Il pipelining: criticità sul controllo ed eccezioni

Architetture Avanzate dei Calcolatori

Valeria Cardellini

Criticità sul controllo

- Criticità sul controllo: tentativo di prendere una decisione sulla prossima istruzione da caricare prima che sia stata valutata la condizione di salto
- Determinata dal pipelining di:
  - Istruzioni di salto condizionato
    - Criticità anche detta *conflitto di salto condizionato*
    - Caso più frequente
  - Altre istruzioni che modificano il PC
    - Salti incondizionati, …
- Statisticamente meno frequente della criticità sui dati
- Degrada le prestazioni rispetto allo speedup ideale ottenibile con il pipelining poiché può rendere necessario lo stallo della pipeline

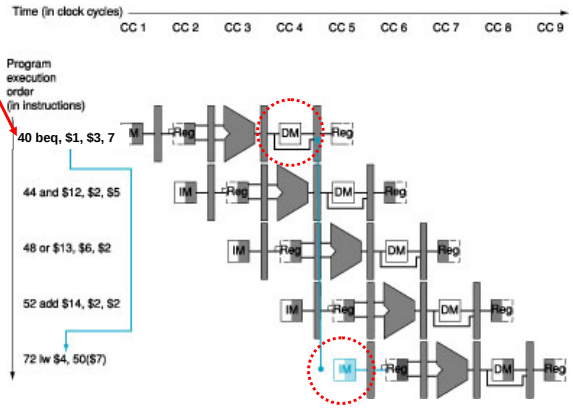
Salto condizionato nel MIPS

- Implementazione vista finora dell'istruzione beq:
  - La decisione sul salto avviene nello stadio MEM

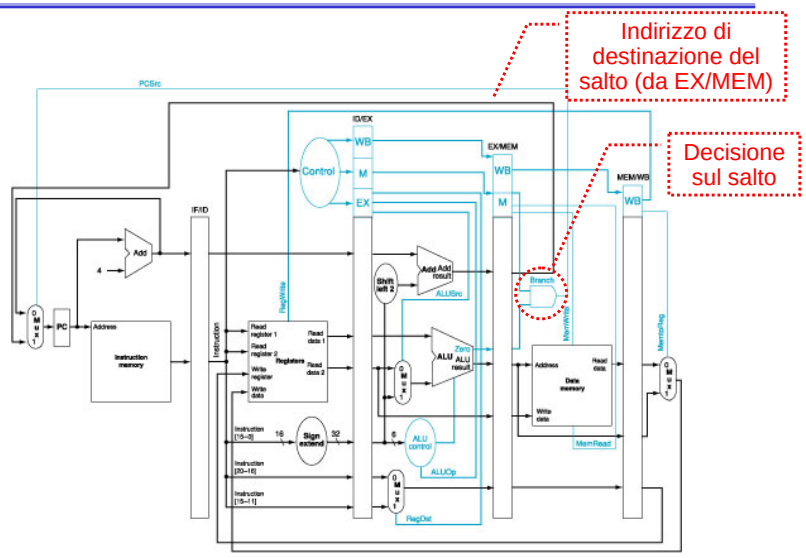
Salto eseguito: dopo l'esecuzione di beq, salto all'istruzione di indirizzo 72 (40+4+7*4)

Salto eseguito (*taken*): il salto causa il trasferimento del controllo alla destinazione del salto

Salto non eseguito (*not taken*): al salto seguono le istruzioni successive del programma (*fall-through*)



Salto condizionato nel MIPS (2)

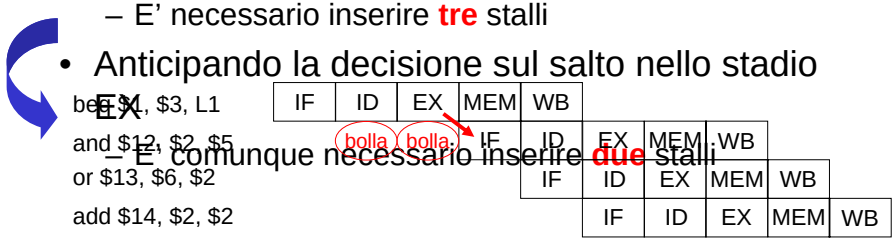


Soluzioni possibili

- Stallo della pipeline
- Ipotizzare che il salto condizionato non sia eseguito (branch not taken)
- Ridurre i ritardi associati ai salti
- Predizione dei salti

Stallo della pipeline

- Prima soluzione: si blocca la pipeline finché non viene presa la decisione sul salto e poi si carica la pipeline con l'istruzione corretta (**stalling until resolution**)
- Decisione sul salto nello stadio MEM
  - E' necessario inserire **tre** stalli
- Anticipando la decisione sul salto nello stadio EX
  - E' comunque necessario inserire **due** stalli



Salto non eseguito

- Seconda soluzione: si assume che il salto non sia eseguito (**branch not taken**)
  - Si continuano a caricare nella pipeline le istruzioni successive a quella di salto condizionato
- Se il salto non è effettivamente eseguito
  - Non c'è nessuna penalizzazione
- Se invece il salto è eseguito (la predizione è errata)
  - Si scartano le istruzioni che sono state nel frattempo caricate nella pipeline
    - Nell'esempio del lucido 2, 3 istruzioni (and, or, add) da scartare
  - Si puliscono gli stadi IF, ID, EX
    - *Flushing* (annullamento) delle istruzioni
  - La pipeline viene caricata a partire dall'istruzione di destinazione del salto (nell'esempio del lucido 2 l'istruzione lw)
  - Non è stato modificato nessun registro perché nessuna istruzione successiva al salto ha raggiunto lo stadio WB
  - Riduzione del throughput

