Le architetture a pipeline

M. Sonza Reorda

Politecnico di Torino

Dip. Automatica e Informatica

Introduzione

Per migliorare le prestazioni di un processore si può agire in due direzioni:

- **aumentando la velocità dei componenti**
- **modificando l'architettura.**

L'uso di architetture a pipeline segue la seconda strada.

Processori RISC e superscalari

I processori tradizionali richiedono un certo numero di periodi di clock per l'esecuzione di ogni istruzione.

Grazie all'introduzione delle pipeline, in ogni periodo di clock

- **i processori *RISC* possono completare un'istruzione**
- **i processori *superscalari* possono completare anche più di un'istruzione.**

CPI

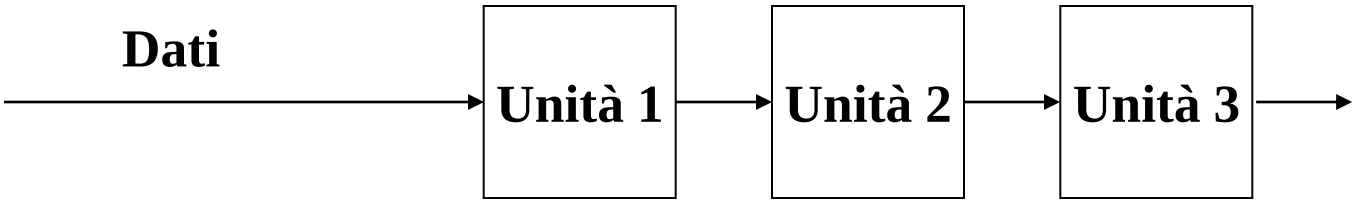
Per misurare l'efficienza di un processore si usa a volte il parametro CPI (*Clocks per Instruction*):

- **per processori CISC ⇒ CPI > 1**
- **per processori RISC ⇒ CPI ~ 1**
- **per processori superscalari ⇒ CPI < 1.**

Pipeline

La pipeline è l'equivalente elettronico della *catena di montaggio*.

Permette di eseguire in parallelo operazioni diverse sullo stesso flusso di dati.



Pipeline ideale

Se

- **tutti i dati devono passare attraverso gli stessi stadi**
- **tutti gli stadi richiedono sempre lo stesso tempo per eseguire la relativa elaborazione**

allora l'uso di una pipeline di *n* stadi può portare a regime ad un miglioramento di *n* volte nelle prestazioni, misurate in termini di quantità di lavoro svolto nell'unità di tempo (*throughput*).

La pipeline nei processori

L'esecuzione di ciascuna istruzione può essere suddivisa in varie fasi, quali:

