CALCOLATORI
La pipeline

Marco Roveri
marco.roveri@unitn.it

Lezione basata su materiale preparato dai Prof. Luigi Palopoli e Giovanni Iacca

UNIVERSITÀ DEGLI STUDI DI TRENTO
Dipartimento di Ingegneria
e Scienza dell'Informazione

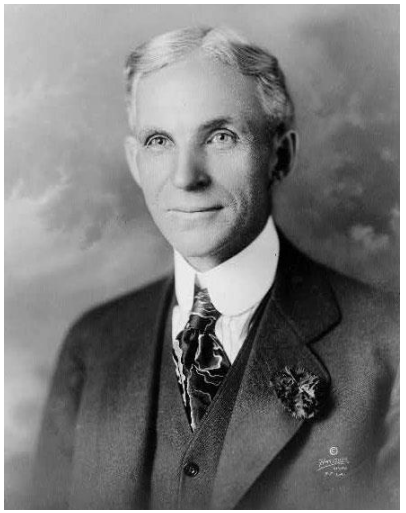
Ripartiamo da questo....

- Abbiamo visto come realizzare un semplice processore che esegue le istruzioni in un ciclo
- Questo non si fa più nelle implementazioni moderne perché:
  - A dettare il periodo di clock (comune a tutte istruzioni) sono le istruzioni più lente (ovvero istruzioni di accesso alla memoria)
  - Se si mettono istruzioni più complesse di quelle che abbiamo visto, le prestazioni peggiorano ulteriormente (esempio gestione operazioni su floating point)
  - Non si riescono a fare ottimizzazioni aggressive sulle cose fatte più di frequente.

Che fare allora?

- Il cosa fare ce lo insegnò Henry Ford con la catena di montaggio (assembly line):

"If everyone is moving forward together, then success takes care of itself."



"Coming together is a beginning. Keeping together is progress. Working together is success."

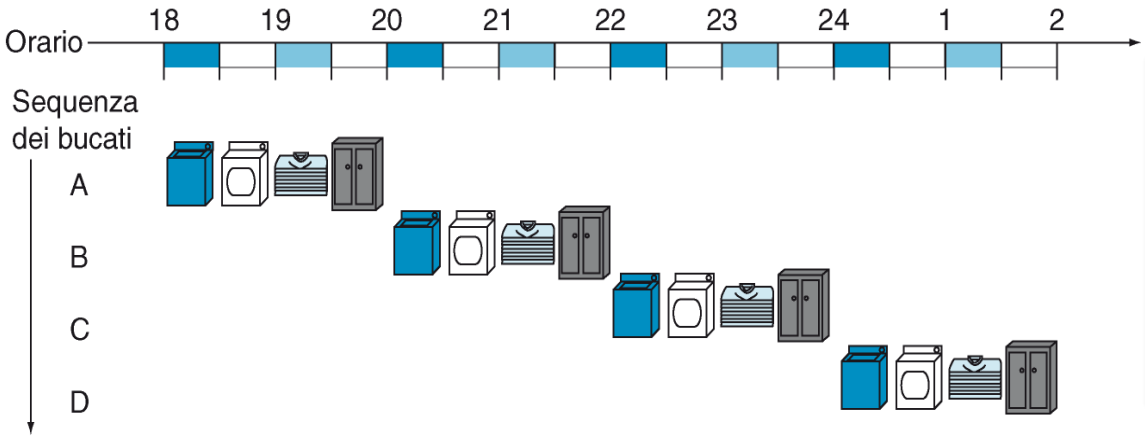
Un esempio

- Supponiamo di dover fare il bucato e che questo consista nelle seguenti attività:

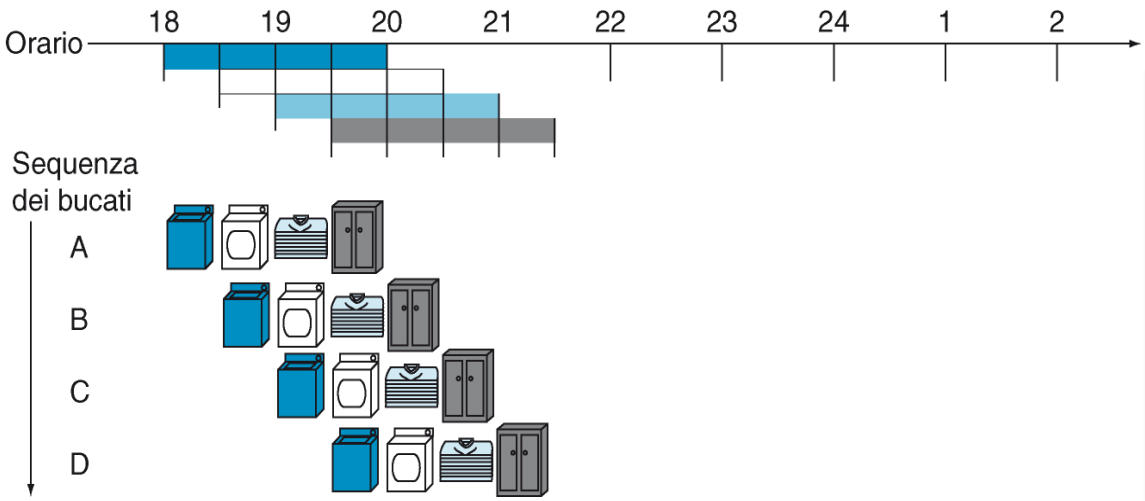
1. Mettere la biancheria nella lavatrice
2. Terminato il lavaggio mettere i panni nell'asciugatrice
3. Terminata l'asciugatura mettere i panni sull'asse da stiro e procedere alla stiratura
4. Finita la stiratura chiedere al proprio co-inquilino di riporre i panni

- Abbiamo un grande vantaggio se mentre la lavatrice lava un bucato, l'asciugatrice ne asciuga un altro e io ne stiro un altro ancora

Una rappresentazione grafica



Supponendo che ciascuna fase duri 30 minuti, il ciclo di lavaggio dura due ore (tra le 18 e le 20 per il primo).
Con il secondo metodo fino alle 21.30 ne faccio 4.