Riduzione del costo

- Terza soluzione: si anticipa la decisione sul salto ad uno stadio precedente a MEM
- Occorre anticipare tre azioni
  - Calcolare l'indirizzo di salto
  - Valutare la decisione del salto
    - Per beq e bne occorre confrontare i registri
  - Aggiornare il PC
- Occorre aggiungere delle risorse hardware

1. Per il calcolo dell'indirizzo di salto
- Se l'indirizzo di salto è calcolato
  - Alla fine dello stadio EX: due stalli
  - Alla fine dello stadio ID: uno stallo
- Si sposta l'addizionatore per l'indirizzo di salto nello stadio ID

Riduzione del costo del salto (2)

2. Per confrontare i registri
   - Nel caso di beq: l'**unità di confronto** posta nello stadio ID esegue lo XOR bit a bit dei due registri e poi l'OR del risultato dello XOR
- Occorre gestire la propagazione all'ingresso dell'unità di confronto
  - Gli operandi sorgente possono provenire dai registri di pipeline EX/MEM o MEM/WB
- Può essere necessario uno stallo per risolvere una criticità sui dati
  - Es.: l'istruzione immediatamente precedente beq produce uno dei due operandi confrontati

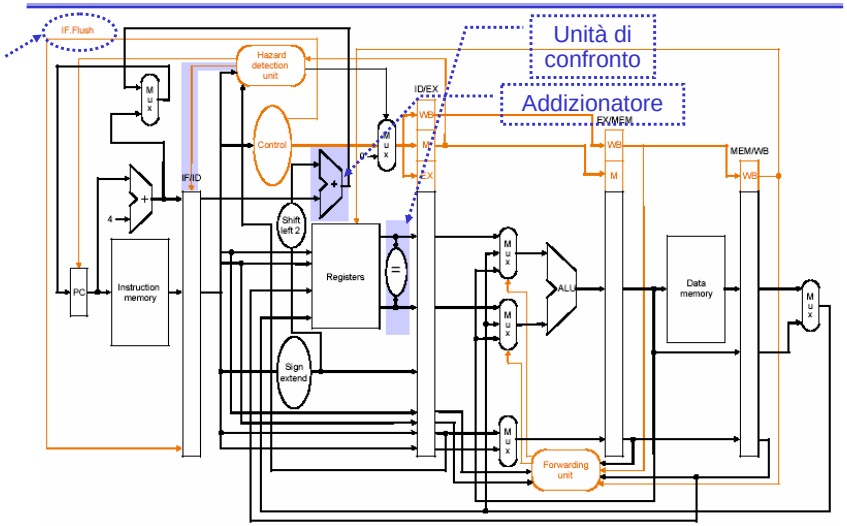
```
addi $6, $6, 4
beq $6, $7, Loop
```


Riduzione del costo del salto (3)

3. Per aggiornare il PC
   - Si sposta nello stadio IF la porta AND con ingressi il segnale di controllo Branch e l'uscita dell'unità di confronto
   - Se il salto è eseguito, il PC è scritto con l'indirizzo di destinazione del salto al termine del ciclo di clock dello stadio ID di beq

- Anticipando la decisione sul salto allo stadio ID si riducono i ritardi associati ai salti (*branch penalty*)
  - Occorre inserire un solo stallo *dopo* ogni salto
  - Oppure svuotare la pipeline di una sola istruzione

Modifica dell'unità di elaborazione



IF.Flush: segnale per azzerare i campi dell'istruzione nel registro di pipeline IF/ID

Esempio

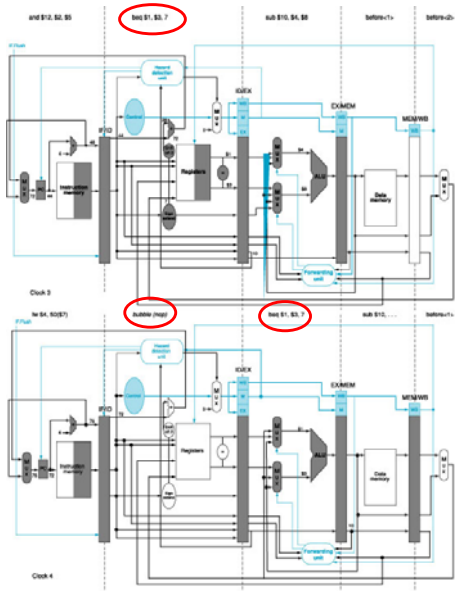
- Consideriamo la sequenza di istruzioni MIPS

```
36    sub $10, $4, $8
40    beq $1, $3, 7     # 40+4+4*7=72
44    and $12, $2, $5
48    or $13, $2, $6
52    add $14, $4, $2
56    slt $15, $6, $7
      …
72    lw $4, 50($7)
```

- Assumiamo che:
  - la pipeline sia ottimizzata per salti non eseguiti
  - l'esecuzione del salto sia stata anticipata allo stadio ID

