

## Lezione 30-09-08

### Pipeline (vedi slide da 78 a 84)

“In informatica col termine pipeline si indica un processo di esecuzione (ad esempio l’esecuzione di un’istruzione) realizzato in più fasi, dette stadi.. Ogni fase, fisicamente realizzata in un blocco circuitale, provvede a ricevere in ingresso un dato, ad elaborarlo e poi a trasmetterlo all’elemento successivo. Il flusso di dato, nel nostro caso l’istruzione, percorre tutti gli elementi della pipeline.” Le architetture pipeline realizzano un incremento delle prestazioni a livello di Instruction Level Parallelism (ILP).

I processori antecedenti al Pentium realizzavano la pipeline “imperfetta”; dal Pentium in poi si ha una pipeline (quasi) perfetta.

E’ pertanto necessario suddividere le istruzioni in elementi atomici (non ulteriormente suddivisibili), assegnando ciascun elemento ad uno stadio.

Il principio di funzionamento di una pipeline è paragonabile a quello di una catena di montaggio.

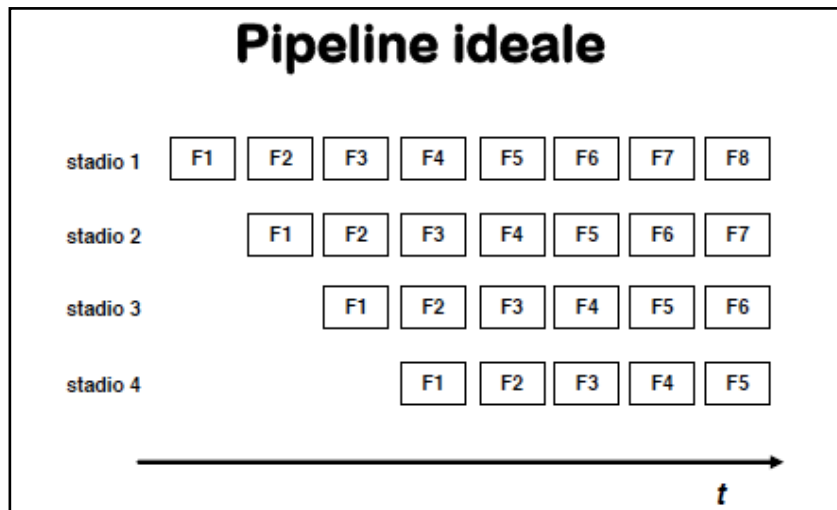
L’assemblaggio è caratterizzato da diversi stadi da eseguire in un ordine fisso e da un determinato tempo di esecuzione. I vari stadi produttivi sono posti in cascata e lavorano in modo che l’uscita di uno stadio coincida con l’ingresso dello stadio seguente. A regime, ad ogni ciclo di clock viene terminata un’istruzione.

Si ha una pipeline ideale (o perfetta) quando la durata di esecuzione di tutte le fasi è identica (costante ed uguale). Se ciò non avviene si hanno dei periodi di inattività (pipeline imperfetta).



Nel caso in cui il tempo di stadio sia uguale in tutti gli stadi:

$$[\text{NumeroElementiPipeline}] \cdot [\text{TempoStadio}] = [\text{DurataIstruzione}]$$



Nella progettazione di pipeline verso il pipeline perfetto si devono trovare gli stadi di pipeline per un'istruzione, vedere se sono omogenei e determinarne la durata massima. La frequenza di clock di riferimento è legata alla durata degli stadi di pipeline o più propriamente al maggiore periodo di tempo impiegato tra i diversi stadi. Più il tempo di un stadio è breve più il clock è veloce.

Il numero di stadi determina il throughput o meglio il numero di istruzioni eseguite al secondo.

Maggiori sono gli stadi di pipeline, più è difficile omogeneizzare la durata di tutti gli stadi (ottimizzare la pipeline).

Sia detto:

- $T_s$  il tempo di esecuzione di un'istruzione in un sistema senza pipeline
- $T_p$  il tempo di esecuzione di un'istruzione in un sistema con pipeline

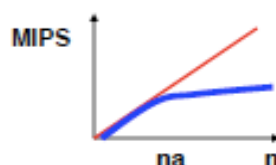
Si ha che  $T_s \leq T_p$ : infatti, il tempo di esecuzione di un singolo stadio è dimensionato al tempo massimo di esecuzione nello stadio tra tutte le istruzioni del processore e pertanto la durata di una singola istruzione in una architettura pipeline può risultare maggiore (avviene tra l'altro nel caso di pipeline imperfetto o nel caso di blocco del pipeline perfetto).

Se si considera il throughput, le prestazioni vanno valutate in MIPS:

- $MIPS_s = 1/T_s$  (sistema senza pipeline)
- $MIPS_p = n/T_p$  (sistema con pipeline), con  $n$  numero di stadi

Si ha che  $MIPS_s \ll MIPS_p$ .

In teoria, le prestazioni crescono linearmente al crescere di  $n$ . In realtà, per  $n \geq n_a$  l'aumento di prestazioni è quasi nullo poiché al crescere di  $n$  non si riesce più a soddisfare il vincolo di pipeline ottimo.



## Valutazione delle prestazioni (vedi slide 85-86)

Siano:

- $k$  = numero stadi pipeline
- $\tau_i$  = tempi dell' $i$ -esimo stadio
- $\tau_{\max} = \max(\tau_i)$

Se consideriamo le  $N$  istruzioni, si ha:

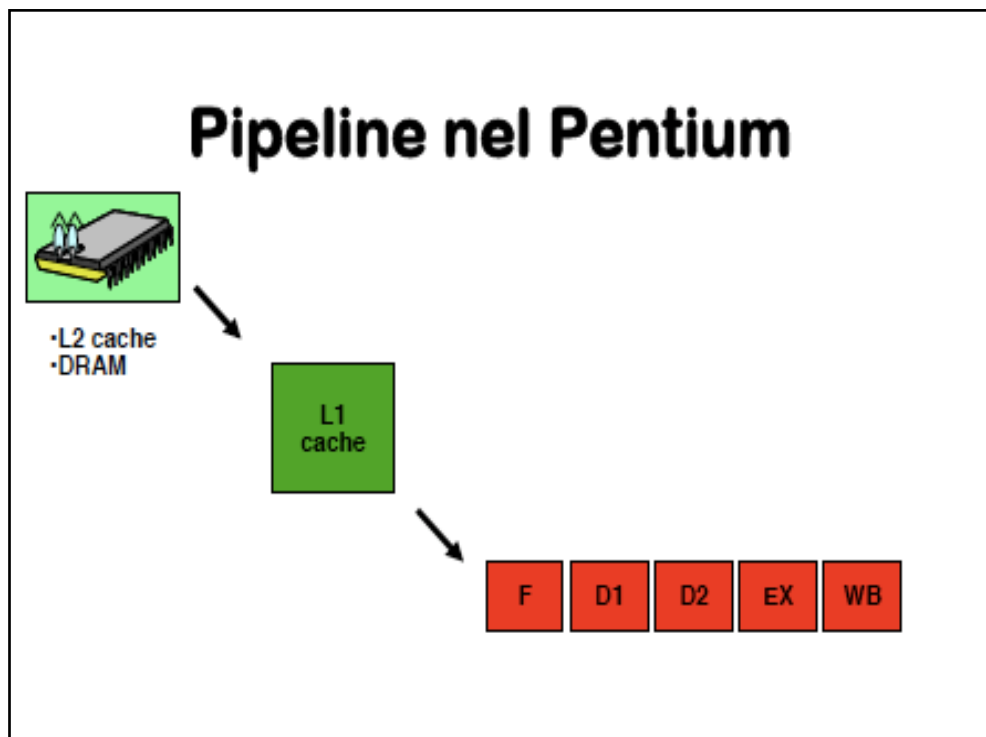
- Tempo di esecuzione di una istruzione:  $k\tau_{\max}$
- Tempo di esecuzione di  $N$  istruzioni:  $Nk\tau_{\max}$
- Tempo di esecuzione di  $N$  istruzioni con pipeline:  $k\tau_{\max} + (N-1)\tau_{\max}$

