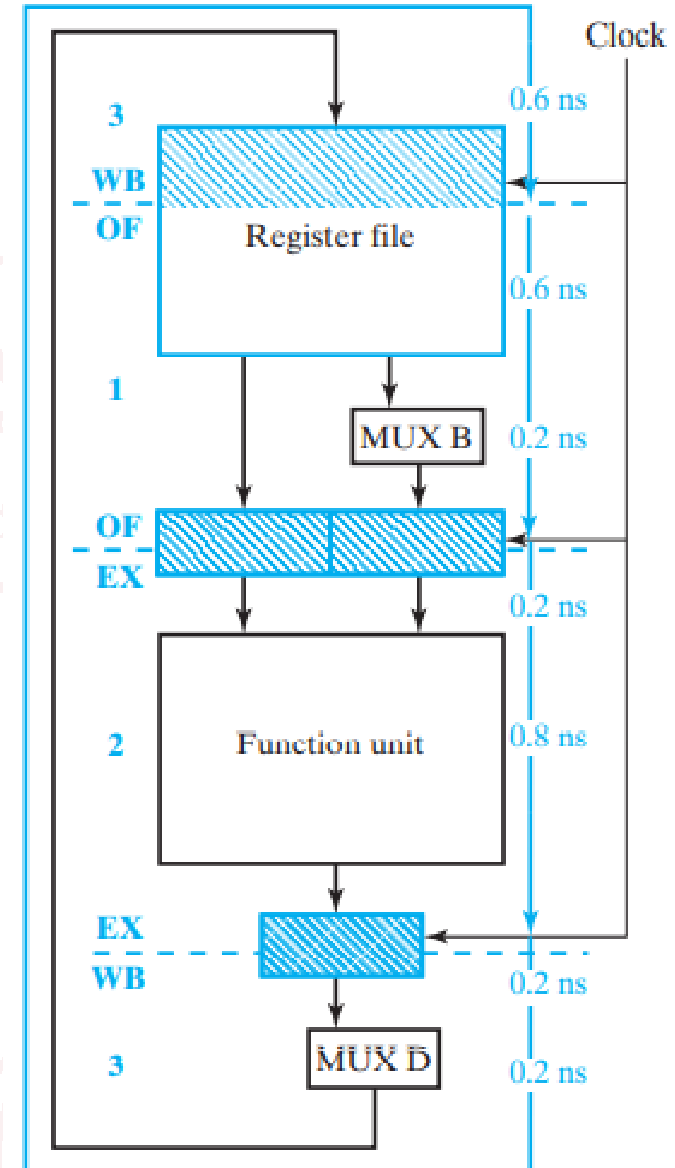
- Abbiamo visto che il tempo di esecuzione di una istruzione dipende dalla somma dei ritardi dei componenti attraversati nel datapath
- es.: somma = 2.4ns, freq max = 416 MHz

- Esiste però la possibilità di aumentare il clock e aumentare il numero di istruzioni eseguite



