Corso di Architettura degli Elaboratori e Laboratorio (M-Z)

Pipelining

Nino Cauli

UNIVERSITÀ
degli STUDI
di CATANIA

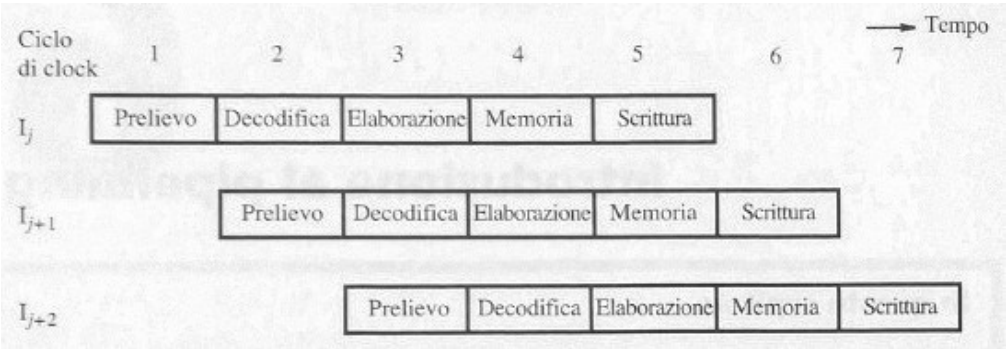
Dipartimento di Matematica e Informatica

Paralellizzare un architettura a 5 stadi

- Per svolgere compiti sempre più complessi in tempi ridotti abbiamo bisogno di aumentare le prestazioni di un calcolatore
- Essere in grado di parallelizzare l'esecuzione di istruzioni dividendola in stadi è la soluzione adottata dalle architetture RISC
- L'idea è usata da tempo è usata nelle catene di montaggio delle fabbriche
- Sebbene un'automobile potrebbe impiegare una giornata ad essere prodotta, la possibilità di lavorare su diverse auto in stadi differenti permette di produrne una ogni pochi minuti