$k\tau_{\max}$  è il tempo di esecuzione della prima istruzione che deve percorrere tutta la pipeline.

Si può definire il fattore di accelerazione come il rapporto tra il tempo di esecuzione di  $N$  istruzioni senza pipeline e il tempo di esecuzione di  $N$  istruzioni con pipeline.

$$FattAccelerazione = \frac{N \cdot k \cdot \tau_{\max}}{k \cdot \tau_{\max} + (N-1) \cdot \tau_{\max}} \text{ per } N \text{ grandi tende a } k. \text{ In MIPS } \rightarrow \frac{1}{\tau_{\max}}.$$

## Pipeline nel Pentium (vedi slide da 87 a 89)



Originariamente esistevano 5 stadi di pipeline, nelle successive versioni si passa da 12 a 30. In generale, si cerca di avere stadi con “densità di lavoro” simile e frequenza di avanzamento dettata da  $\tau_{\max}$ .

Gli stadi di pipeline sono:

- Fetch: Non è la fase di prelevamento dalla memoria in quanto il tempo di accesso a quest’ultima è molto elevato rispetto alla velocità del clock; l’istruzione viene prelevata dalla L1 cache o dalla coda di prefetch
- Decoding: Se lavoriamo in modo reale 2 stadi di decodifica risultano eccessivi, invece in modo protetto dobbiamo eseguire 2 cose:
  - passaggio da indirizzo virtuale (logico) ad indirizzo reale (calcolo degli indirizzi); devono essere esaminate le opportune tabelle (D2)
  - verificare l’eseguibilità e la legittimità (privilegi) dell’istruzione (D1)
- Execution: Vengono eseguite le operazioni della ALU e settati i flag del PSW (per l’FPU c’è un altro pipeline)
- Write-Back: risultati scritti o nel registro o nella L1 cache o in memoria

Ogni stadio richiede un ciclo di clock; in alcune condizioni in uno stadio si inseriscono più periodi di clock, come nel caso dell’accesso alla memoria. In altre situazioni, ad esempio salti condizionati, si può determinare uno stallo completo e la conseguente necessità di svuotare la pipeline.

Esempio:

**ADD AX, [BX]**

- F: Devo andare a prelevare l’istruzione puntata da CS:IP, mandare sulla BIU (Bus Interface Unit) interna questo indirizzo e capire se si trova in memoria, in coda o in cache. Concettualmente viene preso il contenuto di CS:IP ed inserito nell’IR (fetch di 2 byte)
- D1: Dato che qui siamo in modo reale ed è tutto legittimo c’è solo l’interpretazione del codice operativo.
- D2: Si procura il dato relativo a [DS:BX]; passa dall’indirizzo segmentato a quello da spedire nell’ABUS (calcolo dell’indirizzo) e deve fare il fetch del dato (2 byte di AX). Si può avere un’altra situazione di stallo legata ai MISS; quindi i MISS possono verificarsi sia in F che in D2.

- **EX**: Inserisce il valore del dato nel registro temporaneo dell'ALU. I Pentium hanno un moltiplicatore hardware. Si possono verificare criticità (stallo di esecuzione).
- **WB**: In questo caso non ci sono possibili criticità in quanto andiamo a scrivere in AX. Deve essere calcolato anche il nuovo Program Counter.

Il tempo reale di questa istruzione è il tempo di permanenza lungo la fase di pipeline:

$$T_{ADD} = \tau_F + \tau_{D1} + \tau_{D2} + \tau_{EX} + \tau_{WB}$$

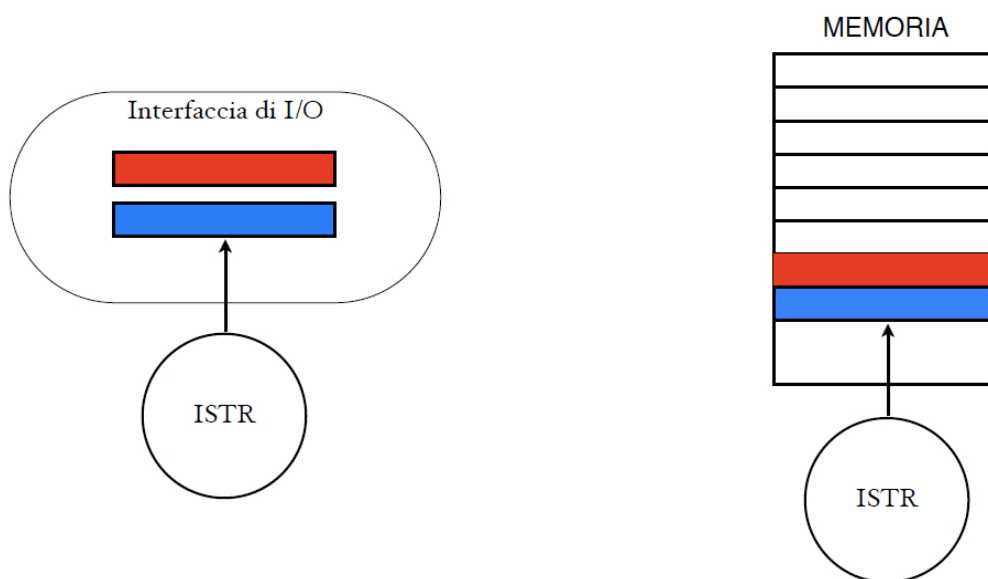
$$\text{Best case: } T_{ADD} = T_p + T_p + T_p + T_p + T_p = 5 \cdot T_p$$

$$\text{Worst case: } T_{ADD} = m \cdot T_p + T_p + m \cdot T_p + T_p + T_p = 3 \cdot T_p + 2 \cdot m \cdot T_p = (3 + 2 \cdot m) \cdot T_p$$

dove  $m$  rappresenta il più piccolo multiplo intero di  $T_p$  che racchiude il tempo di accesso alla DRAM.

