



Architettura degli Elaboratori

Informatica per la Comunicazione Digitale

Università degli Studi di Milano

## CPU a pipeline: il concetto

Marco Tarini

1




## CPU monociclo (riassunto)

- Nel modello visto di CPU, ad ogni ciclo di clock si esegue tutta un'istruzione, comprese le fasi di
  - ▶ Fetch (accesso della memoria istruzioni al PC)
  - ▶ Decode (da parte della CU)
  - ▶ Estrazione dei valori dal banco dei registri
  - ▶ Eventuale operazione ALU, etc
  - ▶ Fino a memorizzare il risultato nei registri e nel PC (fase di "write back")
- Il limite di questa soluzione è che l'intero datapath è molto lungo
  - ▶ Periodo di clock lunghi sono richiesti, in modo da dare abbastanza tempo al datapath (e alla CU) di produrre i risultati finali da memorizzare (per es, in registri utente, PC, etc) per quando arriva il fronte del clock
- Una soluzione generale al problema di circuito combinatori troppo lunghi è "spezzarli" in sotto-circuiti, memorizzando i dati intermedi in nuovi appositi registri
  - ▶ in questa lezione panoramica, vediamo come applicare questa tecnica al datapath e più in generale alla CPU

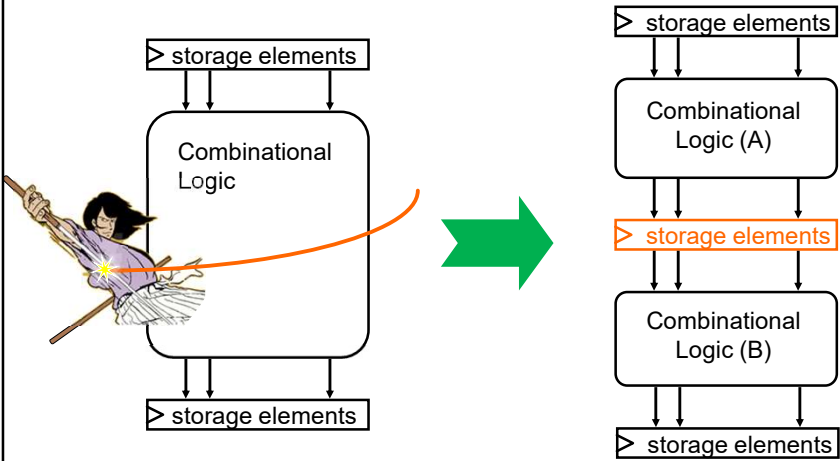
- 2 -

2



Riduzione del tempo di ciclo

- Si spezza la rete combinatoria in due parti (con registri in mezzo)
- Il lavoro che si faceva in un ciclo lento viene fatto in due cicli, più veloci