Notare che il tempo per fare un bucato (tempo di risposta) rimane invariato (due ore).
Il throughput passa da 1/4 a 4/7 (miglioramento di 2.3).
Se ci fossero molto più di 4 bucati, si potrebbe arrivare ad un fattore di circa 4 di miglioramento.

Andiamo su un esempio più serio

- Tornando al RISC-V, le fasi di esecuzione di un'istruzione sono le seguenti:

1. Prelievo dell'istruzione dalla memoria
2. Lettura dei registri e decodifica dell'istruzione
3. Esecuzione di un'operazione (tipo R) o calcolo di un indirizzo
4. Accesso a un operando nella memoria dati (se richiesto)
5. Scrittura del risultato in un registro (se richiesto)

- Conseguentemente la pipeline dovrà avere cinque stadi

Un esempio

- Limitiamoci per il momento a una pipeline che sia in grado di effettuare le seguenti operazioni:
  - LOAD (ld), STORE (sd)
  - Somma (add), Sottrazione (sub), AND (and), OR (or)
  - Branch on equal (beq)

Tempi

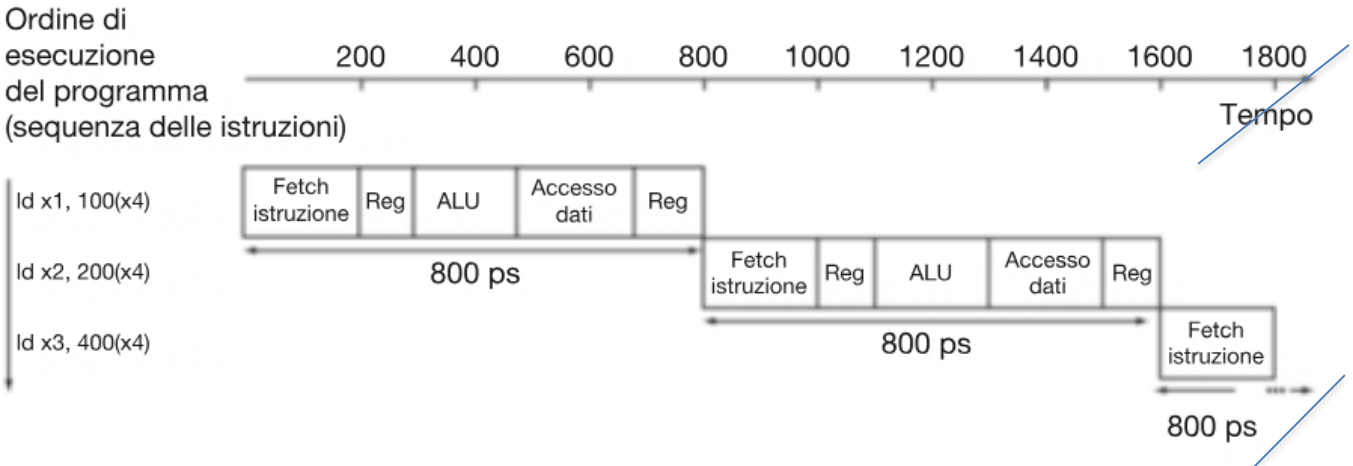
- I tempi richiesti per le varie fasi sono i seguenti:

Tipo di istruzione	Lettura dell'istruzione	Lettura dei registri	Operazione con la ALU	Accesso ai dati in memoria	Scrittura del register file	Tempo totale
Load doubleword (ld)	200 ps	100 ps	200 ps	200 ps	100 ps	800 ps
Store doubleword (sd)	200 ps	100 ps	200 ps	200 ps		700 ps
Formato R (add, sub e or)	200 ps	100 ps	200 ps		100 ps	600 ps
Salto condizionato (beq)	200 ps	100 ps	200 ps			500 ps

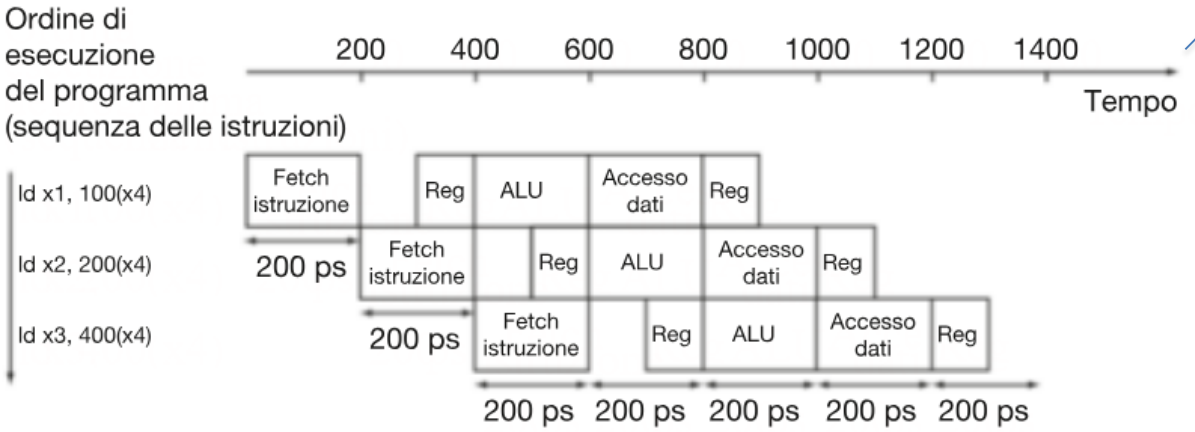
- L'istruzione più lenta è la ld. Quindi ci si deve adeguare ad essa per la scelta del clock

