Pipeline
prestazioni ideali

Le prestazioni ideali di una pipeline si possono calcolare matematicamente come segue

- Sia τ il tempo di ciclo di una pipeline necessario per far avanzare di uno stadio le istruzioni attraverso una pipeline. Questo può essere determinato come segue:

$$\tau = \max_i [\tau_i] + d = \tau_m + d \quad 1 \le i \le k$$

  - τ_m = massimo ritardo di stadio (ritardo dello stadio più oneroso)
  - k = numero di stadi nella pipeline
  - d = ritardo di commutazione di un registro, richiesto per l'avanzamento di segnali e dati da uno stadio al successivo

Pipeline
prestazioni ideali

Poiché $\tau_m >> d$, il tempo totale T_k richiesto da una pipeline con k stadi per eseguire n istruzioni (senza considerare salti ed in prima approssimazione) è dato da

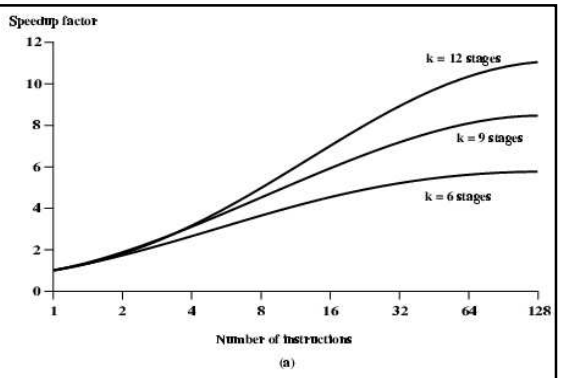
$$T_k = [k + (n\text{-}1)]\tau$$

in quanto occorrono k cicli per completare l'esecuzione della prima istruzione e n-1 per le restanti istruzioni, e quindi il *fattore di velocizzazione* (speedup) di una pipeline a k stadi è dato da:

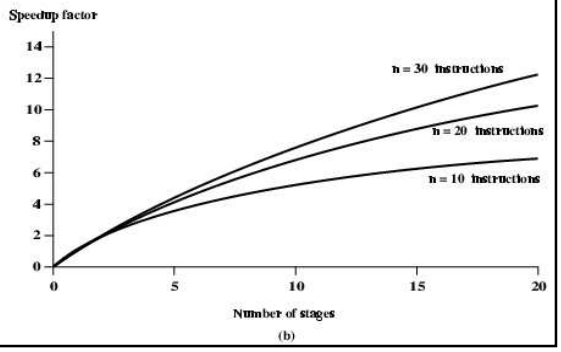
$$S_k = \frac{T_1}{T_k} = \frac{nk\tau}{[k+(n-1)]\tau} = \frac{nk}{[k+(n-1)]}$$

Speedup

Calcolato in funzione del numero di istruzioni



Calcolato in funzione del numero di stadi



Pipeline
Problemi 1

- Vari fenomeni pregiudicano il raggiungimento del massimo di parallelismo teorico (**stallo**)
  - **Sbilanciamento delle fasi**
    - Durata diversa per fase e per istruzione
  - **Problemi strutturali**
    - La sovrapposizione totale di tutte le (fasi di) istruzioni causa conflitti di accesso a risorse limitate e condivise (ad esempio la memoria per gli stadi FI, FO, WO)

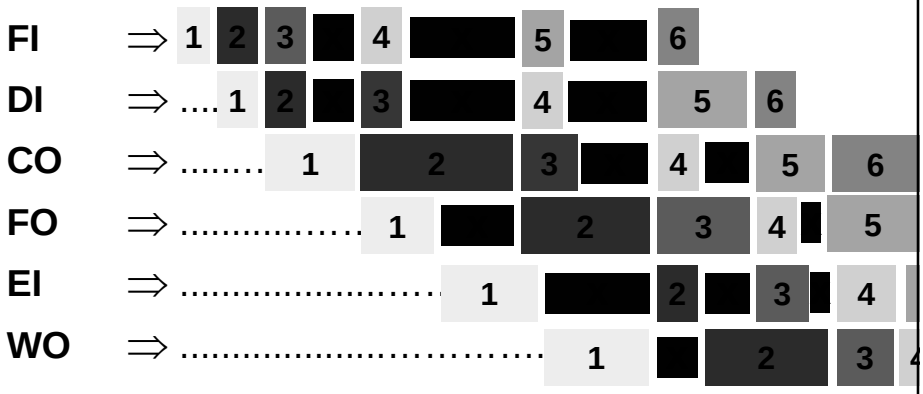
Pipeline
Problemi 2

- **Dipendenza dai dati**
  - L'operazione successiva dipende dai risultati dell'operazione precedente
- **Dipendenza dai controlli**
  - Istruzioni che causano una violazione di sequenzialità (p.es.: salti condizionali) invalidano il principio del *pipelining* sequenziale