## **I/O (vedi slide 90)**

Programmare un dispositivo significa scrivere un'opportuna sequenza di bit all'interno dei registri della sua interfaccia. Questi registri, a livello assembler, possono essere visti come indirizzi a cui corrisponde un'interfaccia.



Possiamo avere due differenti filosofie progettuali:

- Isolated I/O: L'accesso alla periferica avviene attraverso un insieme di codici operativi a sé stante, ovvero attraverso speciali istruzioni di I/O.
- Memory Mapped: I registri delle interfacce della periferica vengono mappati nella memoria. In questo caso non vengono più adoperate istruzioni specifiche per gestire gli I/O, ma le tradizionali istruzioni della memoria (es: MOV AL,6).  
Questo sistema viene adoperato solo nei sistemi semplici in quanto, se voglio aggiungere memoria, ho il vincolo di non avere una contiguità fisica degli indirizzi.

### **FPU (Floating Point Unit) (vedi slide da 91 a 95)**

Ha registri espressi su 80 bit. E' una macchina a stack. IEEE754 specifica il formato dei bit (32 o 64). Lavora su 80 bit, poi approssima ai 24 più significativi.

### **MMX (MultiMedia/Matrix Math/Multiple Path eXtension) (vedi slide da 96 a 100)**

Sono state introdotte funzioni per l'elaborazione multimediale e dei pixel.

La CPU non è più scalare, ma vettoriale (SIMD), ovvero la CPU anziché utilizzare un solo operando, lavora con un vettore di n operandi (Es.: elaborazione finestre di pixel sovrapposte).

## SSE -Streaming SIMD Extensions

Somma due vettori di 4 elementi su architettura scalare

```
vec_res.x=v1.x+v2.x;  
vec_res.y=v1.y+v2.y;  
vec_res.z=v1.z+v2.z;  
vec_res.w=v1.w + v2.w
```

Somma due vettori di 4 elementi su architettura SSE

```
movaps xmm0,address-of-v1  
addps xmm0,address-of-v2  
Movaps,address-of-vec_res,xmm0
```

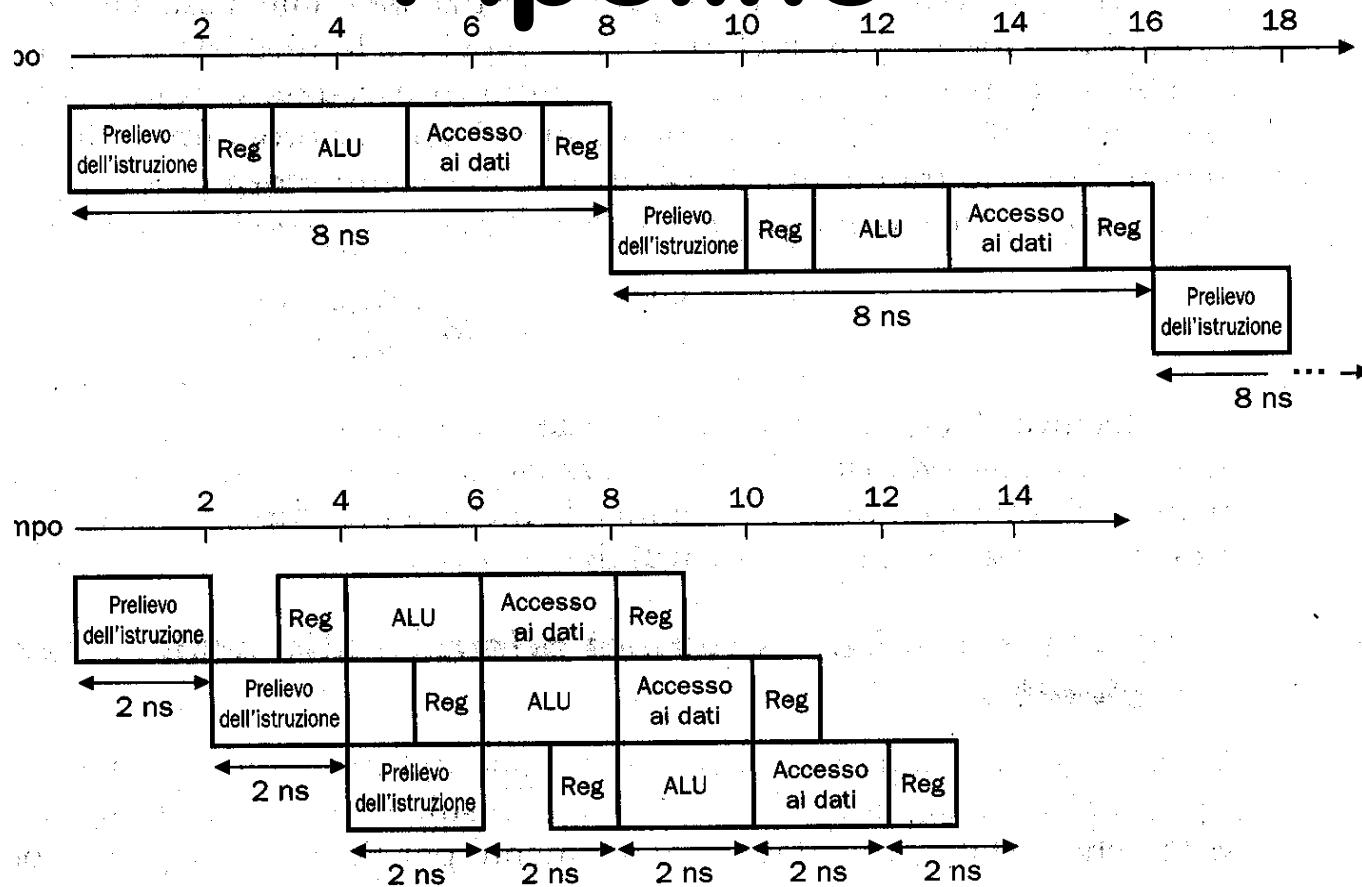
65

Prof. Marco Mezzalana - a.a. 08/09

SSE2 lavorano anche in floating point e si possono usare con i supercomputer.