Architettura degli elaboratori

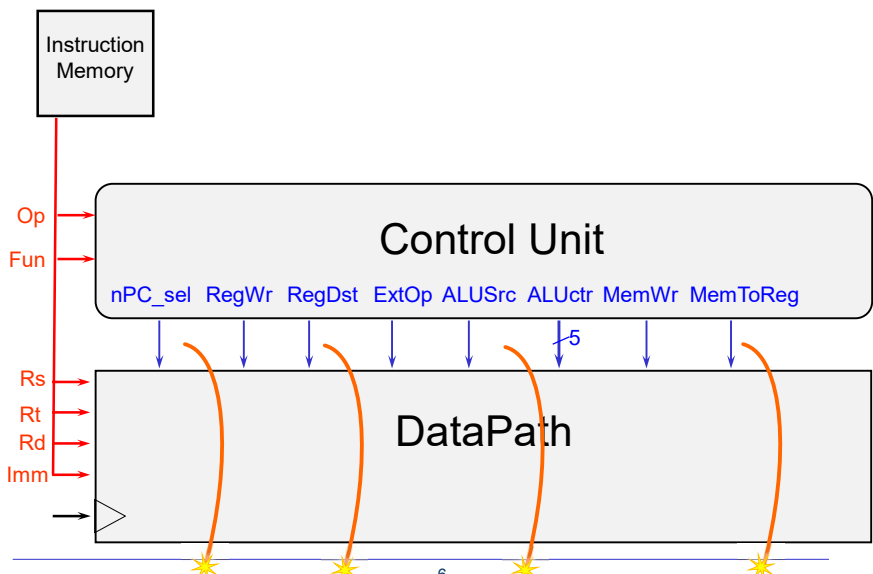
- 5 -

CPU multiciclo

5

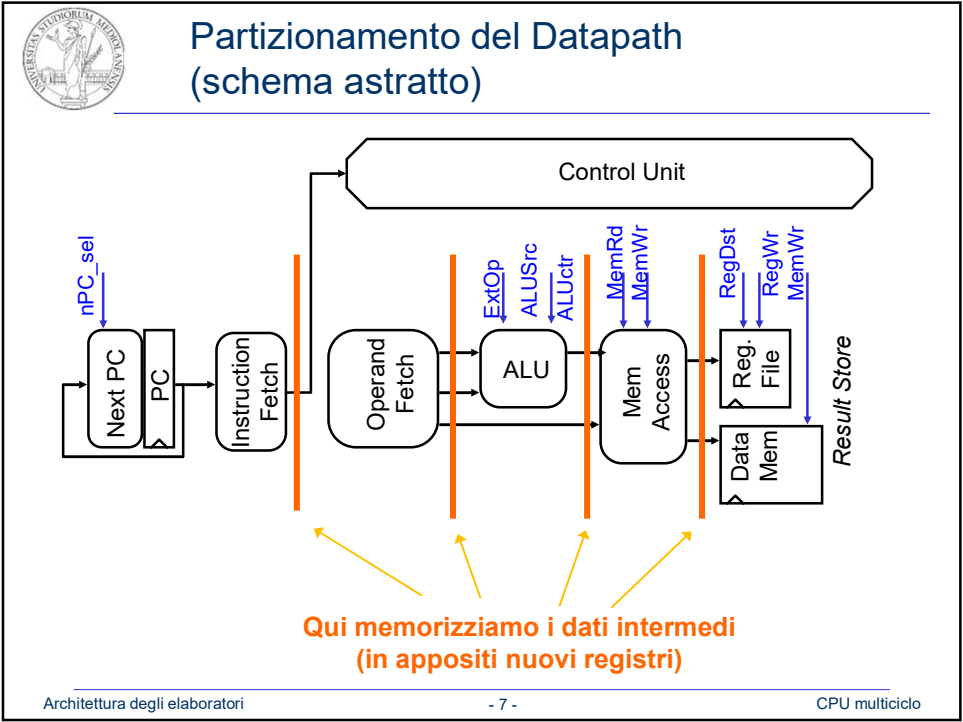


Esempio di CPU monociclo  
Quadro complessivo

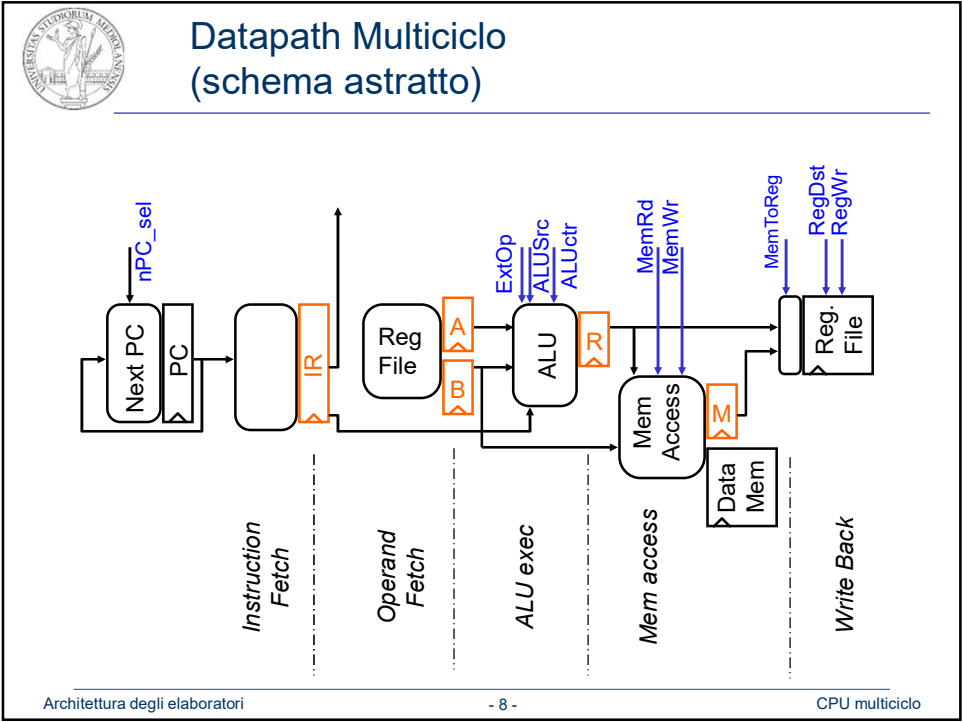


- 6 -


6



7



8



## CPU multiciclo: vantaggi ulteriori


- Nella CPU a 1 ciclo, abbiamo dovuto replicare alcuni componenti del datapath, come:
  - ▶ RAM. Infatti:
    - per un'istruzione "LW", in uno stesso ciclo devo sia leggere un'istruzione dalla RAM (come per tutte le istruzioni) sia leggere un dato dalla RAM:
    - Nostra soluzione finora (poco adottata in realtà): una RAM ulteriore, separata dalla RAM dati, solo per le istruzioni :-)
  - ▶ ALU. Infatti:
    - per un'istruzione "BEQ", in uno stesso ciclo devo sia computare la condizione (di eguaglianza), sia eseguire una somma per calcolare il valore del nuovo PC
    - nostra soluzione finora (poco adottata in realtà): un addizionatore ulteriore, separato da quello nella ALU, solo per il computo del nuovo PC dopo il salto :-)
- Invece, nella CPU multi-ciclo, la stessa RAM e la stessa ALU possono assolvere funzioni diverse in sotto-fasi diverse di una stessa istruzione! :-)
- Il ulteriore grande opportunità consiste nello svolgere le fasi avanzate dell'istruzione corrente in parallelo con le fasi iniziali dell'istruzione successiva,
  - ▶ Quest'idea porta al concetto di pipeline (catena di montaggio)
  - ▶ Vediamone il funzionamento, partendo da un esempio metaforico

