

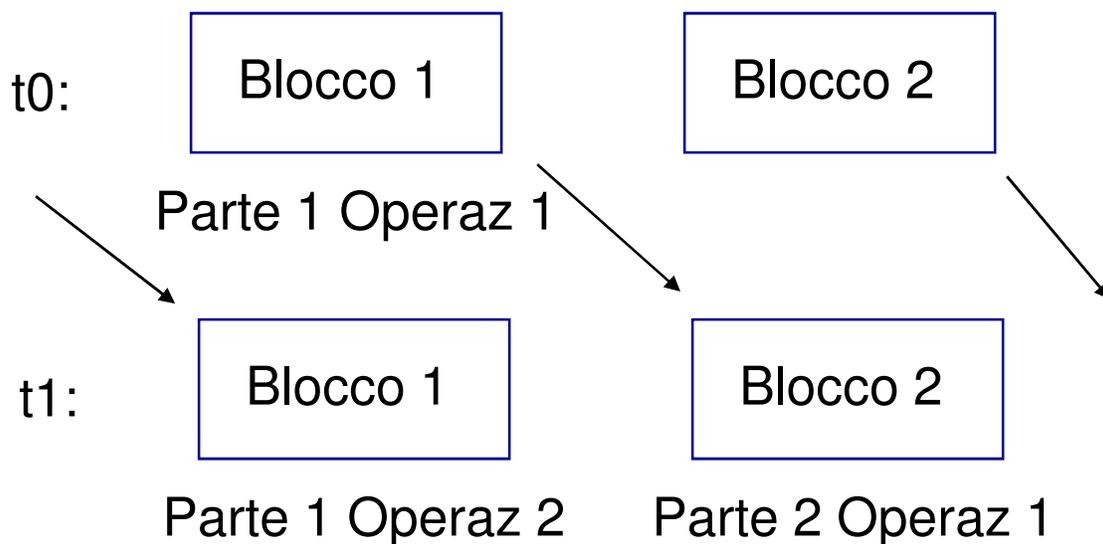
## Architetture “moderne”

- ◆ Esecuzione delle istruzioni in pipeline
- ◆ Predizione dei branch
- ◆ Multiple issue
- ◆ Register renaming
- ◆ Esecuzione fuori ordine
- ◆ Cache non bloccanti

# Pipelining

## PIPELINING

- ◆ utile per eseguire una sequenza di operazioni simili
- ◆ ciascuna operazione viene scomposta in parti
- ◆ le parti sono eseguite in successione su blocchi funzionali diversi
- ◆ la sequenza di operazioni è eseguita in modo che ogni blocco operi contemporaneamente agli altri su parti diverse di operazioni in sequenza



# Pipelining nell'esecuzione delle istruzioni

Il pipelining può essere utilmente applicato per velocizzare l'esecuzione delle istruzioni,

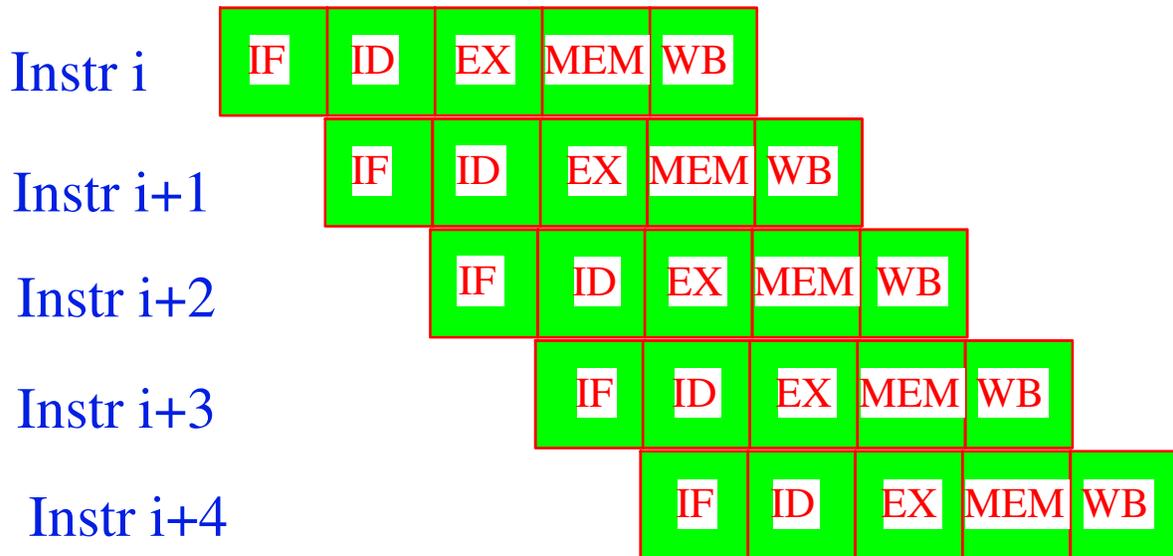
Se:

- ◆ la sovrapposizione di istruzioni non crea conflitti di risorse (i blocchi hanno risorse indipendenti)
- ◆ tutte le parti hanno la stessa velocità (carico bilanciato)
- ◆ i registri possono essere letti e scritti contemporaneamente
- ◆ ogni stadio prevede REGISTRI LATCH che memorizzano il data path e le istruzioni con i relativi segnali di controllo tra ogni stadio ed il successivo

→ lo speedup tra esecuzione sequenziale delle istruzioni e esecuzione in pipeline è pari al numero di stadi della pipeline

# Esecuzione in pipeline

**DLX:**

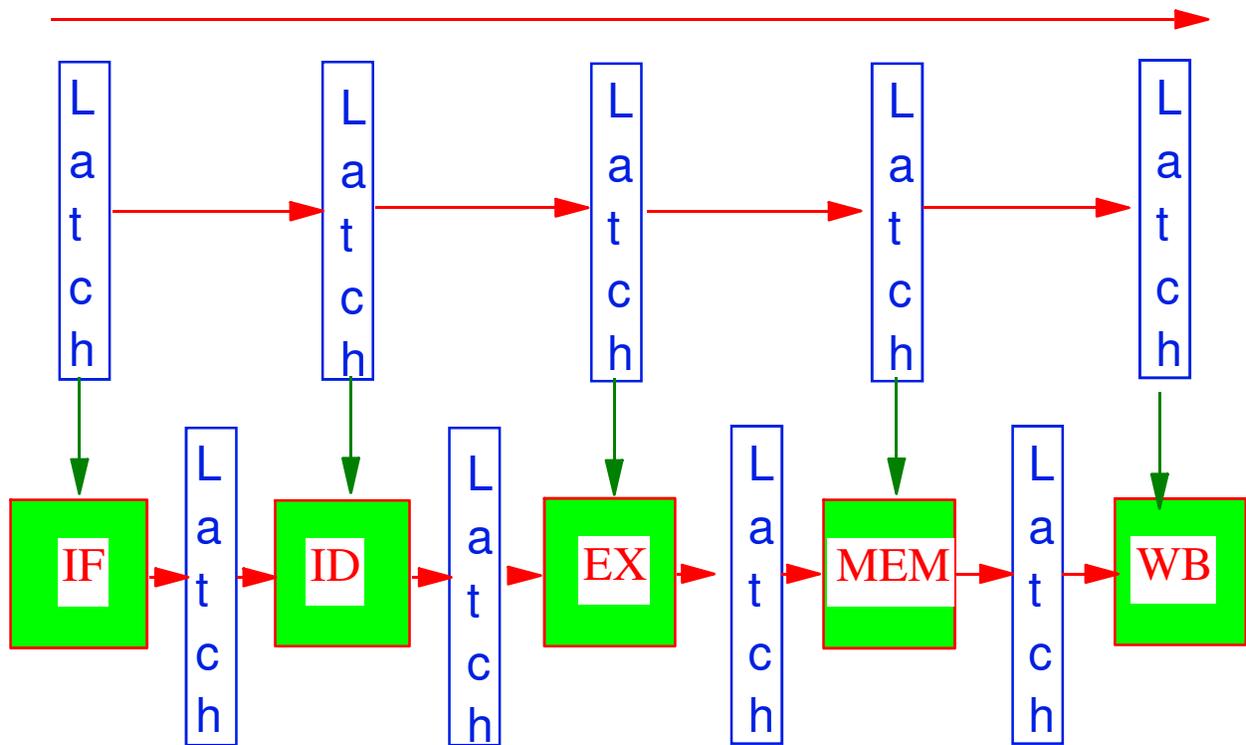


$$T_{\text{exec pipeline}} = \frac{T_{\text{exec seq}}}{N_{\text{stadi}}}$$

A parità di tempo di esecuzione della singola istruzione l'aumento degli stadi aumenta lo speedup