# LEZIONE 30 Settembre 2008

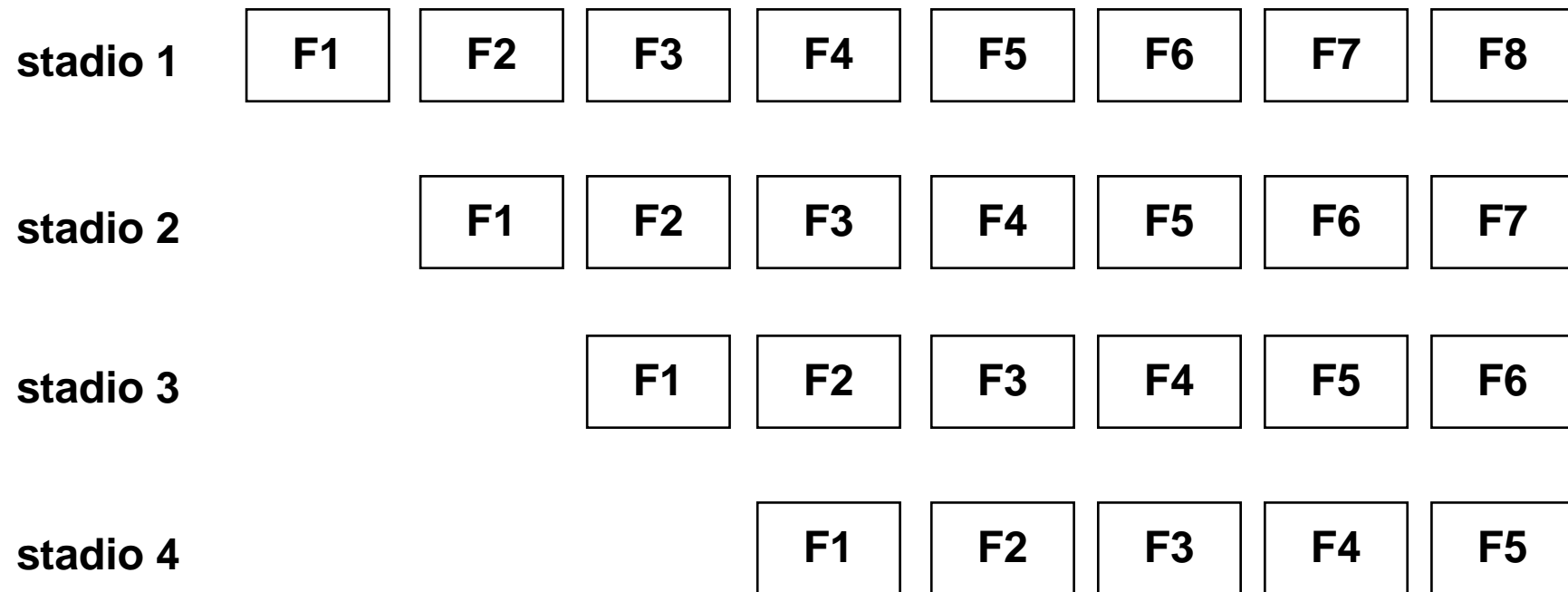
# Pipeline



Si noti che si può avere: **pipeline perfetto o ideale**, o **pipeline imperfetto** (parallelismo a livello di istruzioni)



# Pipeline ideale



$t$

# Pipeline

Sia detto:

**T<sub>s</sub>** il tempo di esecuzione di una istruzione in un sistema senza pipeline

**T<sub>p</sub>** il tempo di esecuzione di una istruzione in un sistema con pipeline

Si ha che:

**$T_s \leq T_p$**  (si veda la figura precedente)

Ciò avviene nel caso di pipeline imperfetto o nel caso di blocco del pipeline perfetto

# Pipeline

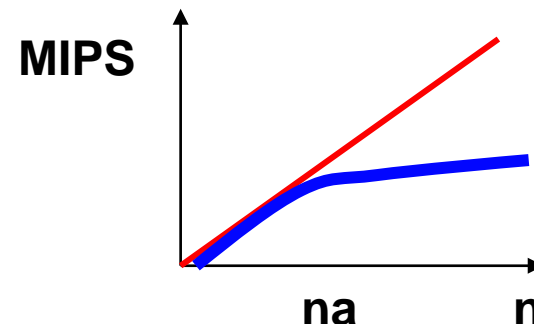
**Prestazioni**, valutate in MIPS:

**MIPS<sub>s</sub> = 1/T<sub>s</sub>** (sistema senza pipeline)

**MIPS<sub>p</sub> = n/T<sub>p</sub>** (sistema con pipeline), con n numero stadi

$$\text{MIPS}_s \ll \text{MIPS}_p$$

In teoria le prestazioni crescono linearmente con n, in realtà esiste un asindoto per  $n \geq n_a$  poiché al crescere di n non si riesce più a soddisfare il vincolo del pipeline ottimo.



# Pipeline

Un meccanismo di pipeline *ideale* è costituito da una serie di stadi che eseguono un compito specifico in un tempo di esecuzione *costante* ed *uguale* per tutti gli stadi.

Una catena di montaggio di una fabbrica automobilistica funziona con il meccanismo di pipeline. L'assemblaggio di una automobile è caratterizzata da diversi stadi da eseguire in un ordine fisso e caratterizzato da un determinato tempo di esecuzione (ad es. creazione scocca, montaggio del motore, sistemazione degli interni e verniciatura).

I vari stadi produttivi sono posti in cascata e lavorano in modo che l'uscita di uno stadio coincida con l'ingresso dello stadio seguente.

# Pipeline

(segue)

A regime, ogni stadio è caratterizzato da una coda di prodotti in attesa di essere processati.

Se i tempi di esecuzione delle varie fasi sono uguali il sistema funziona in maniera ideale: ogni stadio è sempre attivo e la sua coda è sempre vuota; al termine di un'operazione il prodotto passa allo stadio a valle e l'unità produttiva è pronta a processare il prodotto che gli arriva dallo stadio a monte.

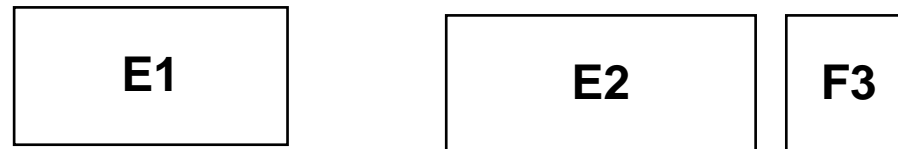
*In una **pipeline ideale** il sistema globale riduce il tempo medio di fabbricazione di un prodotto di un fattore pari al numero di stadi della pipeline.*

# Pipeline

## imperfetto nell'8086,386,486



processamento sequenziale



processamento in parallelo (I singoli stadi non hanno tempi di esecuzione uguali)

# Pipeline

## (valutazioni Prestazioni)

Siano:

$k$  = n. stadi pipeline

☐  $\tau_i$  = tempo dello  $i$ -esimo stadio

☐  $\tau_{\max} = \max(\tau_i)$

Se si considerano  $N$  istruzioni, si ha:

1. Tempo di esecuzione di una istruzione =  $k\tau_{\max}$

2. Tempo di esecuzione di  $N$  istruzione, senza pipeline, =  $Nk\tau_{\max}$

3. Tempo di esecuzione di  $N$  istruzione, con pipeline, =  $k\tau_{\max} + (N - 1) \tau_{\max}$

( $k\tau_{\max}$  è tempo di esecuzione della prima istruzione che deve percorrere tutta pipeline)

# Pipeline

## (valutazioni Prestazioni)

*Fattore di accelerazione:*

(Tempo di esecuzione di N istruzioni, senza pipeline)/(Tempo di esecuzione di N istruzioni, con pipeline) =

$$Nk\tau_{\max}/k\tau_{\max} + (N - 1)\tau_{\max} =$$

$$Nk / (k + (N - 1))$$

*Si noti che per N grandi tende a k*

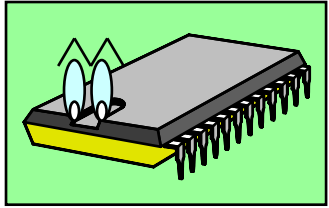
Ne deriva che i MIPS di picco di una architettura pipeline risultano:

$$\tau_{\max} = m \text{ (periodi clock cpu)} = m / (\text{frequenza clock cpu})$$

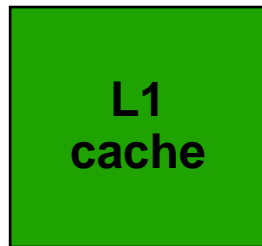
$$\text{MIPS} = 1 / \tau_{\max} = (\text{frequenza clock cpu}) / m$$



# Pipeline nel Pentium



- L2 cache
- DRAM



# Pipeline nel Pentium

Nel processore Pentium esiste una vera pipeline di 5 stadi:

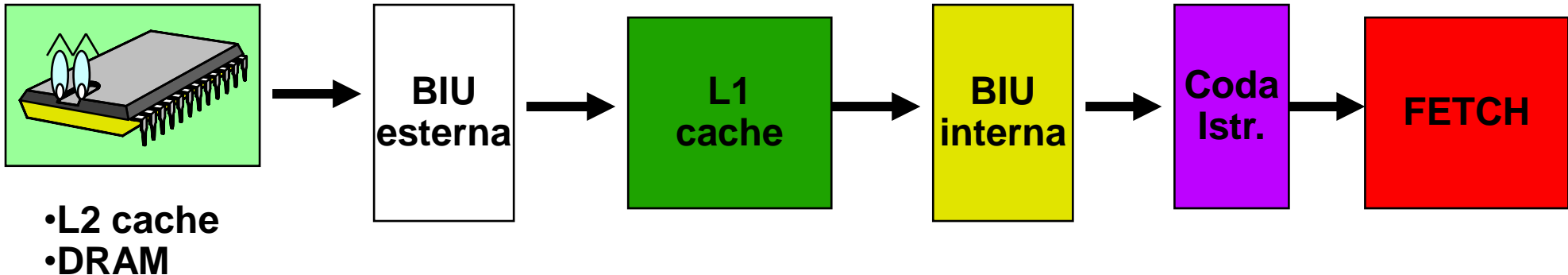
- la fase di **fetch** (eseguita dalla coda o da L1 cache)
- la fase di **decodifica** dell'istruzione, in cui vengono anche eseguiti i controlli di privilegio e di attributi in modo protetto(D1)
- La fase di **calcolo degli indirizzi degli operandi**, in cui sono calcolati gli indirizzi reali tramite segmentazione e paginazione(D2)
- La fase di **esecuzione**, in sono eseguite le operazioni della ALU e settati i flag
- La fase di **write-back**, in cui sono scritti i risultati nei registri o nella L1 cache o in memoria.

Ogni stadio richiede 1 CLK

In alcune condizioni in uno stadio si inseriscono più periodi di CLK, es. accesso alla memoria.

# Gestione della memoria

Durante il *prefetch* delle istruzioni, quando vi siano almeno due byte liberi nella coda delle istruzioni, la BIU preleva dalla memoria, o dalla L1 cache una parola allineata all'indirizzo pari.



# I/O

L'accesso alle periferiche avviene spesso attraverso speciali locazioni associate ad un certo indirizzo.

L'accesso a tali locazioni può avvenire nei processori 80x86 sia in modo *memory mapped* sia in *isolated I/O*. Nel primo caso l'accesso alla periferica avviene attraverso una normale istruzione, nel secondo attraverso speciali istruzioni di I/O.

Lo spazio di indirizzamento dell'I/O è pari al più a 64K, in quanto le istruzioni di IO esprimono indirizzo IO al più su 16 bit.

## Unità aggiuntive - REGISTRI SPECIALI

**Nell'architettura 80x86 esistono oltre ai citati registri di cpu, registri contenuti in due unità speciali: il coprocessore matematico e l'unità di gestione delle istruzioni di tipo SIMD**

**Il coprocessore matematico, FPU, fu introdotto fin dall'inizio, mentre le estensioni MMX e SSE vennero introdotte in seguito (Pentium Pro e Pentium IV), anticipate da AMD**