Esempio: cicli di clock 3 e 4

- L'istruzione and entra nella pipeline (branch not taken)
- Nello stadio ID si determina che il salto deve essere eseguito (predizione errata)
- Viene selezionato 72 come prossimo valore del PC
- Viene azzerato il caricamento di and nella pipeline (segnale IF.Flush)
- L'istruzione lw (corrispondente alla destinazione del salto) viene caricata nella pipeline



Tecniche per la predizione dei salti

- In generale, il problema della predizione dei salti peggiora al crescere della profondità della pipeline
  - Aumenta il costo di una predizione errata (il salto è risolto 4 o più stadi dopo l'ID)
- Obiettivo delle tecniche di predizione: predire quanto più presto possibile il risultato del salto
- Le prestazioni di una tecnica di predizione dipendono da:
  - *Accuratezza della predizione*: misurata in termini di percentuale di predizioni errate
  - *Costo di una predizione errata*: misurato in termini di tempo sprecato per eseguire istruzioni inutili
- Occorre considerare anche la frequenza dei salti
  - L'importanza di una tecnica di predizione accurata è maggiore in programmi che hanno un'elevata frequenza di salti

Tecniche per la predizione dei salti (2)

- Tecniche ***statiche*** per la predizione dei salti
  - Predizione fissata per ogni salto *a tempo di compilazione* e per tutta l'esecuzione del programma
  - Esempio: salto non eseguito (soluzione già considerata)
  - Soluzioni semplici, adatte per pipeline con pochi stadi
  - Con pipeline di profondità maggiore, il ritardo associato al salto diventa considerevole
- Tecniche ***dinamiche*** per la predizione dei salti
  - Predizione effettuata *a tempo di esecuzione*, usando informazioni sull'esecuzione di salti passati
  - Implementate in hardware
- In entrambi i casi: attenzione a non cambiare lo stato del processore finché non è noto il risultato del salto!

Tecniche di predizione statiche

- Usate prevalentemente in processori in cui il comportamento del salto può essere predetto correttamente con buona probabilità *a tempo di compilazione*
- Utilizzate anche come supporto a tecniche di predizione dinamiche
- Tipologie di tecniche statiche
  - **Salto non eseguito** (Branch Always Not Taken)
    - Già considerata
  - **Salto eseguito** (Branch Always Taken)
  - **Salto all'indietro eseguito in avanti non eseguito** (Backward Taken Forward Not Taken)
  - **Predizione guidata da profili** (profile-driven prediction)
  - **Salto ritardato** (delayed branch)

Salto eseguito (Branch Always Taken)

- Approccio alternativo al Branch Always Not Taken
- Non appena l'istruzione di salto viene decodificata e viene calcolato l'indirizzo di salto, si assume che il salto sia sempre eseguito
  - Si carica nella pipeline l'istruzione corrispondente alla destinazione del salto
- Schema applicabile in pipeline in cui l'indirizzo di salto è noto prima del risultato del confronto
- Nel MIPS l'indirizzo di salto non è noto prima del risultato del confronto
  - Non c'è vantaggio nell'adottare questo approccio nel MIPS

Backward Taken Forward Not Taken Predizione guidata da profili

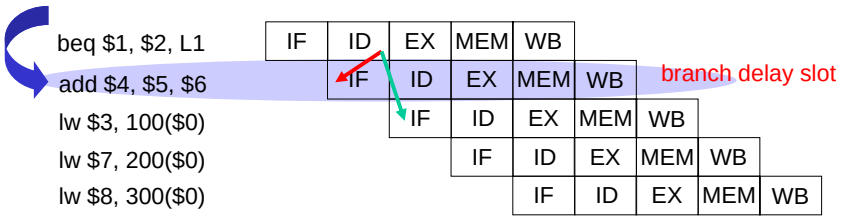
- Backward Taken Forward Not Taken
  - La predizione dipende dalla direzione del salto
  - Se il salto è all'indietro (*backward*) la tecnica predice che il salto verrà eseguito
    - Es.: il salto al termine di un ciclo per tornare indietro alla successiva iterazione del ciclo
  - Se il salto è in avanti (*forward*) la tecnica predice che il salto non verrà eseguito
- Predizione guidata da profili (profile-driven prediction)
  - La predizione è basata su informazioni relative all'esito dei salti acquisite in esecuzioni precedenti del programma

Salto ritardato

- Il compilatore seleziona un'istruzione da porre nel *branch delay slot*, riorganizzando il codice
  - Branch delay slot: la posizione dopo l'istruzione di salto
- L'istruzione nel branch delay slot viene eseguita sempre, indipendentemente dall'esito del salto
  - Non si devono scaricare dalla pipeline le istruzioni che seguono il salto!
- Se assumiamo di avere un *branch delay* di un ciclo, abbiamo solo un branch delay slot
  - E' il caso del processore MIPS
  - E' possibile avere un branch delay maggiore di un ciclo, ma tutti i processori che adottano questa soluzione hanno un solo delay slot

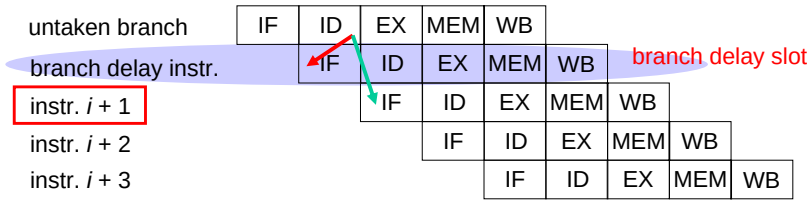
Salto ritardato (2)