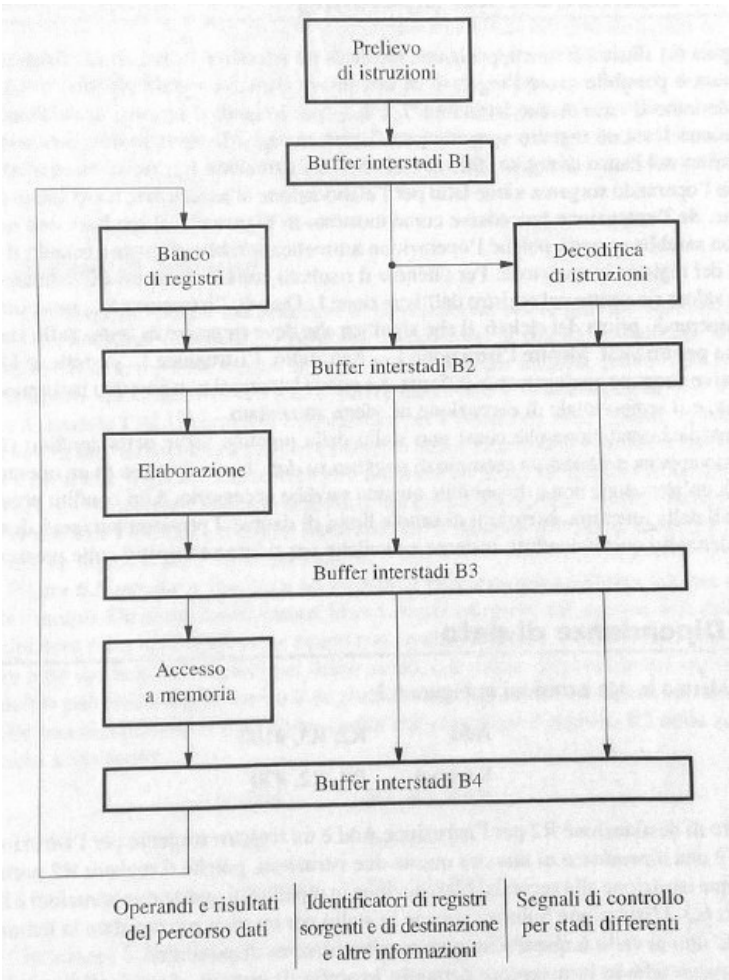
Pipelining

- Data la semplicità della codifica delle istruzioni, i processori RISC possono essere strutturati in un architettura a stadi
- Abbiamo visto una possibile suddivisione in 5 stadi: **Prelievo – Decodifica – Elaborazione – Memoria - Scrittura**
- Nel caso migliore (senza cache miss) si hanno 5 istruzioni eseguite in parallelo
- Sebbene un'istruzione impieghi 5 cicli di clock per essere eseguita, una volta che 5 istruzioni sono in esecuzione parallela, ogni istruzione viene ultimata ogni ciclo di clock



Pipelining

- Per gestire l'esecuzione in **pipeline** di più istruzioni, è necessario mantenere delle informazioni tra uno stadio e l'altro
- Queste informazioni vengono mantenuti nei **buffer interstadi**
- I buffer interstadi contengono
  - i rispettivi registri interstadio (RA, RB, PC_Temp, etc.)
  - Il registro IR (Per mantenere gli identificatori dei registri sorgente e destinazione)
  - Segnali di controllo per i vari stadi



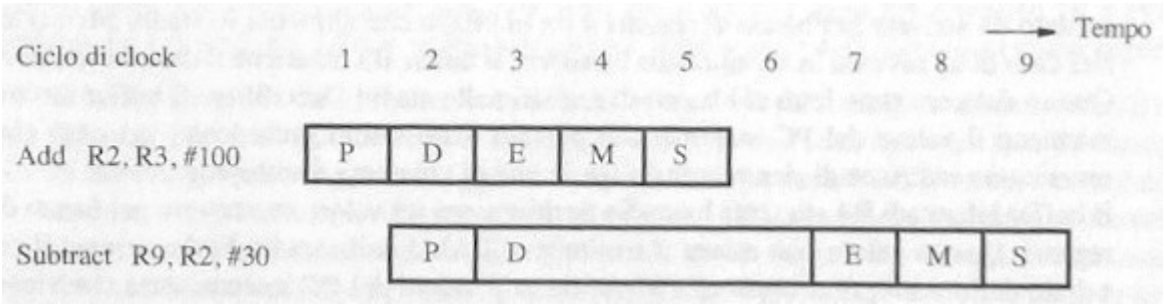
Problemi del Pipelining

- Non sempre è possibile avere la situazione ideale in cui si eseguono 5 istruzioni in parallelo
- Spesso avvengono dei **conflitti** che ritardano l'ingresso di nuove istruzioni nella pipeline
- I possibili tipi di conflitto sono:
  - **Dipendenze di dato**
  - **Ritardi nell'accesso alla memoria**
  - **Ritardi nei salti**
  - **Limiti di risorse**

Dipendenze di dato

- Una dipendenza di dato avviene quando un istruzione contiene un registro sorgente che non è stato ancora aggiornato dalle istruzioni precedenti

- Si considerino le seguenti istruzioni:



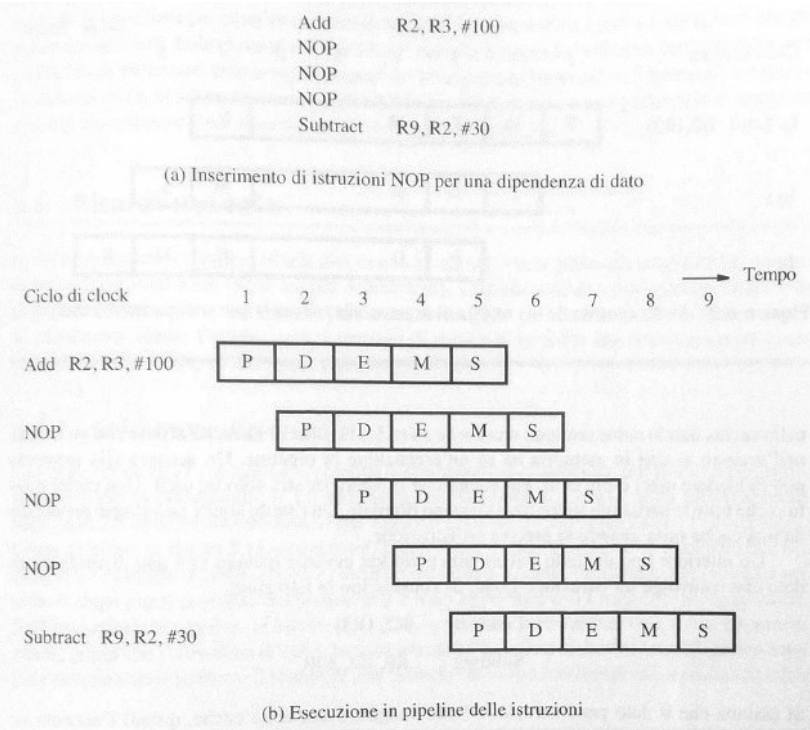
- L'istruzione Add aggiornerà il contenuto di R2 alla fine della sua fase di scrittura

- L'istruzione Subtract legge il contenuto di R2 nella sua fase di decodifica, ma l'istruzione Add è ancora alla fase di esecuzione

- L'istruzione Subtract dovrà rimanere in stallo finché R2 non sarà aggiornato (3 cicli di clock)

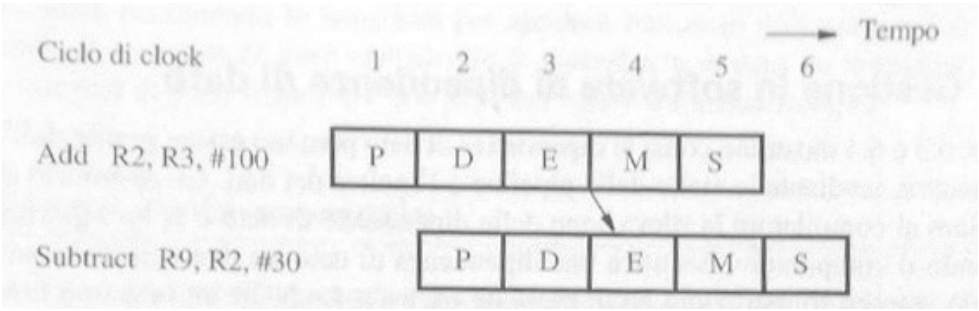
Istruzioni NOP e bolle

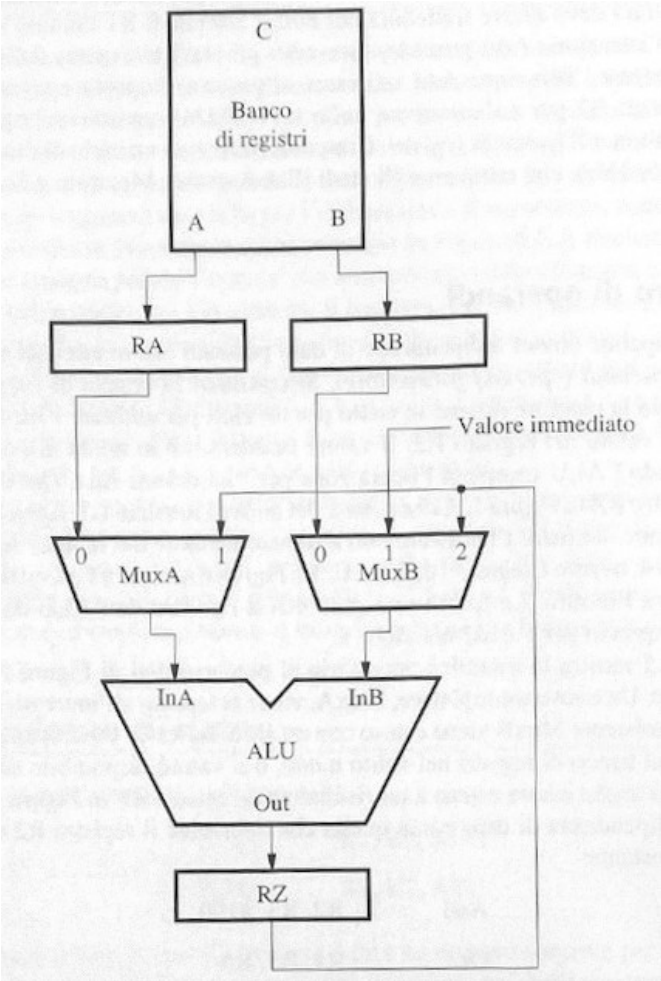
- Per porre in stallo un'istruzione si possono inserire delle **istruzioni nulle (NOP)** tra le due istruzioni in conflitto
- Ciascuna NOP crea un ciclo di inattività chiamato **bolla**
- Le istruzioni nulle possono essere generate via **software dal compilatore** o **via hardware attraverso dei circuiti di controllo più complessi**
- I compilatori ottimizzanti possono riordinare il codice in modo da spostare istruzioni utili nelle posizioni delle NOP