Un datapath più veloce

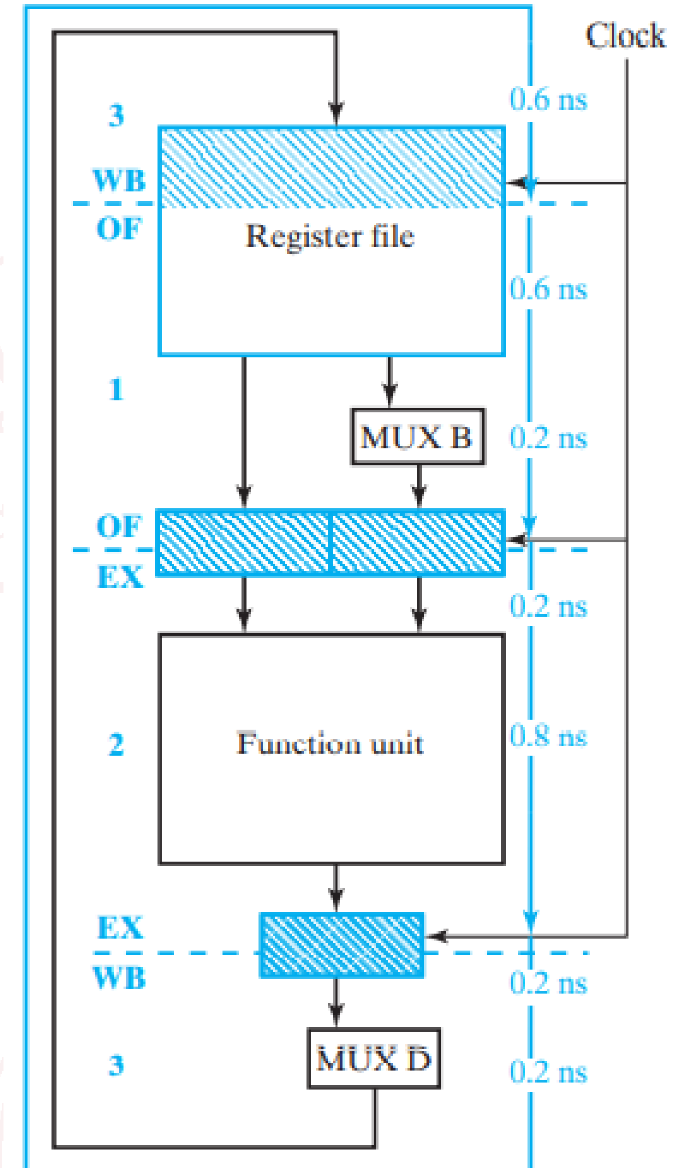
- Dividiamo il datapath:
  - Una si occupa del fetch dell'operando (OF)
  - Una si occupa dell'esecuzione dell'operazione (EX)
  - Una si occupa dell'output del risultato (WB)
- Ogni sezione termina con un registro dove il dato viene memorizzato temporaneamente in attesa di essere prelevato come input della sezione successiva
- Ognuna di queste sezioni è attivata e tenuta in sincronia con le altre sezioni tramite un clock
- Come prima il clock è dato dal ritardo più lungo
- Ora è di 1.0 ns (Freq: 1GHz)
- Una operazione, per essere completata, ha bisogno di più cicli di clock: 3 (3 ns, freq 333 MHz)



Un datapath più veloce

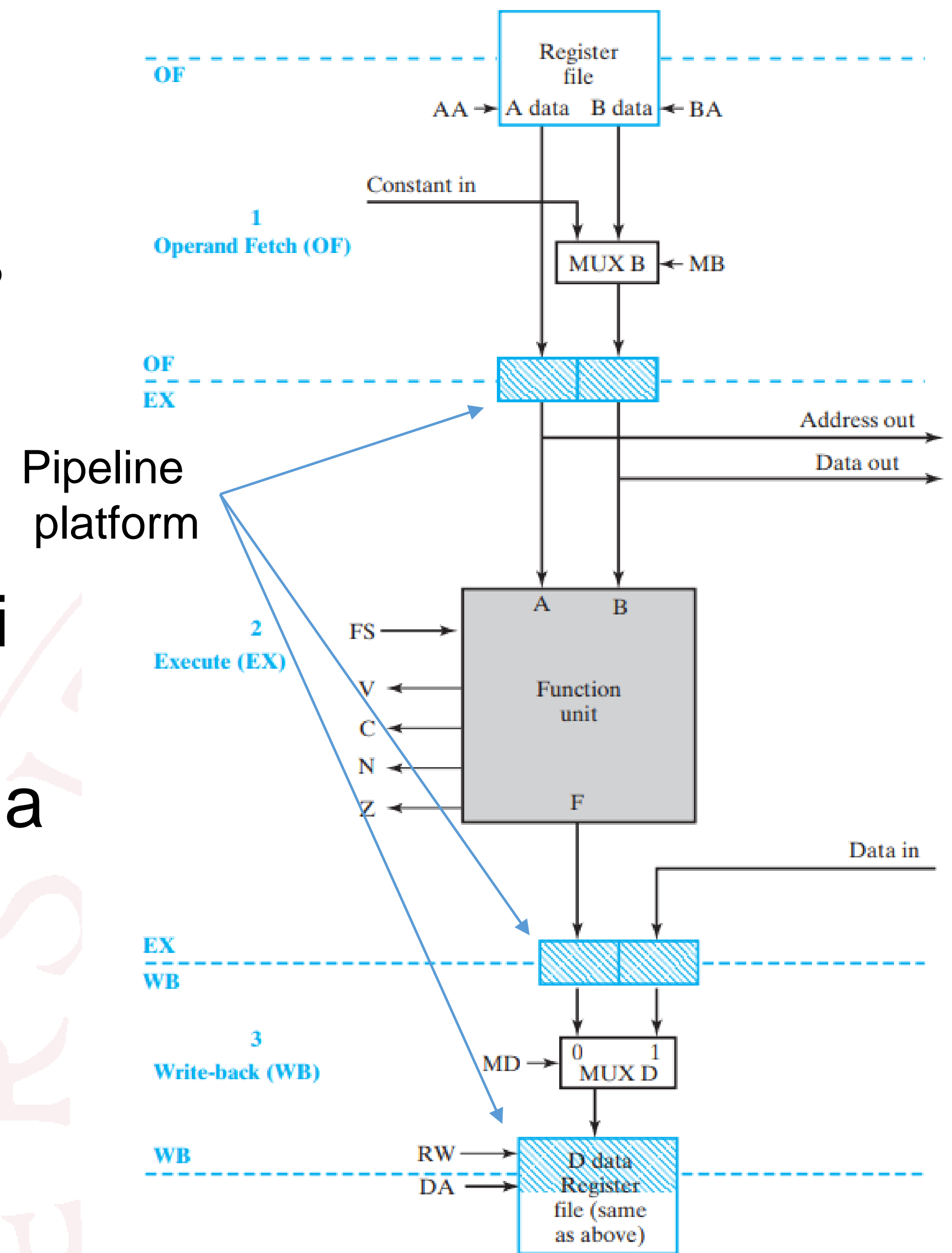
- Sembra che non abbiamo guadagnato niente, anzi abbiamo peggiorato il throughput della nostra rete logica
- Se guardiamo bene però, mentre una operazione viene eseguita, è attiva una sola sezione alla volta
- Le altre sezioni sono «libere»
- È possibile quindi pensare di usare le stazioni libere per iniziare a processare un’altra operazione intanto che quella precedente viene completata

