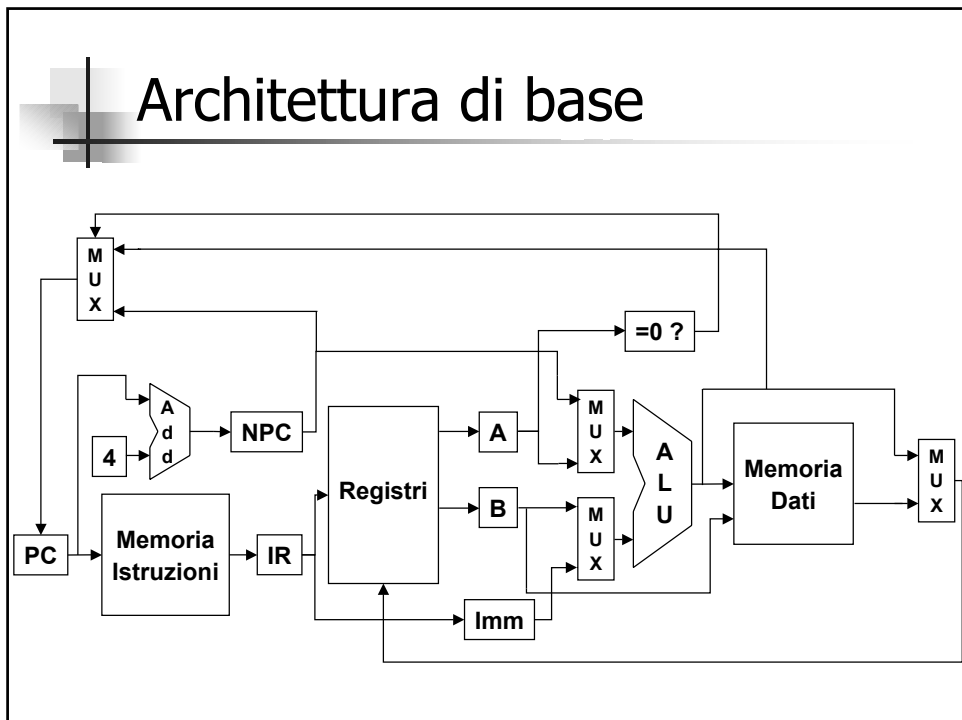


Architettura di una CPU "moderna"



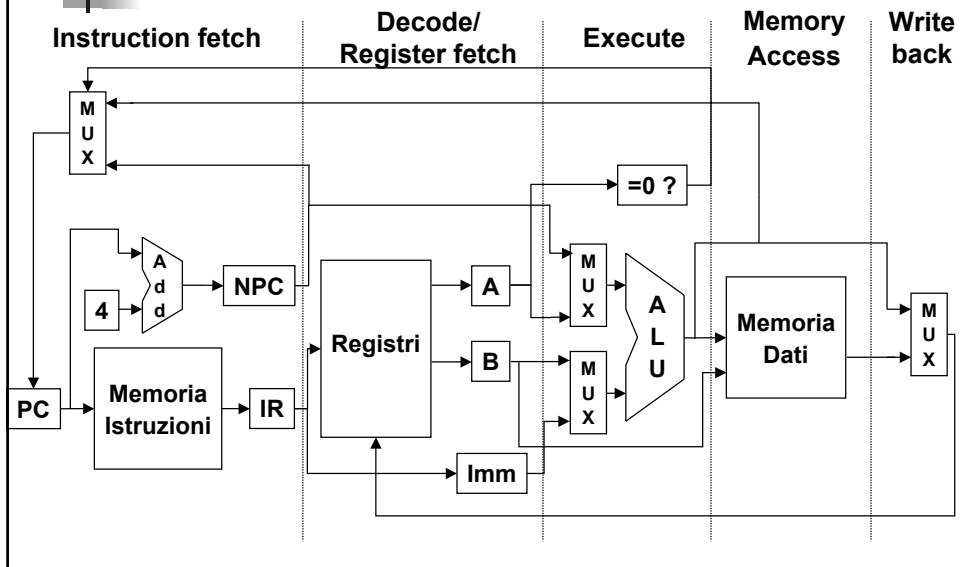
Architettura di base

- Dati e Istruzioni sono memorizzati in due memorie separate: cache dati e cache istruzioni.
- Le istruzioni sono codificate su 32 bit.
- Next Program Counter (NPC): contiene l'indirizzo della prossima istruzione.
- La CPU contiene un insieme di registri che contengono i dati su cui la ALU lavora.