Inoltro degli operandi (forwarding)

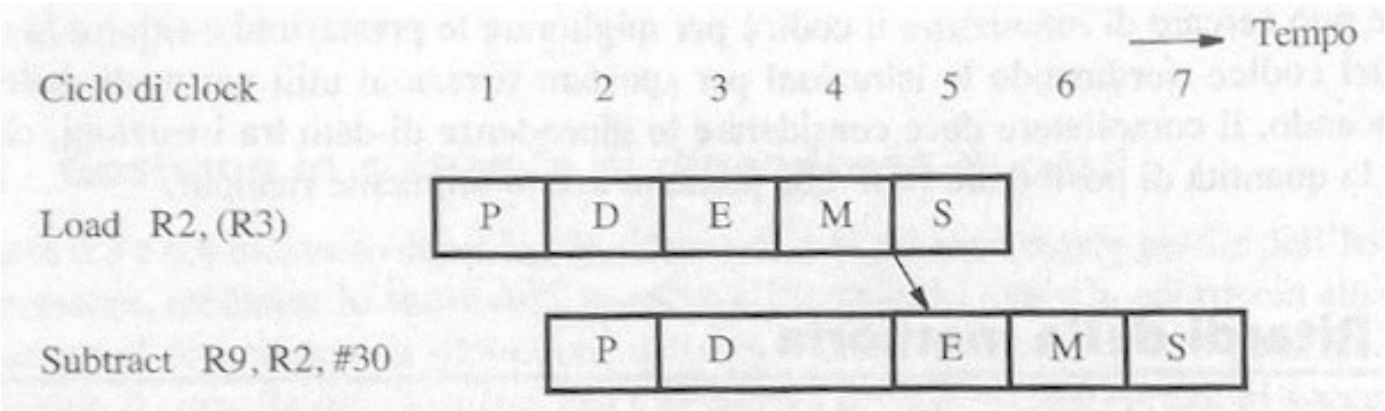
- Per risolvere il problema si può ricorrere all'inoltro degli operandi (**operand forwarding**)
- In questo caso registri interstadio successivi vengono inoltrati a stadi precedenti
- Per esempio, RZ può essere reso disponibile direttamente nello stadio di esecuzione attraverso dei multiplatori
- In questo modo si elimina lo stallo del caso precedente