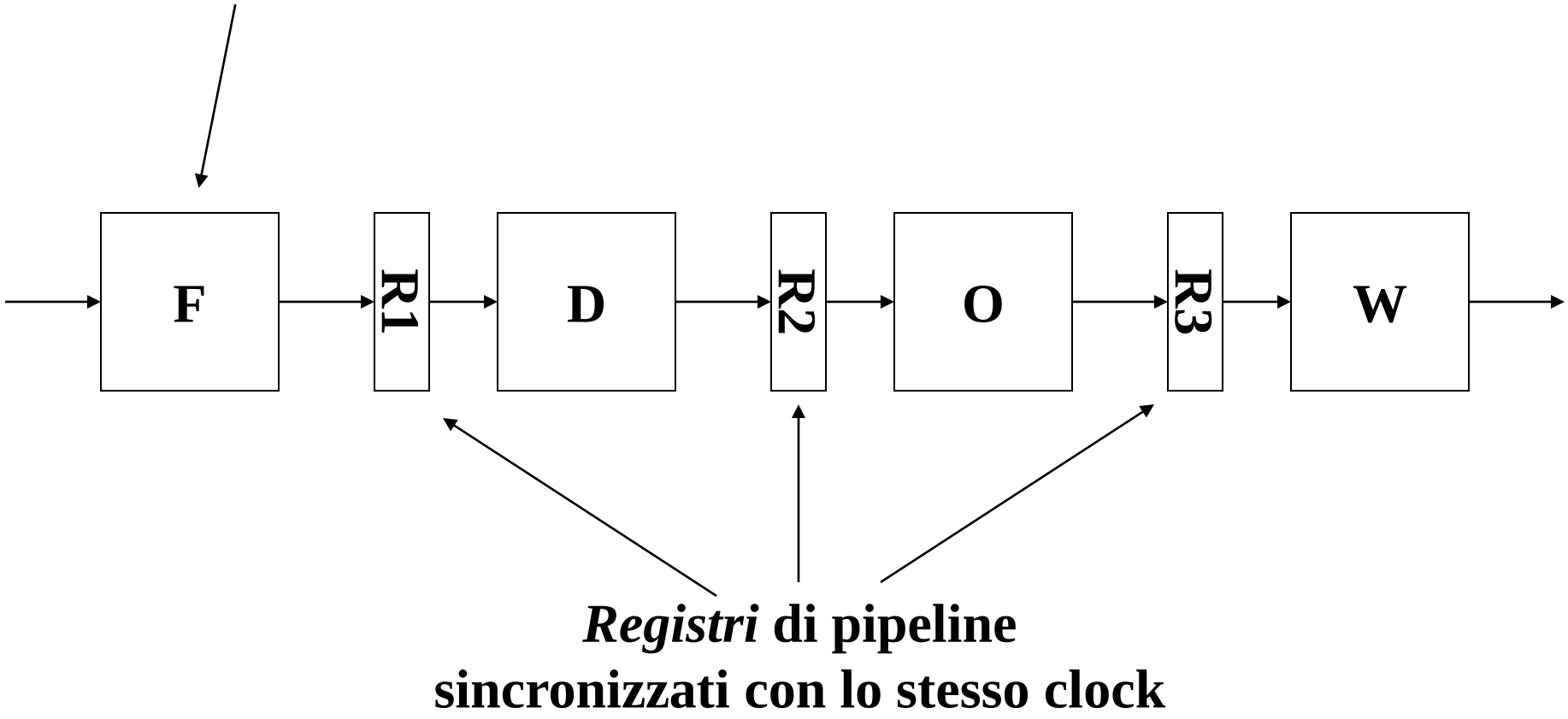
- ***Fetch*** **(o prelievo): caricamento dell'istruzione dalla memoria**
- ***Decode*** **(o decodifica): decodifica dell'istruzione e lettura degli operandi sorgente**
- ***Operate*** **(o elaborazione): esecuzione dell'operazione**
- ***Write*** **(o scrittura): scrittura del risultato.**

Ciascun processore può adottare una suddivisione in un numero di fasi diverso:

- **Processori ARM di prima generazione: 3 fasi**
- **Processore MIPS: 5 fasi**
- **Processori Intel Netburst (Pentium 4): 31 fasi.**