Varie implementazioni

- Diagramma temporale per sequenza istruzioni:



Implementazione a singolo ciclo



Implementazione a pipeline

Tempo totale tra fine prima a fine terza istruzione passa da (3*800) a 1400 ps.

Confronto di prestazioni

- Un confronto di prestazioni può essere fatto con la seguente formula (valida se la pipeline opera in condizioni ideali)

$$\text{Tempo tra due istruzioni con pipeline} = \frac{\text{Tempo tra due istruzioni senza pipeline}}{\text{Numero di stadi della pipeline}}$$

- La formula suggerisce che con cinque stadi dovremmo arrivare a un tempo tra due istruzioni (inverso del *throughput*) che dovrebbe essere 1/5.

Osservazioni

- Nell'esempio precedente siamo passati da 2400 a 1400 (miglioramento 1.7) come mai non 5?
- *Prima osservazione*: per ottenere una prestazione di 1/5 dovremmo portare il clock a 160 ps (non possibile) perché ci sono alcune fasi che durano 200 ps (al più 4 volte)
- *Seconda osservazione*: abbiamo considerato poche istruzioni e non abbiamo fatto in tempo a riempire la pipeline.

Comportamento al limite

- Se consideriamo molte più istruzioni (ad esempio ne aggiungiamo **1000000** all'esempio di prima) per un totale di **1000003** si passa da un tempo di esecuzione di 1000003*800ps a un tempo di esecuzione di 1400 + 1000000*200ps.
- Se facciamo il rapporto vediamo che l'incremento prestazionale in termini di *throughput* si avvicina al 400%

Vantaggi del RISC

Vantaggi delle architetture RISC

- ***Primo vantaggio:*** Tutte le istruzioni hanno la stessa lunghezza. Questo facilita di molto il prelievo (sempre una word).
- ***Secondo vantaggio***: I codici degli operandi sono in posizione fissa. Questo permette di accedervi leggendo il register file in parallelo con la decodifica dell'istruzione.
- ***Terzo vantaggio***: Gli operandi residenti in memoria sono possibili solo per ld/lw e sd/sw. Ciò permette di usare la ALU per il calcolo di indirizzi (cosa che non sarebbe possibile se dovessimo usare le ALU in due fasi della stessa istruzione).
- ***Quarto vantaggio***: L'uso di accessi allineati fa sì che gli accessi in memoria avvengano sempre in un ciclo di trasferimento (impegnando un solo stadio della pipeline).

Hazard

- In condizioni normali la pipeline permette di eseguire un'istruzione per ciclo di clock.
- Alle volte questo non è possibile per il verificarsi di «condizioni critiche» (detti «hazard»).
- Passiamo in rassegna alcuni tipologie di hazard.

Hazard Strutturali

- Una condizione di hazard strutturale è una condizione per la quale l'architettura dell'elaboratore rende impossibile l'esecuzione di alcune sequenze di istruzioni in pipeline.
- Ad esempio, se io disponessi di un'unica memoria, non potrei nello stesso ciclo, caricare istruzioni e memorizzare (o prelevare) operandi dalla memoria.

Hazard sui dati

- Questo tipo di hazard si verifica quando la pipeline deve essere messa in stallo per ottenere delle informazioni dagli stadi precedenti
- Nell'esempio della stiratura, se mi accorgo che manca un calzino, devo interrompere la stiratura dell'altro e andare a cercarlo (bloccando anche le fasi precedenti).
- Nel caso del RISC-V, consideriamo la seguente sequenza

```
add x1, x2, x3
sub x4, x1, x5
```

