ARCHITETTURA DEGLI ELABORATORI

MAGGIO 2020

PROF. TRAMONTANA
www.dmi.unict.it/~tramonta

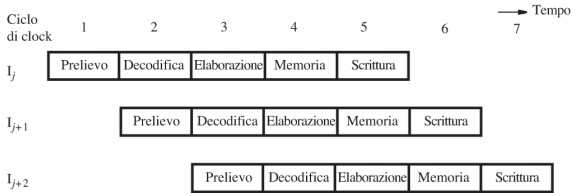
Capitolo 6
Introduzione al pipelining

Obiettivi

- Pipelining per migliorare le prestazioni
- Possibili situazioni che limitano il guadagno di prestazioni in processori con pipeline e modi per alleviare il loro effetto
- Implicazioni hardware e software del pipelining
- Influenza del pipelining sulla progettazione dell'insieme di istruzioni
- Processori superscalari
- Pipelining in processori CISC

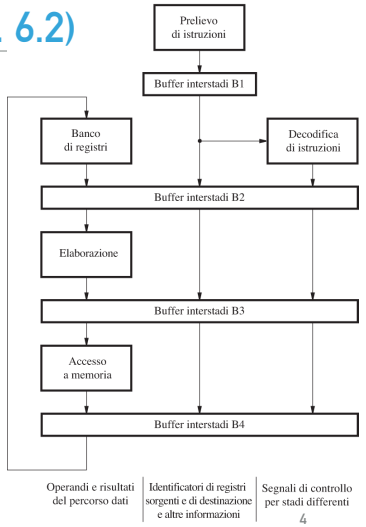
Pipeline ideale (S. 6.1)

- Pipelining: organizzazione a stadi di un processo produttivo per l'esecuzione parallela. Sovrapposizione di stadi diversi del processo per prodotti diversi
- Esempio diffuso nella manifattura: catena di montaggio. Nello stesso momento si hanno tanti prodotti a un diverso stadio. La realizzazione di un prodotto può richiedere più tempo, ma il **throughput** (quantità di prodotti che escono nell'unità di tempo) è maggiore
- Con l'organizzazione hardware a cinque stadi vista precedentemente



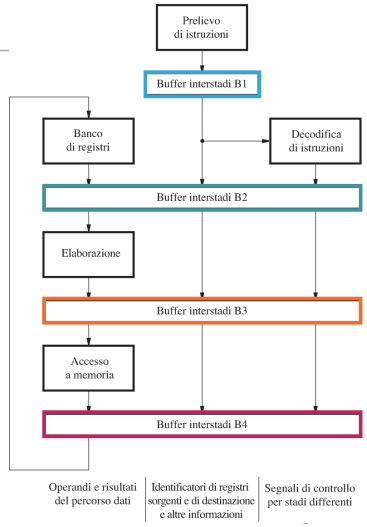
Organizzazione in pipeline (S. 6.2)

- I buffer interstadi contengono i registri RA, RB, RM, RY, RZ, IR, PC-Temp
- Il Buffer B1 alimenta lo stadio **Decodifica**
- Il Buffer B2 alimenta lo stadio **Elaborazione** con i due operandi letti dal banco registri, il valore immediato, il valore del PC, e i segnali di controllo
- Il Buffer B3 tiene il risultato del calcolo svolto dall'ALU, e nel caso di scrittura in **Memoria** il dato da scrivere
- Il Buffer B4 alimenta lo stadio **Scrittura** con un valore da scrivere nel banco dei registri (risultato dell'ALU o dell'accesso in memoria)



```
100 Add R2, R3, #100
101 Or R4, R5, R6
102 Subtract R9, R2, #30
```