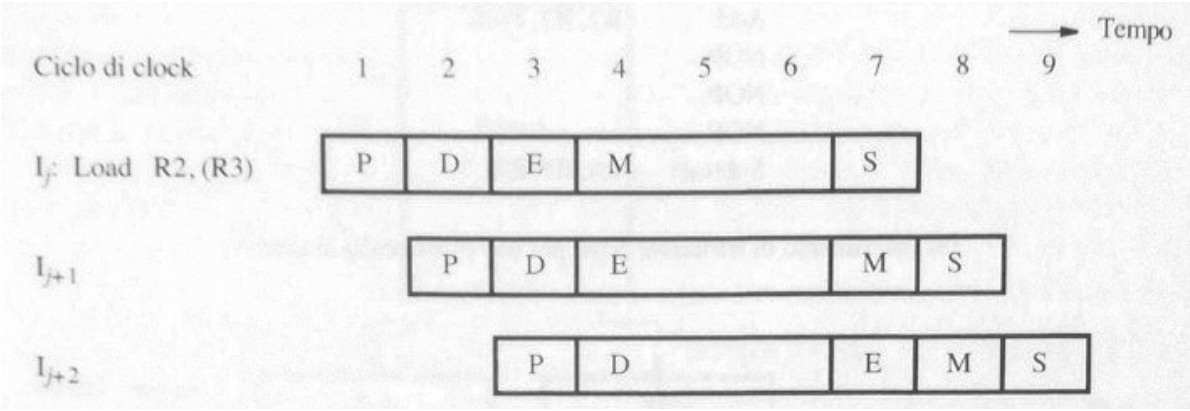
Inoltro degli operandi (forwarding)

- L'inoltro può avvenire anche con il registro RY
- In questo caso si risolvono ritardi in situazioni simili alla seguente:



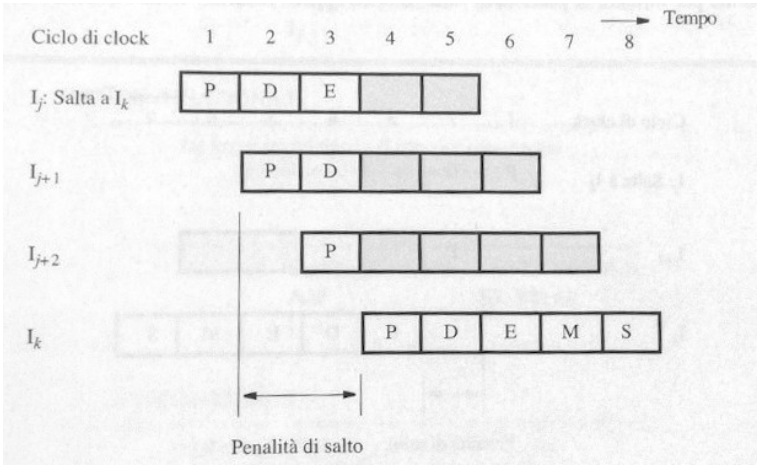
Ritardi della memoria

- Gli accessi alla memoria alcune volte necessitano di diversi cicli di clock
- Nei casi in cui un dato da leggere non sia nella cache (**cache miss**), si possono avere ritardi di 10 o più cicli di clock
- In questi casi tutte le istruzioni successive dovranno essere ritardate dello stesso numero di passi



Ritardi nei salti

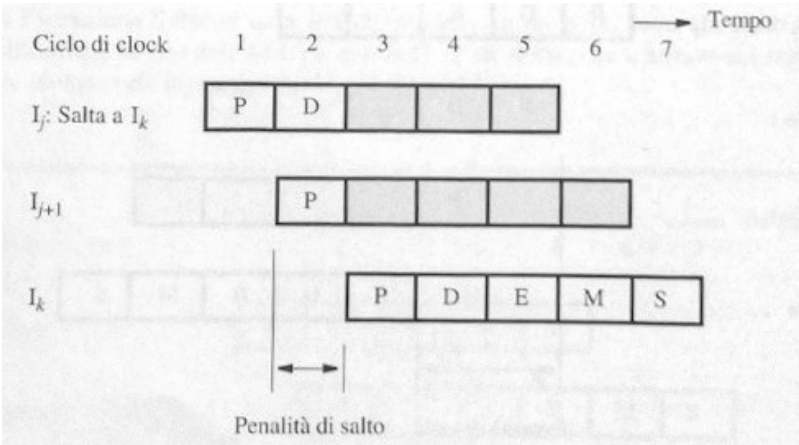
- Durante un'istruzione di salto incondizionato, l'indirizzo di destinazione viene caricato nel PC durante il passo 3
- Le due istruzioni successive entrate nella pipeline verranno quindi scartate
- Avviene quindi una penalità di salto di due cicli di clock
- Lo stesso problema avviene nei salti condizionati quando la condizione è vera