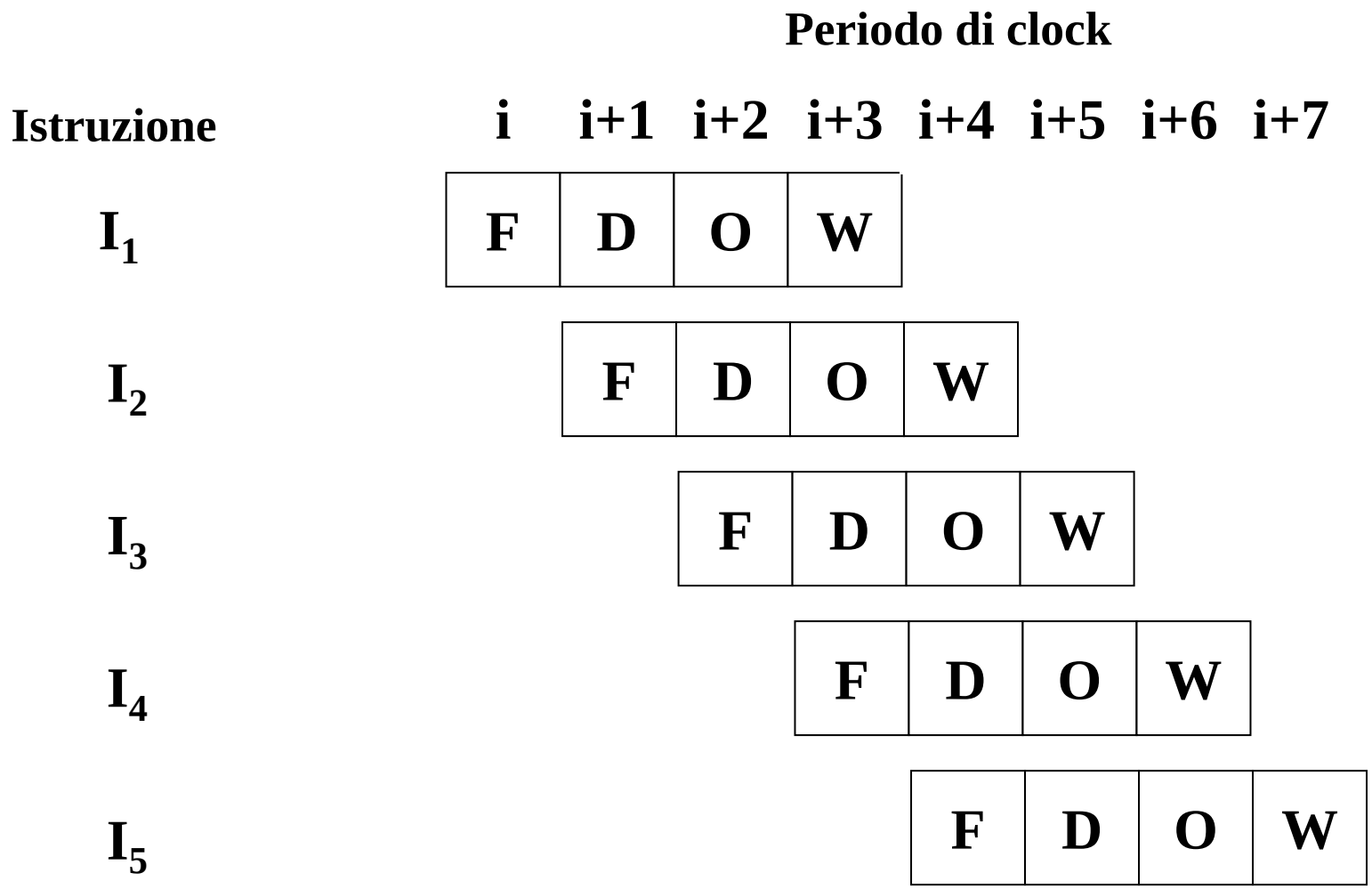
Architettura

Stadio **della pipeline:**
in ogni periodo di clock
esegue una fase
su un'istruzione

