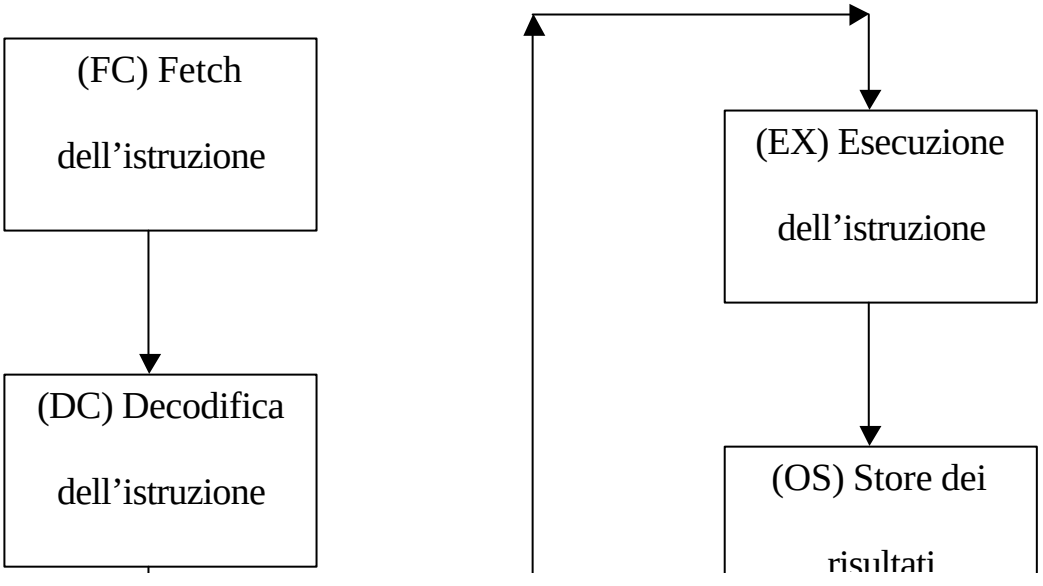
## Architetture pipeline

Un'architettura pipeline è un miglioramento di un'architettura tradizionale di un microprocessore in cui si cerca di introdurre una sorta di parallelismo temporale fra i vari componenti del microprocessore.

Supponiamo di poter suddividere il microprocessore in più stadi indipendenti fra di loro. Un esempio può essere la suddivisione del ciclo di esecuzione di un'istruzione in più fasi, ad esempio



Supponiamo di poter suddividere il microprocessore in più stadi, ciascuno dei quali si occupi di una fase dell'esecuzione dell'istruzione.

In un microprocessore tradizionale, quando uno di questi stadi termina l'esecuzione delle operazioni di sua competenza si pone in stato di attesa o idle finché tutti gli stadi non hanno completato l'esecuzione dell'istruzione come nello schema temporale seguente. Supponendo che ogni stadio completi le sue elaborazioni in un ciclo macchina occorrono, per l'esecuzione di una singola istruzione, tanti cicli macchina quanti sono gli stadi (nell'esempio 5) . Vedi foglio

| Ciclo di clock | **T 1** | **T 2** | **T 3** | **T 4** | **T 5** | **T 6** | **T 7** | **T 8** | **T 9** | **T 10** | **T 11** | **T 12** | **T 13** | **T 14** | **T 15** |
|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|
| FASI DI ESECUZIONE | FC 1 | | | | | FC 2 | | | | | FC 3 | | | | |
| | | DC 1 | | | | | DC 2 | | | | | DC 3 | | | |
| | | | FO 1 | | | | | FO 2 | | | | | FO 3 | | |
| | | | | EX 1 | | | | | EX 2 | | | | | EX 3 | |
| | | | | | OS 1 | | | | | OS 2 | | | | | OS 3 |
| | Istruzione 1 | | | | | Istruzione 2 | | | | | Istruzione 3 | | | | |

In un’architettura pipeline ideale, invece, ogni stadio, per evitare questi tempi morti, dopo aver completato le elaborazioni di sua competenza per l’istruzione in corso, passa ad elaborare l’istruzione successiva, come nel seguente schema temporale. In questo caso l’esecuzione della prima istruzione necessita ancora di tanti cicli macchina quanti sono gli stadi, ma la seconda istruzione verrà completata soltanto dopo un solo ciclo di clock aggiuntivo, così come tutte le istruzioni seguenti. (vedi foglio)

