Pipeline
Dipendenza dai controlli

Alcune soluzioni per salti condizionati

- flussi multipli (multiple streams) ➡
- prelievo anticipato della destinazione (prefetch branch target) ➡
- buffer circolare (loop buffer) ➡
- predizione del salto (branch prediction) ➡
- salto ritardato (delayed branch) ➡

Pipeline
Dipendenza dai controlli

Flussi multipli: replicare le parti iniziali della pipeline, una che contenga l'istruzione successiva a quella corrente di salto (nel caso il salto non avvenga), e l'altra l'istruzione destinazione (*target*) del salto (nel caso in cui il salto avvenga)

Problemi di questa soluzione:

- conflitti nell'accesso alle risorse (registri, memoria, ALU,…) da parte delle 2 pipeline
- presenza di salti condizionali in sequenza che entrano nelle 2 pipeline prima che si sia risolta la condizione del primo salto condizionale (occorrerebbero 2 pipeline aggiuntive per ogni ulteriore salto condizionale…)

Pipeline
Dipendenza dai controlli

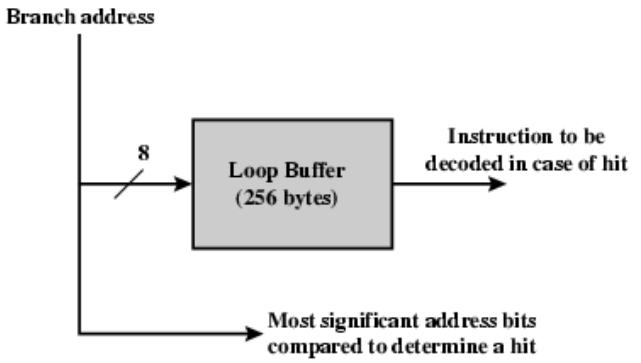
Prelievo anticipato della destinazione: quando si incontra un salto condizionato si effettua il fetch anticipato della istruzione di destinazione del salto in modo da trovarla già caricata nel caso in cui il salto debba avvenire.

Problemi di questa soluzione:

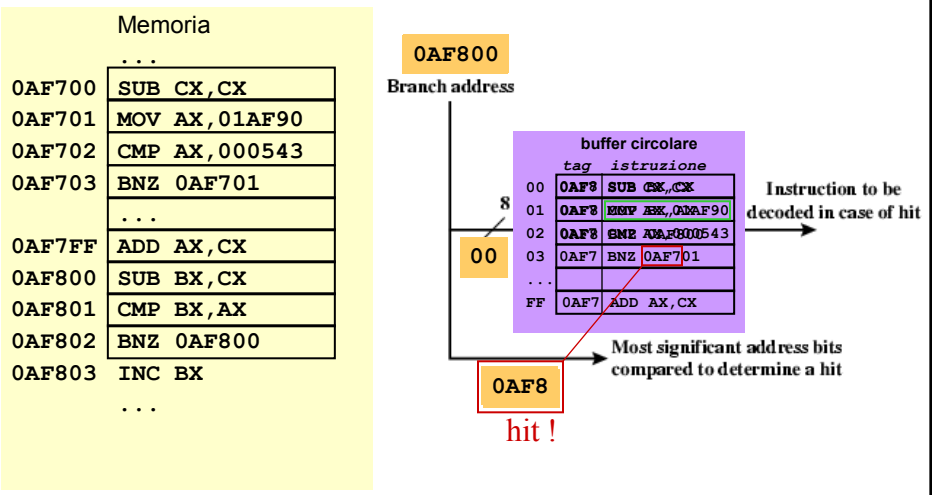
- non evita l'eventuale svuotamento della pipeline con conseguente perdita di prestazioni

Pipeline
Dipendenza dai controlli

Buffer circolare: si utilizza una memoria piccola e molto veloce (il buffer circolare) dove mantenere le ultime *n* istruzioni prelevate. In caso di salto, si controlla se l'istruzione destinazione è già presente nel buffer, così da evitare il fetch della stessa.