# Pipelining

Percorso delle istruzioni



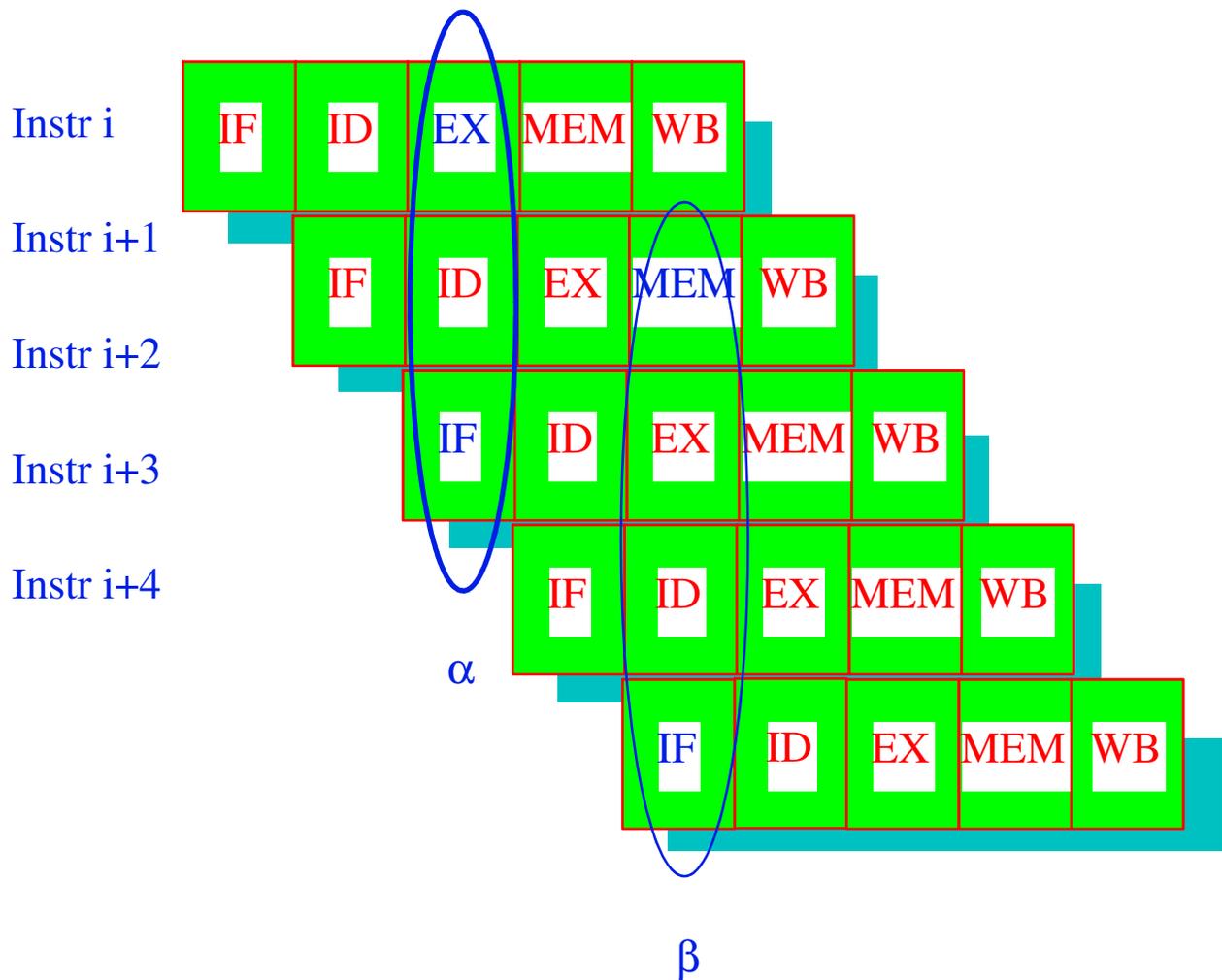
Percorso dei dati

# Alee nelle pipeline

Esistono tre limiti all'aumento del numero degli stadi della pipeline (non idealità):

- **Alee strutturali:** conflitto di risorse: a due blocchi occorre la stessa risorsa, esempio la **ALU**
- **Alee di dato:** a un'istruzione serve un **risultato non ancora pronto**, in quanto l'istruzione che la precede nel programma è ancora in corso di esecuzione
- **Alee di controllo:** il processore **non sa dove trasferire il controllo** ("saltare") perché non è ancora stato determinato se e dove saltare

# Alee strutturali



$\alpha$  – Conflitto sulla ALU fra EX (operazioni) e IF (incremento PC)