- Abbiamo creato una ***PIPELINE***



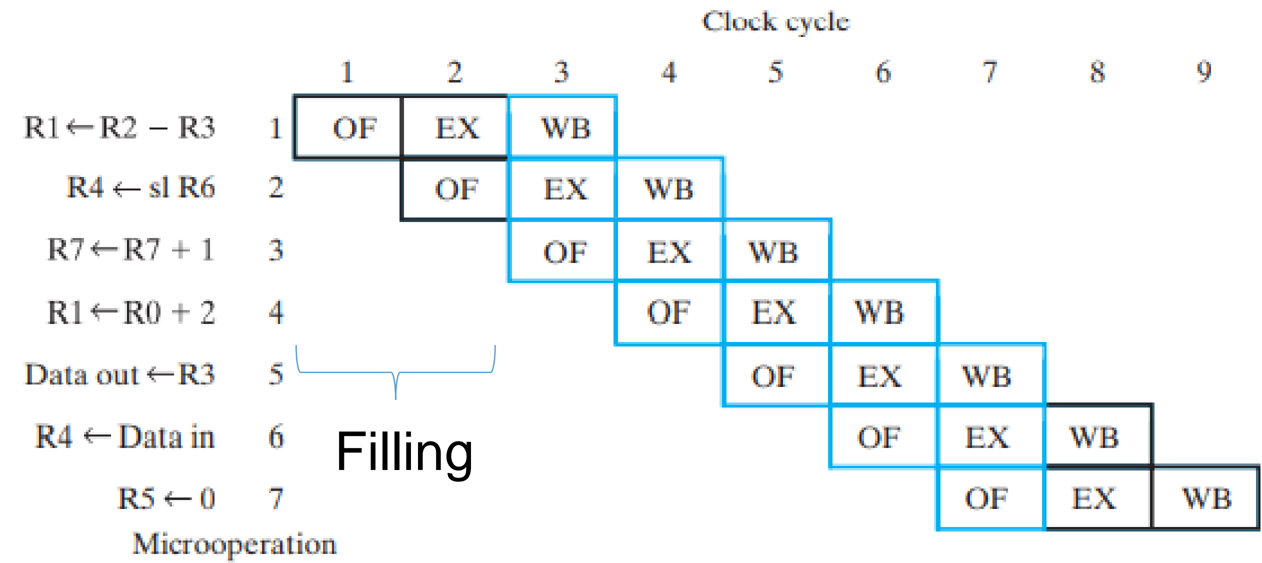
Pipelined datapath

- La nostra pipeline impiega 3 cicli da 1.0 ns ad eseguire una microoperazione
- Funziona con un clock di 1GHz
- Mentre esegue una microoperazione elabora parzialmente altre microoperazioni
- Quando la prima microoperazione è terminata, ne ha una eseguita per 2/3 e una per 1/3
- Una volta terminata la prima, riesce da eseguire una operazione ad ogni ciclo di clock, quindi 1 ogni 1.0 ns !!



