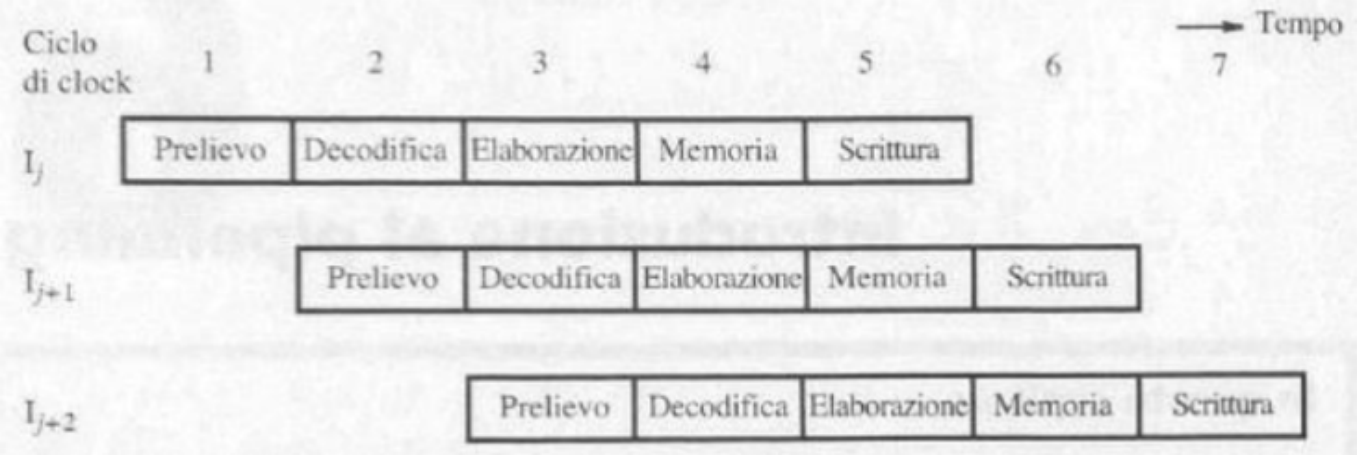
2 Pipelining

spiegazione pipeline breve

Pipeline

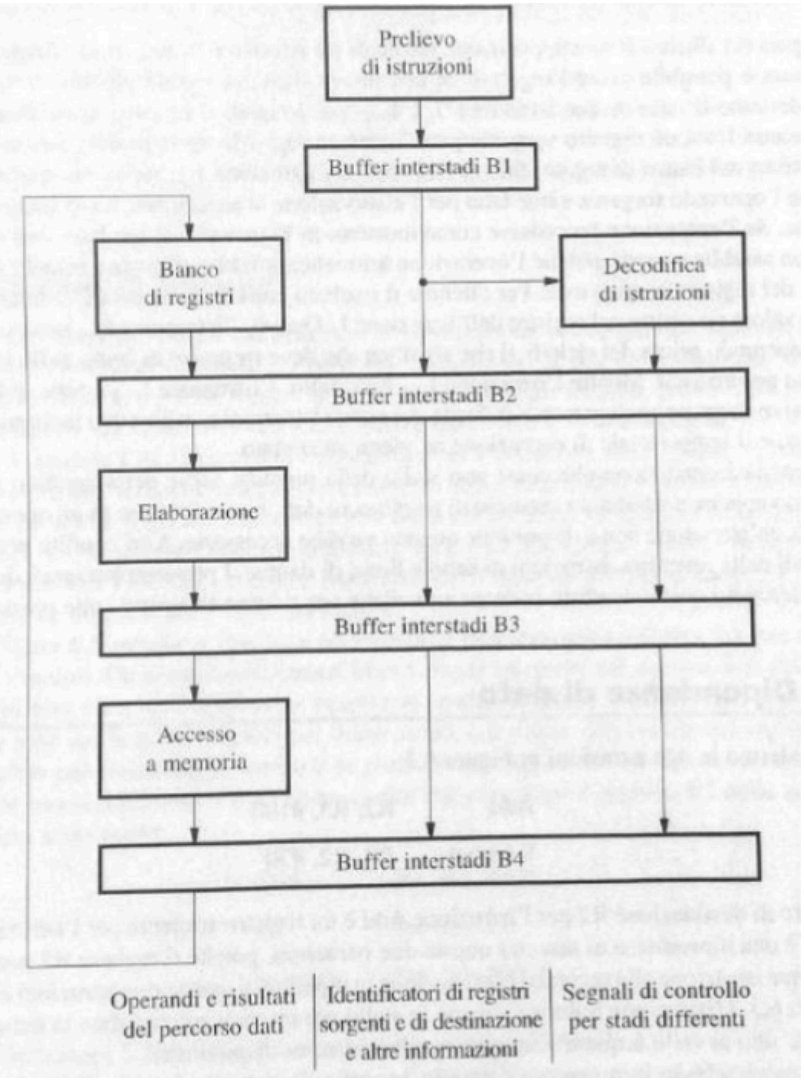
Il pipelining è una tecnica usata per migliorare l'efficienza della gestione delle istruzioni, le istruzioni vengono gestite in modo che ci siano più istruzioni eseguite contemporaneamente.
(processori con pipeline hanno un throughput maggiore rispetto a quelli che non la adottano)
(la pipeline si adatta meglio ai processori RISC perché loro usano istruzioni più semplici)