Architettura di base

- Gli operandi su cui la ALU lavora vengono prelevati dai registri e messi in due registri speciali A e B.
- Se l'operazione specifica un indirizzamento immediato, il dato viene messo nel registro Imm.
- La ALU esegue operazioni aritmetiche/logiche e di salto.
- I dati su cui opera sono memorizzati in:
 - A, Imm: trasferimento dati o aritmetico/logica
 - A, B: aritmetico/logica
 - NPC, Imm: trasferimento del controllo.

Architettura di base



Instruction fetch (IF)

- Preleva dalla IM l'istruzione che deve essere eseguita espressa su 32 bit;
- Incrementa il PC.

Decode/Register fetch (ID)

- Decodifica l'istruzione.
- Accede ai registri general purpose per prelevare gli operandi e li memorizza nei registri dedicati A e B;
- Se l'indirizzamento è immediato, memorizza l'operando nel registro Imm.

Execute (EX)

- Esegue l'operazione richiesta
- Può essere:
 - calcola il risultato di una operazione aritmetica;
 - calcola il risultato di una operazione logica;
 - calcola l'indirizzo in DM per una operazione di trasferimento dati;
 - calcola l'indirizzo in IM a cui si deve fare accesso in seguito ad un salto.

Memory access (MEM)

- Nel caso di operazione di trasferimento dati accede alla DM;
- Nel caso di salto condizionato o meno aggiorna il PC.

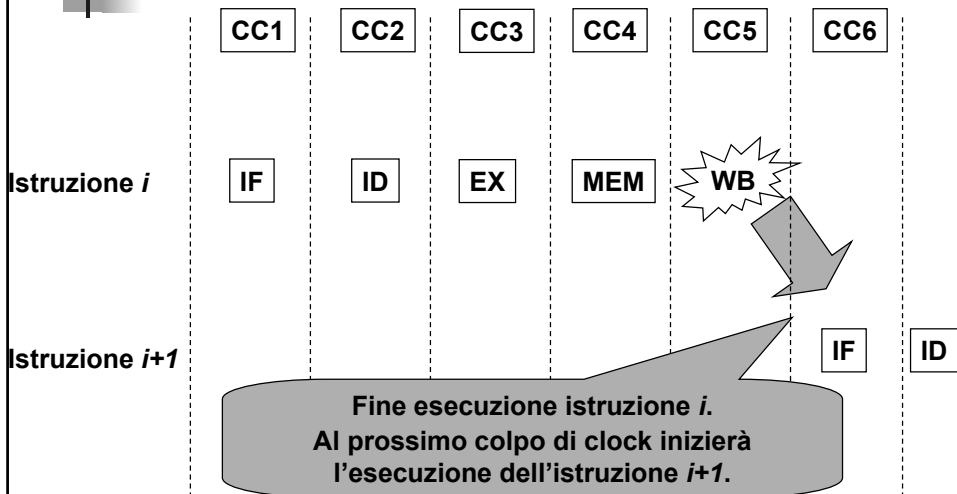
Write back (WB)

- Aggiorna i registri general purpose sulla base del risultato della ALU.