□ **FIGURE 10-2**
Block Diagram of Pipelined Datapath

Pipeline



- La prima operazione richiede 3 ns
- La seconda è pronta in 4 ns
- La terza è pronta al 5 ns
- …e via dicendo
- L'incremento teorico massimo di performance di una pipeline a n stadi è di 1/n
- È massimo quando la pipeline è piena

Emptying

Fattore di **SPEEDUP**

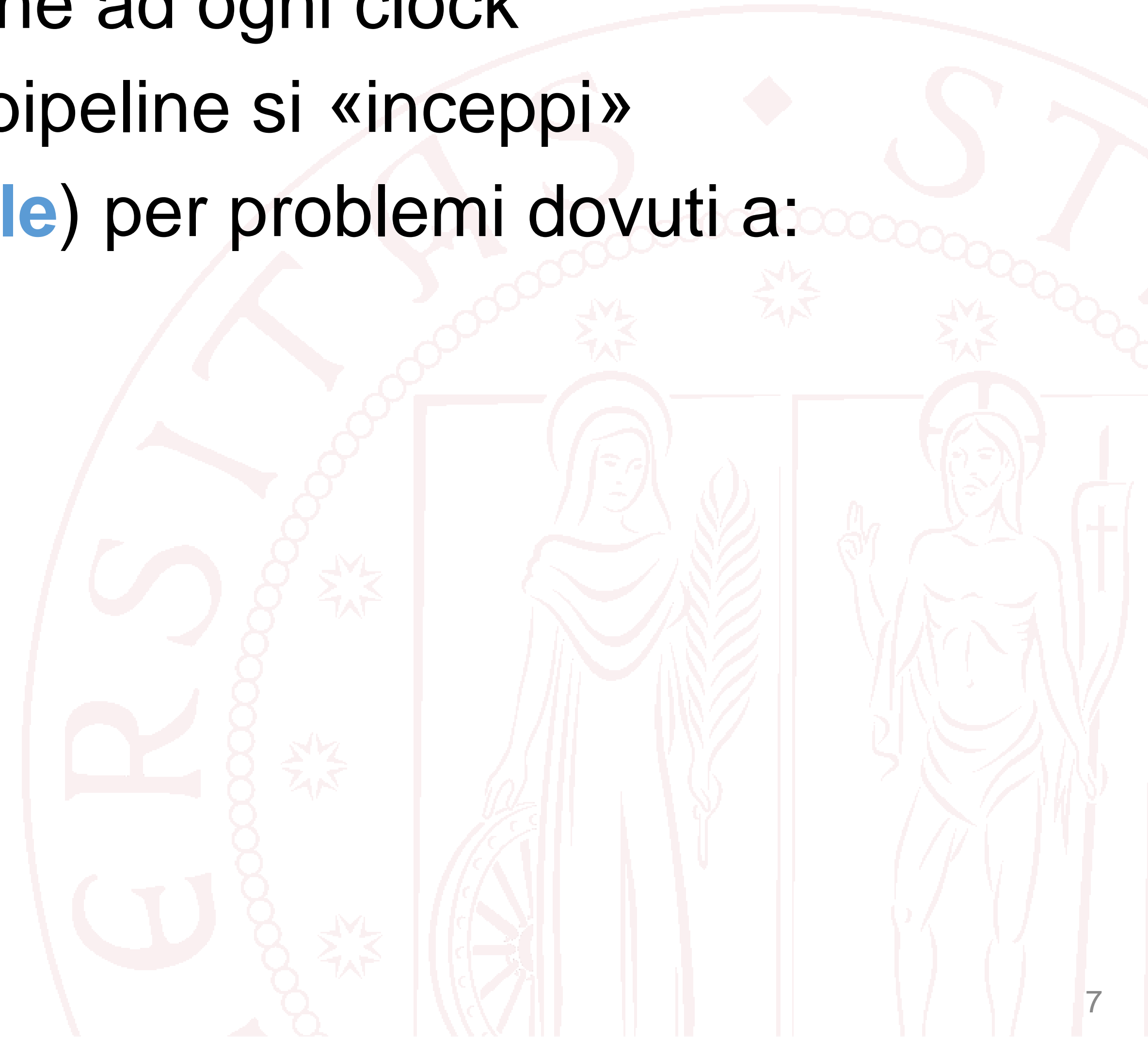
Tempo totale per eseguire N operazioni:

Senza pipeline: $t_1 = N \cdot k \cdot t$ (k=numero di stadi, t=durata di uno stadio)

Con pipeline a k stadi: $t_k = (k + (N - 1)) \cdot t$

Problemi nella pipeline

- La pipeline funziona al massimo della performance quando è piena e possiamo eseguire una microoperazione ad ogni clock
- Purtroppo però può succedere che la pipeline si «inceppi»
- La pipeline può incepparsi (**stall/bubble**) per problemi dovuti a:
  - accessi alla memoria (cache miss)
  - conflitti sulle risorse (resource hazard)
  - dipendenze tra dati (data hazard),
  - salto condizionati (branch hazard);



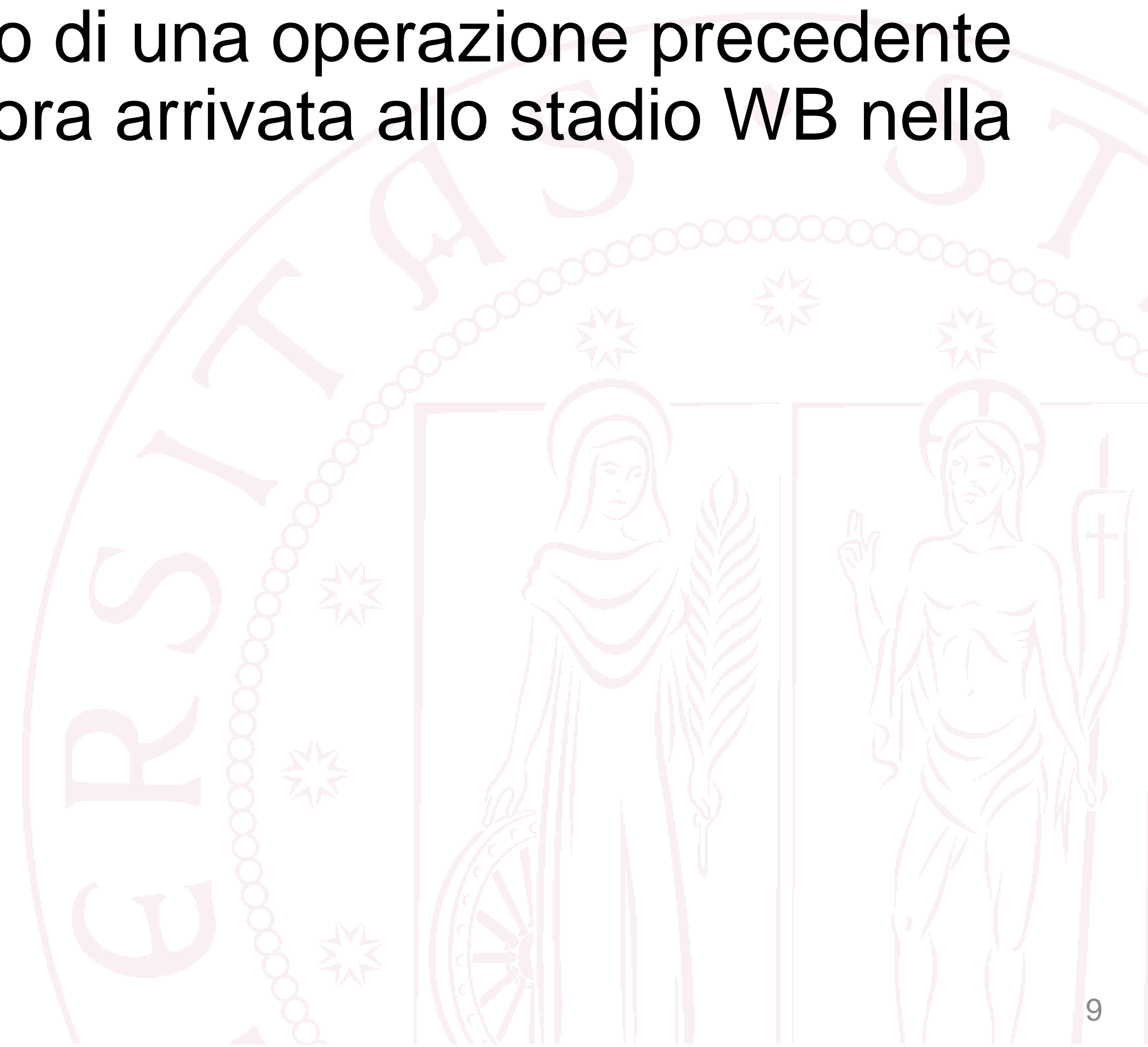
Stall per memoria o risorse

- Le fasi che accedono alla memoria (EX, WB) hanno una durata pari alle altre fasi (1 ciclo), solo se gli accessi alla memoria si risolvono nella cache (cache hit).