Buffer circolare (senza prefetch)



Pipeline

Dipendenza dai controlli

Buffer circolare: si utilizza una memoria piccola e molto veloce (il buffer circolare) dove mantenere le ultime *n* istruzioni prelevate. In caso di salto, si controlla se l'istruzione destinazione è già presente nel buffer, così da evitare il fetch della stessa.

Vantaggi:

- anticipando il fetch, alcune delle istruzioni successive a quella corrente saranno già presenti nel buffer e se non si ha salto non ci sarà bisogno di caricarle dalla memoria
- se si salta in avanti di poche istruzioni (vedi trattamento del costrutto IF-THEN-ELSE), l'istruzione destinazione sarà già presente nel buffer
- se il salto condizionale realizza un ciclo le cui istruzioni possono essere tutte contenute nel buffer, non c'è bisogno di effettuare fetch ripetuti delle stesse istruzioni

Pipeline
Dipendenza dai controlli

Predizione dei salti: si cerca di prevedere se il salto sarà intrapreso oppure no.

Varie possibilità:

- previsione di saltare sempre
- previsone di non saltare mai
- previsione in base al codice operativo

} *approcci statici*

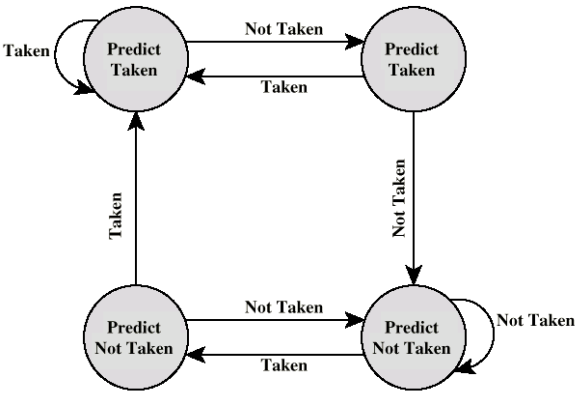
- bit *taken/not taken*
- tabella della storia dei salti

} *approcci dinamici*

Approcci dinamici di predizione: cercano di migliorare la qualità della predizione sul salto memorizzando la storia delle istruzioni di salto condizionato di uno specifico programma.

bit *taken/not taken* :

- ad ogni istruzione di salto condizionato si associano uno o più bit che codificano la storia recente.
- bit memorizzati non in memoria centrale ma in una locazione temporanea ad accesso molto veloce



esempio con 2 bit

Approcci dinamici di predizione: cercano di migliorare la qualità della predizione sul salto memorizzando la storia delle istruzioni di salto condizionato di uno specifico programma

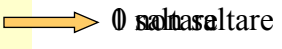
esempio:

```
        ……
LOOP:   ……
        ……
        ……
        BNZ   LOOP
```

- **Predizione con 1 bit**: si predice il comportamento osservato l'ultima volta
  - dopo la prima esecuzione del ciclo, in uscita dal ciclo, il bit assegnato a BNZ ricorderà che il salto **non è stato preso**, così che, quando si rientra nel ciclo si avrà un primo errore per la prima iterazione del ciclo (che invece è preso), le successive predizioni saranno giuste, tranne l'ultima, quando si esce dal ciclo: in totale **2 errori**
- **Predizione con 2 bit**: vedi lucido precedente
  - dopo la prima esecuzione del ciclo, si commette **un solo errore** di predizione all'uscita del ciclo

Predizione dinamica 1 bit

```
        ...
LOOP2   SUB CX,CX
LOOP1   MOV AX,0001AF90
        CMP AX,00000543
        BNZ LOOP1
        INC CX
        ADD AX,CX
        SUB BX,CX
        CMP BX,AX
        BNZ LOOP2
        INC BX
        ...
```