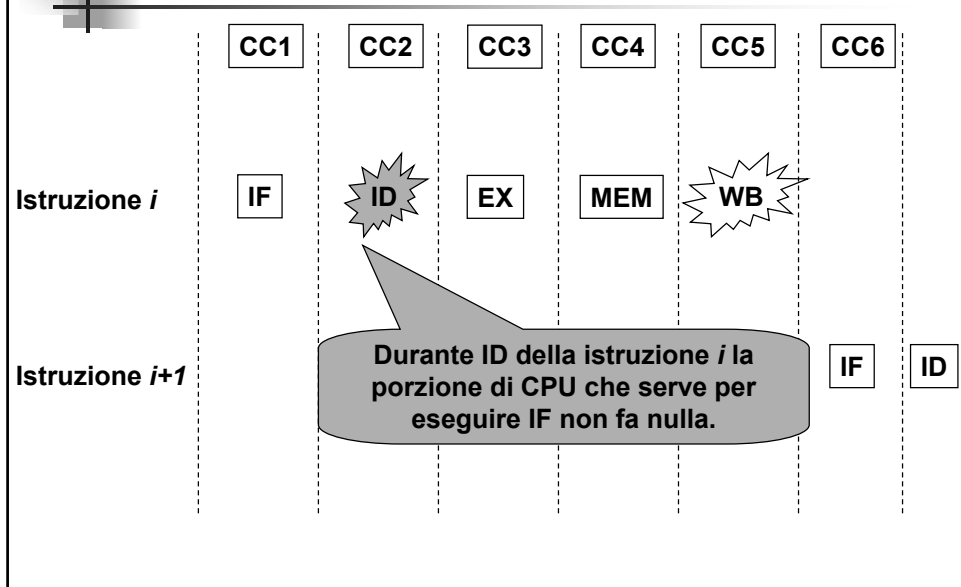
Architettura di base

- I moduli IF, ID, EX, MEM, WB impiegano lo stesso tempo ad eseguire il loro compito:
 - un colpo di clock;
- Il processore descritto esegue:
 - tutte le istruzioni in 5 colpi di clock
 - le istruzioni di salto in 4 colpi di clock.

Esempio di funzionamento

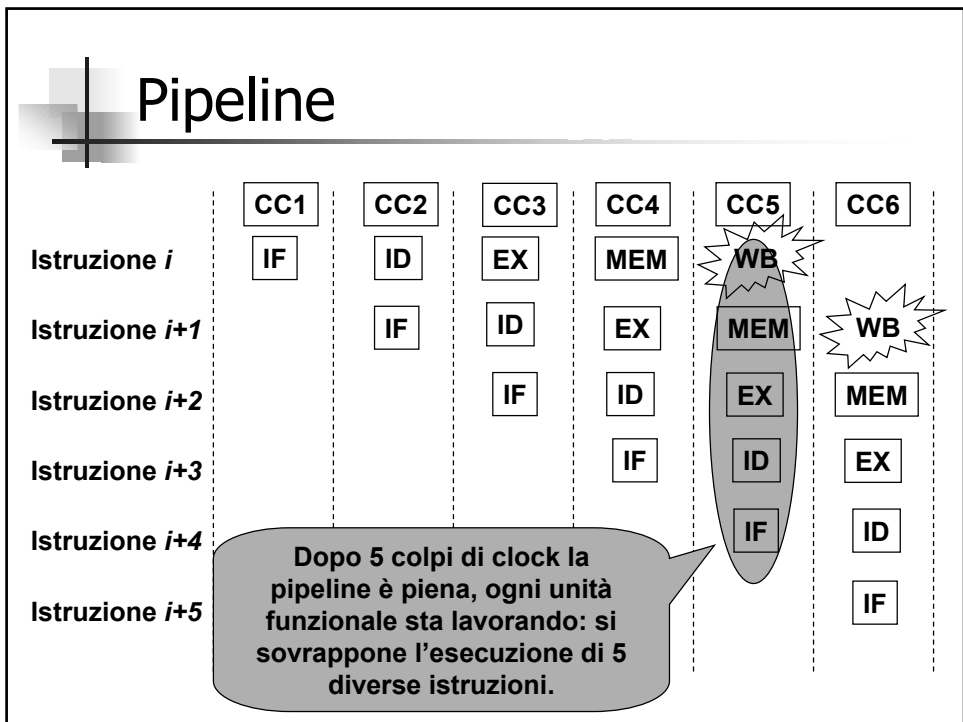
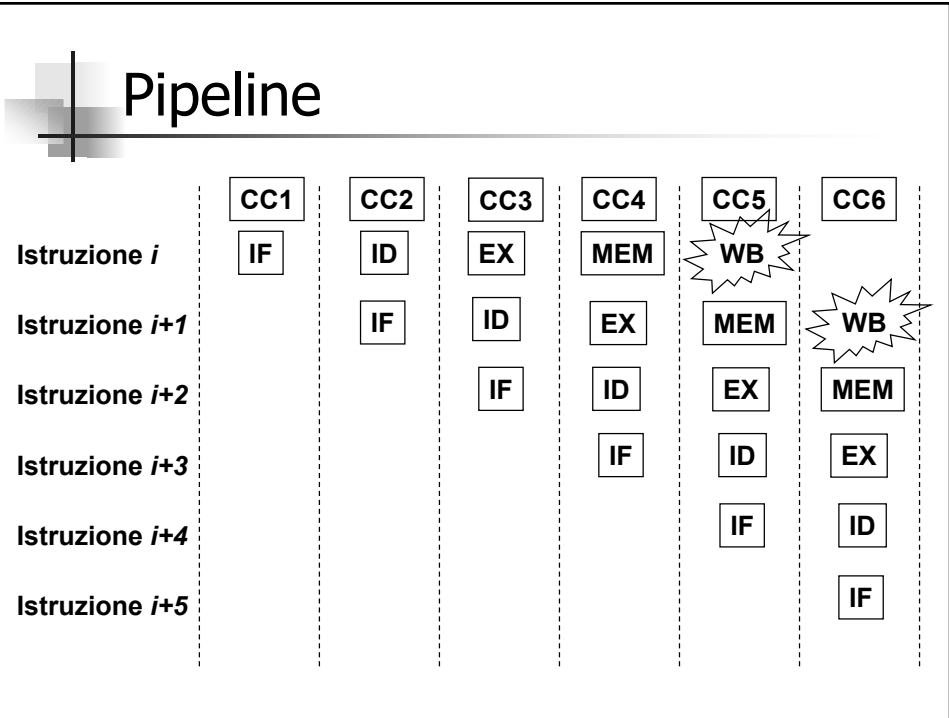