Anticipare la valutazione del salto

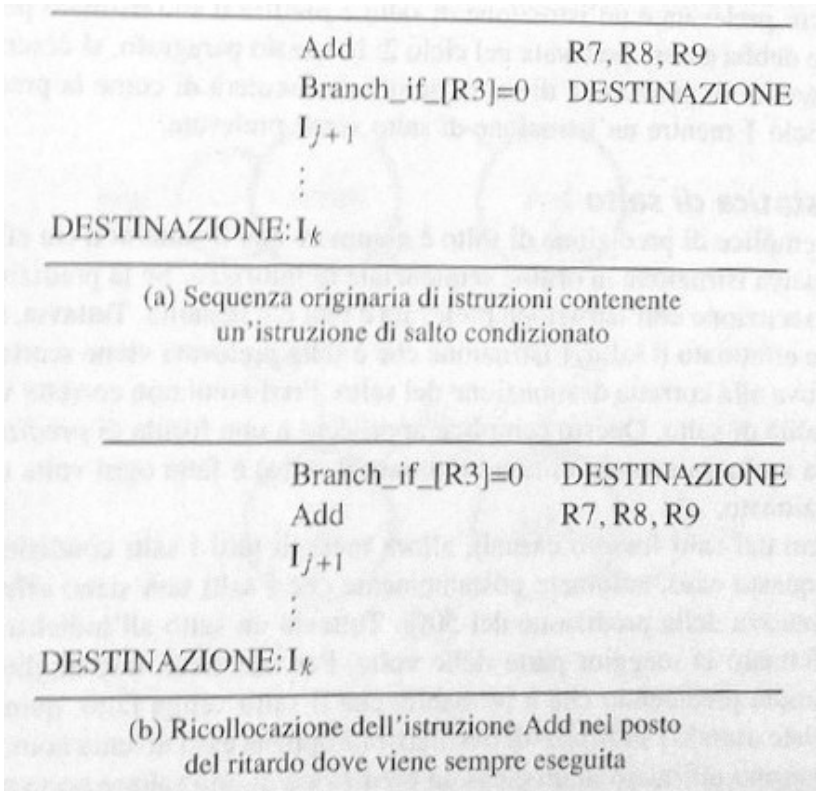
- Per ridurre la penalità di salto ad 1 cliclo di clock, si può modificare l'hardware visto finora in modo da **valutare ed eseguire il salto nello stadio di Decodifica**

- Si devono anticipare 2 operazioni:
  - Determinare l'indirizzo di destinazione (aggiungere un sommatore nel passo 2)
  - Valutare la condizione di salto (spostare il circuito comparatore nel passo 2, usando come ingressi direttamente le uscite del banco dei registri)



Salto differito

- Normalmente, se avviene un salto le istruzioni prelevate dopo l'istruzione di salto vengono scartate
- Nel **Salto Differito** le istruzioni successive salto vengono eseguite in ogni caso
- Il compilatore tenterà di riorganizzare le istruzioni in modo da posizionare dopo il salto istruzioni da eseguire in ogni caso
- Nel caso il compilatore non trovi istruzioni valide, inserisce delle NOP



	Add	R7, R8, R9
	Branch_if_[R3]=0	DESTINAZIONE
	I_{j+1}	
	⋮	
DESTINAZIONE:	I_k	

(a) Sequenza originaria di istruzioni contenente un'istruzione di salto condizionato

	Branch_if_[R3]=0	DESTINAZIONE
	Add	R7, R8, R9
	I_{j+1}	
	⋮	
DESTINAZIONE:	I_k	

(b) Ricollocazione dell'istruzione Add nel posto del ritardo dove viene sempre eseguita

Predizione di salti

- Per i salti non differiti, si ha sempre un ritardo di un'istruzione se la condizione di salto è vera
- Se fossimo in grado di predire il risultato del salto potremmo caricare l'indirizzo di destinazione direttamente al passo di prelievo
- Vediamo 2 modalità di predizione:
  - **Predizione statica**
  - **Predizione dinamica**