| Ciclo di clock | **T 1** | **T 2** | **T 3** | **T 4** | **T 5** | **T 6** | **T 7** | **T 8** | **T 9** | **T 10** |
|---|---|---|---|---|---|---|---|---|---|---|
| FASI DI ESECUZIONE | FC 1 | FC 2 | FC 3 | FC 4 | FC 5 | FC 6 | FC 7 | FC 8 | FC 9 | FC 10 |
| | | DC 1 | DC 2 | DC 3 | DC 4 | DC 5 | DC 6 | DC 7 | DC 8 | DC 9 |
| | | | FO 1 | FO 2 | FO 3 | FO 4 | FO 5 | FO 6 | FO 7 | FO 8 |
| | | | | EX 1 | EX 2 | EX 3 | EX 4 | EX 5 | EX 6 | EX 7 |
| | | | | | OS 1 | OS 2 | OS 3 | OS 4 | OS 5 | OS 6 |
| | ISTRUZIONE 1 | ISTRUZIONE 2 | ISTRUZIONE 3 | ISTRUZIONE 4 | ISTRUZIONE 5 | ISTRUZIONE 6 | ISTRUZIONE 7 | ISTRUZIONE 8 | ISTRUZIONE 9 | ISTRUZIONE 10 |

Il pipeline dell’esecuzione delle istruzioni è solo un possibile esempio (i processori di questo tipo sono detti sistemi look-ahead). Ci sono altri esempi di architettura pipeline in cui ad esempio, è l’ALU ad essere organizzata con architettura pipeline.

In generale, supponendo che il pipeline sia costituito da k stadi, l’esecuzione di n processi avviene in un numero di cicli macchina pari a

$$T_K=k+(n-1)^1$$

Mentre senza pipeline ne sarebbero occorsi

$$T_1=k*n$$

Il rapporto fra queste due grandezze

$$S_K = \frac{T_1}{T_K} = \frac{kn}{k+(n-1)}$$

viene detto fattore di accelerazione o speed-up.

Come si può intuire, tale fattore di accelerazione aumenta al crescere del numero di processi elaborati, in quanto diminuisce il peso del tempo necessario ad eseguire il primo processo.ma quest'accelerazione ha un limite superiore. Infatti, se calcoliamo il limite

$$\lim_{n\longrightarrow\infty} S_K = \lim_{n\longrightarrow\infty} \frac{T_1}{T_K} = \lim_{n\longrightarrow\infty} \frac{kn}{k+(n-1)} = k$$

vediamo che l'accelerazione ideale è pari al numero di stadi del pipeline.

## Problemi del pipeline

La situazione precedentemente descritta è ideale. Nella pratica la progettazione di un sistema pipeline ci si scontra con tre tipologie di problemi

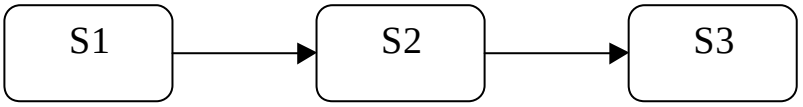
---

[1] Occorrono un numero di cicli macchina pari al numero degli stadi per eseguire la prima istruzione (k) e un ciclo macchina aggiuntivo per ogni altro processo restante (e sono rimasti n-1)

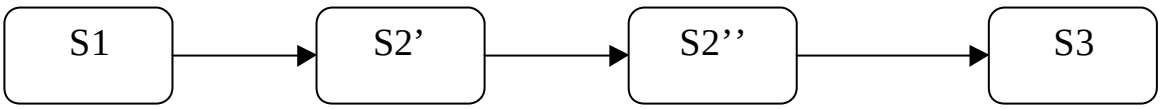
**Bottleneck**

Si ha il problema del collo di bottiglia quando uno degli stadi in cui è suddiviso il pipeline, risulta molto più lento degli altri. Per sincronizzare i vari stadi del pipeline occorre che la durata del clock del pipeline sia cadenzata sullo stadio più lento