Funzionamento



Funzionamento

- Durante il colpo di clock CC_j dell'istruzione *i* solo una della unità funzionali sta lavorando, le altre sono inerti;
- Sovrapponendo le operazioni necessarie per eseguire istruzioni differenti:
 - si mantengono operative tutte le unità funzionali della CPU
 - si aumentano le prestazioni della CPU.



Pipeline

- Il risultato dell'istruzione i viene prodotto dopo 5 colpi di clock:
 - latenza della pipeline.
- Quando la pipeline è piena i risultati delle istruzioni vengono scritti ad ogni colpo di clock: è come se la CPU eseguisse una istruzione un 1 colpo di clock.

Pipeline

- La pipeline aumenta il numero di istruzioni che la CPU è in grado di completare nell'unità di tempo: throughput.
- La pipeline NON riduce il tempo necessario per eseguire una singola istruzione.
- In generale la gestione della pipeline comporta un overhead: il tempo di esecuzione di una istruzione AUMENTA.

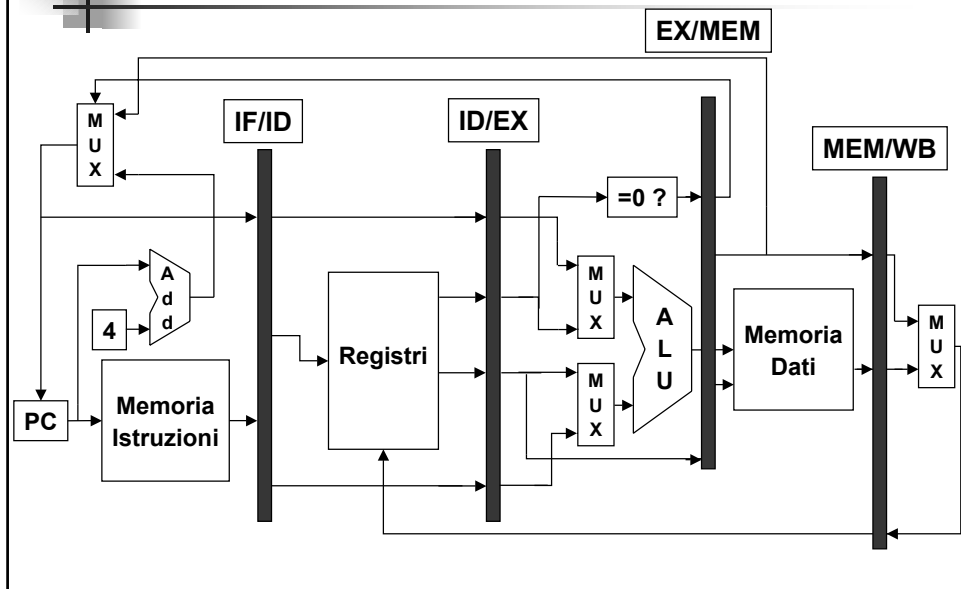
Tempistica

- Ogni operazione nella pipeline avviene in un colpo di clock.
- La durata del colpo di clock dipende dalla unità più lenta tra quelle presenti nella CPU:
 - si può dare il colpo di clock solo quando tutte le unità funzionali hanno completato il loro lavoro.

Esempio

- CPU con frequenza di clock di 10 ns
- Tutte le istruzioni richiedono 5 colpi di clock
- Overhead dovuto alla pipeline 1 ns
- Tempo di esecuzione di una istruzione:
 - senza pipeline: $5 \cdot 10 \text{ ns} = 50 \text{ ns}$
 - con pipeline: $1 \text{ ns} + 10 \text{ ns} = 11 \text{ ns}$
- Miglioramento: $50/11 = 4.5$

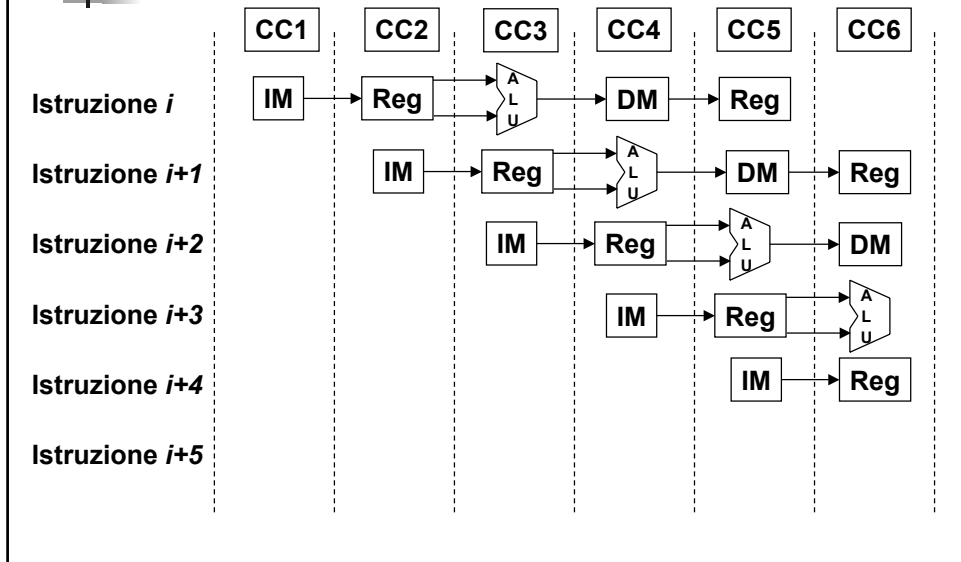
Processore con pipeline



Processore con pipeline

- Vengono inseriti appositi registri tra le unità funzionali della CPU.
- I registri contengono dati e segnali di controllo.
- Le unità funzionali comunicano tra di loro solo attraverso i registri della pipeline.

Esempio di funzionamento



Osservazione 1

- È necessario accedere contemporaneamente alla memoria istruzioni (IM) ed alla memoria dati (DM):
 - esempio in CC4
- Conviene usare due cache separate per dati e istruzioni in modo da eliminare conflitti di accesso ad un'unica memoria istruzioni/dati.

Conseguenza

- Data una frequenza di clock, in un sistema con pipeline come quello visto, la gerarchia di memoria deve essere più efficiente che nello stesso sistema senza pipeline;
- Accessi in memoria:
 - senza pipeline: 2 ogni 5 colpi di clock
 - con pipeline: 7 ogni 5 colpi di clock.