Predizione statica

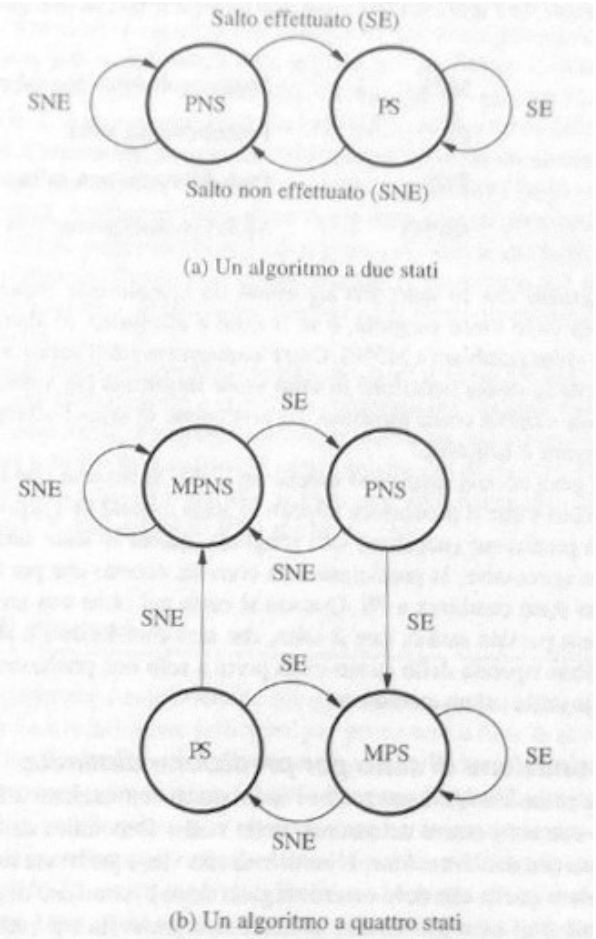
- Modo più semplice in cui viene sempre supposto che un salto condizionato non sia effettuato (o al contrario che venga sempre effettuato)
- Nel caso di salti all'indietro a fine ciclo è più probabile che il salto avvenga, viceversa nei salti in avanti che non avvenga
- Il processore può determinare il segno dello spiazzamento per decidere come comportarsi
- Un'altra soluzione è quella di includere nell'istruzione macchina un bit di predizione

Predizione dinamica

- Nella **predizione dinamica** si usa l'attuale comportamento di salto per influenzare la predizione

- La forma più semplice presenta 2 stati descritti dalla macchina a stati (a):
  - **PS**: Probabilmente salta
  - **PNS**: Probabilmente non salta

- Una forma più elaborata presenta 4 stati (macchina a stati (b)):
  - **MPS**: Molto probabilmente salta
  - **PS**: Probabilmente salta
  - **PNS**: Probabilmente non salta
  - **MPNS**: Molto probabilmente non salta



Buffer di destinazione di salto

- Per poter eseguire la predizione nello stadio di Fetch (il primo stadio) si ha bisogno di una memoria piccola e veloce chiamata **Buffer di destinazione di salto**
- Il buffer di destinazione di salto conterrà una tabella con tutte le istruzioni di salto del programma. Per ogni istruzione saranno salvate le seguenti informazioni:
  - **Indirizzo dell'istruzione di salto**
  - **Uno o due bit di stato per l'algoritmo di predizione**
  - **Indirizzo di destinazione del salto**
- Una volta prelevata un'istruzione, il suo indirizzo verrà cercato nella tabella
- Se l'istruzione prelevata è un salto si useranno le informazioni in tabella per aggiornare il PC
- Per grandi programmi la tabella non contiene tutte le istruzioni di salto, ma viene aggiornata man mano

Limiti di risorse

- La pipeline può andare in stallo quando una risorsa hardware è richiesta da più istruzioni contemporaneamente
- Esempio:
  - In ogni ciclo di clock si ha un accesso alla cache per prelevare la prossima istruzione
  - Quando un istruzione accede alla memoria (Load o Store) si avrà uno stallo
- Per evitare il problema si possono avere cache separate per istruzioni e dati