Pipeline
Sbilanciamento delle fasi 1

- La suddivisione in fasi va fatta in base all'istruzione più onerosa
- Non tutte le istruzioni richiedono le stesse fasi e le stesse risorse
- Non tutte le fasi richiedono lo stesso tempo di esecuzione
  - P.es.: lettura di un operando tramite registro rispetto ad una mediante indirizzamento indiretto

Pipeline
Sbilanciamento delle fasi 2



Tempo di attesa forzata dovuta allo sbilanciamento delle fasi

Pipeline
Sbilanciamento delle fasi 3

Possibili soluzioni allo sbilanciamento:

- Decomporre fasi onerose in più sottofasi
  - Costo elevato e bassa utilizzazione
- Duplicare gli esecutori delle fasi più onerose e farli operare in parallelo
  - CPU moderne hanno una ALU in aritmetica intera ed una in aritmetica a virgola mobile

Pipeline
Problemi strutturali

Problemi

- Maggiori risorse interne (**severità bassa**): l'evoluzione tecnologica ha spesso permesso di duplicarle (es. registri)
- Colli di bottiglia (**severità alta**): l'accesso alle risorse esterne, p.es.: memoria, è molto costoso e molto frequente (anche 3 accessi per ciclo di clock)

Soluzioni

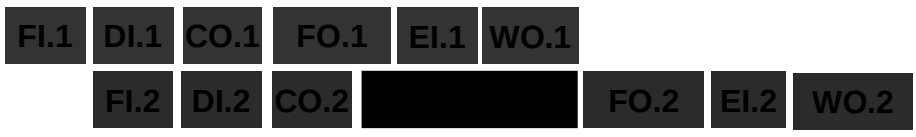
- Suddividere le memorie (accessi paralleli: introdurre una memoria cache per le istruzioni e una per i dati)
- Introdurre fasi non operative (***nop***)

Pipeline
Dipendenza dai dati 1

- Un dato modificato nella fase **EI** dell'istruzione corrente può dover essere utilizzato dalla fase **FO** dell'istruzione successiva

  INC [0123]

  CMP [0123], AL



Ci sono altri tipi di dipendenze ?

Pipeline
Dipendenza dai dati 2

Soluzioni

- Introduzione di fasi non operative (***nop***)
- Individuazione del rischio e prelievo del dato direttamente all'uscita dell'ALU (**data forwarding**) ➡
- Risoluzione a livello di compilatore **(vedremo esempi per l'architettura MIPS)**
- Riordino delle istruzioni (**pipeline scheduling**)

Pipeline
Dipendenza dai controlli

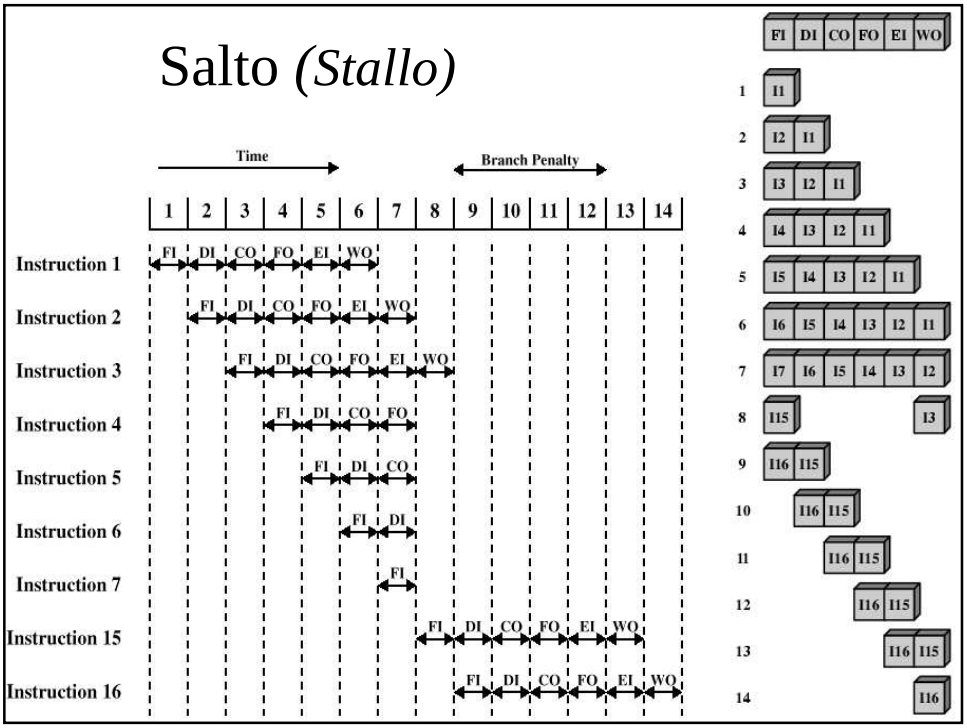
- Tutte le istruzioni che modificano il PC (salti condizionati e non, chiamate a e ritorni da procedure, interruzioni) invalidano il pipeline
- La fase **fetch** successiva carica l'istruzione seguente, che può *non essere* quella giusta
- Tali istruzioni sono circa il 30% del totale medio di un programma