Architettura degli elaboratori





- 9 -

CPU multiciclo

9

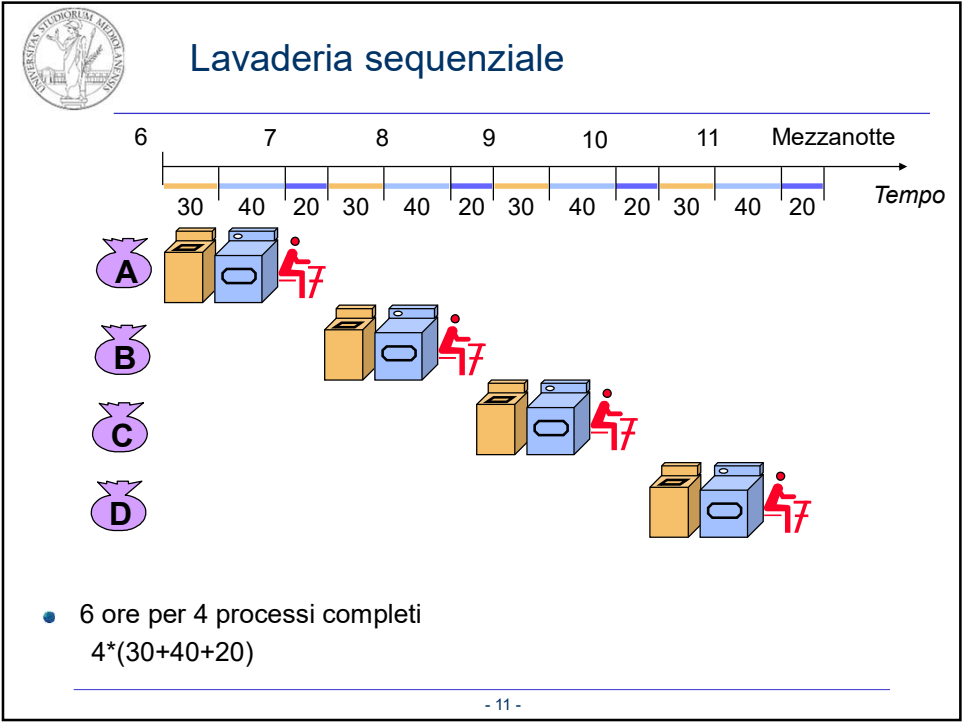


## Il pipelining è una soluzione generale: esempio di una lavanderia

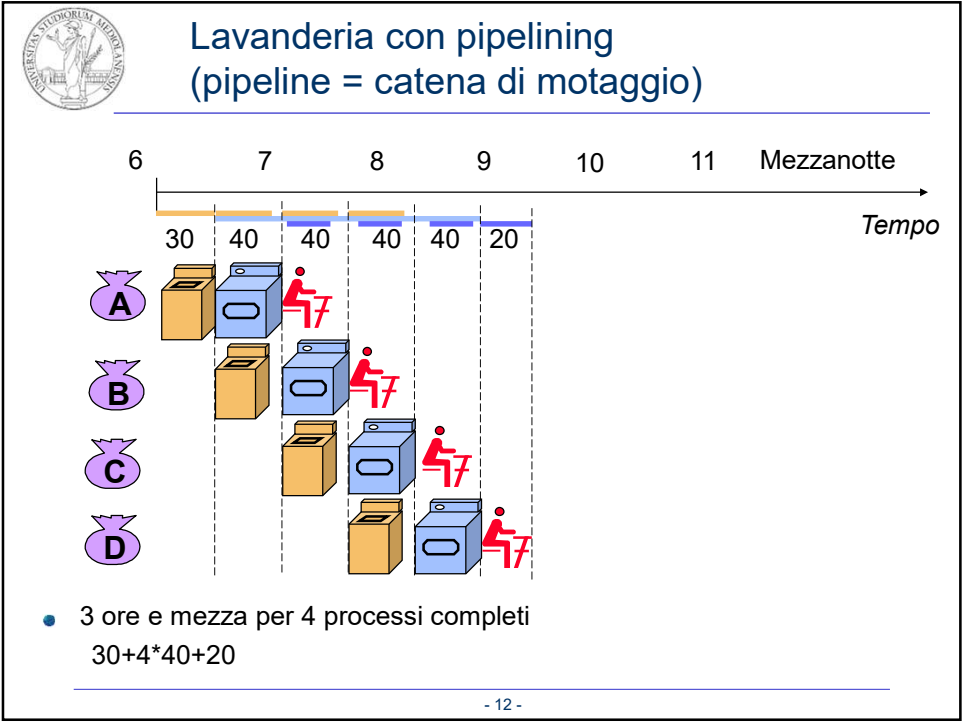
- Anna, Bernardo, Caterina e Davide hanno ciascuno dei panni da lavare, asciugare e piegare 
- ▶ Il lavaggio impiega 30 minuti 
- ▶ L'asciugamento impiega 40 minuti 
- ▶ Per la piegatura ci vogliono altri 20 minuti 

- 10 -


10



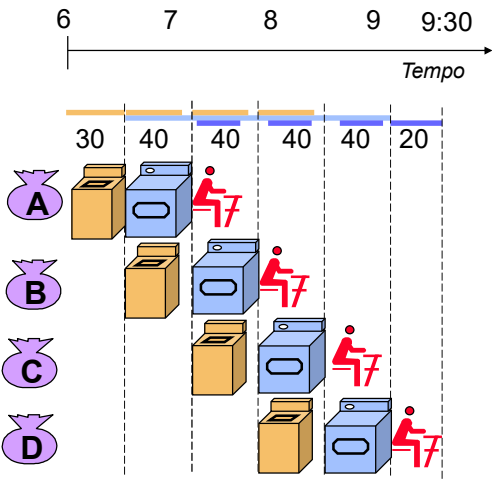
11



12



## Pipelining: caratteristiche




Tempo

- Il singolo task *non* viene velocizzato.
- L'intero processo è velocizzato
- Il tempo di una fase dipende dalla fase più lenta (detta «collo di bottiglia»)
- Diversi task che usano risorse diverse sono attivi in parallelo
- Più sono le fasi e migliore è l'efficienza complessiva (potenzialmente – vedi dopo)
- Fasi di lunghezza diversa riducono il miglioramento
- A regime, termino un intero bucato nella durata di una fase (x slot da 40min → x bucati)