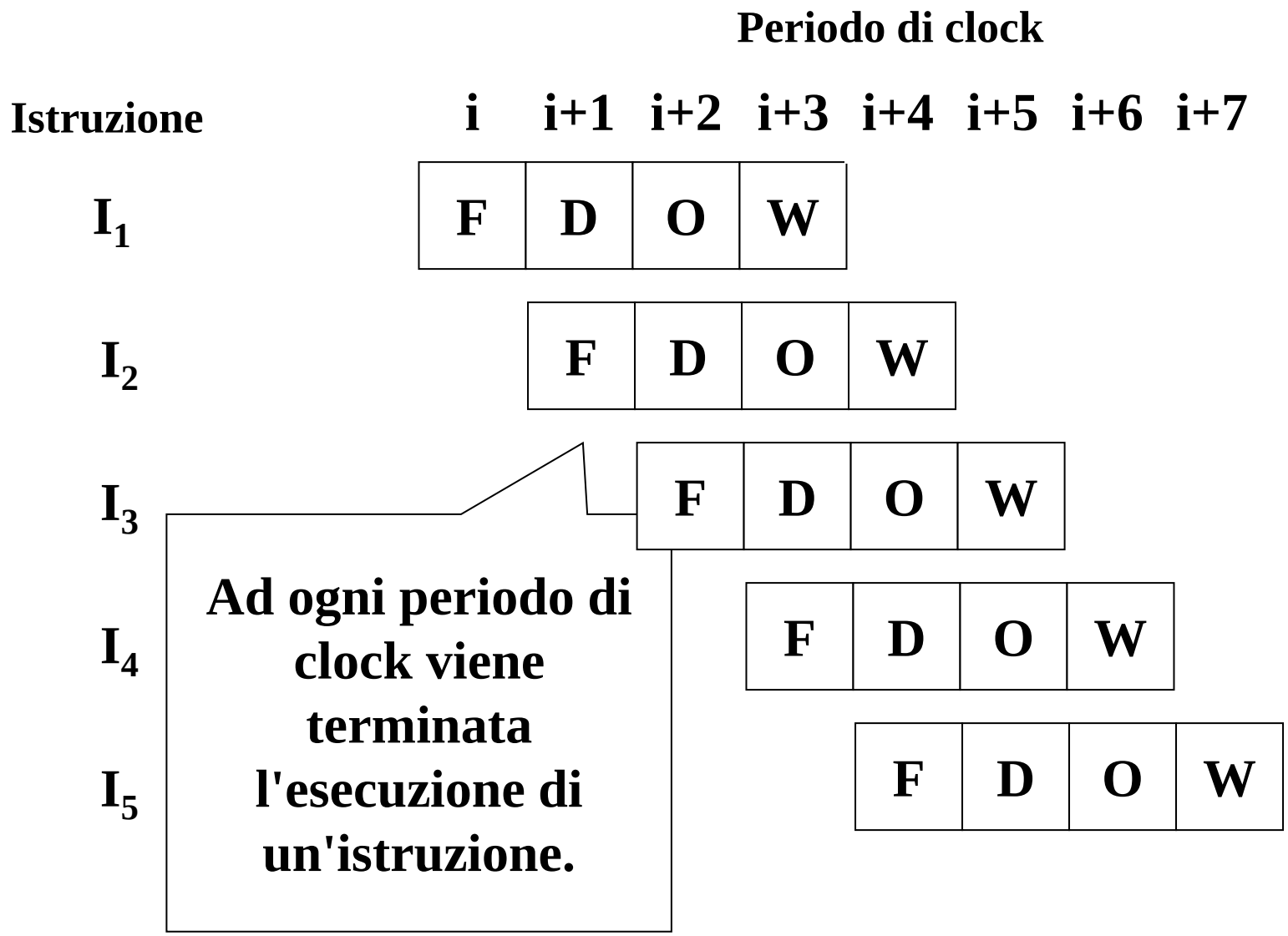


Registri **di pipeline**
sincronizzati con lo stesso clock

Esempio


Periodo di clock
Istruzione
i
i+1
i+2
i+3
i+4
i+5
i+6
i+7
I1
I2
I3
I4
I5
F
D
O
W

Esempio


Periodo di clock
Istruzione
i
i+1
i+2
i+3
i+4
i+5
i+6
i+7
I1
I2
I3
I4
I5
F
D
O
W
Ad ogni periodo di clock viene terminata l'esecuzione di un'istruzione.

Stallo

In un processore reale, è possibile che a volte uno stadio non riesca a completare il proprio compito in un periodo di clock.