- In caso di cache miss, l'operazione può richiedere 2 o 3 cicli.
- Di conseguenza il pipeline si inceppa e l'esecuzione delle istruzioni viene ritardata.

- La pipeline si può anche inceppare nel caso in cui due operazioni chiedano l'accesso alla stessa risorsa nello stesso istante
- Ad esempio allo stesso dato di memoria (WB e OF di una operazione successiva)

Data Hazard

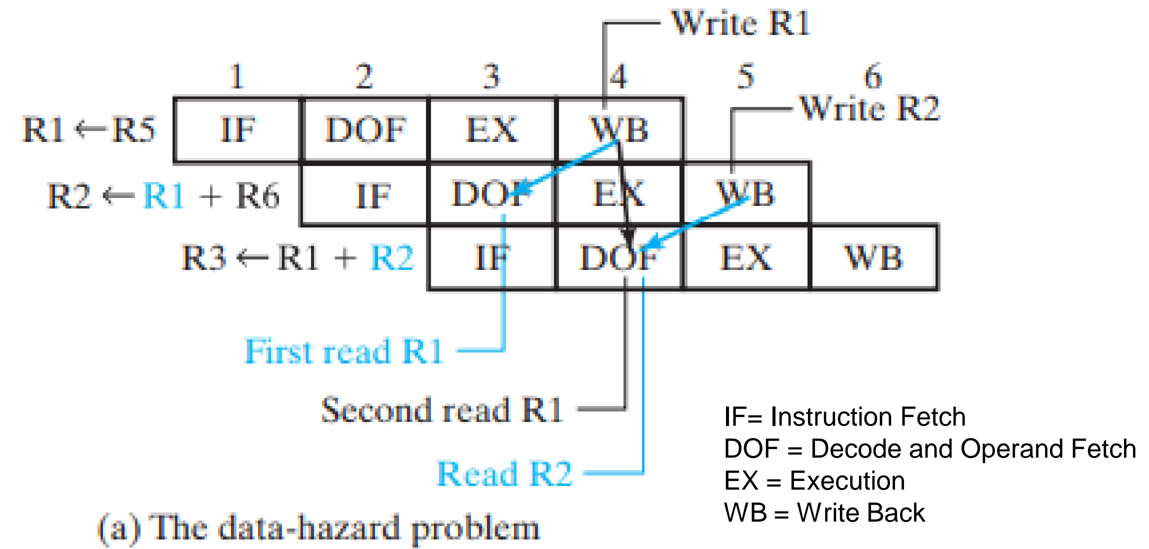
- E’ un problema di timing
- Una istruzione cerca di usare il risultato di una operazione precedente come operando, ma questa non è ancora arrivata allo stadio WB nella pipeline
- Soluzione:
  - soluzione **NOP**
  - Soluzione data stall / **bubble** (hardware)
  - Soluzione **data-forward** (hardware)