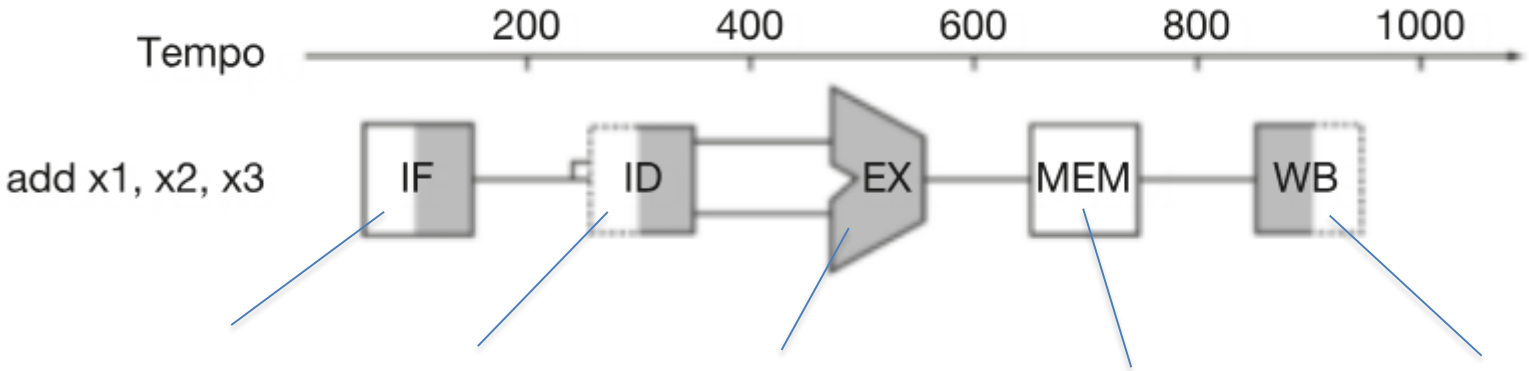
- Il problema è che x1 viene memorizzato nella quinta fase, mentre la sub dopo ne ha bisogno nella seconda fase… quindi è costretta ad aspettare per 3 cicli di clock!

Come risolverlo?

- L'hazard precedente blocca il completamento della seconda istruzione per tre cicli di clock
- In certi casi possiamo cavarcela a livello di compilazione invertendo alcune istruzioni
- Tuttavia il compilatore può risolvere il problema solo in alcuni casi
- In generale, è utile osservare che non occorre aspettare di aver memorizzato il risultato

Torniamo all'esempio

- Una rappresentazione grafica per l'operazione della pipeline su una delle istruzioni è la seguente:



Instruction fetch (quadratino memoria istruzioni)

Instruction decode (quadratino spezzato register file)

Instruction Execute

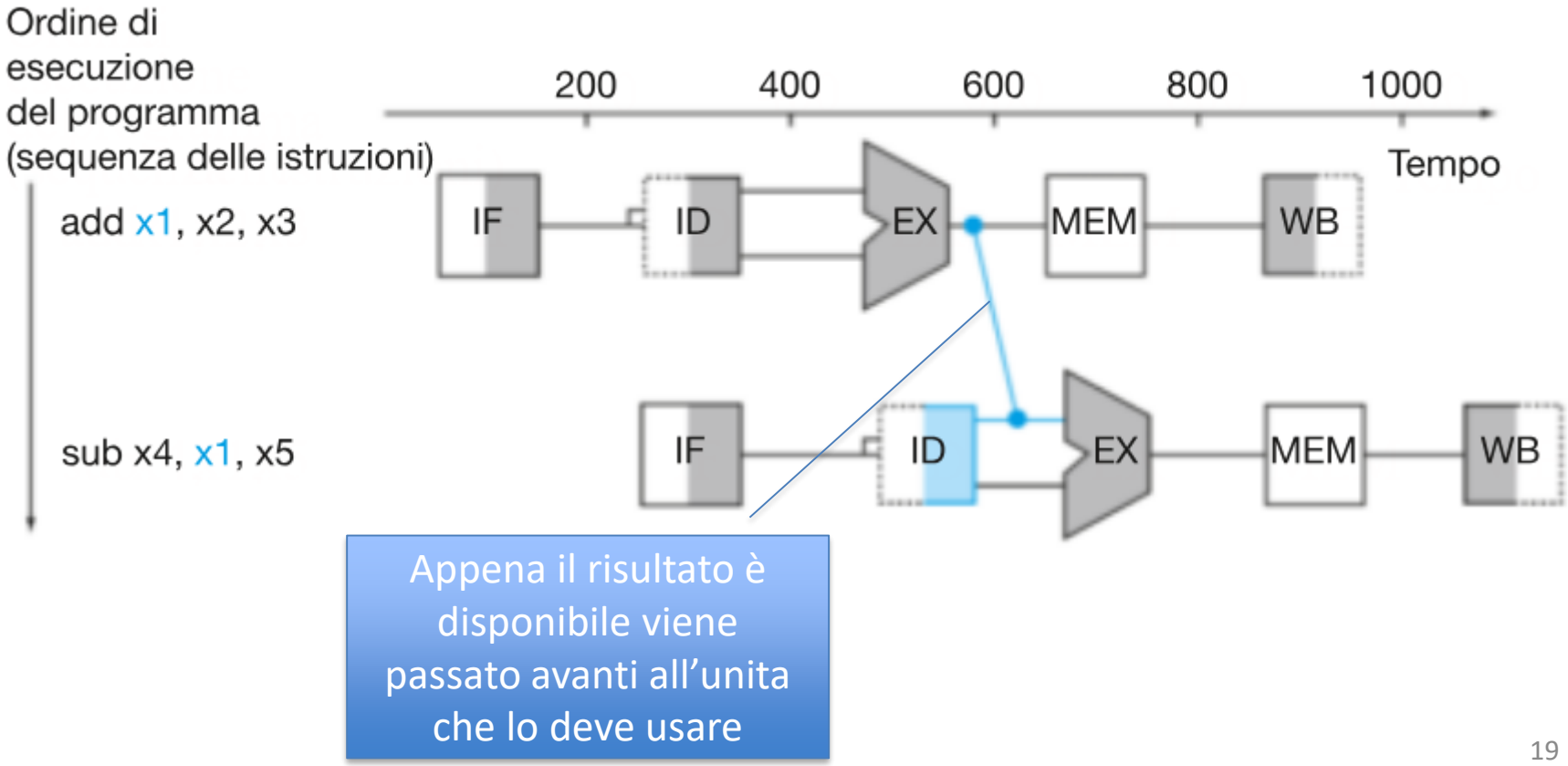
Accesso alla memoria

Write back (scrittura risultato nel register file)