In tal caso

- **gli altri stadi a monte devono arrestarsi, ossia andare *in stallo*, in quanto i registri in cui devono scrivere non sono liberi**
- **gli stadi a valle possono invece continuare a lavorare, ma solo per smaltire le istruzioni già processate dallo stadio critico.**

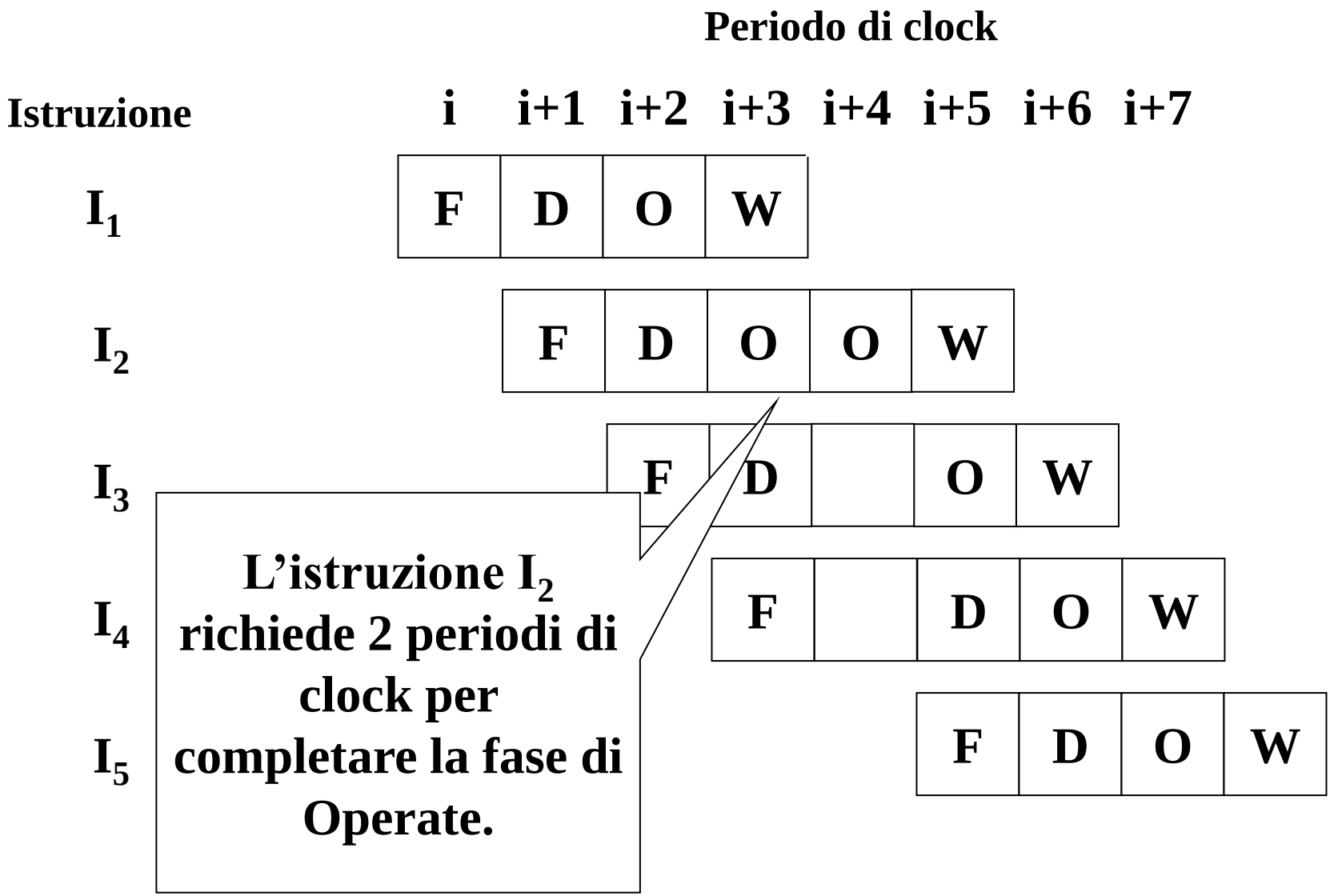
In tal modo le prestazioni del processore scendono rispetto a quelle ideali previste.

Cause di stallo

Esempi

- **Lentezza negli accessi a memoria (fasi F, D e W)**
- **Lentezza nell'esecuzione di operazioni interne (fase O).**

Esempio


Periodo di clock
Istruzione
i
i+1
i+2
i+3
i+4
i+5
i+6
i+7
I1
F
D
O
W
I2
F
D
O
O
W
I3
F
D
O
W
I4
F
D
O
W
I5
F
D
O
W
L'istruzione I2 richiede 2 periodi di clock per completare la fase di Operate.