0 saltare

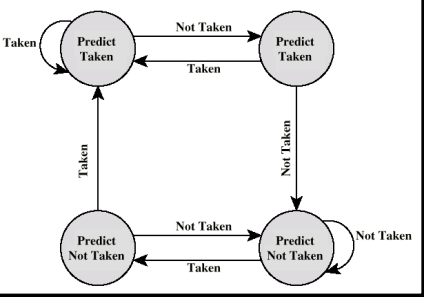
errori di predizione 0

Predizione dinamica 2 bit

```
        ...
LOOP2   SUB CX,CX
LOOP1   MOV AX,0001AF90
        CMP AX,00000543
        BNZ LOOP1
        INC CX
        ADD AX,CX
        SUB BX,CX
        CMP BX,AX
        BNZ LOOP2
        INC BX
        ...
```

errori di predizione 0

10 saltare

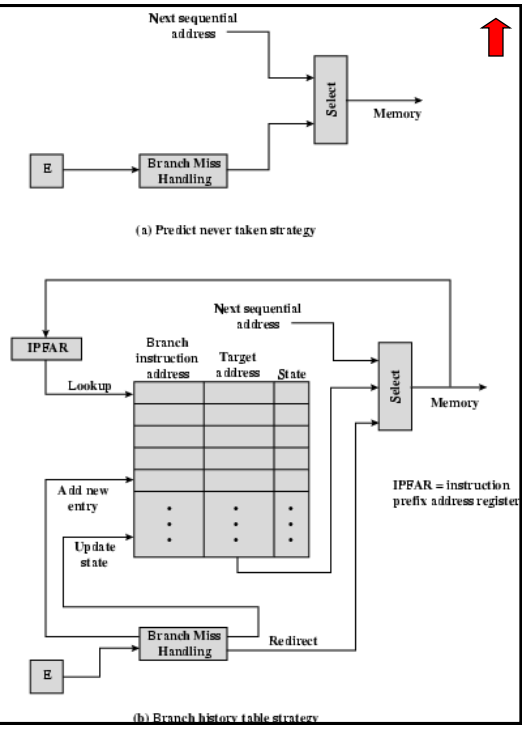


Problemi di bit *taken/not taken* :

- quando si decide di saltare, bisogna aspettare la decodifica dell'indirizzo destinazione prima di poter prelevare l'istruzione destinazione
- si può anticipare il prelievo a patto di salvare opportune info nel *branch target buffer* o *branch history table*

tabella della storia dei salti:

- piccola memoria associata allo stadio fetch della pipeline
- ogni riga della tabella è costituita da 3 elementi:
  1. indirizzo istruzione salto,
  2. l'indirizzo destinazione del salto (o l'istruzione destinazione stessa),
  3. alcuni bit di storia che descrivono lo stato dell'uso dell'istruzione



Salto ritardato (delayed branch)

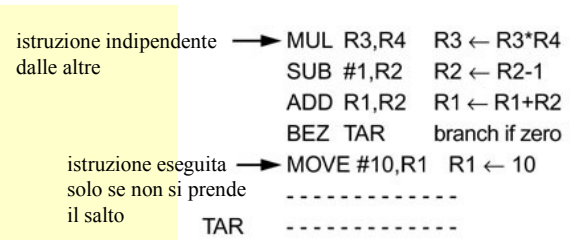
Idea base: utilizzare gli stadi inattivi a causa dello stallo per fare del lavoro utile

Delayed branch:

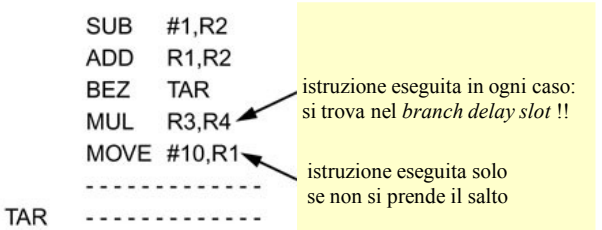
- La CPU esegue **sempre** l'istruzione che segue il salto e solo dopo altera, se necessario, la sequenza di esecuzione delle istruzioni
- L'istruzione che segue quella di salto si dice essere posta nel *branch delay slot*
- Il **compilatore** cerca di allocare nel *branch delay slot* una istruzione "opportuna"