Osservazione 2

- Quando la pipeline è a regime (da CC5 in poi) si verificano operazioni contemporanee di lettura e scrittura nei registri
- Cosa succede se la CPU deve leggere e scrivere contemporaneamente dallo stesso registro?

Osservazione 3

- Per eseguire il fetch di una nuova istruzione ad ogni colpo di clock è necessario stabilire ad ogni colpo di clock il valore di PC.
- Il valore di PC può essere cambiato da una istruzione di salto durante la fase MEM.
- Come gestire i salti?

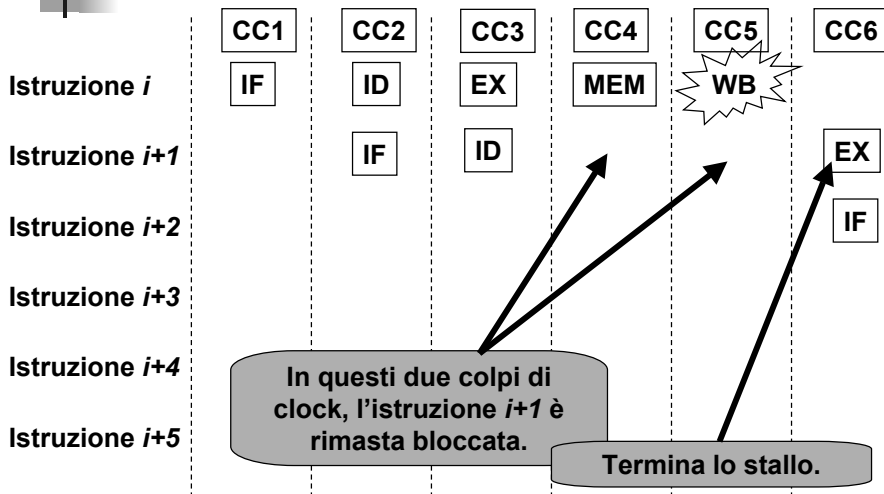
Problemi legati alla pipeline

- Esistono situazioni, dette Hazard, che impediscono il corretto funzionamento della pipeline
- Esistono tre categorie di hazard:
 - structural hazard;
 - data hazard;
 - control hazard.

Stallo della pipeline

- Quando l'istruzione i produce un hazard è necessario mandare in stallo la pipeline:
 - tutte le istruzioni inserite nella pipeline prima di i vengono completate;
 - tutte le istruzioni inserite nella pipeline dopo i vengono bloccate;
 - nessuna nuova istruzione viene inserita nella pipeline.
- Al termine di i , si riprende l'esecuzione delle istruzioni bloccate.