Cache

Le cache sono fondamentali nei processori che usano una pipeline, in quanto permettono di accedere a dati e istruzioni in un solo periodo di clock.

Bolle

Dopo che uno stadio è andato in stallo, si crea all'interno della pipeline una situazione di inattività (denominata *bolla*) che viene eliminata gradualmente.

Coda delle istruzioni

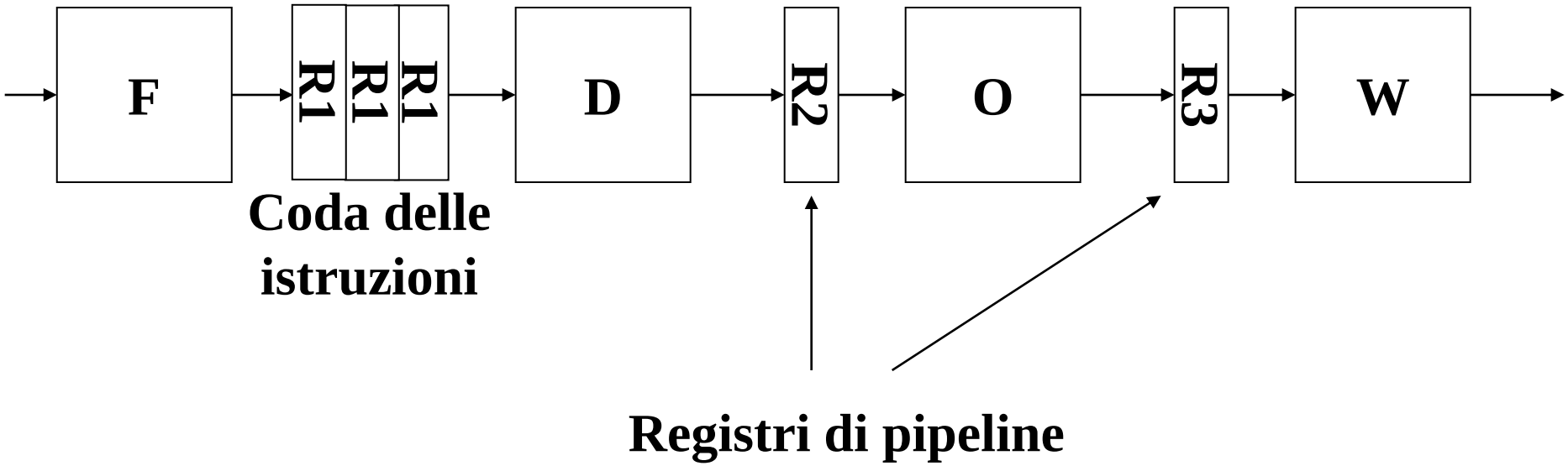
Quando l'unità di fetch non trova un'istruzione in cache, manda in stallo l'intera pipeline.

Per evitare questa situazione, il buffer tra le unità di fetch e di decodifica è sostituito da una coda, detta *coda delle istruzioni*.

L'unità di fetch cerca di mantenere sempre piena tale coda, caricando anche più di un'istruzione per periodo di clock.

In tal modo si compensano eventuali ritardi dovuti a situazioni di miss nella cache.

Architettura



Dipendenze di dato

La presenza di una pipeline rende parallelo un processo (quello di esecuzione delle istruzioni) che è originariamente sequenziale.

Affinché il risultato sia lo stesso del processo originario, bisogna che non vi siano dipendenze tra dati.