# FPU

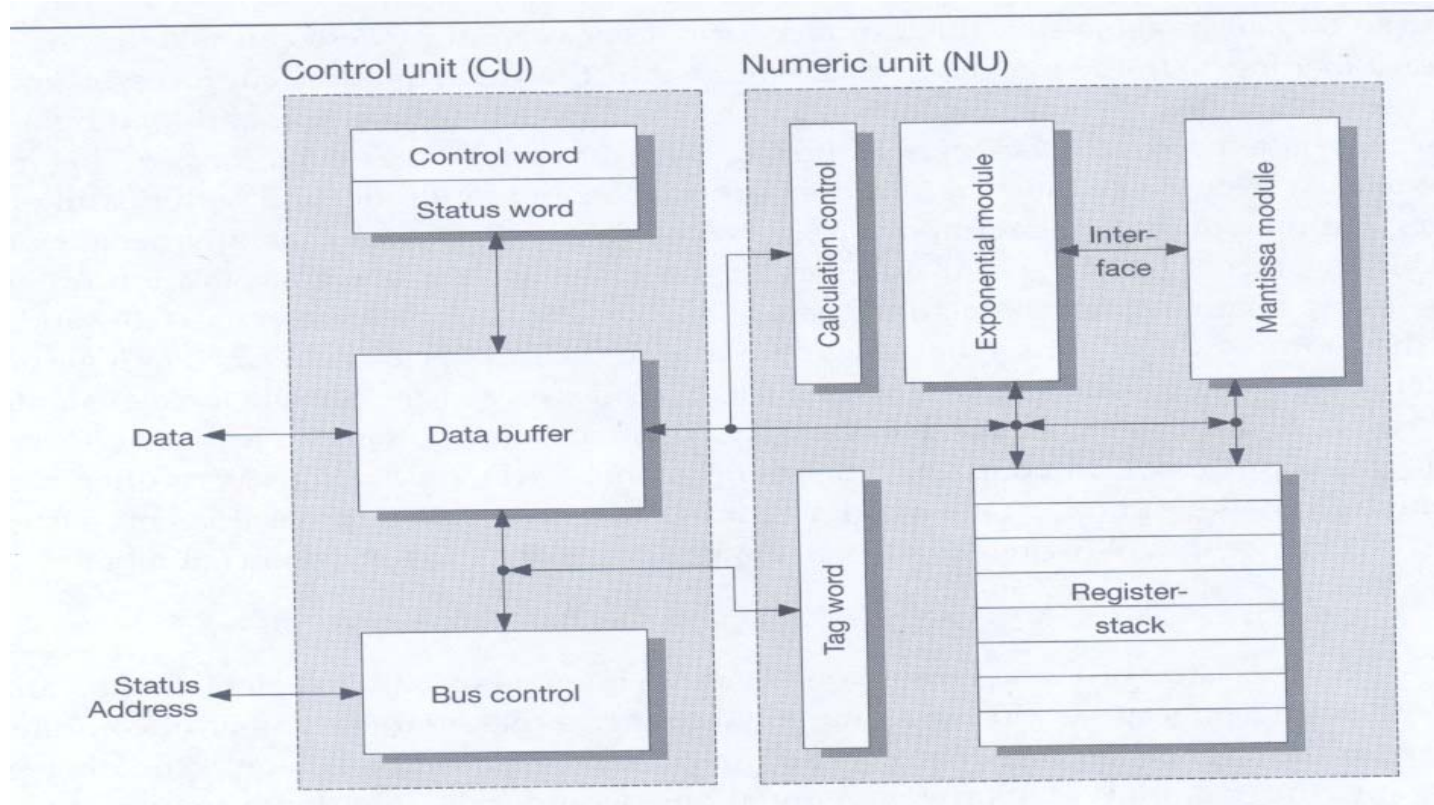


Fig. 9.2.1 The 80387 has one control unit for addressing the bus and for controlling the numeric unit. The n

**Esegue sia operazioni in fixed point, floating point, sia il calcolo di Funzioni trascendentali e trigonometriche**

**E' un processore con architettura a stack**

# FPU – registri dati

Register stack

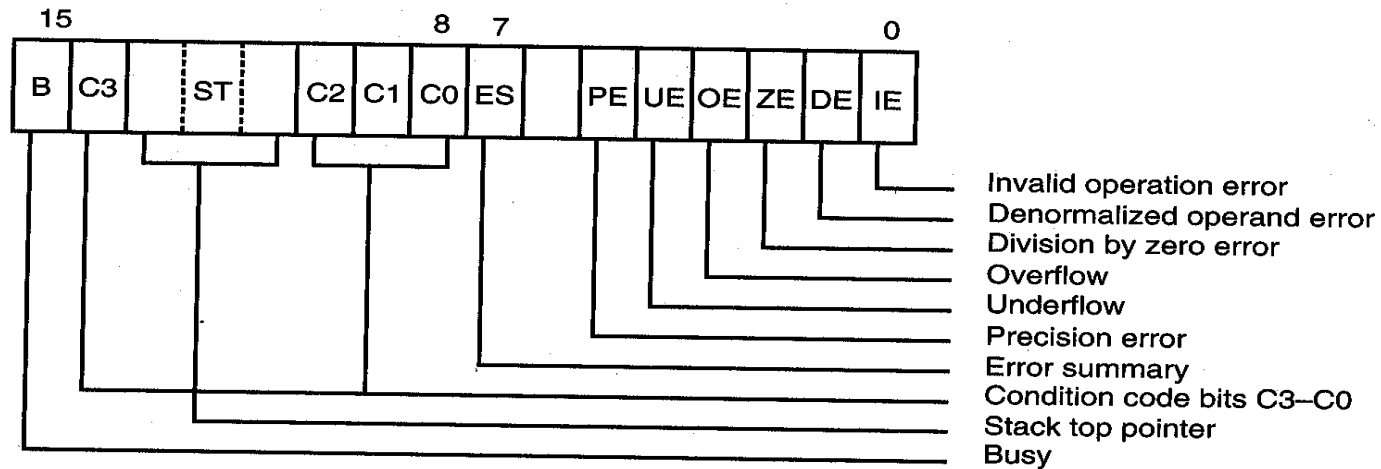
	79	78		64	63		0		1	0
S	Exponent					Mantissa			Tag	
S	Exponent					Mantissa			Tag	
S	Exponent					Mantissa			Tag	
S	Exponent					Mantissa			Tag	
S	Exponent					Mantissa			Tag	
S	Exponent					Mantissa			Tag	
S	Exponent					Mantissa			Tag	
S	Exponent					Mantissa			Tag	

Control and status register

15		1		31		16	15		0
Control register				Instruction pointer					
Control register				Data pointer					
Tag word									

Tag = 00 (valid), 01 (zero), 10 (invalid), 11 (empty)