Supponiamo di avere un pipeline su tre stadi



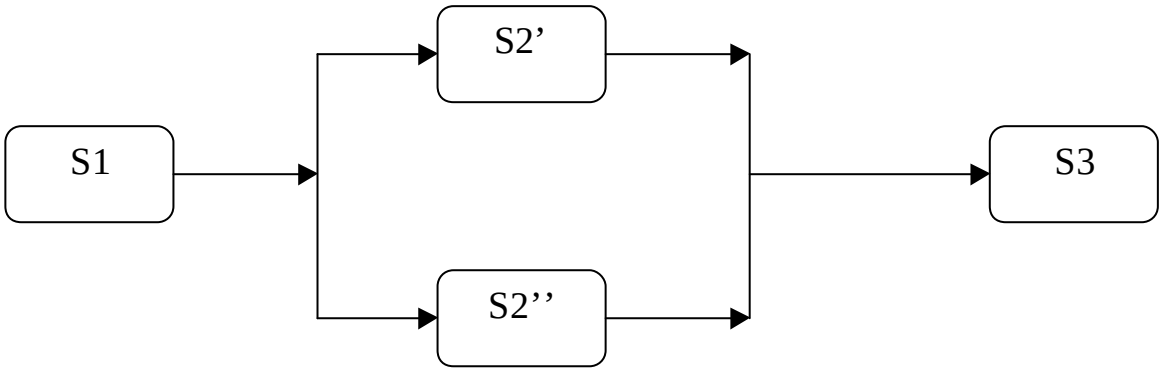

E che il primo e il terzo stadio impieghino per le loro elaborazioni 10 nanosecondi mentre il secondo stadio impiega 100 nanosecondi. L'esecuzione di un processo senza pipeline, poiché i tre stadi non sono costretti a sincronizzarsi, impiegherà 220 nanosecondi. In un'architettura pipeline, invece, i tre stadi vanno sincronizzati per cui occorre che essi agiscano tutti con la stessa velocità. In tal caso l'esecuzione di un'istruzione necessiterà di 300 nanosecondi. Per eseguire 5 processi, senza il pipeline avremo bisogno di 5*(10+100+10)=600 nanosecondi mentre con pipeline avremo bisogno di [3+(5-1)]*100 = 700 nanosecondi. Ci può essere un miglioramento delle prestazioni con il pipeline ( vedi 1 e 2) ma comunque si nota come le prestazioni del pipeline sono profondamente deteriorate. Tale problema può essere risolto se si tenta di suddividere lo stadio più lento in più stadi




Se riusciamo a suddividere tale stadio in due stadi che impiegano un tempo di 50 nanosecondi ciascuno, il tempo necessario per eseguire 5 processi diventa

[4+(5-1)]*50 = 400 nanosecondi

IL risultato massimo si avrebbe se riuscissimo a fare in modo che questi due sottostadi potessero agire in parallelo

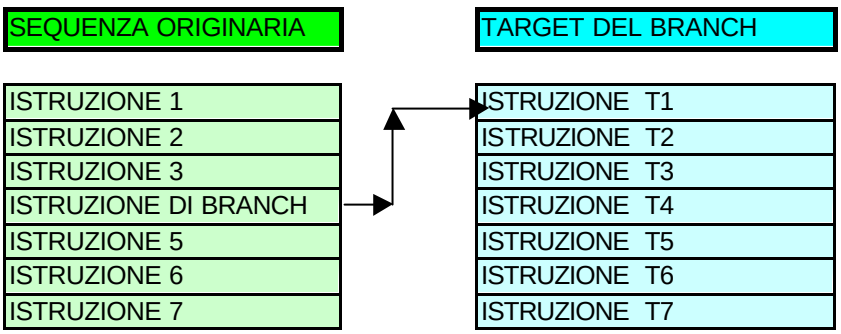



In tal caso per eseguire 5 processi sarà necessario un tempo [3+(5-1)]*50 = 350 nanosecondi. (Vedi 1 e 2)

**Trattamento delle istruzioni di branch**

Per istruzioni di branch si intendono tutte quelle istruzioni che modificano la normale sequenza di esecuzione di un programma come istruzioni di salto condizionato e incondizionato, chiamate di sottoprogrammi, ritorno da sottoprogramma.

Queste istruzioni costituiscono un problema per il pipeline poiché esse comportano, se eseguite, uno sconvolgimento della normale sequenza di esecuzione delle istruzioni e l’esecuzione di istruzioni presenti in un’altra area di memoria detta target dell’istruzione di branch.



nell’esempio di figura dopo l’istruzione di branch dovrebbe essere eseguita l’istruzione 5 ma se l’istruzione di branch viene eseguita, la prossima istruzione deve essere l’istruzione T1. nel frattempo però, il pipeline si è riempito delle istruzioni 5, 6 e così via che, se il salto viene effettuato, non dovrebbero essere eseguite.

Occorre dunque ripulire il pipeline da queste istruzioni e riempirlo con le istruzioni del target.

Ciò comporta naturalmente un rallentamento del pipeline.

**Dipendenza dai dati**

Si ha il problema della dipendenza dai dati quando un’istruzione dipende dai risultati di un’istruzione precedente. Ad esempio

- SOMMA A,B
- INCREMENTA A

L’istruzione due non può essere eseguita finché l’istruzione precedente non è stata completata, poiché altrimenti non agirebbe sul contenuto di A+B come richiesto dal programma.

Una soluzione è il cosiddetto tagging delle risorse. In sostanza le risorse del processore quali i registri vengono dotate di un bit busy che, ad esempio, se è settato ad 1 indica che la risorsa è utilizzata da un’istruzione, mentre se è a zero, indica che la risorsa non è impegnata. Nel nostro esempio l’istruzione di somma provocherebbe il settaggio dei bit di busy dei registri A e B. La seconda istruzione troverebbe che il registro A è impegnato da un’istruzione già in esecuzione e si bloccherebbe finche il registro ridiventa disponibile.