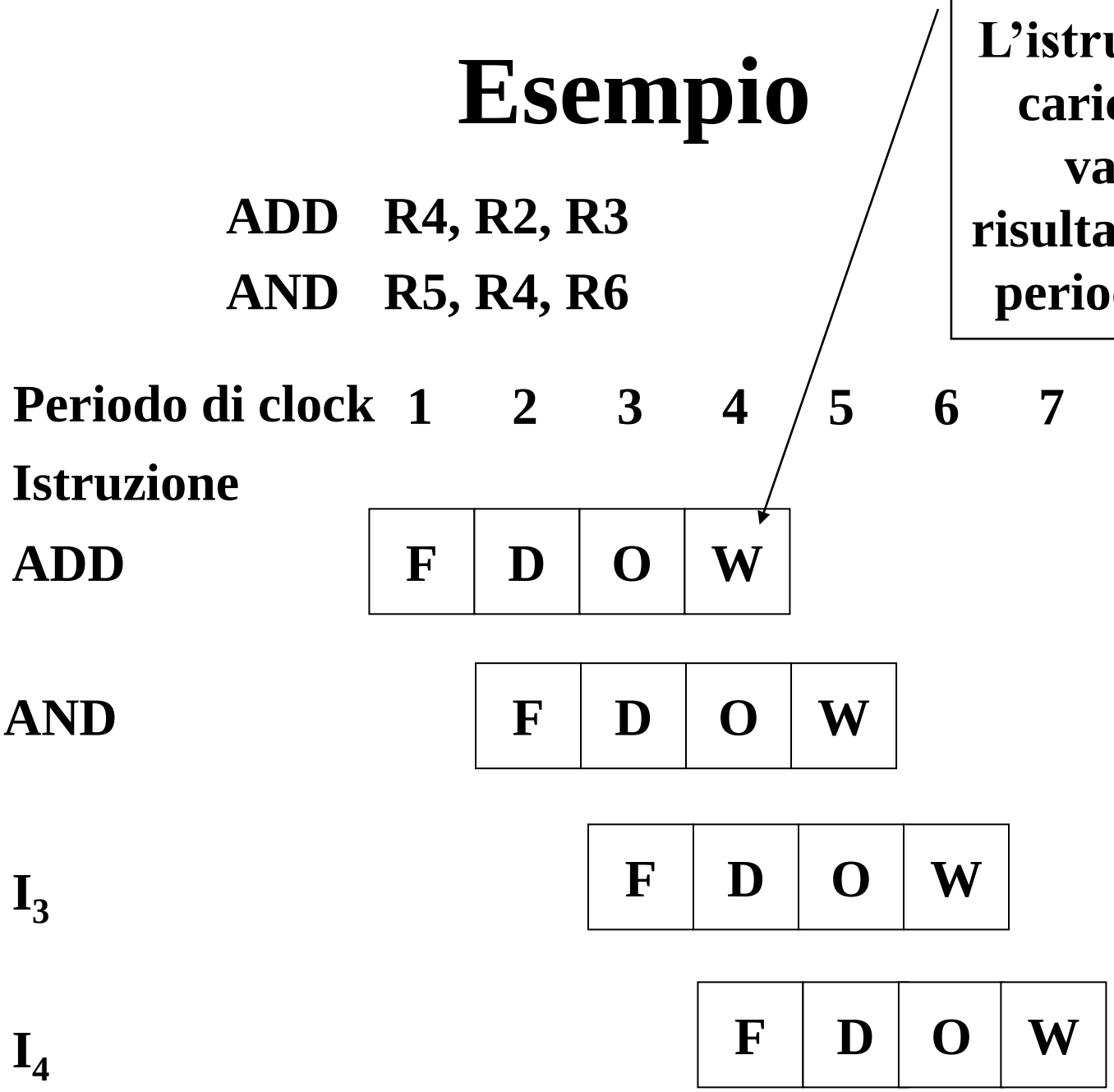
Ad esempio bisogna che gli operandi sorgente di un'istruzione siano già stati prodotti dalle istruzioni precedenti.