Ombra = unità usata per istruzione

Operand forwarding

- L'operand forwarding (detto anche propagazione, o anche bypass) viene usato per rendere il risultato disponibile bypassando l'operazione di storage

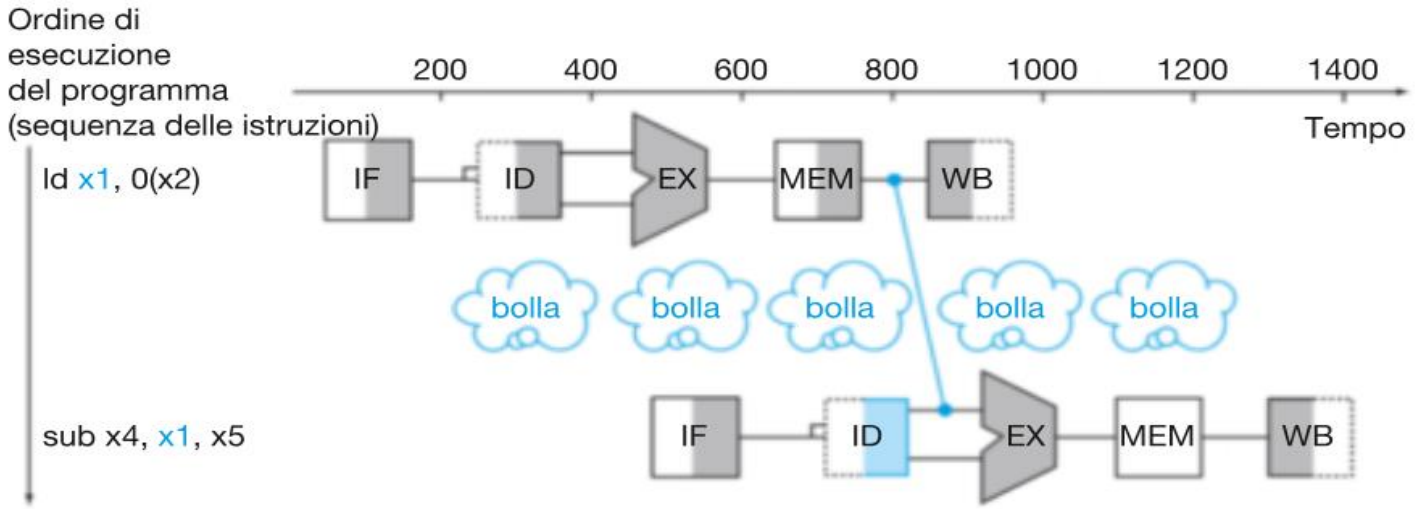


Operand forwarding continued

- L'operand forwarding funziona solo se lo stadio a cui il dato viene propagato è successivo nel tempo allo stadio dal quale viene prelevato
  - Non si può propagare tra l'uscita della fase di accesso alla memoria di istruzione e l'ingresso della fase di esecuzione dell'istruzione successiva:
    - ✓Occorrerebbe propagare un dato indietro nel tempo

Operand forwarding continued

- Supponiamo di avere sequenza:
  ld x1, 0(x2)
  sub x4, x1, x5
  - x1 sarebbe disponibile solo dopo il quarto stadio della prima istruzione
    - ✓ Troppo tardi per essere usato come input al terzo stadio della sub
    - ✓ Dovremmo imporre uno «stallo della pipeline» della durata di un ciclo di clock («hazard sui dati» di una load)