- Il compilatore MIPS cerca di posizionare dopo il salto un'istruzione indipendente dall'esito del salto
- Esempio
  - Per il branch delay slot viene selezionata un'istruzione add precedente a beq (senza effetti su beq)

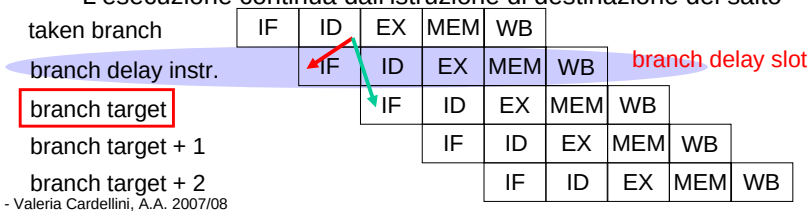


Salto ritardato (3)

- Se il salto non è eseguito
  - L'esecuzione continua con l'istruzione successiva al salto

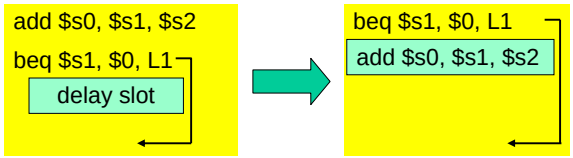


- Se il salto è eseguito
  - L'esecuzione continua dall'istruzione di destinazione del salto



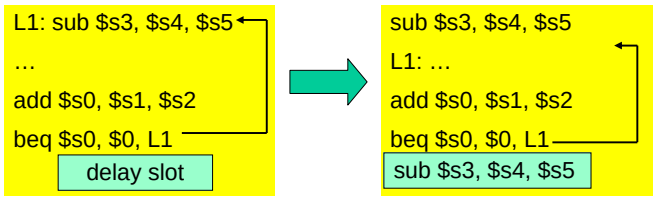
Selezione del delay slot

- Compito del compilatore è selezionare un'istruzione da porre nel delay slot che sia valida ed utile
- Tre strategie per selezionare l'istruzione da spostare
  - Da prima del salto
  - Dalla destinazione del salto
  - Tra salto e destinazione del salto
- Strategia 1: da prima del salto
  - Nel delay slot si posiziona un'istruzione indipendente proveniente dalla parte di codice prima del salto
  - L'istruzione nel delay slot viene *sempre* eseguita
  - E' la strategia migliore



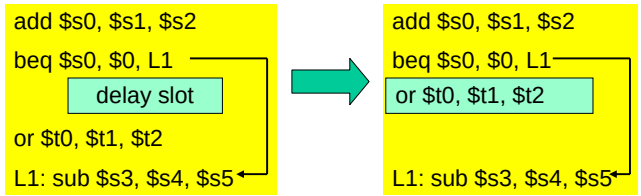
Selezione del delay slot (2)

- Strategia 2: dalla destinazione del salto
  - Non è possibile scegliere un'istruzione prima del salto
  - Nel delay slot si posiziona l'istruzione di destinazione del salto
  - Questa istruzione viene generalmente copiata perché può essere raggiunta anche tramite un altro percorso
  - Strategia utile per salti all'indietro (elevata probabilità che il salto sia eseguito)



Selezione del delay slot (3)

- Tra salto e destinazione del salto
  - Non è possibile scegliere un'istruzione prima del salto
  - Nel delay slot si posiziona un'istruzione proveniente dalla sequenza tra salto e destinazione del salto
  - Strategia utile per salti in avanti



- Nelle ultime due strategie deve essere possibile eseguire l'istruzione spostata quando il salto va nella direzione inattesa

Tecniche di predizione dinamiche

- Idea: usare informazioni sull'esito di *salti passati* per predire il futuro
- Uso di hardware per effettuare la predizione del salto
  - La predizione dipende dall'esito del salto *a tempo d'esecuzione* e cambia se il salto modifica il proprio comportamento durante l'esecuzione
- Esaminiamo inizialmente uno schema semplice, per poi considerare approcci che incrementano l'accuratezza della predizione

Meccanismi per la predizione dinamica

La predizione dinamica è basata su due meccanismi:

1. Predizione dell'esito del salto
   - Si predice la direzione del salto: *taken* o *not taken*
   - Si utilizza la tabella di storia del salto (***Branch Prediction Buffer*** o BPB, anche detto ***Branch History Table*** o BHT)
2. Predizione della destinazione del salto
   - Si predice l'indirizzo di destinazione in caso di salto eseguito
   - Si utilizza il buffer di destinazione del salto (***Branch***

Predizione dell'esito del salto

- Analizziamo le seguenti tecniche:
  - Predizione ad 1 bit *(1-bit BHT)*
  - Predizione a 2 bit *(2-bit BHT)*
  - Predizione ad n bit *(n-bit BHT)*
  - Predittore correlato
  - Predittore di salto a torneo *(tournament predictor)*