Data Hazard: soluzione NOP

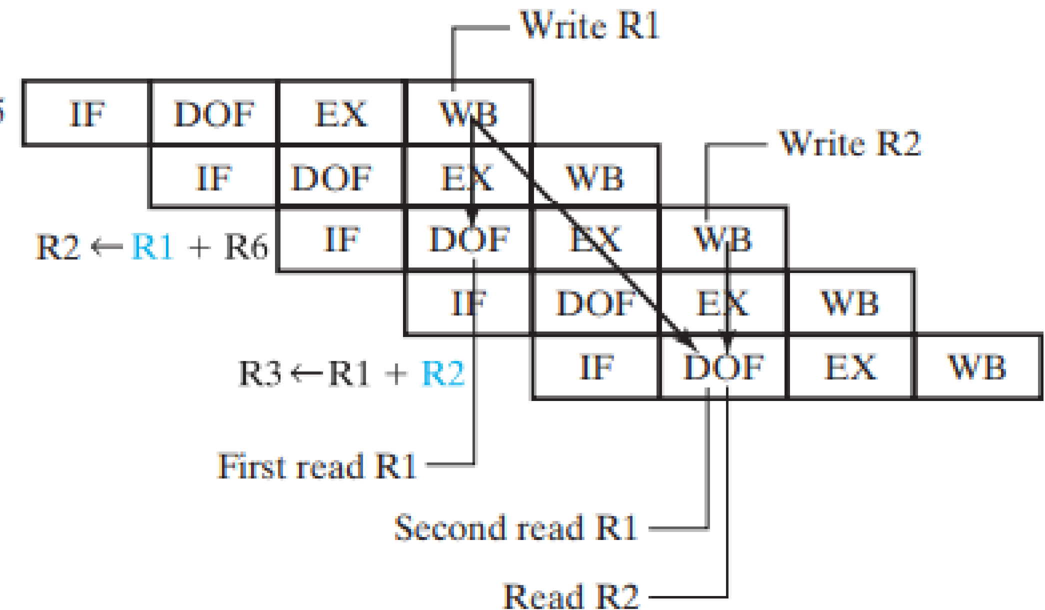
- il compilatore aggiunge NOP al programma
- Mentre viene «eseguita» l’istruzione NOP, il dato diventa pronto
- Il compilatore deve avere profonda conoscenza della struttura della pipeline
- Il codice compilato viene più lungo e anche più lunga è l’esecuzione

MOVA R1, R5
ADD R2, R1, R6
ADD R3, R1, R2



(a) The data-hazard problem

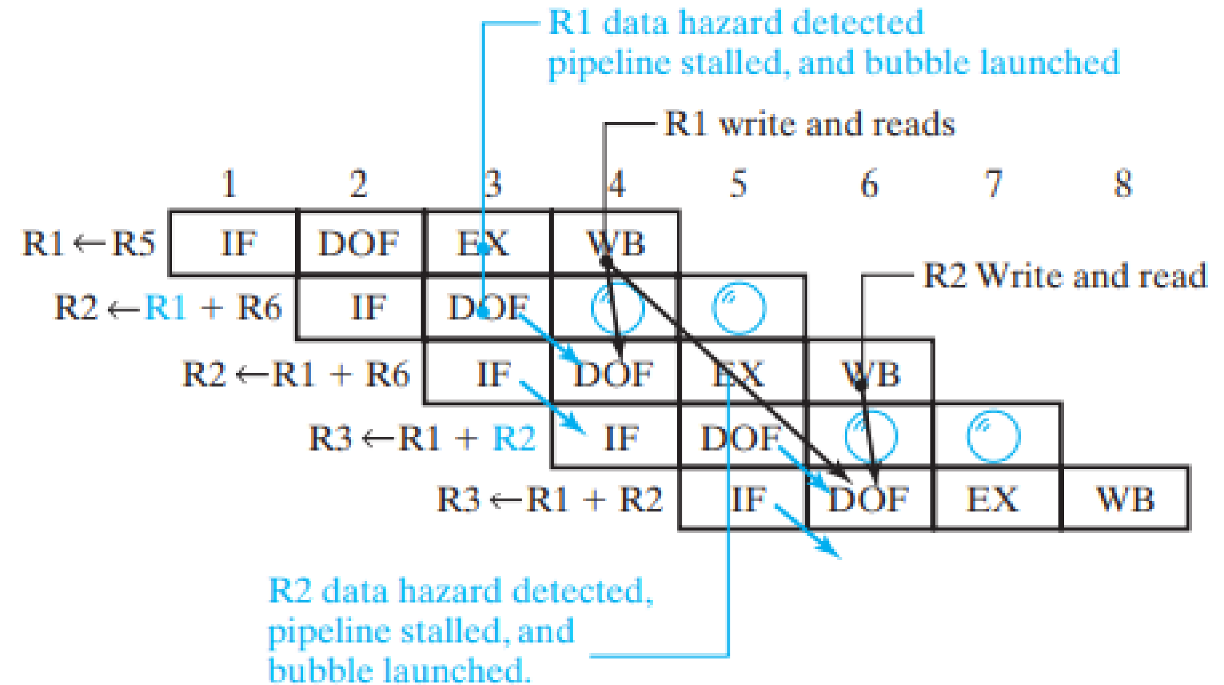
MOVA R1, R5
NOP
ADD R2, R1, R6
NOP
ADD R3, R1, R2