Pipeline
Dipendenza dai controlli

Soluzioni

- Mettere in **stallo** il pipeline fino a quando non si è calcolato l'indirizzo della prossima istruzione
  - Pessima efficienza, massima semplicità
- Individuare le istruzioni critiche per anticiparne l'esecuzione, eventualmente mediante apposita logica di controllo
  - Compilazione complessa, hardware specifico

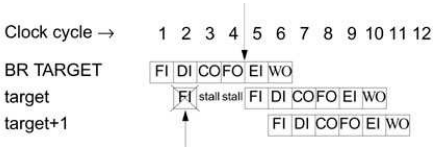
Salto *(Stallo)*



Salto *(Stallo)*

Salto incondizionato

indirizzo di salto conosciuto qui

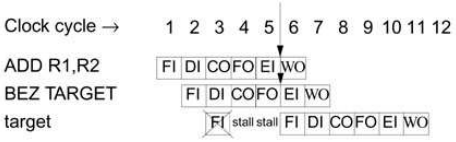


inizialmente caricata istruzione successiva a BR TARGET, poi eliminata quando si riconosce il salto incondizionato alla fine della fase DI

Salto condizionato

Salto preso

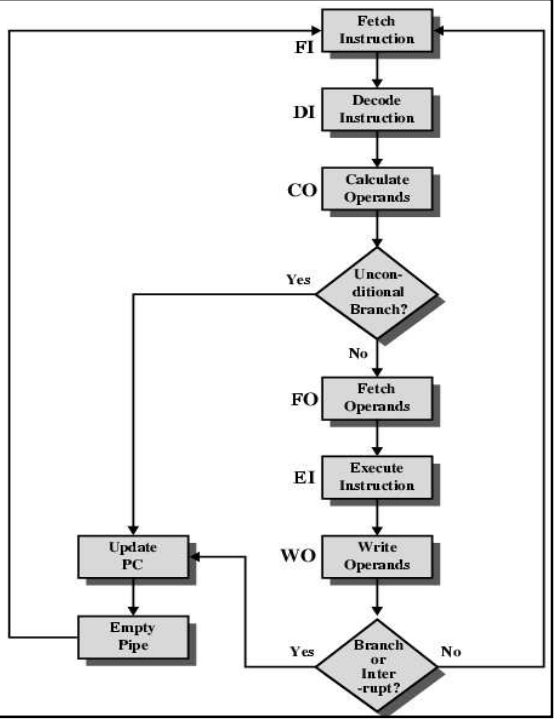
condizione ed indirizzo di salto conosciuti qui



Salto non preso

condizione di salto conosciuta qui

Funzionamento pipeline a 6 stadi con trattamento dei salti ed interrupt tramite svuotamento

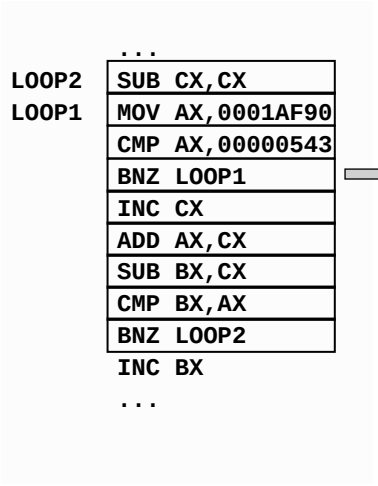


Pipeline
Dipendenza dai controlli

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) ➡

Predizione dinamica 1 bit

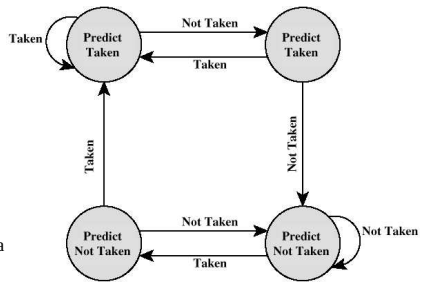


Predizione dinamica 2 bit

```
          ...
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



Architettura

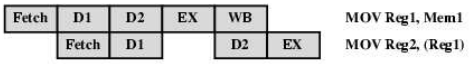
Intel 80486 Pipelining

- 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
  - Independente 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

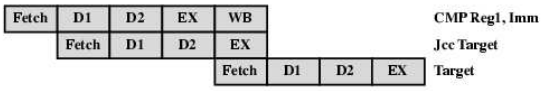
Architettura degli elaboratori -1 Pagina 91

80486 Instruction Pipeline: esempi

(a) No Data Load Delay in the Pipeline



(b) Pointer Load Delay



(c) Branch Instruction Timing