1-bit Branch History Table

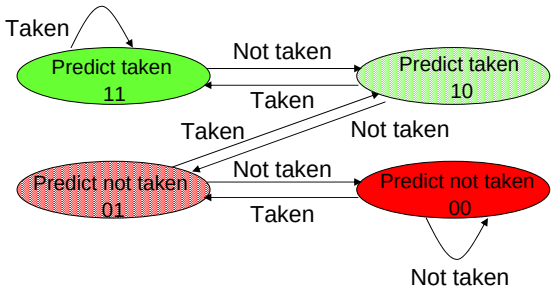
- Nella forma più semplice di predizione dinamica, si tiene conto dell'esito dell'ultima esecuzione del branch
- La tabella sulla storia dei salti (**1-bit BHT**):
  - Contiene righe da 1 bit, ciascuna della quali indica se recentemente il salto corrispondente è stato effettuato o meno
  - Tabella indicizzata tramite i bit meno significativi dell'indirizzo dell'istruzione di salto
- Predizione: è un suggerimento che si assume sia corretto
  - Se si scopre che la predizione è sbagliata, il bit di predizione è invertito e riscritto nella BHT; si svuota la pipeline e si esegue la sequenza di istruzioni corretta
- Il bit di predizione può essere stato scritto da un'altra istruzione di salto, il cui indirizzo ha gli stessi bit usati per indicizzare la tabella rispetto al salto che si sta considerando

Accuratezza della 1-bit Branch History Table

- Si ha una predizione errata quando
  - Per quel salto la predizione è sbagliata
  - La stessa riga della tabella è acceduta da due salti diversi e la storia precedente (il bit) si riferisce all'altro salto
    - Soluzioni possibili: ampliare il numero di righe della BHT per ridurre il numero di interferenze o usare una funzione di hashing per l'indicizzazione (adottata nel predittore GShare)
- Svantaggio della BHT ad 1 bit
  - In un loop, anche se il salto viene quasi sempre effettuato e solo una volta non viene effettuato (alla fine del loop), si ha un doppio errore
    - Alla fine dell'iterazione del loop e all'inizio dell'iterazione successiva dello stesso loop
  - Es.: salto di un loop effettuato 9 volte e non effettuato 1 volta
    - L'accuratezza della predizione è pari all'80% (invece di 90%)

2-bit Branch History Table

- Per aumentare l'accuratezza si usa una tabella a 2 bit
- La predizione deve essere errata per due volte consecutive prima che venga invertita
- I due bit sono usati per codificare i quattro stati di una macchina a stati finiti



- Contatore a 2 bit in saturazione (valore min 00, valore max 11)
- Contatore incrementato se branch taken
- Contatore decrementato se branch not taken

n-bit Branch History Table

- La BHT a 2 bit può essere generalizzata ad n bit usando un contatore ad n bit a saturazione
  - Valori del contatore compresi tra 0 e 2^n-1
  - Quando il valore del contatore è maggiore o uguale a 2^{n-1}, si predice che il salto sia eseguito (taken)
- Analogamente alla BHT ad 2 bit, il contatore ad n bit a saturazione viene incrementato se il salto viene eseguito (taken), decrementato altrimenti
- Studi condotti sui predittori ad n bit hanno dimostrato che i predittori a 2 bit si comportano quasi sempre bene
  - In generale, si usano predittori a 2 bit

Correlazione tra i salti

- La BHT a 2 bit si basa soltanto sul comportamento recente di un salto per predire il comportamento futuro di quel salto (solo informazione *locale*)
- Idea: il comportamento di salti recenti è *correlato*: il comportamento recente di *altri* salti può influenzare il comportamento del salto corrente (anche informazione *globale*)
- Esempio:

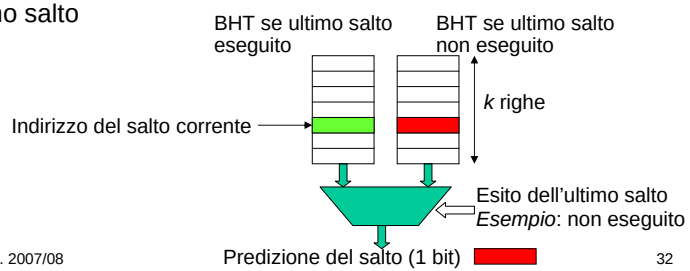
```
if (a == 2) a=0;   // bb1
if (b == 2) b=0;   // bb2
if (a != b) { };   // bb3
```

```
        addi $s3, $s1, -2
        bne $s3, $zero, L1      # bb1
        add $s1, $zero, $zero
L1:     addi $s3, $s2, -2
        bne $s3, $zero, L2      # bb2
        add $s2, $zero, $zero
L2:     sub $s3, $s1, $s2
        beq $s3, $zero, L3      # bb3
```

Il salto bb3 è correlato ai salti precedenti bb1 e bb2: se bb1 e bb2 sono eseguiti, allora bb3 non è eseguito

Predittore correlato

- Un predittore correlato combina il comportamento *locale* di un salto con una informazione *globale* relativa al comportamento recente di *altri* salti
  - Anche detto *predittore a due livelli*
  - Al primo livello si identifica la situazione in cui ci si trova
  - Al secondo livello si effettua la predizione vera e proprio
- Esempio: **predittore a due livelli (1,1)**
  - Si memorizzano gli esiti degli ultimi *k* salti (BHT con *k* righe)
  - Si seleziona uno dei due predittori ad 1 bit in base all'esito dell'ultimo salto

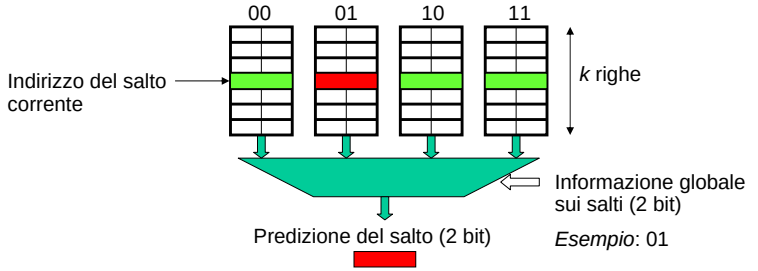


Predittore correlato (2)