(b) A program-based solution

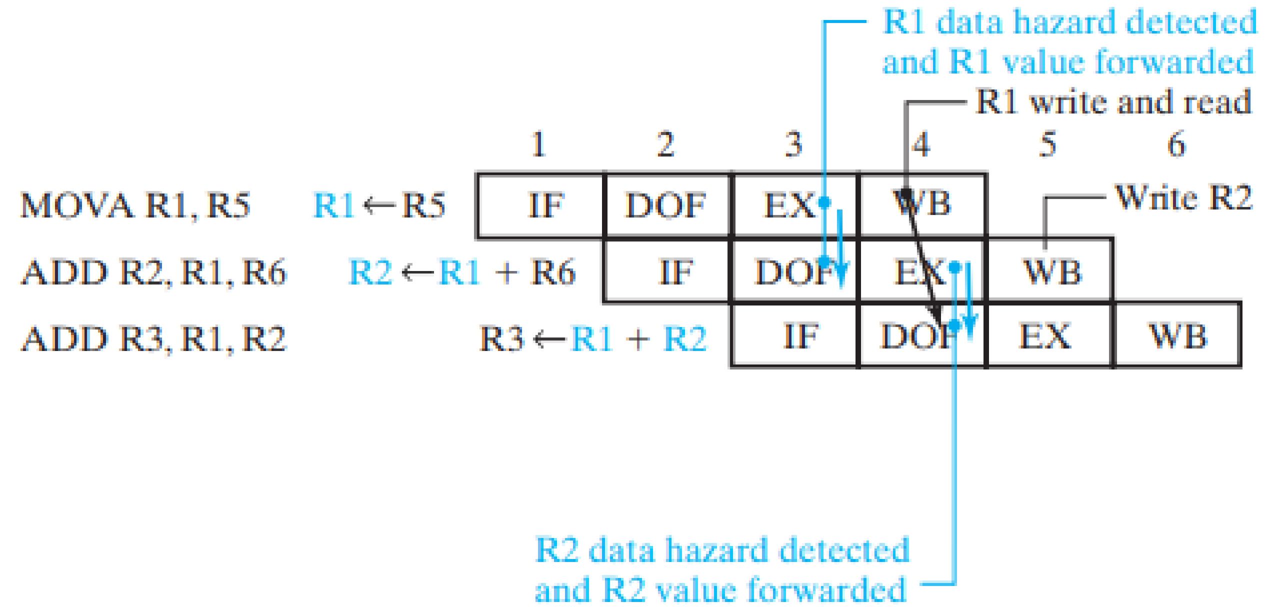
Data Hazard: soluzione BUBBLE

- viene rilevata la richiesta di un operando che non è ancora stato scritto
- viene generata una bubble
- l'operazione non viene fatta avanzare
- Simile al NOP, ma gestita via hardware



Data Hazard: soluzione DATA-FORWARD

- se il dato che si cerca è già disponibile (ad es. è su un bus) viene letto dalla sua posizione, senza aspettare l'esecuzione della sua fase di write-back (WB)
- Soluzione possibile solo se l'hardware lo permette



Control Hazard

- Problemi legati al flusso di esecuzione
- Per vedere se un branch va preso o no, questo deve arrivare alla fase EX
- Intanto però ho caricato le istruzioni successive in pipeline
- Se il branch deve essere preso, le istruzioni successive non vanno eseguite: bisogna bloccarne l'esecuzione (e questo inceppa la pipeline)
- Soluzione:
  - Soluzione nop: lato compilatore sono inseriti dei NOP dopo il branch, oppure il programmatore mette operazioni che sa che vanno comunque eseguite (**delayed-branch**)
  - Soluzione bubble: (**branch-hazard-stall**)
  - Soluzione **branch-prediction**

```
if( day_of_week == 'giovedi' ) {
    open('www.epic.com');
    search( 'free games' );
    get_all_games();
}
```

Control Hazard: branch prediction

- Si presume che il branch NON sarà preso, e si esegue il codice successivo
- Se non è vero, sono inserite bubble annullando le righe da non eseguire
- Questo evita di avere di NOP, quindi la performance peggiora solo se il branch si deve fare
- Si può anche ragionare al contrario (branch sempre preso), questo però richiede di calcolare l’indirizzo di salto e memorizzarlo