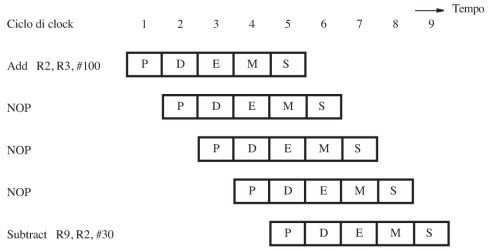
Gestione software di dipendenze di dati

- L'inserimento di NOP può essere demandato al compilatore
- Controllo hardware più semplice, ma nessun guadagno di throughput, e dipendenza del progetto del compilatore dalla microarchitettura (non solo dall'ISA)
- Un compilatore ottimizzante può effettuare un riordino delle istruzioni che sposti istruzioni utili in luogo delle NOP, se possibile

```
Add       R2, R3, #100
NOP
NOP
NOP
Subtract  R9, R2, #30
```

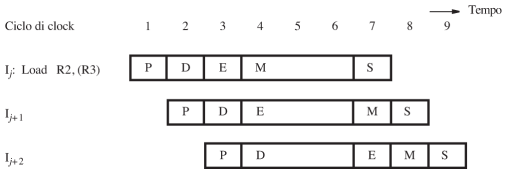
(a) Inserimento di istruzioni NOP per una dipendenza di dato

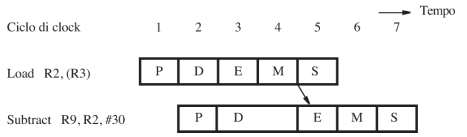


(b) Esecuzione in pipeline delle istruzioni

Ritardi della memoria (S. 6.5)

- La cache miss di un operando o di un'istruzione provoca uno stallo
- In `Load R2, (R3)` una cache miss del dato in memoria alla locazione `(R3)` provoca uno stallo dopo il ciclo 4, fino a che il dato sia caricato
- Inoltre la dipendenza da un dato letto dalla memoria, anche se trovato in cache, provoca uno stallo (bolla) se l'istruzione successiva usa il dato appena letto, poiché la scrittura di RY in B4 avviene al ciclo 4, e la scrittura in R2 al ciclo 5
- La `Subtract` deve restare in stallo per un ciclo per permettere di inoltrare RY all'ingresso dell'ALU nel ciclo 5



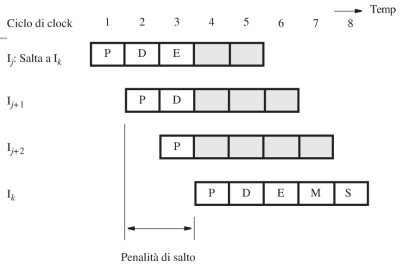


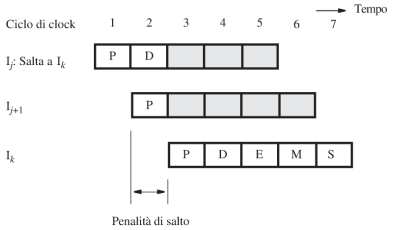
Ritardi nei salti (S. 6.6)

- Un'istruzione di salto cambia la sequenza di esecuzione, ma la destinazione di salto è nota solo dopo la sua esecuzione
  - Le istruzioni prelevate nel frattempo sono da scartare
- Penalità di salto: cicli di clock impiegati nel prelievo di istruzioni da scartare
- Sommario
  - salti incondizionati
  - salti condizionati
  - posto del ritardo nel salto
  - predizione di salto: statica, dinamica, buffer di destinazione

Salti incondizionati

- L'istruzione di salto è prelevata al ciclo 1, decodificata al ciclo 2 e la destinazione di salto, istruzione k, è calcolata al ciclo 3 (la destinazione di salto è la somma di PC e spiazzamento, ottenuta al ciclo 3)
- Quindi l'istruzione k è prelevata al ciclo 4, quando il PC è stato aggiornato
- Le istruzioni j+1 e j+2 devono essere scartate
- Il ritardo di due cicli è la **penalità di salto**
- Le istruzioni di salto si verificano frequentemente, sono circa il 20% del conteggio dinamico di istruzioni. Una penalità di salto di due cicli fa aumentare del 40% il tempo di esecuzione
- Per ridurre la penalità si anticipa il calcolo della destinazione di salto allo stadio di decodifica, ciclo 2
- Occorre aggiungere un sommatore, si collega lo spiazzamento nell'istruzione al sommatore, e allo stadio 2 si calcola la somma. Quindi si riduce di 1 la penalità di salto





Salti condizionati

- Istruzioni come `Branch_if_[R5]=[R6] CICLO`
- Con il percorso dati (capitolo 5), il risultato del confronto nel terzo passo determina se effettuare il salto
- Si anticipa allo stadio di decodifica (secondo passo) il confronto, insieme al calcolo della destinazione di salto. La decisione se saltare o meno si prende al secondo passo
- La penalità di salto è un solo ciclo per tutte le istruzioni di salto

Posto del ritardo del salto

- Al fine di ridurre la penalità di salto, dopo il prelievo dell'istruzione di salto si può prelevare un'istruzione che, indipendentemente dal salto, sia comunque utile eseguire. Tale istruzione deve essere un'istruzione che precede l'istruzione di salto, e che rispetta le dipendenze di dato. Se nessuna istruzione può essere inserita viene inserita una `NOP`
- La posizione che segue un'istruzione di salto è detta **posto del ritardo di salto** (branch delay slot)
- La pipeline deve essere organizzata in modo da eseguire comunque l'istruzione nel posto del ritardo, anziché eliminarla nel caso di salto
- Nell'esempio mostrato sotto, le istruzioni sono riordinate in modo che il posto del ritardo di salto viene occupato dalla `Add`. L'esecuzione procede come se il salto venisse dopo la `Add`, poiché il salto è effettuato un'istruzione dopo rispetto a dove appare: *salto differito*
- E' il compilatore che deve riordinare le istruzioni, i dati raccolti indicano che il compilatore può riempire un posto di ritardo di salto nel 70% dei casi