Salto ritardato (delayed branch)

codice scritto dal programmatore



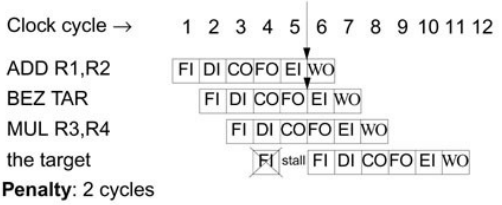
codice ottimizzato dal compilatore



Salto ritardato (delayed branch)

Salto preso

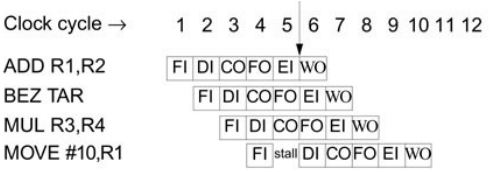
condizione ed indirizzo di salto conosciuti qui



Penalty: 2 cycles

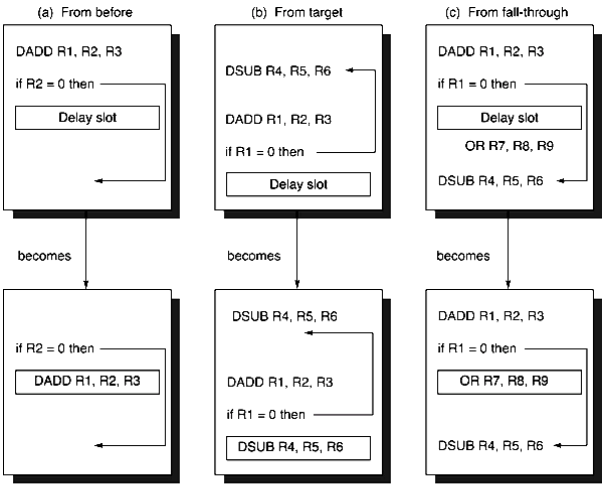
Salto non preso

condizione di salto conosciuta qui



Penalty: 1 cycle

Salto ritardato (delayed branch)



b) e c) legali solo se R4 e R7 sono registri temporanei il cui contenuto può essere "sporcato" senza cambiare la semantica del programma

Intel 80486 Pipelining

- Fetch
  - Istruzioni prelevate dalla cache o memoria esterna
  - Poste in uno dei due buffer di prefetch da 16 byte
  - Carica dati nuovi appena quelli vecchi sono "consumati"
  - Poiché le istruzioni sono a lunghezza variabile (1-11 byte), in media carica 5 istruzioni per ogni caricamento da 16 byte
  - Indipendente dagli altri stadi per mantenere i buffer pieni
- Decodifica 1 (D1)
  - Decodifica codice operativo e modi di indirizzamento
  - Le informazioni di sopra sono codificate (al più) nei primi 3 byte di ogni istruzione
  - Se necessario, indica allo stadio D2 di trattare i byte restanti (dati immediati e spiazzamento)
- Decodifica 2 (D2)
  - Espande i codici operativi in segnali di controllo per l'ALU
  - Provvede a controllare i calcoli per i modi di indirizzamento più complessi
- Esecuzione (EX)
  - Operazioni ALU, accesso alla cache (memoria), aggiornamento registri
- Retroscrittura (WB)
  - Se richiesto, aggiorna i registri e i flag di stato modificati in EX
  - Se l'istruzione corrente aggiorna la memoria, pone il valore calcolato in cache e nei buffer di scrittura del bus

80486 Instruction Pipeline: esempi

(a) No Data Load Delay in the Pipeline

(b) Pointer Load Delay

(c) Branch Instruction Timing