- In generale si usa un predittore a due livelli **(*m*,*n*)**
  - Si memorizzano gli esiti degli ultimi *m* salti per scegliere tra 2^m BHT possibili, ciascuna delle quali è un predittore ad *n* bit
  - Richiede $2^m \times n \times k$ bit per la memorizzazione

Esempio: **predittore a due livelli (2,2)**

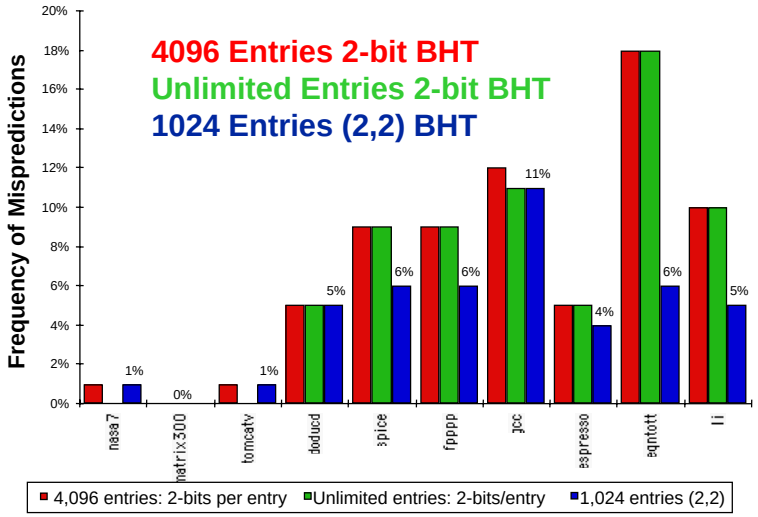
- Utilizza quattro BHT a 2 bit
- Si utilizzano due bit di storia globale per scegliere tra quattro possibili predittori per ciascun indirizzo di salto



Accuratezza dei predittori correlati

- Un predittore 2-bit BHT senza storia globale è un predittore a due livelli (0,2)
- Confronto mediante il benchmark SPEC89 tra un predittore (2,2) avente 1K di righe ed un semplice predittore 2-bit BHT avente 4K di righe
  - Predittore (2,2) richiede $2^2 \times 2 \times 1K = 8K$ bit per la memorizzazione
  - Predittore (0,2) richiede $2^0 \times 2 \times 4K = 8K$ bit per la memorizzazione
- Si ottiene che…
  - Il predittore (2,2) ha un'accuratezza sempre migliore del predittore (0,2) con 4K righe
    - Per il predittore (2,2) il numero di errori di predizione varia da 0% a 6%, mentre per il predittore (0,2) varia da 0% a 18%
  - Il predittore (2,2) ha un'accuratezza quasi sempre migliore di un predittore (0,2) con un numero illimitato di righe

Accuratezza dei predittori correlati (2)



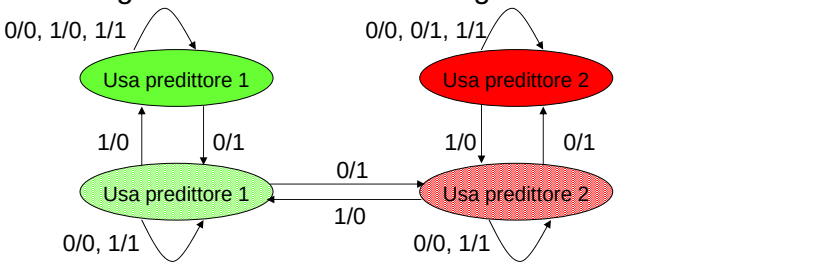
Predittori ibridi

- Predittori di salto a torneo *(tournament predictor)*
- Idea: predizioni multiple per ogni salto
- Si utilizzano livelli multipli di predittori (basati su informazioni globali e locali) insieme ad un meccanismo di selezione per scegliere il predittore migliore

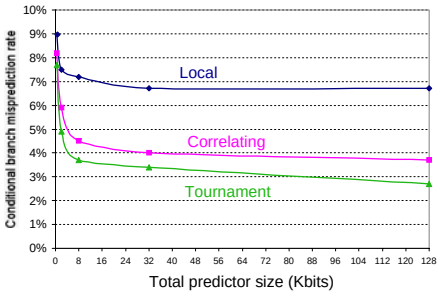
- Esempio (processore Alpha 21264)
  - Tournament predictor con un contatore in saturazione a 2 bit per salto per scegliere tra due diversi predittori: i quattro stati del contatore indicano il predittore da scegliere
  - Il contatore è incrementato quando il predittore prescelto è corretto e l'altro è sbagliato
  - Il contatore è decrementato quando il predittore prescelto è sbagliato e l'altro è corretto

Predittori ibridi (2)

- Diagramma di transizione tra gli stati



- Tasso di predizioni errate all'aumentare della dimensione totale del predittore



