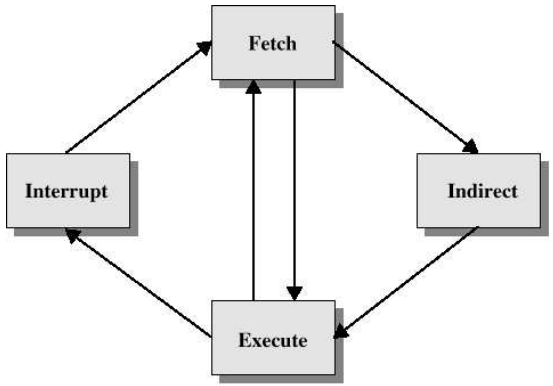
Ciclo esecutivo delle istruzioni: Fetch/Execute

- Lo avete visto nel corso di "Introduzione alle Architetture degli Elaboratori"
- Stallings, Capitolo 3
- Ne vediamo una versione revisionata

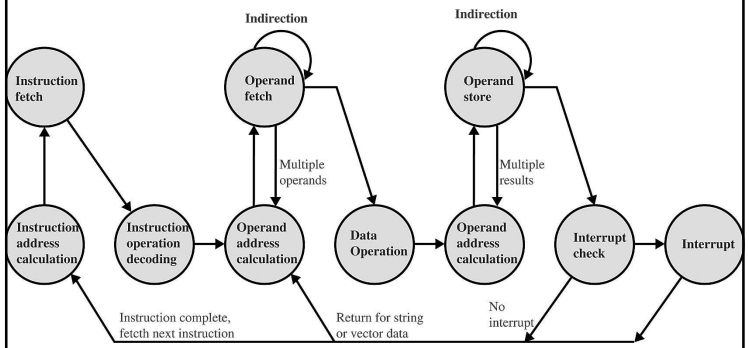
Indirettezza

- Per recuperare gli operandi di una istruzione può essere necessario accedere alla memoria
- La modalità di indirizzamento indiretto per specificare la locazione in memoria degli operandi richiede più accessi in memoria
- L'indirettezza si può considerare come un sottociclo del ciclo fetch/execute

Introduzione della Indirettezza (indirect)



Fetch/Execute: ancora più in dettaglio



Flusso dei dati (Instruction Fetch)

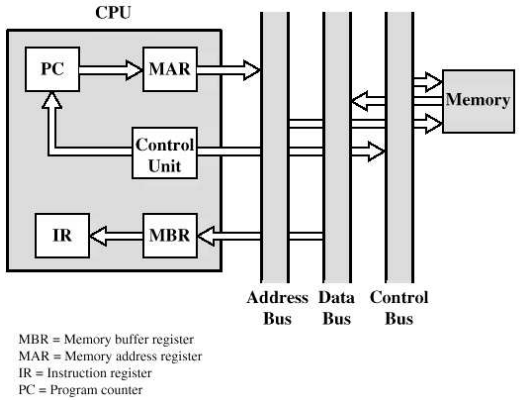
Dipende dalla architettura della CPU, in generale:

- Fetch
  - PC contiene l'indirizzo della istruzione successiva
  - Tale indirizzo viene spostato in MAR
  - L'indirizzo viene emesso sul bus degli indirizzi
  - La unità di controllo richiede una lettura in memoria principale
  - Il risultato della lettura in memoria principale viene inviato nel bus dati, copiato in MBR, ed infine in IR
  - Contemporaneamente PC viene incrementato di 1

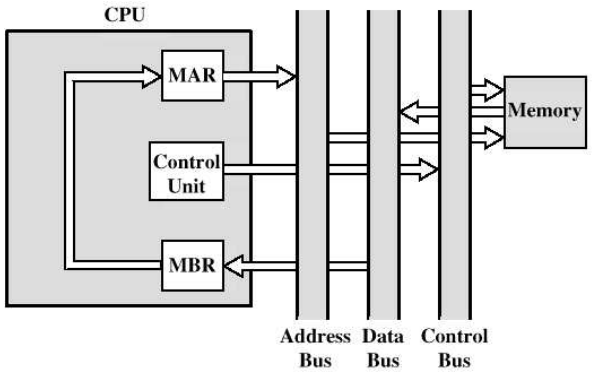
Flusso dei dati (Data Fetch)

- IR è esaminato
- Se il codice operativo della istruzione richiede un indirizzamento indiretto, si esegue il ciclo di indirettezza
  - gli N bit più a destra di MBR vengono trasferiti nel MAR
  - L'unità di controllo richiede la lettura dalla memoria principale
  - Il risultato della lettura (indirizzo dell'operando) viene trasferito in MBR

Flusso dei dati (Diagramma di Fetch)



Flusso dei dati (Diagramma di Indirettezza)



Flusso dei dati (Execute)