	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

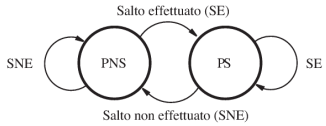
- La decisione di salto viene presa al ciclo 2, e l'istruzione che segue l'istruzione di salto potrebbe essere scartata dalla pipeline (quando si salta)
- Determinando al ciclo 1: se l'istruzione prelevata è di salto, e se il salto avviene, si potrebbe ridurre la penalità di salto
- Occorrerebbe predire il risultato dell'istruzione di salto
  - Predizione statica
  - Predizione dinamica
  - Buffer di destinazione di salto per predizione dinamica

Predizione statica di salto

- La più semplice forma di predizione statica è la predizione costante
- Es. si predice che il salto avvenga, la probabilità di indovinare è del 50% (apparentemente), quindi nel 50% dei casi non si ha penalità, altrimenti si ha penalità
- Per i salti condizionati per i cicli, si ha maggiore accuratezza se si predice
  - Si salta per i salti all'indietro, ovvero per i cicli in cui la condizione è messa alla fine, spesso si torna a fare un'altra passata del ciclo
  - Non si salta per i salti in avanti, ovvero per i cicli in cui la condizione è all'inizio, spesso si entra nel ciclo per fare almeno una passata
- La predizione è fatta in base al segno dello spiazzamento di salto, oppure il compilatore può inserire un bit nell'istruzione di salto per codificare la predizione

Predizione dinamica di salto (due stati)

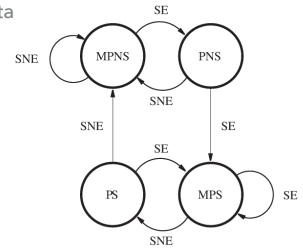
- Per migliorare l'accuratezza della predizione, si usa l'attuale comportamento. Nella forma più semplice: il processore assume che la prossima volta che l'istruzione è eseguita la decisione sul salto sia la stessa dell'ultima volta
- Macchina a due stati: PS, Probabilmente Salta; e PNS, Probabilmente Non Salta
- Richiede un solo bit per rappresentare la storia dell'istruzione
- Funziona bene all'interno di un ciclo, ma la predizione è sbagliata per l'ultima passata e per la prima passata, se si incontra nuovamente l'istruzione
- Si assuma che lo stato sia inizialmente PNS, si aggiorna a PS se salta e rimane in PS fino all'ultima passata



(a) Un algoritmo a due stati

Predizione dinamica di salto (quattro stati)

- Per migliorare l'accuratezza della predizione, si usa la storia dell'esecuzione. Si hanno quattro stati: MPS, Molto Probabilmente Salta; PS, Probabilmente Salta; PNS, Probabilmente Non Salta, MPNS, Molto Probabilmente Non Salta
- Si assuma che lo stato sia inizialmente PNS, se l'istruzione di salto viene eseguita lo stato si aggiorna in MPS, altrimenti in MPNS
- Per un ciclo, che ha la condizione alla fine, si inizia con lo stato PNS e si cambia in MPS alla prima passata, la previsione sarà corretta per le altre passate
- Per l'ultima passata la previsione è sbagliata e lo stato viene aggiornato a PS, non avendo effettuato il salto
- Quando si incontra il ciclo nuovamente, si parte da PS e la predizione sarà corretta se si fa almeno una passata



(b) Un algoritmo a quattro stati

Buffer di destinazione di salto

- La destinazione del salto, ovvero l'istruzione da prelevare, è conosciuta allo stadio 2 Decodifica
- Per evitare la penalità di salto si dovrebbe anticipare allo stadio 1 di Prelievo il riconoscimento di un'istruzione di salto e la predizione. Come si può prelevare, riconoscere l'istruzione e predire se si salta nello stesso ciclo?
- Si usa un buffer di destinazione di salto (branch target buffer): una piccola tabella, sulla quale viene cercato il valore di PC dell'istruzione appena prelevata
- La tabella contiene record che hanno: indirizzo dell'istruzione di salto, indirizzo di destinazione di salto, bit di predizione (uno o più bit)
- Ogni volta che si preleva un'istruzione, si controlla in tabella se l'istruzione è un salto, quindi si usa il bit di predizione di salto, e nel ciclo 2 si usa il risultato della predizione per prelevare la prossima istruzione
- Tabelle piccole per avere ricerche veloci, tipicamente la tabella ha 1024 elementi

Limiti di risorse (S. 6.7)

- Il parallelismo nella pipeline si ottiene quando istruzioni diverse accedono a risorse separate
- Se due istruzioni accedono alla stessa risorsa nello stesso ciclo di clock, un'istruzione deve andare in stallo per permettere all'altra istruzione di andar avanti
- Caso tipico: gli stadi di Prelievo e di Memoria accedono entrambi alla cache
- Ogni ciclo di clock ha un'istruzione nello stadio di Prelievo, quindi si avrebbe uno stallo a ogni accesso di dato in memoria (istruzioni `Load` e `Store`). Se il 25% delle istruzioni eseguite accedono alla memoria, gli stalli aumentano il tempo di esecuzione del 25%
- L'uso di *cache separate per istruzioni e dati* permette di procedere simultaneamente senza stalli