Predizione della destinazione del salto

- Tecnica generalmente usata nello stadio IF per predire la prossima istruzione da caricare dalla memoria istruzioni
  - Obiettivo: avere un branch penalty pari a 0
- Utilizza il Branch Target Buffer (BTB)
- Se il salto è non eseguito
  - Si incrementa il PC e si continua in modo sequenziale
- Se il salto è eseguito
  - Accedendo al BTB si ottiene l'indirizzo di destinazione del salto, con il quale si aggiorna il PC

Branch Target Buffer

- Il BTB è una piccola memoria (cache: vedi lezioni successive) che contiene l'*indirizzo di destinazione* del salto predetto
  - Indirizzo dell'istruzione successiva a quella di salto nel caso in cui il salto è eseguito
- L'accesso al BTB avviene solitamente nello stadio IF, utilizzando l'indirizzo dell'istruzione appena caricata (un possibile salto!) per indicizzare il BTB
  - Si anticipa la conoscenza dell'indirizzo di destinazione del salto di un ciclo (da ID a IF)
- Riga tipica del BTB

Indirizzo esatto del salto	Indirizzo di destinazione predetto

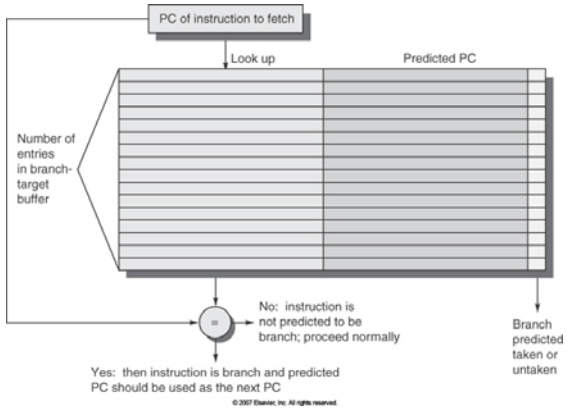
Branch Target Buffer (2)

- Branch History Table: nella pipeline a 5 stadi, vi si accede nello stadio ID, al termine del quale si conoscono:
  - La predizione del salto
  - L'indirizzo di destinazione del salto (calcolato in ID)
  - L'indirizzo fall-through (ovvero PC+4, calcolato in IF)

  Bastano per caricare la prossima istruzione predetta!

- Branch Target Buffer: vi si accede nello stadio IF
  - Se l'accesso al BTB ha successo, ovvero l'indirizzo dell'istruzione caricata coincide con un indirizzo presente nel BTB:
    - L'istruzione caricata è un salto
    - L'indirizzo predetto è noto al termine dello stadio IF: un ciclo prima rispetto a quando viene usata solo la BHT
    - L'indirizzo predetto viene usato come prossimo PC

Struttura di un BTB

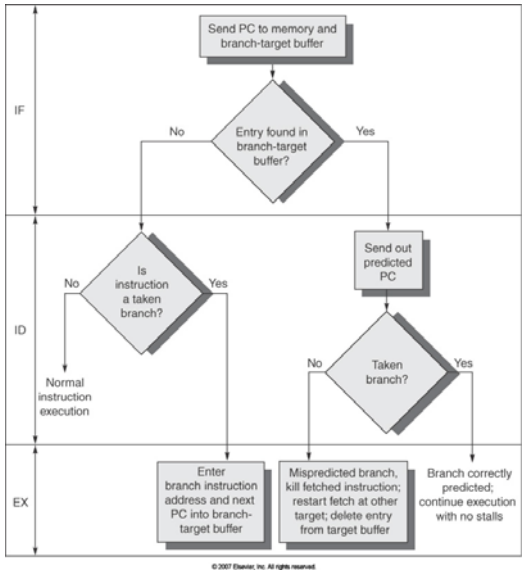


- L'indirizzo predetto viene inviato alla memoria istruzioni *prima della decodifica* dell'istruzione: occorre sapere se l'istruzione predetta è un salto
- BTB è una memoria cache di dimensioni modeste (tipicamente con organizzazione completamente associativa)

Uso del Branch Target Buffer

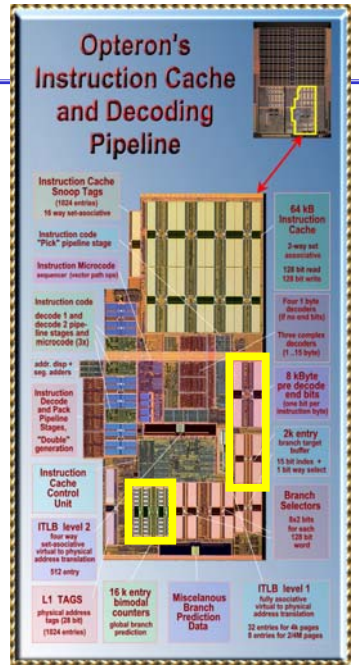
- Se la ricerca nel BTB ha successo (*hit* nella terminologia della cache)
  - Il caricamento della prossima istruzione avviene usando il PC predetto
  - Il matching è necessario: si carica la prossima istruzione *prima* di sapere se l'istruzione attuale è un salto
  - Si memorizzano nel BTB solo gli indirizzi di destinazione dei salti la cui predizione è uguale ad eseguito
    - Se il salto non viene eseguito, il calcolo del prossimo PC è uguale alle altre istruzioni

Gestione di un'istruzione con BTB



Un esempio reale

- Processore AMD Opteron a 64 bit



Influenza sulle prestazioni delle criticità sul controllo

$$\text{Speedup}_{\text{pipeline}} = \frac{\text{Profondità pipeline}}{1 + \text{cicli stallo pipeline per salto}}$$