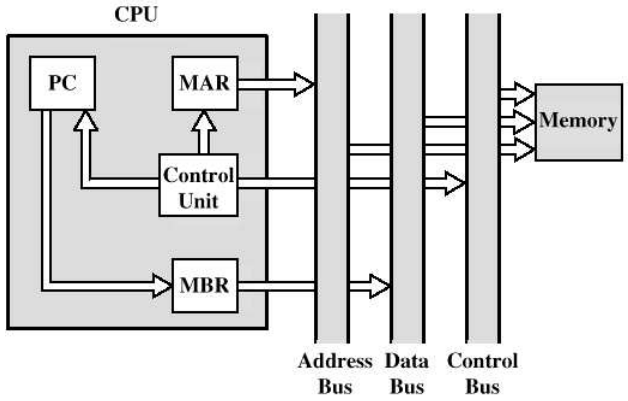
- Può assumere molte forme
- Dipende dalla istruzione da eseguire
- Può includere
  - lettura/scrittura della memoria
  - Input/Output
  - Trasferimento di dati fra registri e/o in registri
  - Operazioni della ALU

Flusso dei dati (Interrupt)

Semplice e prevedibile:

- Contenuto corrente del PC deve essere salvato per permettere il ripristino della esecuzione dopo la gestione dell'interruzione
  - Contenuto PC copiato in MBR
  - Indirizzo di locazione di memoria speciale (es. stack pointer) caricato in MAR
  - Contenuto di MBR scritto in memoria
- PC caricato con l'indirizzo della prima istruzione della routine di gestione della interruzione
- Fetch della istruzione puntata da PC

Flusso dei dati (Diagram. di Interrupt)



Prefetch

- La fase di prelievo della istruzione accede alla memoria principale
- La fase di esecuzione di solito non accede alla memoria principale
- Si può prelevare l'istruzione successiva durante l'esecuzione della istruzione corrente
- Questa operazione si chiama "instruction prefetch"

Miglioramento delle prestazioni

- Il prefetch non raddoppia le prestazioni:
  - La fase di prelievo è tipicamente più breve della fase di esecuzione
    - Prefetch di più istruzioni ?
  - L'esecuzione di istruzioni jump o branch possono rendere vano il prefetch perché si deve caricare una istruzione diversa dalla successiva
- Aggiungere più fasi per migliorare le prestazioni

Evoluzione delle architetture
Evoluzione strutturale

- **Parallelismo**
  - Se un lavoro non può essere svolto più velocemente da una sola persona (unità), allora conviene **decomporlo** in parti che possano essere eseguite da più persone (unità) **contemporaneamente**
    - **Catena di montaggio**

Pipeline
Generalità 1

- Ipotizziamo che per svolgere un dato lavoro **L** si debbano eseguire tre fasi distinte e sequenziali

  L $\Rightarrow$ [*fase1*] [*fase2*] [*fase3*]
- Se ogni fase richiede **T** unità di tempo, un unico esecutore svolge un lavoro **L** ogni **3T** unità di tempo
- Per ridurre i tempi di produzione si possono utilizzare **più esecutori**

Pipeline
Generalità 2

- Soluzione (ideale) a parallelismo totale

  E1 $\Rightarrow$ [fase1.**A**] [fase2.**A**] [fase3.**A**] | [fase1.**D**] …

  E2 $\Rightarrow$ [fase1.**B**] [fase2.**B**] [fase3.**B**] | [fase1.**E**] …

  E3 $\Rightarrow$ [fase1.**C**] [fase2.**C**] [fase3.**C**] | [fase1.**F**] …
- **N** esecutori svolgono un lavoro ogni **3T/N** unità di tempo
- Il problema è **come** preservare la **dipendenza funzionale** nell'esecuzione (di fasi) dei 'lavori' **A**, **B**, **C**, **D**, **E**, **F**, …

Pipeline
Generalità 3

- Soluzione **pipeline** ad **esecutori generici**

E1 ⇒ [fase1] [fase2] [fase3] [fase1] [fase2]

E2 ⇒ ..... [fase1] [fase2] [fase3] [fase1]

E3 ⇒ ..... [fase1] [fase2] [fase3]

- Ogni esecutore esegue un ciclo di lavoro **completo** (*sistema totalmente replicato*)
- A regime, **N** esecutori svolgono un lavoro **L** ogni **3T/N** unità di tempo rispettandone la sequenza