# FPU- registro stato



**FIGURE 14-5** The 80X87 arithmetic coprocessor status register.

I 3 bit ST definiscono quale registro degli 8 deve essere considerato TOS



# FPU- registro controllo

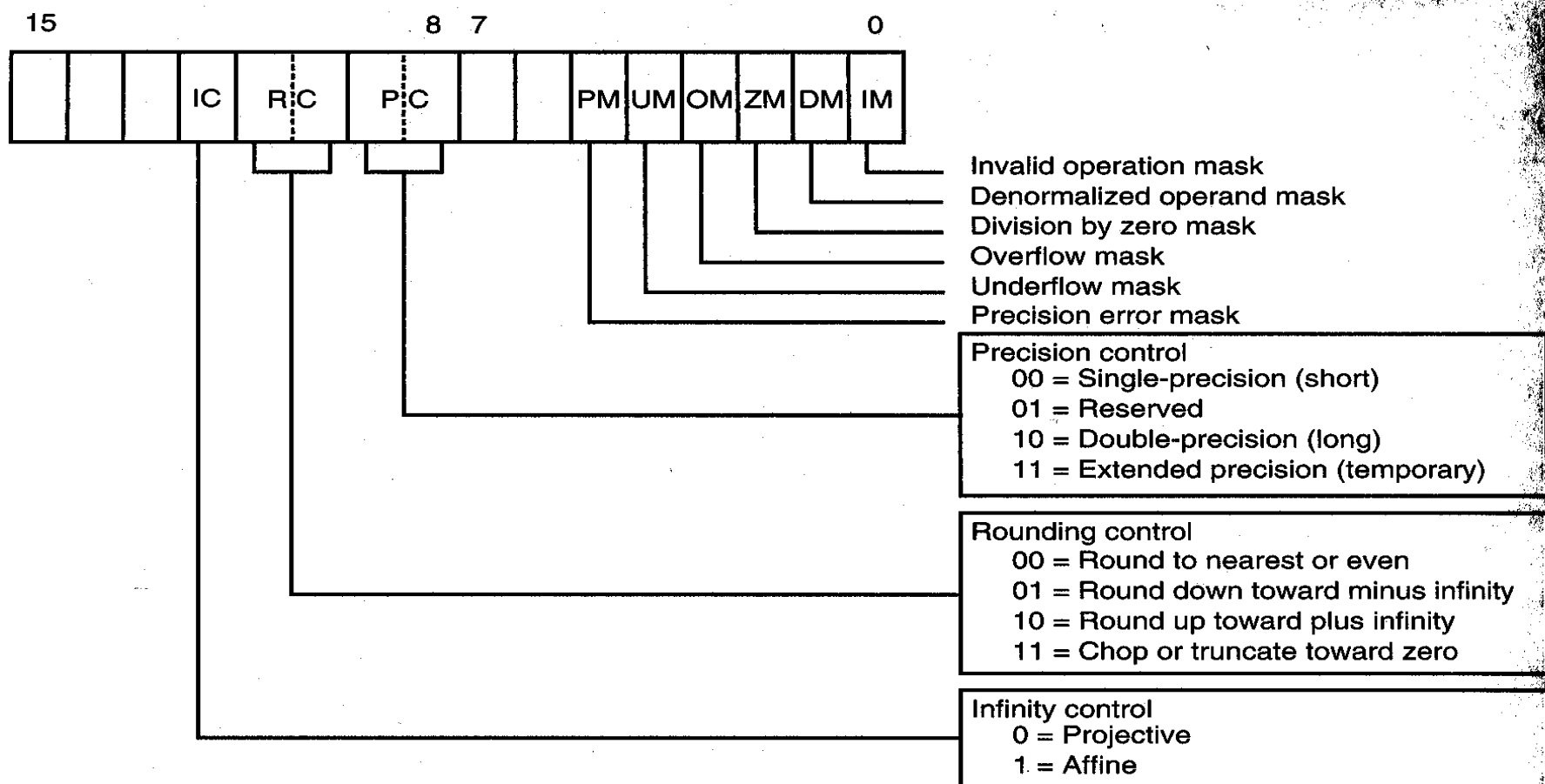


FIGURE 14-6 The 80X87 arithmetic coprocessor control register.

## **MMX(*MultiMedia eXtension* or *Multiple Math* or *Matrix Math eXtension*)**

**Un set di istruzioni che realizzano una architettura SIMD**

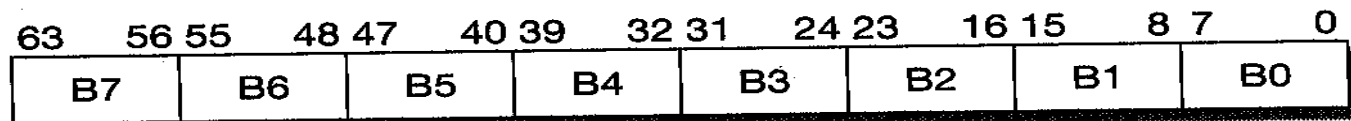
**I registri interessati sono 8 registri da 64 bit, coincidenti fisicamente con parte dei registri dati della FPU. E' pertanto necessario quando si adoperano tali istruzioni salvare il contenuto dei FPU registers e viceversa.**

**Architettura orientata alla grafica (gestione di bit)**

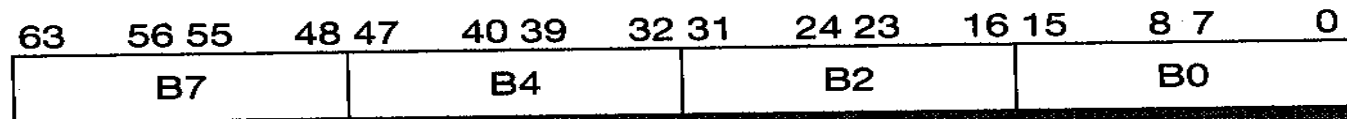
# MMX

I registri da 64 bit, detti XMMi, permettono di gestire i seguenti formati:

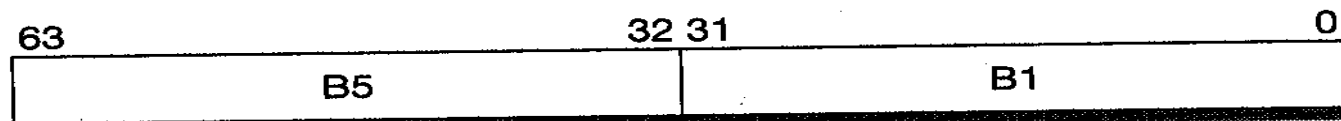
Packed byte (8 bit × 8)



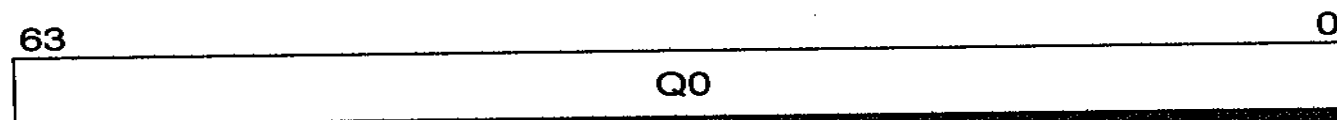
Packed byte (16bit × 4)