- Ciclo di clock 1, prelievo istruzione 1, In **B1**: PC ha 101, IR ha `Add`
- Ciclo di clock 2, prelievo istruzione 2, decodifica istruzione 1
  - In **B1**: PC ha 102, IR ha `Or`
  - In **B2**: PC ha 101, IR ha `Add`, RA ha [R3]
- Ciclo di clock 3, prelievo istruz 3, decodifica istruz 2, elabor istruz 1
  - In **B1**: PC ha 103, IR ha `Subtract`
  - In **B2**: PC ha 102, IR ha `Or`, RA ha [R5], RB ha [R6]
  - In **B3**: PC ha 101, IR ha `Add`, RZ ha [R3]+100
- Ciclo di clock 4, preliev istruz 4, decod istruz 3, elab istruz 2, mem istruz 1
  - In **B1**: PC ha 104, … In **B2**: PC ha 103, IR ha `Subtract`, RA ha [R2]
  - In **B3**: PC ha 102, IR ha `Or`, RZ ha [R5] OR [R6]
  - In **B4**: PC ha 101, IR ha `Add`, RY ha [R3]+100
- Ciclo di clock 5, preliev istruz 5, decod istruz 4, elab istruz 3, mem istruz 2, scritt istr 1
  - In **B1**: PC ha 105, … In **B2**: PC ha 104, …
  - In **B3**: PC ha 103, IR ha `Subtract`, RZ ha [R2]-30
  - In **B4**: PC ha 102, IR ha `Or`, RY ha [R5] OR [R6]
  - In R2 si scrive [R3]+100

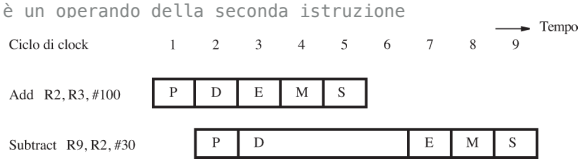


Problematiche del pipelining (S. 6.3)

- Il caso ideale di sovrapposizione degli stadi di esecuzione di istruzioni diverse non sempre è realizzabile in pratica
- Esempio: istruzione I_{j+1} con operando sorgente nel registro destinazione dell'istruzione I_j
  - L'istruzione I_{j+1} non può prelevare i dati dal registro sorgente fino a quando l'istruzione I_j non li abbia scritti, quindi resta in **stallo** nello stadio di decodifica, fino a che i dati non siano scritti
- Un **conflitto** (hazard) è qualsiasi causa di stallo nella pipeline. Nell'esempio prima è un **conflitto di dato**
- Altri conflitti: ritardo della memoria, istruzioni di salto, limiti di risorse

Dipendenze di dato (S. 6.4)

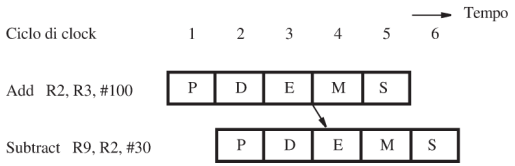
```
Add R2, R3, #100     // R2 è il risultato della prima istruzione
Subtract R9, R2, #30 // R2 è un operando della seconda istruzione
```



- La seconda istruzione va in stallo per tre cicli
- Il circuito di controllo rileva la dipendenza di dato confrontando gli identificatori dei registri delle due istruzioni nei buffer intestati B1 e B2
- L'istruzione in stallo rimane in B1 per tre cicli
- La prima istruzione procede e le successive istruzioni attendono
- Si possono produrre tre cicli di attesa generando NOP virtuali in B2, ovvero **bolle** (bubble) che procedono nella pipeline

Inoltro di operandi (operand forwarding)

- Per le istruzioni precedenti, la prima istruzione calcola R2 (somma di R3 e 100), la seconda istruzione legge R2
- L'ALU mette il risultato di Add in RZ (che è in B3) alla fine del ciclo 3
- Anziché aspettare che il risultato venga scritto in R2 nel ciclo 5, si usa il risultato dell'ALU del ciclo 3 come ingresso per l'ALU nel ciclo 4, ovvero si mettono in connessione diretta uscita e ingressi dell'ALU (notare la freccia)
- L'hardware del percorso dati deve essere modificato



Percorso dati per inoltro operandi

- Modifica del percorso dati per l'inoltro di dati dal registro RZ agli ingressi dell'ALU. E' necessario inserire il **MuxA** che seleziona fra **RA** e **RZ** l'ingresso da dare all'ALU
- L'inoltro su InA può essere esteso a RY per risolvere dipendenze come nella sequenza

```
Add      R2, R3, #100
Or       R4, R5, R6
Subtract R9, R2, #30
```

- In questa sequenza il valore generato da Add è ancora in RY e non in R2 quando serve a Subtract
- Sia MuxA che MuxB vengono estesi con un ulteriore ingresso al fine di farvi arrivare RY