- 13 -

13

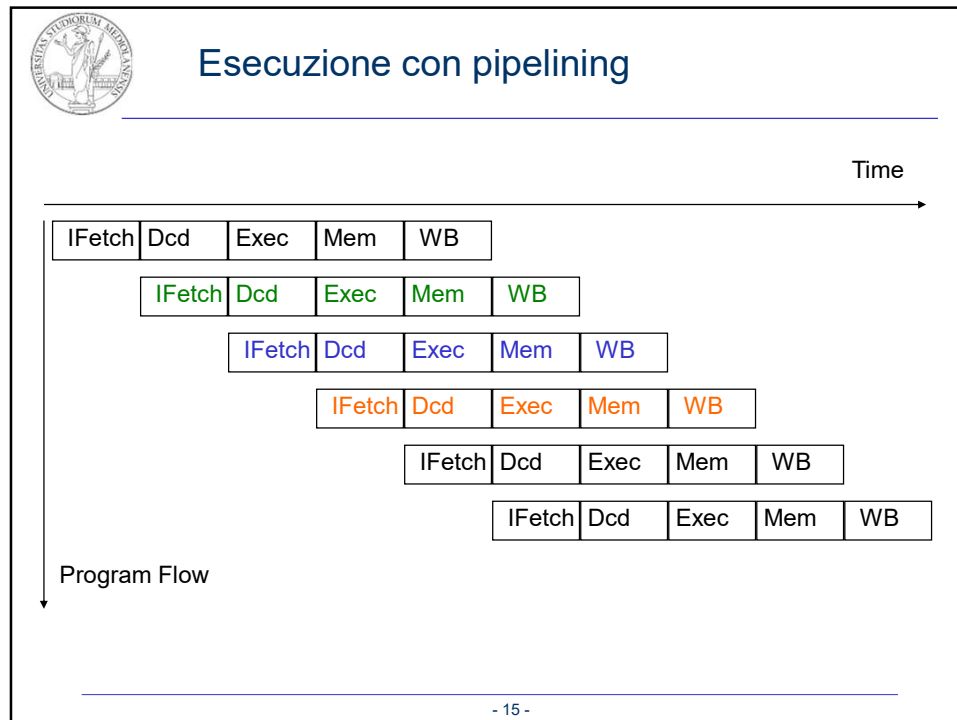


## Fasi di esecuzione di una istruzione in una CPU pipelined

- Instruction Fetch
  - ▶ Accedere alla RAM per ottenere il valore dell'istruzione corrente
  - ▶ Registro intermedio prodotto: contiene l'**istruzione** (questo registro è detto Instruction Register, o IR)
- Decode
  - ▶ La CU produce i **bit di controllo**
  - ▶ Registro intermedio prodotto: tutti i controlli
- Operand fetch
  - ▶ Il banco di registri produce gli operandi della ALU
  - ▶ Registri intermedi prodotti: due nuovi appositi registri interni A e B, con gli operandi
- Exec
  - ▶ La ALU elabora ed espone in uscita il risultato dell'operazione logica o matematica
  - ▶ Registri intermedi prodotti: un registro R contenente il risultato
- Memory access
  - ▶ La memoria produce la parola letta (per le istruzioni che lo richiedono)
  - ▶ Registro intermedio: un altro apposito registro M
- Write back
  - ▶ Scrive dove appropriato il nuovo PC, i valori computati, la parola da scrivere in RAM

- 14 -

14



15

**I problemi che una architettura pipelined deve affrontare (detti «hazard»)**

- **Structural hazard:** tentativo di usare la stessa risorsa (ALU, RAM, etc) di due fasi diverse di due istruzioni diverse, svolte contemporaneamente
- **Data hazard:** tentativo di usare un dato prima che sia pronto
  - ▶ Ad es. un'istruzione scrive un registro \$x, e l'istruzione successiva legge proprio quel registro.
  - ▶ L'istruzione successiva non trova i dati corretti nel banco dei registro prima che la precedente abbia finito la sua «write back»
- **Control hazard:** tentativo di eseguire l'istruzione successiva prima di sapere... quale sia
  - ▶ A partire dall'istruzione fetch, l'istruzione successiva di solito parte da una lettura dal PC attuale incrementato di 4
  - ▶ Nel caso una branch, quale sia l'*effettiva* istruzione successiva è noto solo dopo aver computato la condizione del salto
- Si può sempre risolvere ogni hazard aspettando
  - ▶ Cioè mettendo in pausa le istruzioni successive
  - ▶ L'hazard deve essere individuato

- 18 -

18