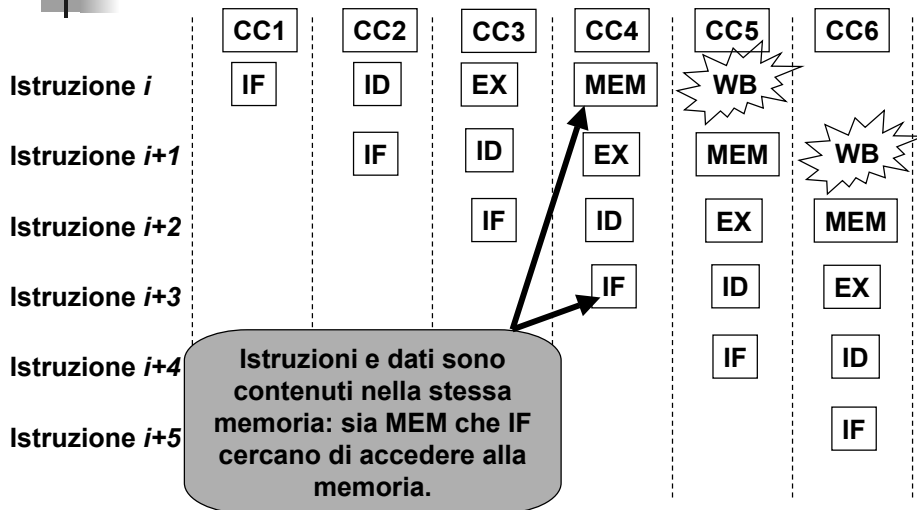
Stallo della pipeline



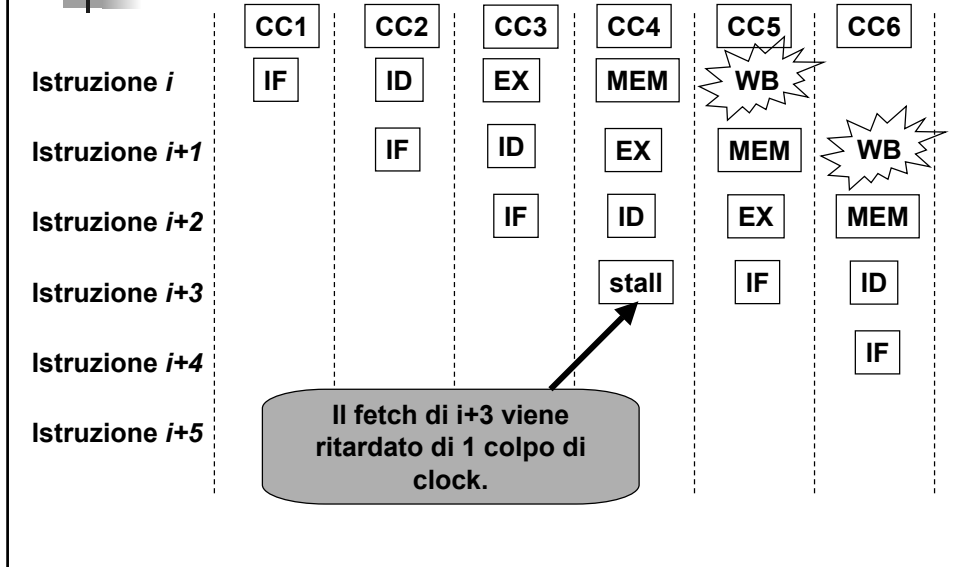
Hazard strutturali

- Non tutte le istruzioni possono essere eseguite con il meccanismo della pipeline.
- Esistono istruzioni che se parzialmente sovrapposte danno luogo a conflitti di risorse.
- Esempio:
 - CPU con un'unica memoria per istruzioni e dati.

Hazard strutturali



Soluzione



Hazard di dato

- La pipeline altera l'ordine temporale delle istruzioni sovrapponendo parzialmente la loro esecuzione.
- Si verifica un data hazard quando la pipeline modifica l'ordine con cui vengono letti/scritti gli operandi in modo tale che l'ordine con cui vengono letti/scritti gli operandi risulti diverso rispetto a quello che si avrebbe con una CPU senza pipeline.

Hazard di dato: esempio

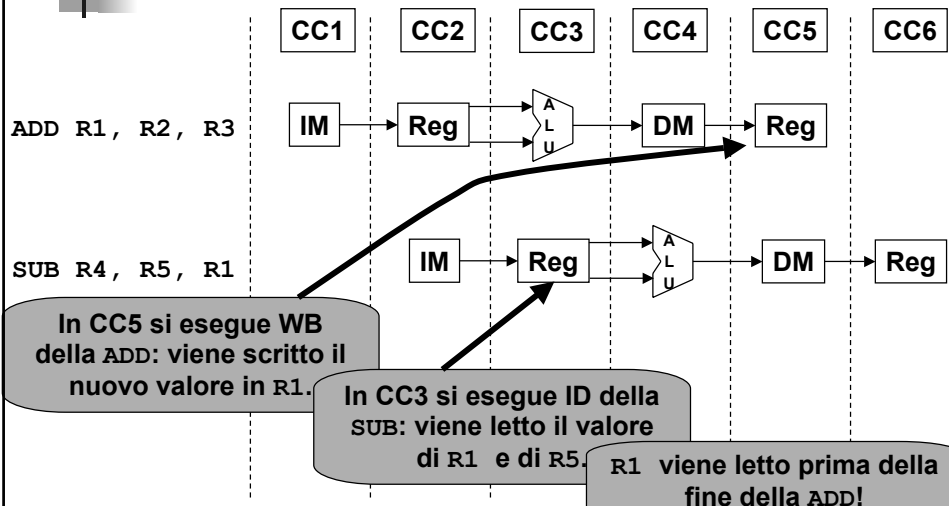
- Programma di esempio:

ADD R1, R2, R3 ; R1 = R2-R3

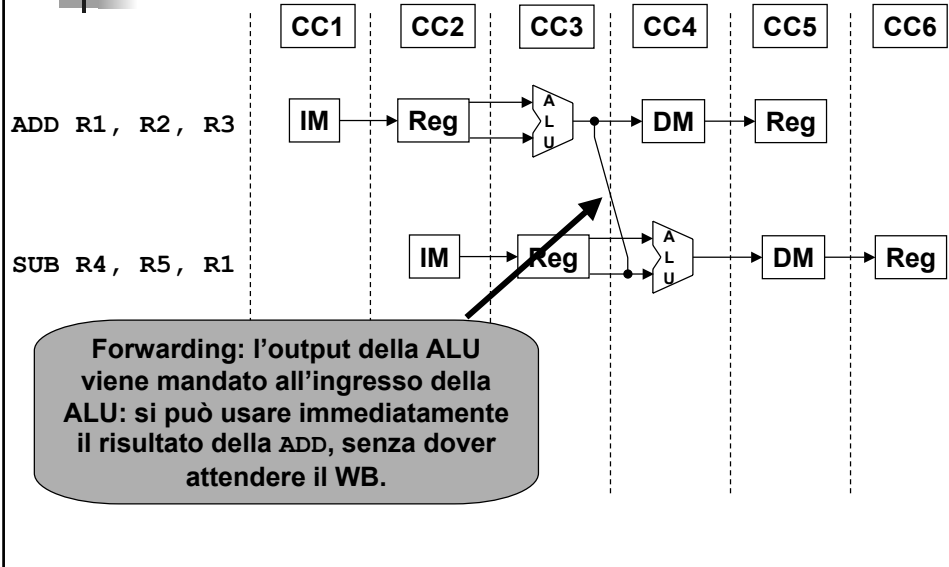
SUB R4, R5, R1 ; R4 = R5-R1

- Osservazione: la SUB utilizza il risultato della ADD, quindi deve essere eseguita solo quando il risultato della ADD è stato scritto in R1.

Hazard di dato



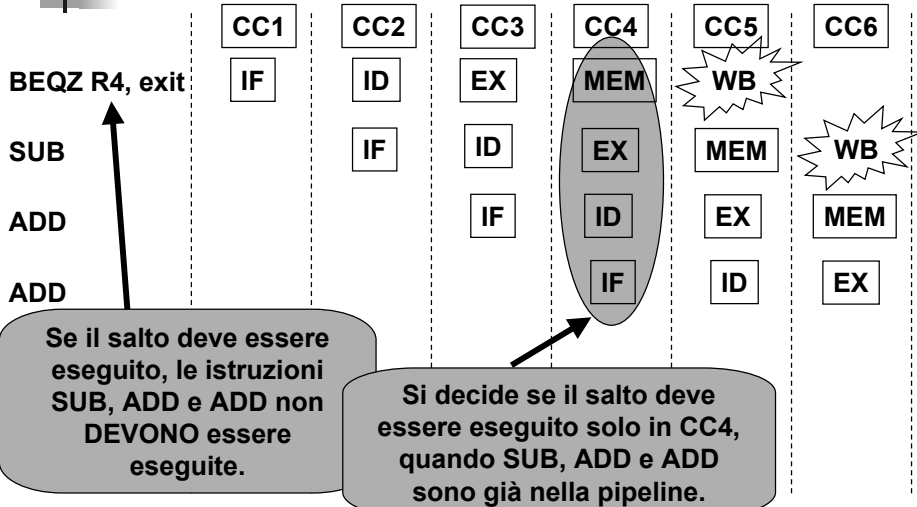
Soluzione



Hazard di controllo

- Si verificano in corrispondenza di una istruzione di salto;
- Esempio:
 - BEQZ R4, exit ; if R4 == 0 then exit
 - SUB R4, R1, R2
 - ADD R5, R6, R7
 - ADD R5, R4, R1

Hazard di controllo



Soluzione