• Per gestire l'esecuzione in pipeline di più istruzioni, è necessario mantenere delle informazioni tra uno stadio e l'altro

• Queste informazioni vengono mantenute nei **buffer interstadi**

• I buffer interstadi contengono
• i rispettivi registri interstadio (RA, RB, PC_Temp, etc.)
• Il registro IR (Per mantenere gli identificatori dei registri sorgente e destinazione)
• Segnali di controllo per i vari stadi



Non sempre è possibile eseguire 5 istruzioni contemporaneamente perché si creano dei conflitti

**TIPI DI CONFLITTI:

1. DIPENDENZE DA DATO (data hazards)

DIPENDENZE **RAW** (read after write)
si verifica quando un'istruzione deve leggere un dato che un'istruzione precedente non ha ancora scritto

- Istruzione1: A = B + C;
- Istruzione2: D = A + 1

- l'Istruzione2 contiene una dipendenza RAW

DIPENDENZE **WAW** (write after write)
si verificano quando due istruzioni vogliono scrivere nello stesso registro o nella stessa posizione di memoria

DIPENDENZE **WAR** (write after read)
Si verifica quando un istruzione cerca di scrivere un registro prima che un'altra abbia finito di leggere.

Esistono delle **tecniche per evitare queste dipendenze:**

Nop → è un istruzione vuota o inattiva che non esegue operazioni, occupa uno slot nel flusso delle istruzioni senza effettuare alcun calcolo o modifica nello stato del processore. E' utilizzata per allineare le operazioni nella CPU . (serve per temporeggiare creando dei ritardi controllati)

Forwarding → bypassa la fase di store nei registri degli operandi (fa passare direttamente il risultato di un operazione all' istruzione successiva, quindi non fa la fase di store)
Il beneficio rispetto al NOP è che evita i ritardi (quindi migliora le performance della CPU)

Stallo → mette in attesa l'istruzione dipendente

Riordinamento delle istruzioni → riordina le istruzioni in modo che non ci siano problemi di dipendenza

Pre-fetching → carica le istruzioni prima Pre-fetching spiegazione

Speculative execution → cerca di predire eventuali problemi di dipendenza e cerca di non farli accadere

2. RITARDI NELL'ACCESSO ALLA MEMORIA

I principali ritardi della memoria si verificano perché:

le memorie hanno una propria latenza costruttiva (memory access latency) il tempo necessario ad accedere ai dati contenuti nella memoria, (influisce anche la velocità del bus)

Il problema sta nel fatto che **la CPU deve aspettare che i dati contenuti nella memoria arrivino** e quindi non può procedere rallentando la pipeline

Esistono delle **tecniche per diminuire questi ritardi:**

1. **Gerarchia della cache** → si usano 3 livelli di cache (L1 L2 L3) dalla più veloce e vicina alla CPU alla più lenta posta sulla scheda madre.
2. **Pre-fetching** → carica in anticipo i dati nella cache Pre-fetching spiegazione
3. **Produrre memoria a bassa latenza** → che siano cache o RAM più sono veloci meglio è.