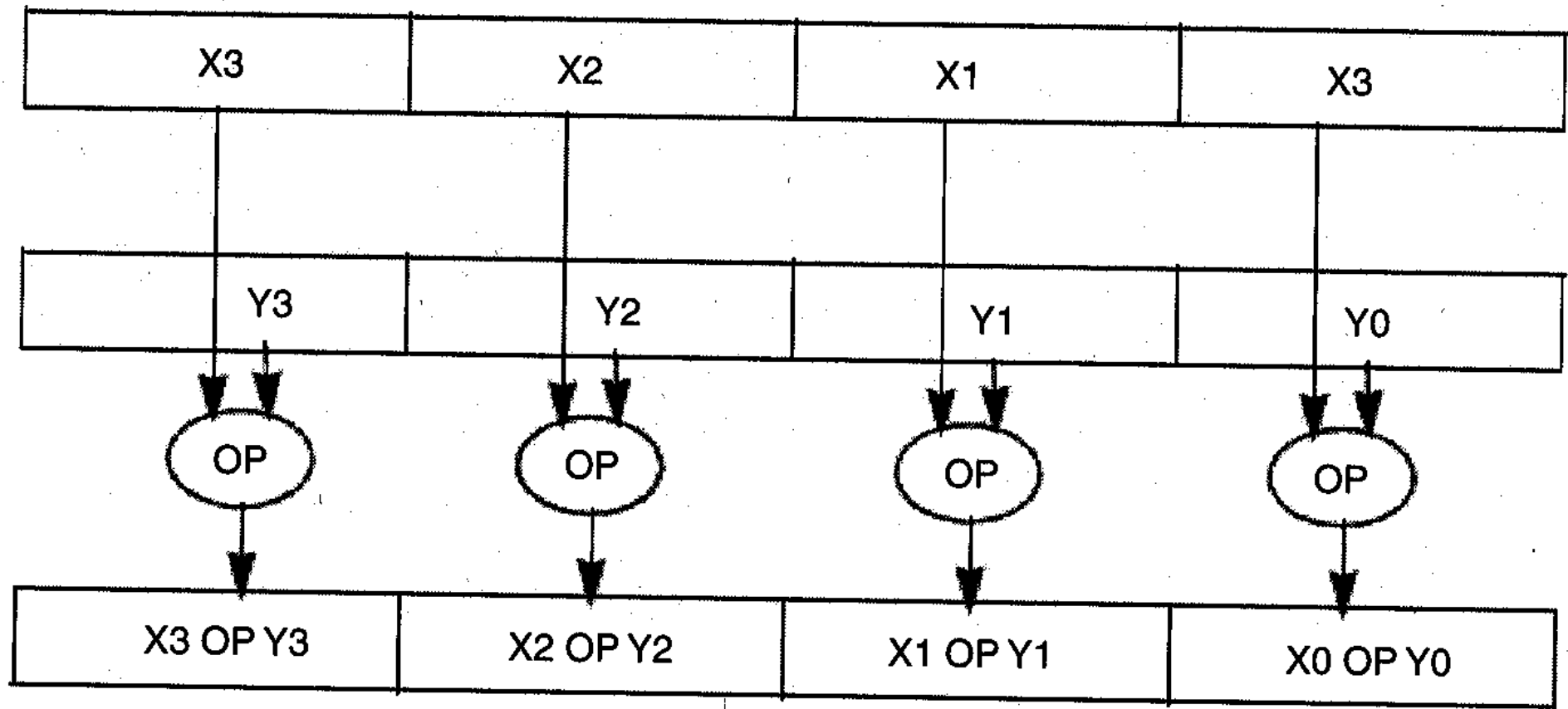


Packed double words (32 bit × 2)



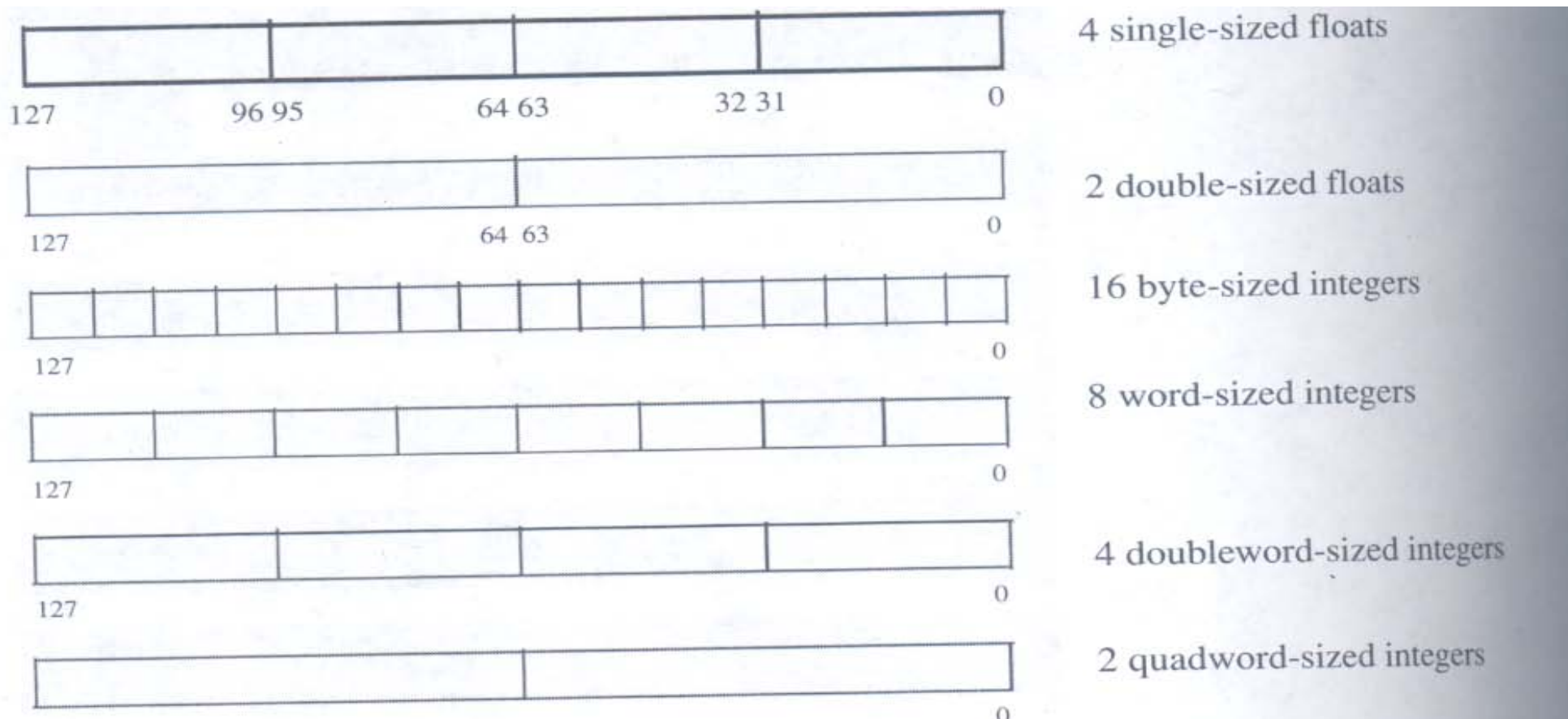
Quadword (64 bit × 1)





# SSE -Streaming SIMD Extensions

Architettura SIMD orientata alla gestione di operazioni su array e realizzata mediante 8+8 registri da 128 bit, che possono contenere i seguenti formati:



# SSE -Streaming SIMD Extensions

Somma due vettori di 4 elementi su architettura scalare

```
vec_res.x=v1.x+v2.x;  
vec_res.y=v1.y+v2.y;  
vec_res.z=v1.z+v2.z;
```

```
vec_res.w=v1.w + v2.w
```

Somma due vettori di 4 elementi su architettura SSE

```
movaps xmm0,address-of-v1
```

```
addps xmm0,address-of-v2
```

```
Movaps,address-of-vec_res,xmm0
```