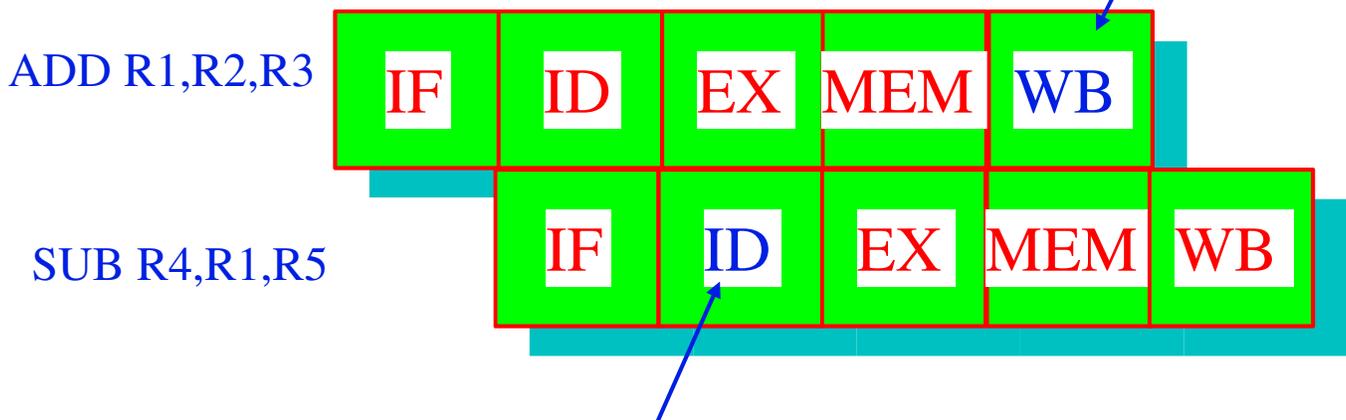
$\beta$  – Conflitto per il bus fra MEM e IF

## Alee di dato

- ADD R2,R3,R1       $R1 \leftarrow R2 + R3$

- SUB R1,R5,R4       $R4 \leftarrow R1 - R5$

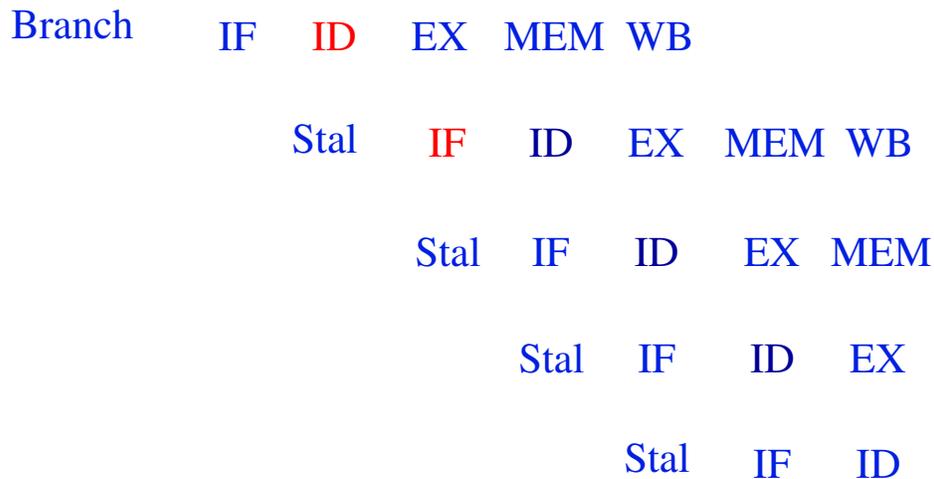
Qui R1 in RF



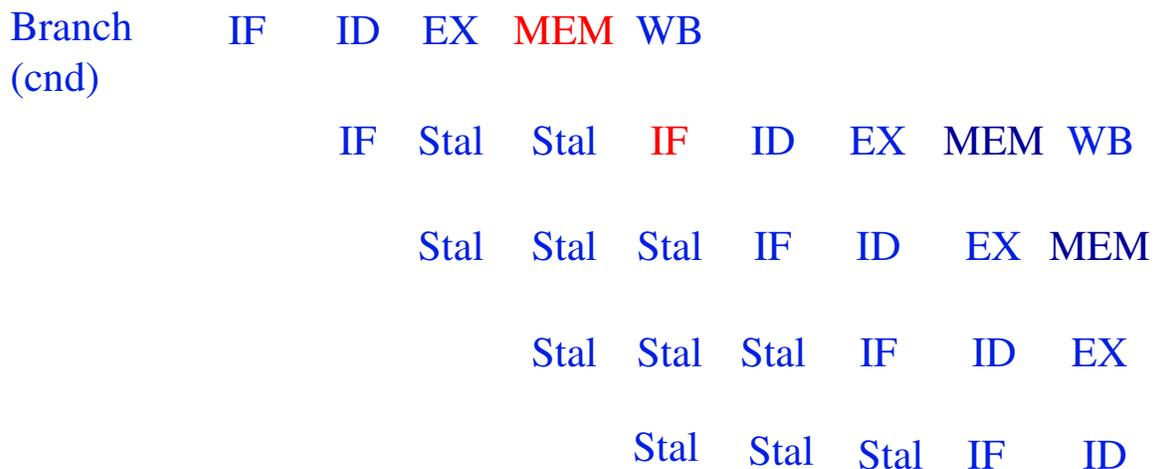
Qui è richiesto R1 (da porre nel registro A di ingresso alla ALU)