- Dato che

  cicli stallo pipeline per salto = frequenza salto $\times$ penalizzazione salto

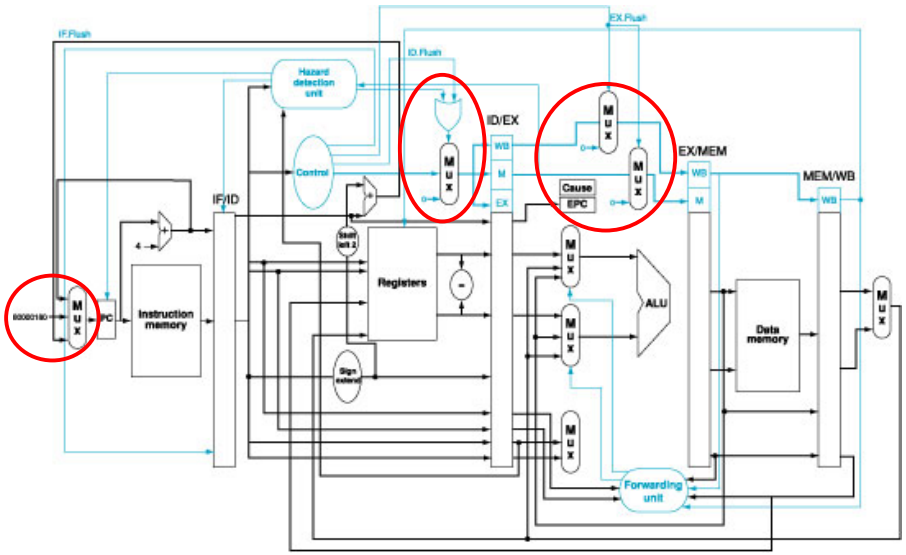
- Si ottiene

$$\text{Speedup}_{\text{pipeline}} = \frac{\text{Profondità pipeline}}{1 + \text{frequenza salto} \times \text{penalizzazione salto}}$$

Eccezioni nel MIPS

- Alcuni tipi di eccezioni
  - Overflow
  - Istruzioni illegali
- E' un'altra forma di criticità sul controllo
- Azioni da intraprendere
  - Caricare il PC con l'indirizzo di gestione dell'eccezione
  - Svuotare immediatamente la pipeline dalle istruzioni caricate successivamente a quella che ha generato l'eccezione
    - Segnali IF.Flush, ID.Flush, EX.Flush
  - Lasciare i registri non modificati
    - Es.: se l'istruzione add $1, $2, $1 causa overflow occorre preservare il vecchio valore di $1

Unità di elaborazione con pipelining e gestione delle eccezioni



Esempio

- Consideriamo la sequenza di istruzioni MIPS

40_{hex}	sub $11, $2, $4
44_{hex}	and $12, $2, $5
48_{hex}	or $13, $2, $6
$4C_{hex}$	add $1, $2, $1
50_{hex}	slt $15, $6, $7
54_{hex}	lw $16, 4($7)
…	

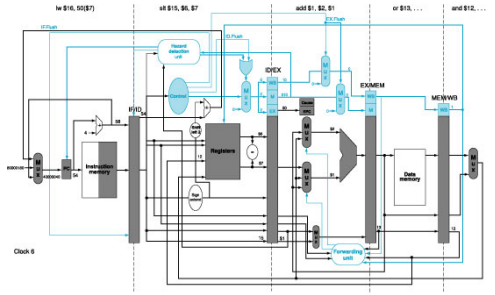
- Assumiamo che le istruzioni eseguite in caso di eccezione siano

40000040_{hex}	sw $25, 1000($0)
40000044_{hex}	sw $25, 1004($0)

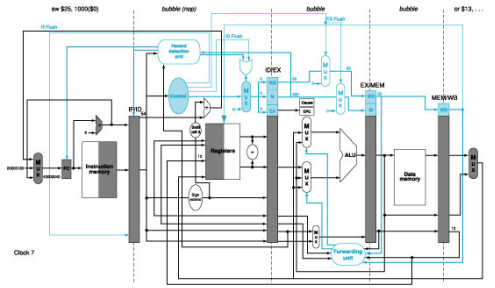
- Analizziamo cosa succede se l'istruzione add genera un'eccezione (nel suo stadio EX: ciclo di clock 6)

Esempio: cicli di clock 6 e 7

- In EPC $4C_{hex}+4=50_{hex}$
- Vengono asseriti tutti i segnali di flushing (IF.Flush, ID.Flush e EX.Flush)
- I segnali di controllo per add vengono messi a 0
- L'istruzione and termina



- Viene caricata nella pipeline la prima istruzione della routine di gestione delle eccezioni
- L'istruzione or termina



Eccezioni imprecise

- In un calcolatore con pipelining è difficile associare sempre in modo corretto un'eccezione all'istruzione che l'ha provocata
- Eccezione *imprecisa*: non è associata all'istruzione esatta che ha causato l'eccezione
  - Esempio con eccezione imprecisa: in EPC 58_{hex} anche se l'istruzione che ha provocato l'eccezione è all'indirizzo $4C_{hex}$
- Eccezione *precisa*: è sempre associata all'istruzione esatta che ha causato l'eccezione
  - L'istruzione che solleva l'eccezione è anche l'istruzione interrotta

L'unità completa di elaborazione e di controllo del processore MIPS con pipeline