3. RITARDI NEI SALTI

Nei processori moderni **i salti, nel pipelining introducono grossi ritardi (branch penalties/delays)**

Esistono delle tecniche per evitare questi ritardi:

La tecnica più importante è sicuramente la **branch prediction** (esecuzione speculativa) --> fa uso delle **predizioni di istruzioni di salto per anticipare una branch penalties** basandosi sui branch penalties precedenti e lo storico delle altre istruzioni, **se la predizione è sbagliata si ha un branch prediction miss**. (In quel caso la CPU esegue il **flushing** ovvero la pulizia della pipeline per evitare che si eseguano istruzioni errate)

1. La **predizione statica** utilizza regole semplici basate su informazioni statiche come la posizione di istruzioni nel codice oppure il tipo di istruzione di salto, per prevedere se il branch verrà preso o no, questo approccio statico è rapido e **semplice ma meno preciso in caso di situazioni complesse.**

2. La **predizione dinamica** è **basata sulla storia dei branch precedenti**, quindi utilizza i branch prediction buffer o tabelle di predizione di branch per memorizzare il comportamento passato dei branch. I moderni processori utilizzano dei metodi di predizione come i predittori a 2 livelli o predittori neurali per migliorare l'accuratezza delle predizioni.

4. LIMITI NELLE RISORSE

La pipeline può andare in **stallo** quando una risorsa hardware è richiesta da più istruzioni contemporaneamente.
lo stallo è un problema critico che **si verifica quando la pipeline viene interrotta a causa delle dipendenze da dato, o per altri problemi come le risorse limitate** (le risorse devono essere condivise. se non ci fossero le risorse condivise, quindi senza accesso concorrente alle risorse non ci sarebbero stalli nella pipeline)).
alcuni processi possono non rilasciare più una risorsa (qui è colpa del programmatore) e quindi creare stalli.

Valutazione delle prestazioni

La valutazione delle prestazioni di un processore viene fatta sia per i processori che usano la pipeline sia per quelli che non la usano, la formula chiaramente cambia. Di seguito la legenda per capire le formule successive:

- N: Numero di istruzioni macchina
- S: Numero di cicli di clock per istruzione (CPI - *Cycles Per Instruction*)
- T_{clock}: Durata di un ciclo di clock ($T_{clock} = \frac{1}{R}$)
- R : frequenza di clock del processore

> TIP
>
> La formula per calcolare T (tempo di esecuzione) di N istruzioni in un processore con pipeline è la seguente: $T = \frac{N*S}{R}$

> TIP
>
> La formula per calcolare il throughput di un processore è la seguente: **Senza pipeline:** $P_{np} = \frac{R}{S*T_{clock}}$ **Con pipeline:** $P_p = \frac{R}{S}$ nel caso ottimale con pipeline il throughput è uguale ad R

Come già detto la pipeline soffre di conflitti, prendendo in considerazione questi conflitti la formula per il calcolo del throughput diventa:

> Formula calcolo del throughput
>
> $P_p = \frac{R}{S+\sigma_{dato}+\sigma_{salto}+\sigma_{miss}}$

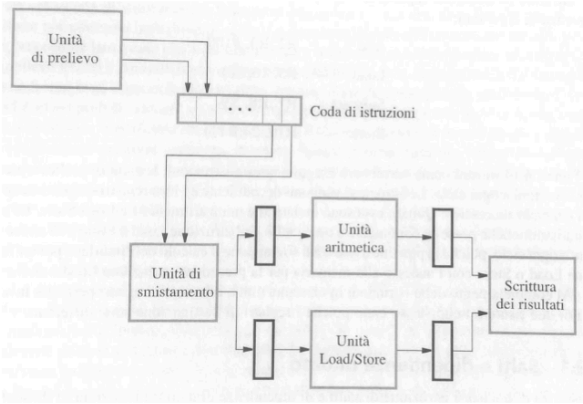
Ogni tipologia di conflitto indipendente aumenta S di un delta δ dato dal numero di occorrenze del conflitto p per il numero medio di cicli di stallo introdotti c per evitarlo:

- Conflitti di dipendenza di dato: $\sigma_{dato} = p_{dato} \cdot c_{dato}$
- Conflitti di salto: $\sigma_{salto} = p_{dato} \cdot c_{dato}$
- Conflitti di cache miss: $\sigma_{miss} = p_{dato} \cdot c_{dato}$

Il **CASO OTTIMALE** della pipeline è 1 istruzione / ciclo di clock
vanno però considerati i problemi legati alle dipendenze ai salti e alle risorse limitate.

I **processori moderni utilizzano più unità aritmetiche (ALU)**, utilizzano l'emissione multipla e vengono chiamati **super scalari**.
I processori superscalari sono formati dalle seguenti componenti e buffer:
• Unità di prelievo (Fetch unit) e Coda di istruzioni
• Unità di smistamento (Dispatch unit) e Stazioni di prenotazione
• Varie unità di esecuzione e Registri temporanei
• Unità di commitment e Buffer di riordino

- Caso di un processore multiscalare con 2 unità di elaborazione:
  - **Unità aritmetica** (per eseguire le istruzioni aritmetiche e logiche)
  - **Unità Load/Store** (per eseguire le istruzioni di accesso alla memoria)
- **L'unità aritmetica** è composta da un solo stadio (esecuzione in un ciclo)
- **L'unità Load/Store** è composta da 2 stadi (un ciclo per il calcolo dell'indirizzo e uno per l'accesso alla memoria)
- Il banco di registri deve permettere la lettura di 4 registri e la scrittura di 2 in un solo passo



Può avvenire che due istruzioni dipendenti vengano inviate a due unità di esecuzione differenti e non se ne possa garantire l'ordine di esecuzione

In questo caso bisogna bloccare l'esecuzione dell'istruzione con dipendenze di dato finché l'altra istruzione non sia stata eseguita

Dei buffer specifici chiamati **stazioni di prenotazione** sono presenti all'ingresso di ciascuna unità di esecuzione. Essi contengono:
• **L'istruzione smistata in attesa di esecuzione**
• **Informazioni e operandi rilevanti a ciascuna istruzione smistata**
le stazioni di prenotazione sono una **tecnica utilizzata nei processori superscalari** con esecuzione fuori ordine per gestire le dipendenze di dato e migliorare l'efficienza della pipeline (sono dei buffer che memorizzano istruzioni e operandi)

• **A causa di cache miss o eccezioni possono avvenire esecuzioni di istruzioni fuori ordine**

• In questi casi **si rischia che un'istruzione che non sarebbe dovuta essere eseguita modifichi i contenuti di registri e/o locazioni di memoria** (eccezioni imprecise)

• **Per evitare il problema, i risultati generati dalle unità di esecuzione vengono memorizzati in registri temporanei che assumono il ruolo dei registri permanenti** (register renaming)

• L'unità di commitment ha il compito di trasferire i risultati nei registri permanenti secondo l'ordine di prelievo

• Per questo scopo viene usata una coda chiamata **buffer di riordino** dove vengono poste in coda le istruzioni nel loro ordine di prelievo.

(esecuzione fuori ordine è una tecnica avanzata utilizzata nei processori per migliorare l'efficienza nell' uso delle risorse. (Introduce una flessibilità nell' esecuzione delle istruzioni (non per forza I1,I2,I3,I4)ma può creare degli stalli.)
si possono eseguire le istruzioni in modo diverso dal modo con cui sono state fornite dal programma.)

(nei processori CISC le operazioni vengono trasformate in operazioni simili a quelle dei RISC e cosi si può utilizzare la pipeline)