- L'istruzione SUB richiede la presenza nel Register File di R1 per eseguire lo stadio ID
- Unica soluzione: **ANTICIPAZIONE (FORWARDING)**

# Alee di controllo



Un salto introduce uno stallo (1 clock sprecato) nella pipeline perché l'indirizzo a cui saltare è noto solo dopo la fase di decode



Un salto condizionale introduce 3 stalli nella pipeline perché la condizione di salto è valutata solo dopo la fase di MEM (vedi lucidi sul DLX)



# Multiple-issue

Obiettivo: **ottenere un CPI < 1**

Una singola pipeline può ottenere al massimo

CPI = 1

→bisogna completare più di un'istruzione per ciclo

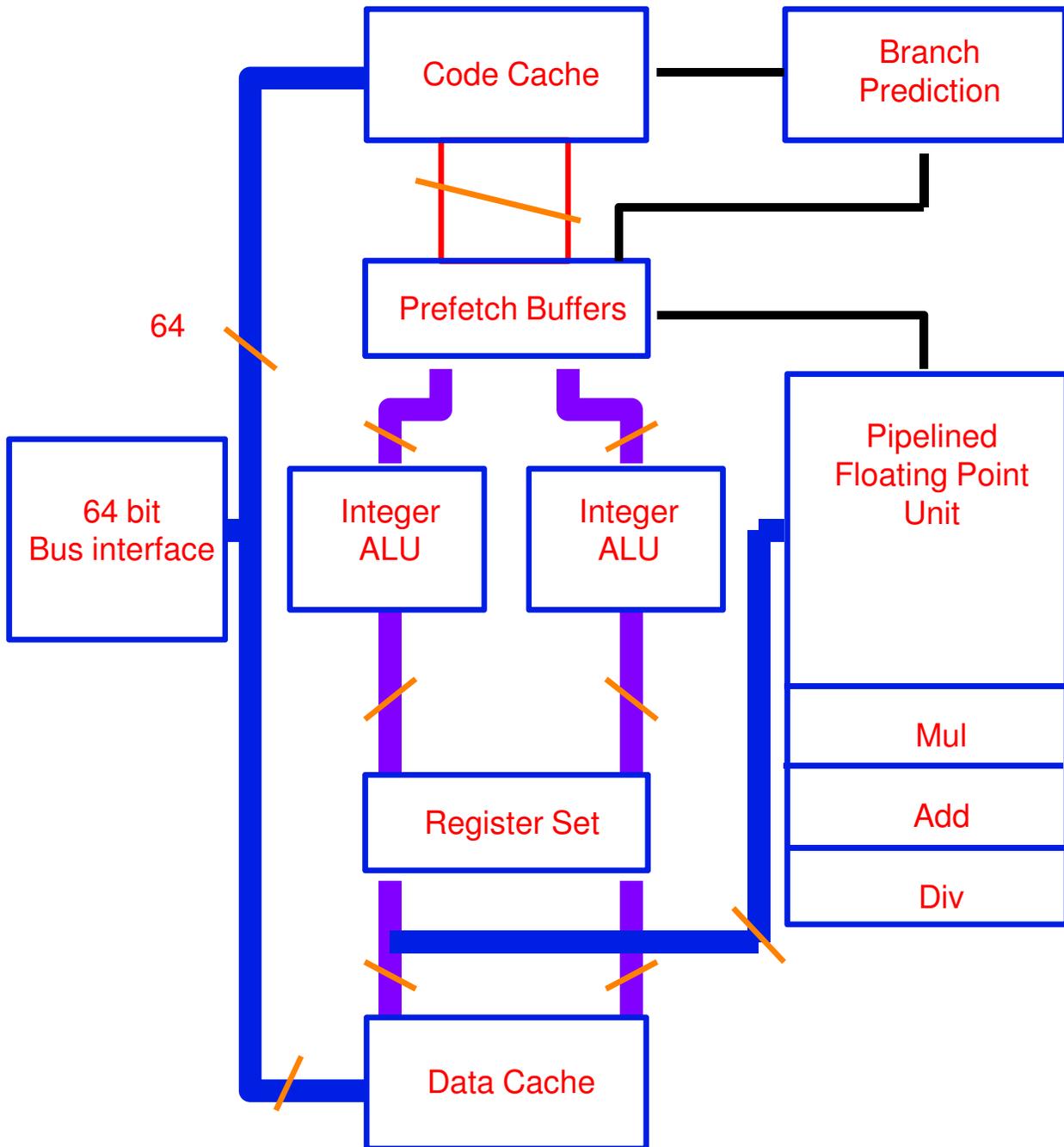
## 1) SUPERSCALARE

- più pipeline che operano contemporaneamente
- il programma è specificato nel comune modo sequenziale
- il processore verifica a run-time se è possibile eseguire sulle varie pipeline diverse istruzioni in parallelo, nel rispetto delle operazioni specificate dal programma sequenziale