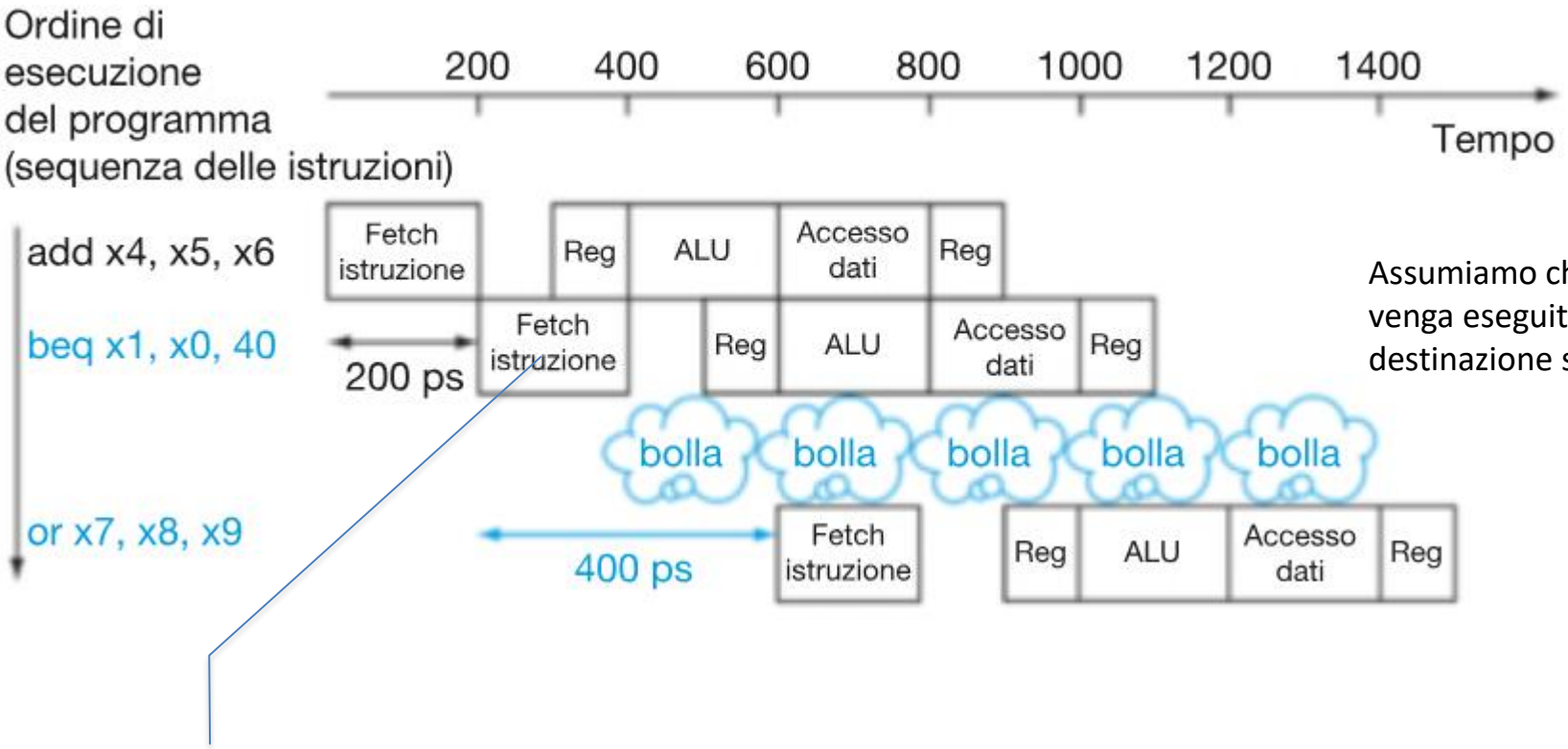
Hazard sul controllo

- Il terzo tipo di hazard riguarda il controllo (sostanzialmente i salti condizionati)
- Torniamo all'esempio del bucato
  - Supponiamo che, a seconda del livello di sporco, si voglia decidere per un lavaggio aggressivo
  - Quello che dovrei fare è verificare la condizione dei panni all'uscita dell'asciugatrice e su questa base cambiare le impostazioni
  - … ma nel far questo, si blocca la pipeline (fino a che non ho finito l'asciugatura non posso procedere al lavaggio della prossima mandata)

Salto condizionato

- Il caso simile a quello del bucato si presenta con i salti condizionati
- Supponiamo di avere un circuito molto sofisticato che ci permette di calcolare l'indirizzo di salto già al secondo stadio
- Comunque, dobbiamo bloccare la pipeline per uno o due cicli

Esempio



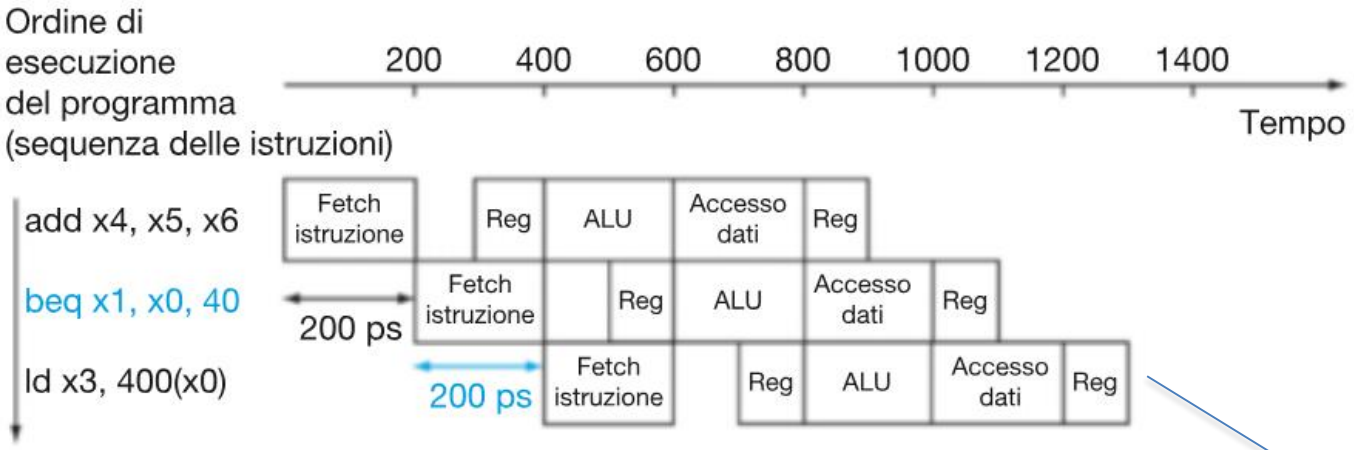
Assumiamo che il salto venga eseguito e che la destinazione sia la or

Alla fine del prelievo capisco che è un beq e aspetto per un ciclo prima di fare il prelievo di quella successiva