Pipeline
Generalità 4

- Soluzione **pipeline** ad **esecutori specializzati**

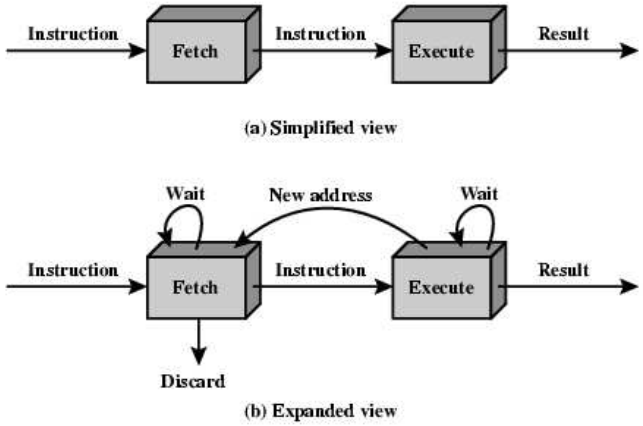
E1 ⇒ [fase1] [fase1] [fase1] [fase1] [fase1]

E2 ⇒ ..... [fase2] [fase2] [fase2] [fase2]

E3 ⇒ ..... [fase3] [fase3] [fase3]

- Ogni esecutore svolge sempre e solo la **stessa** fase di lavoro
- Soluzione più efficace in termini di **uso di risorse** (**3T/N** lavori con **N/3** risorse)

Prefetch come pipeline a due stadi

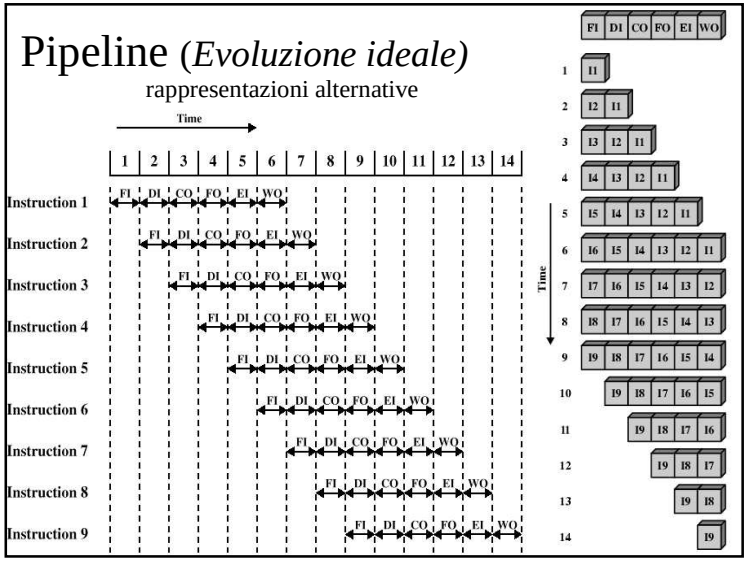


Pipeline
Decomposizione in fasi

- L'esecuzione di una generica istruzione può essere suddivisa nelle seguenti fasi:
  - **fetch (FI)** lettura dell'istruzione
  - **decodifica (DI)** decodifica dell'istruzione
  - **calcolo ind. op. (CO)** calcolo indirizzo effettivo operandi
  - **fetch operandi (FO)** lettura degli operandi in memoria
  - **esecuzione (EI)** esecuzione dell'istruzione
  - **scrittura (WO)** scrittura del risultato in memoria

Pipeline (*Evoluzione ideale)*

rappresentazioni alternative



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

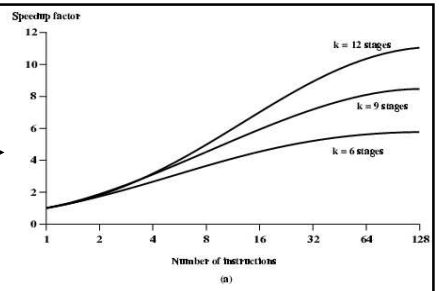
$$T_k = [k + (n\text{-}1)]\tau$$

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:

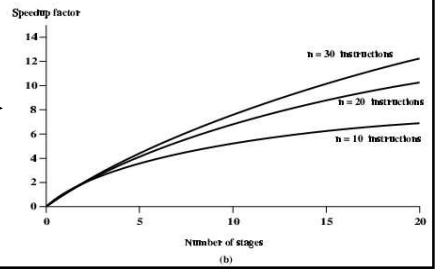
$$S_k = \frac{T_1}{T_k} = \frac{nk\tau}{[k+(n-1)]\tau} = \frac{nk}{[k+(n-1)]}$$

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

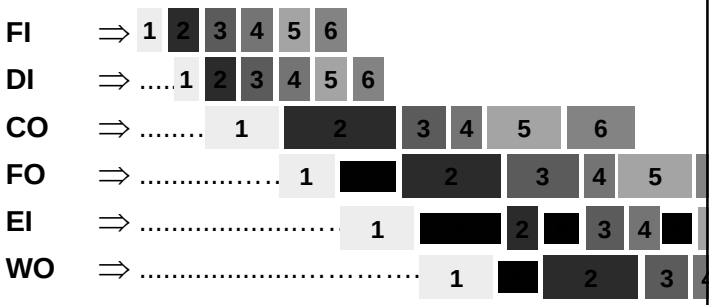
Pipeline
Problemi 2

- **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



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