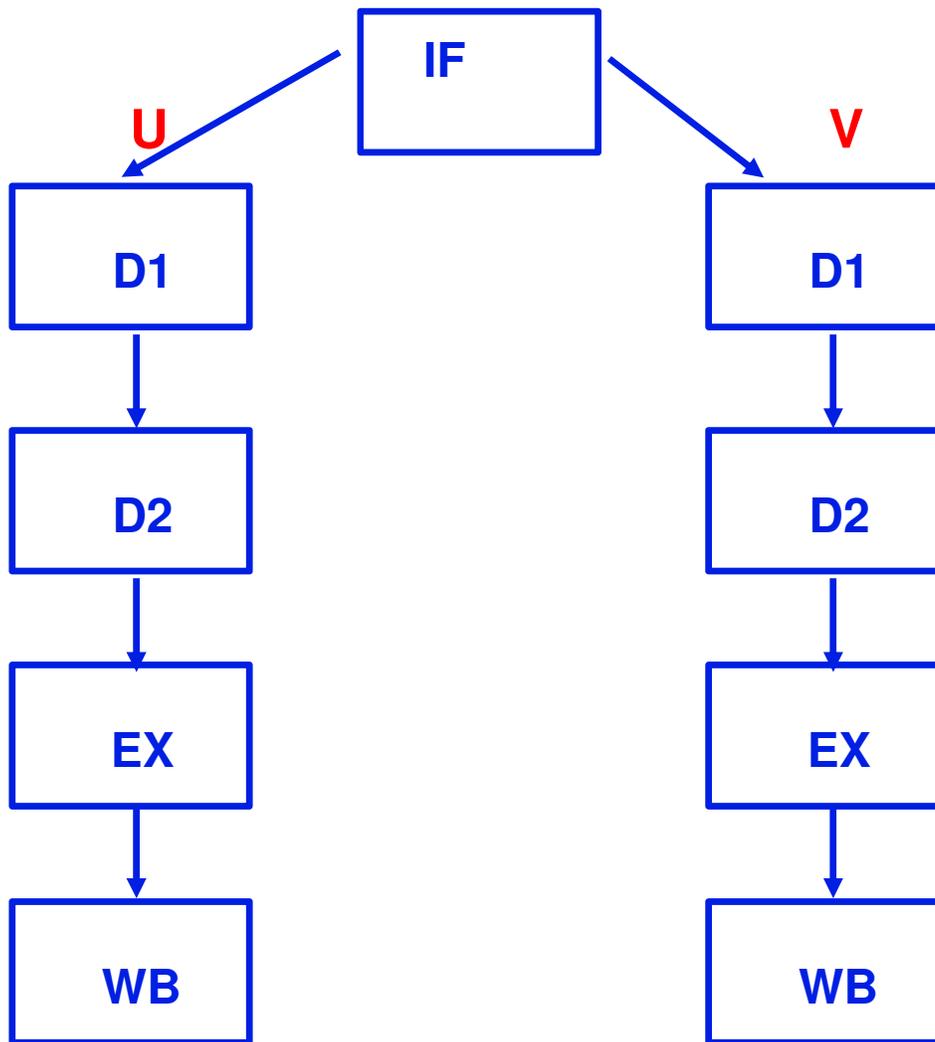
## 2) VLIW (Very Large Instruction Word)

- più unità che operano contemporaneamente
- il programma è specificato in modo parallelo, assegnando a ciascuna unità le operazioni da eseguire
- Speed-up potenziale maggiore, perché il programma ha visibilità delle risorse parallele (parallelismo esplicito)