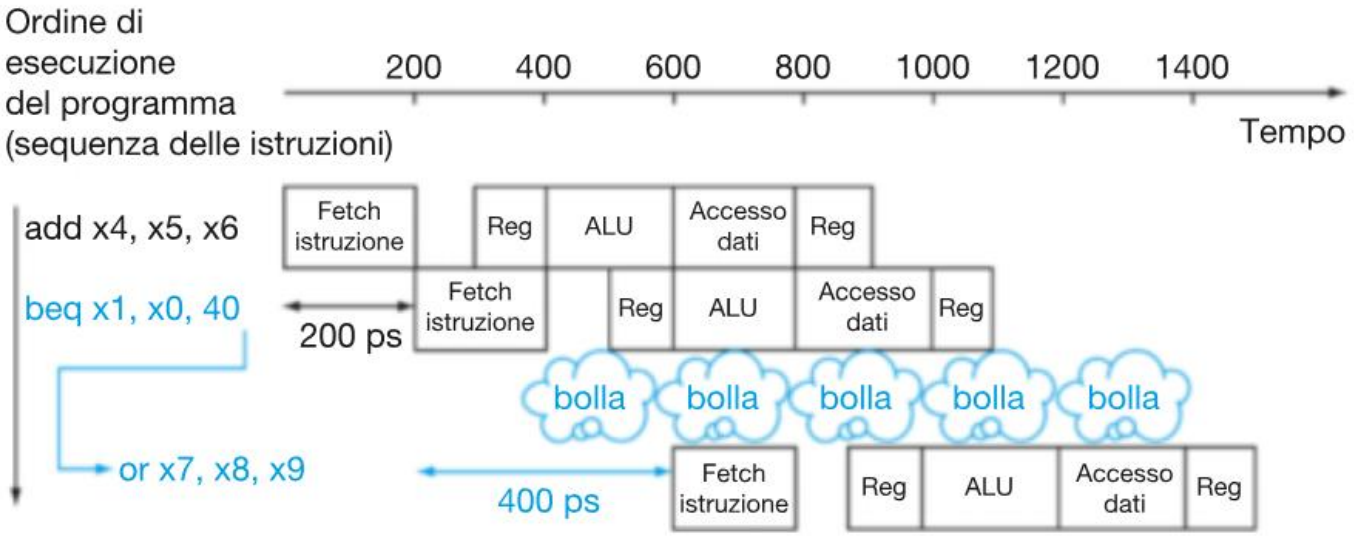
Predizione del salto

- Se la pipeline è più lunga, generare questo stallo risulta in un incremento elevato della durata che risulta in una riduzione delle prestazioni
- Quello che si fa è avere dei circuiti che prevedano i salti
- Ad esempio, nel caso precedente si può assumere che il salto non venga effettuato, e poi correggersi in caso contrario

Esempio precedente



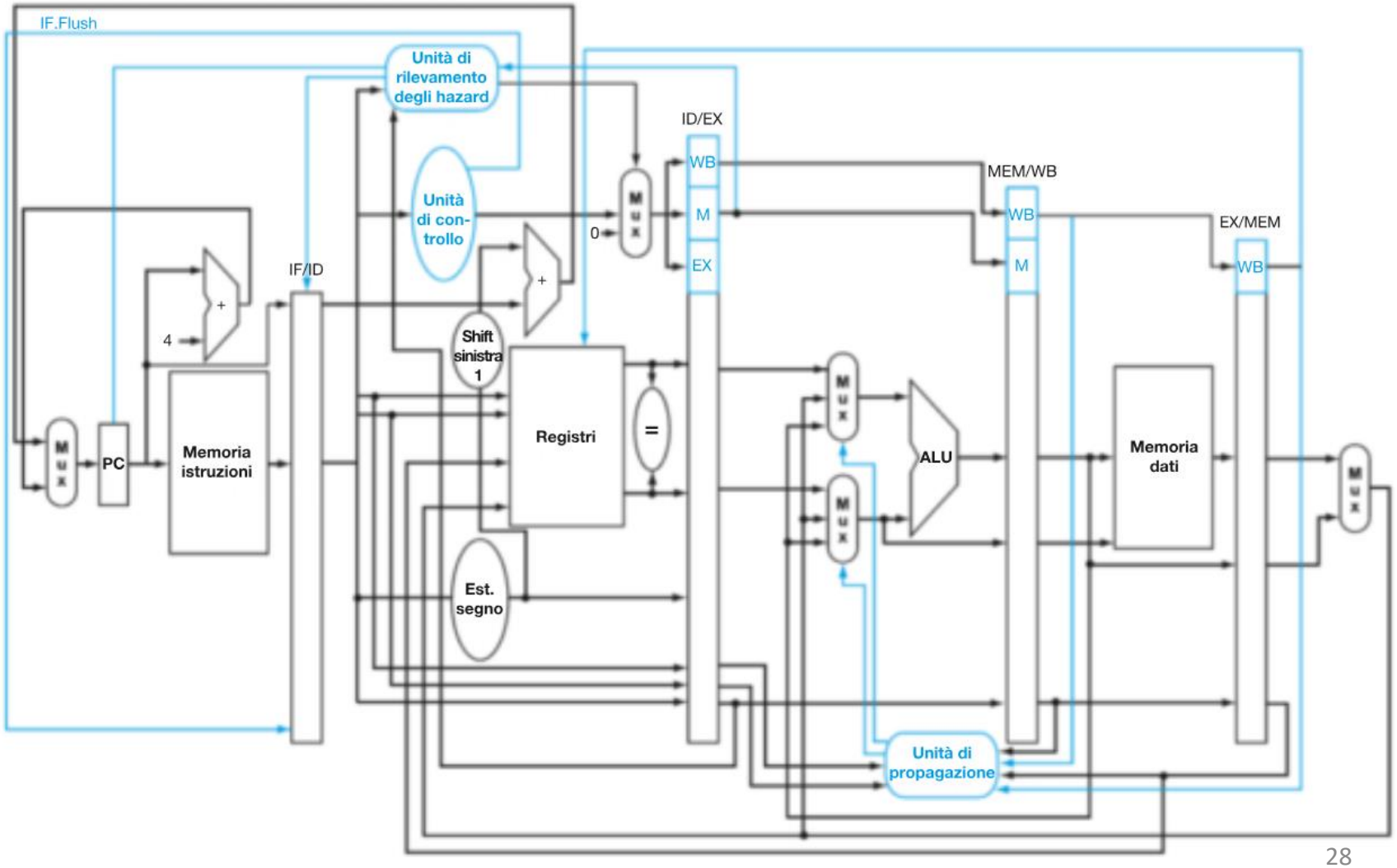
Si assume che il salto non venga effettuato