# Architettura Pentium

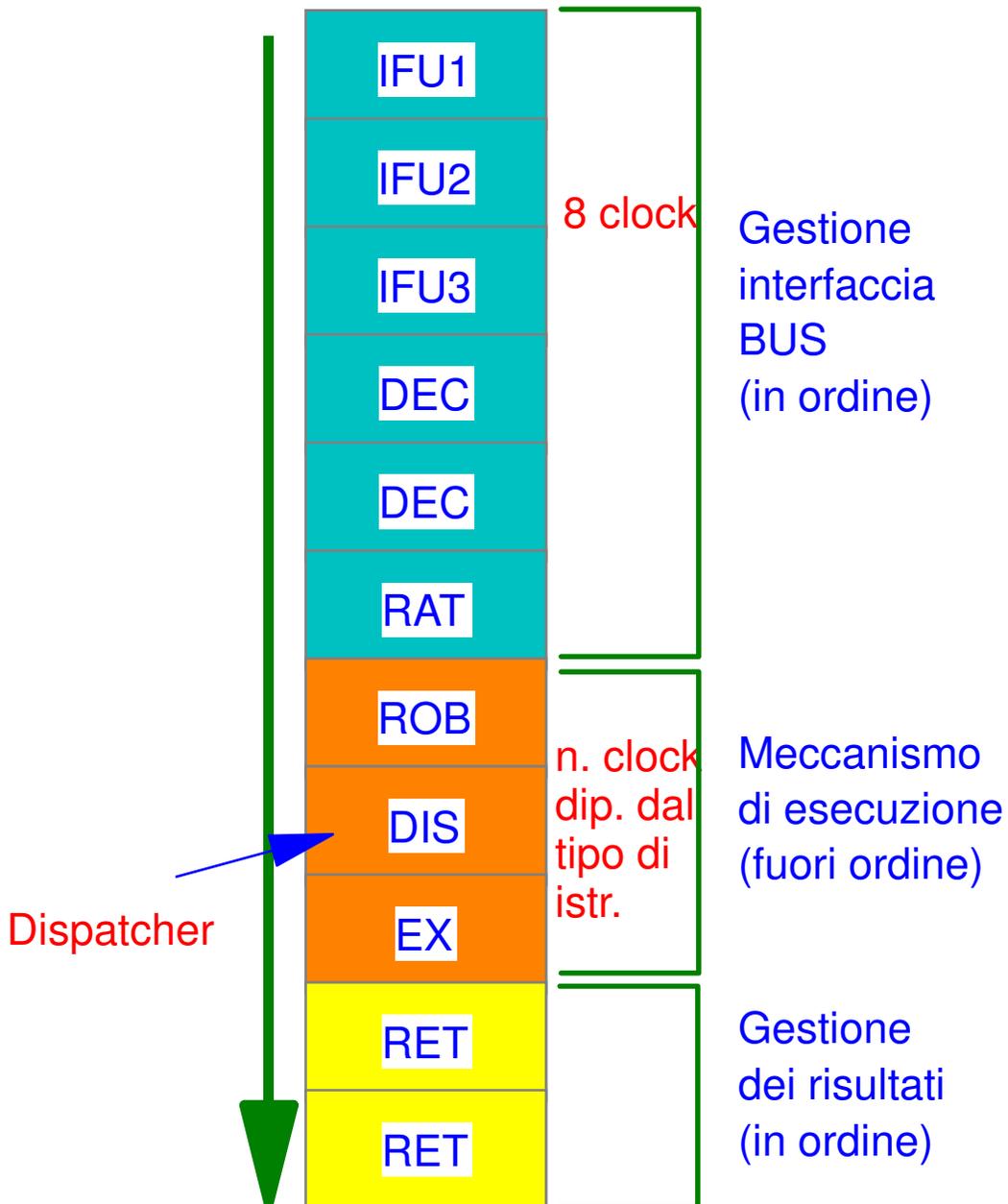


# Pipeline Pentium



Il processore Pentium ha due pipeline di 5 stadi ciascuna operanti in parallelo

# Pipeline P6 (Pentium Pro, II, III)



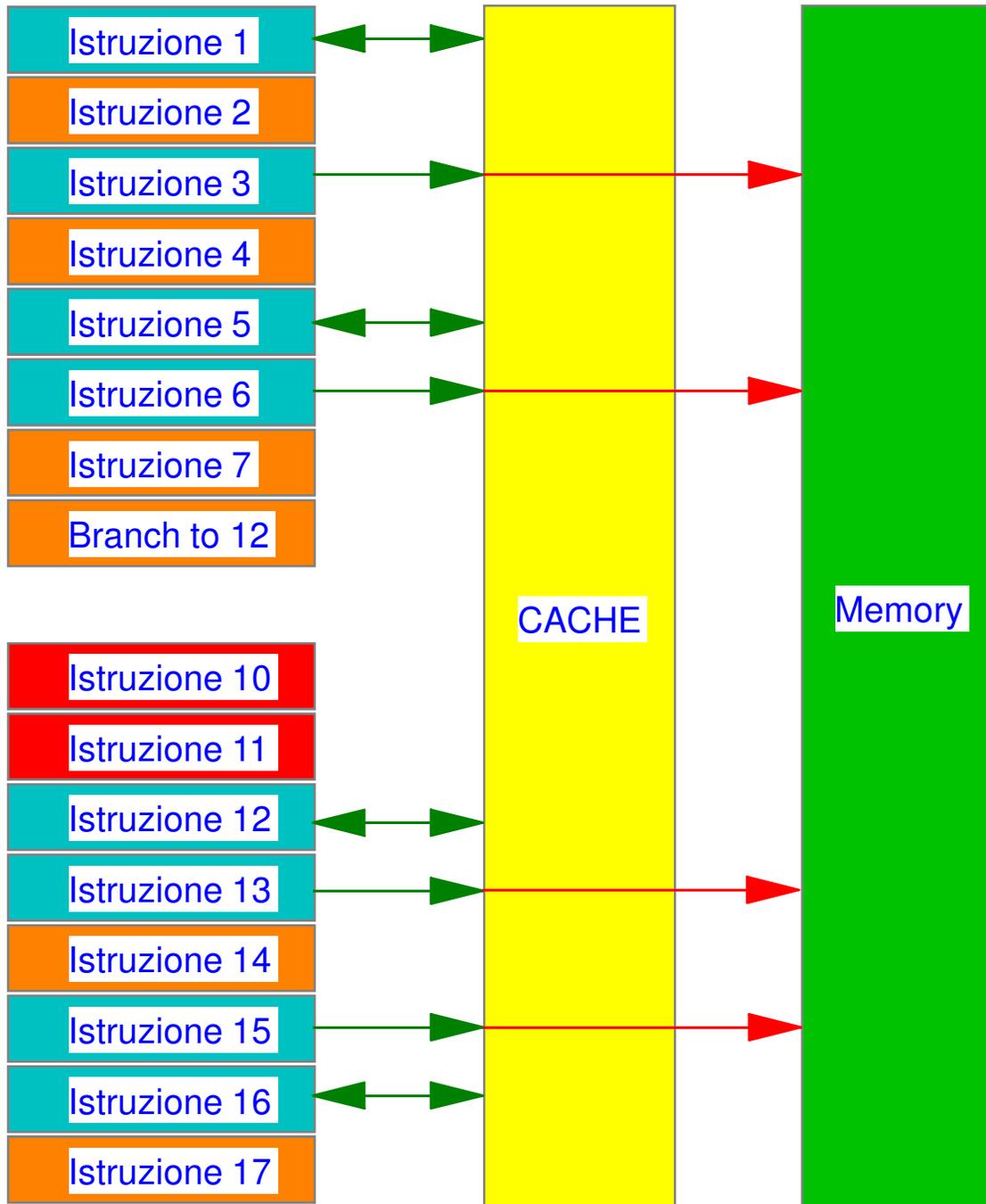
## Descrizione della pipeline

- ◆ IFU1: carica una linea di 32 byte dalla cache delle istruzioni al prefetch buffer
- ◆ IFU2: identifica l'inizio delle varie istruzioni nel blocco e presenta i salti al BTB (predizione branch)
- ◆ IFU3: allinea le istruzioni per presentarle ai decoder
- ◆ DEC1: decodifica le istruzioni nelle corrispondenti micro-ops (istruzioni semplici, **1 o più**). **Le micro-ops sono istruzioni di tipo RISC.** Fino a **3 istruzioni** possono essere decodificate **contemporaneamente** (vedi due pagg. dopo).
- ◆ DEC2: passa le micro-ops nella coda delle istruzioni decodificate
- ◆ RAT (Register Alias Table & Allocator stage): se la micro-op ha operandi sorgente, all'operando sorgente viene associato un registro o una entry della ROB table.

- ◆ ROB (ReOrder Buffer): le istruzioni entrano nel buffer di esecuzione in ordine di programma. **Il ROB è un buffer circolare con al massimo 40 micro-ops, puntato da due puntatori: start-of-buffer, end-of-buffer.**
- ◆ DIS: le micro-ops sono **a 3 alla volta** portate di fronte alle unità di esecuzione (dispatcher)
- ◆ EX: esegue le micro-ops; in quanto istruzioni semplici, sono eseguite quasi tutte in un ciclo di clock.
- ◆ RET1: le istruzioni completate (comprese quelle che dipendono da salti condizionali precedenti) sono marcate “pronte per il ritiro”
- ◆ RET2: quando le istruzioni precedenti hanno completato l'esecuzione, e tutte le micro-ops dell'istruzione corrente sono completate, gli effetti sui registri dell'istruzione corrente sono fissati, in stretto ordine di programma.



## Cache P6



- Le CACHES sono NON BLOCCANTI (4 miss contemporanei) sul bus fino a 8 richieste pendenti