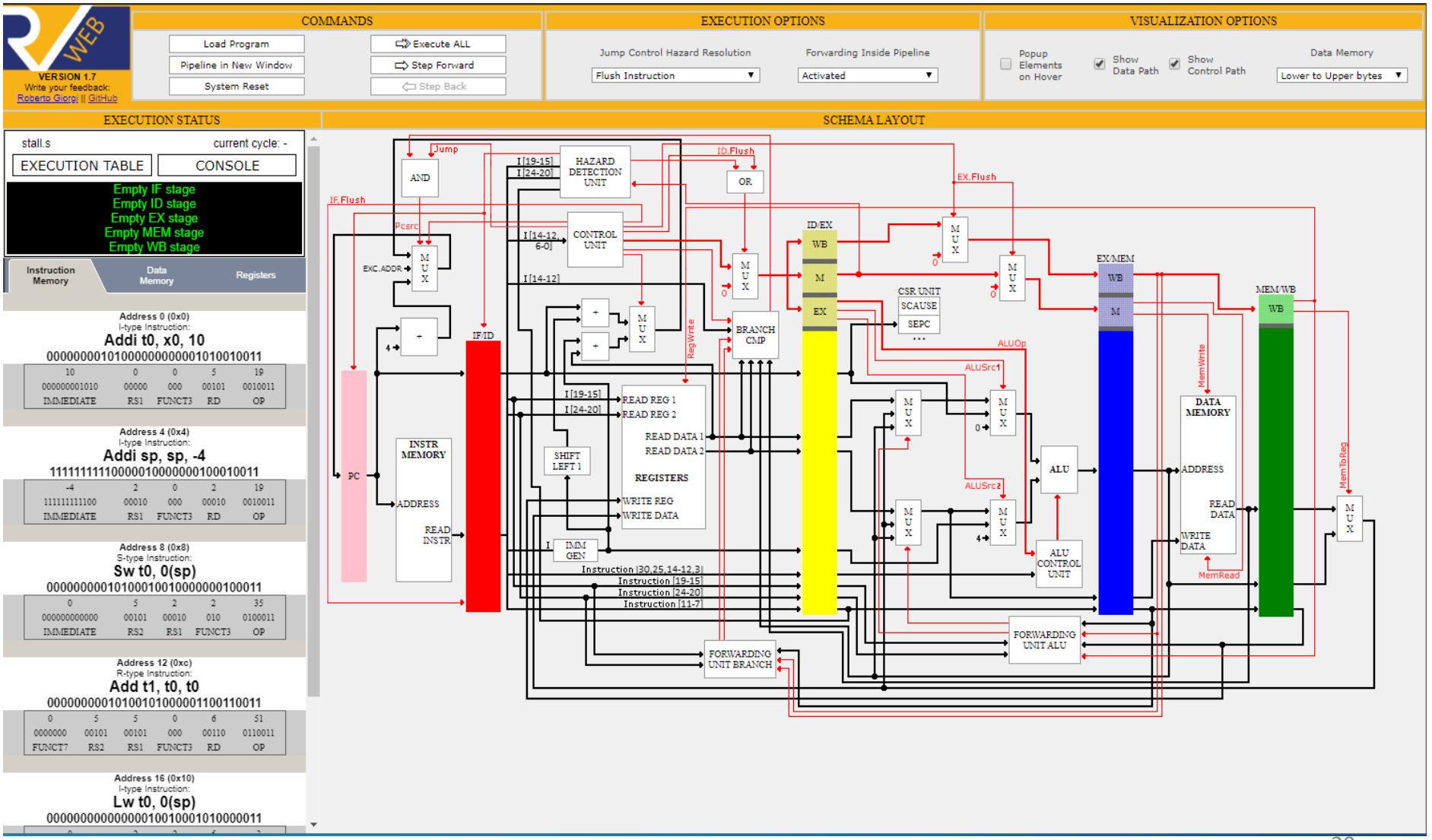
Circuiteria di branch prediction

- L’esempio che abbiamo appena visto non è particolarmente sofisticato (funziona bene solo se il branch non viene effettuato)
- Esistono circuiterie più sofisticate che permettono di memorizzare l’esito del branch precedente e assumere che il comportamento si mantenga coerente

Unità di elaborazione finale e relative unità di controllo



On Line RISC-V Simulator



http://x.dii.unisi.it:8098/~giorgi/WebRISC-V/index.php