Esempio

ADD R4, R2, R3

AND R5, R4, R6

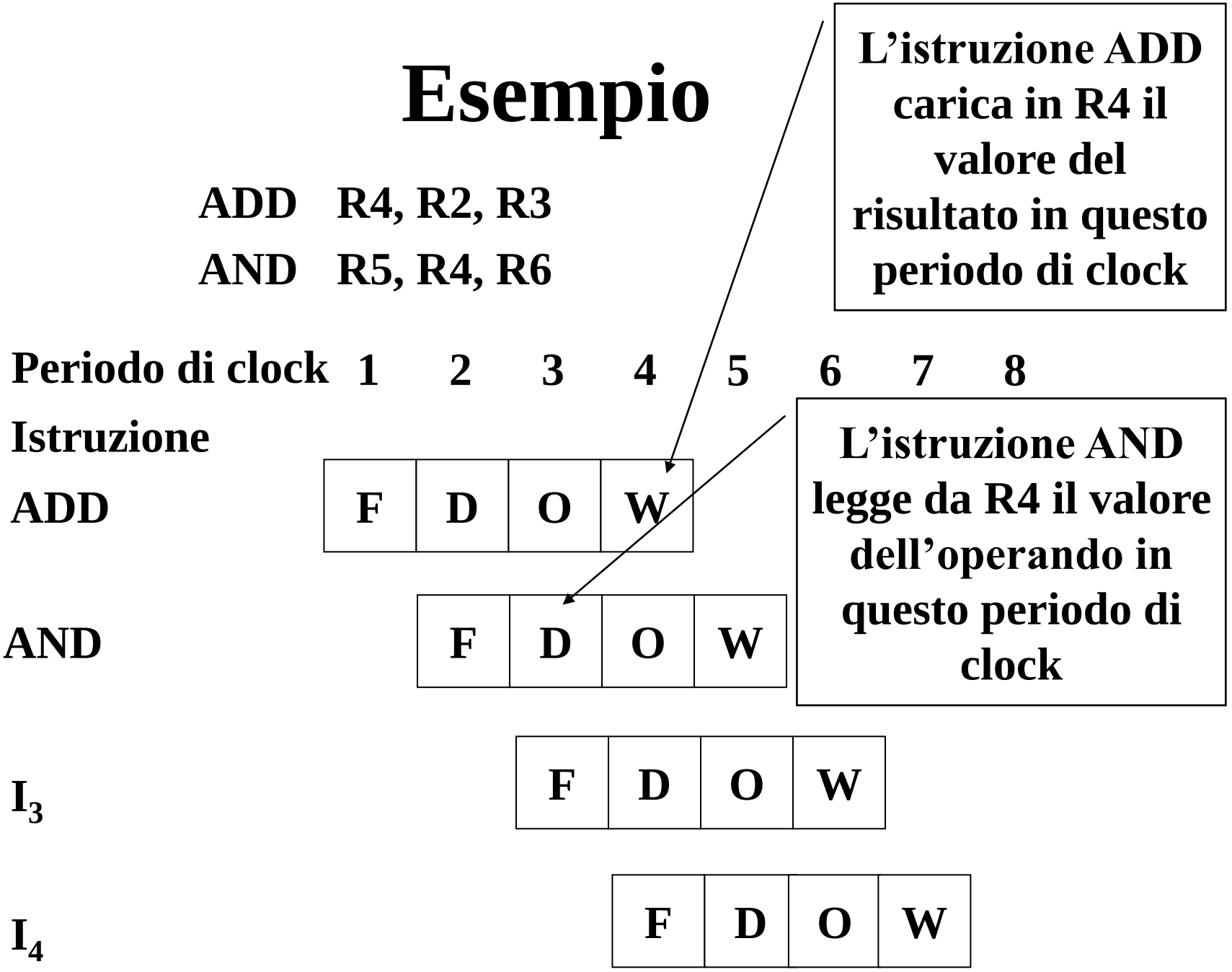
L'istruzione ADD carica in R4 il valore del risultato in questo periodo di clock



Esempio

ADD R4, R2, R3

AND R5, R4, R6



Esempio

L'istruzione ADD carica in R4 il valore del

questo clock

Pe

Ist

AI

AN

AND

valore

do in

do di

Conclusione

È necessario garantire la correttezza del risultato.

Questo può essere fatto

- in HW
- in SW.

I_3

I_4

F	D	O	W

Soluzione HW

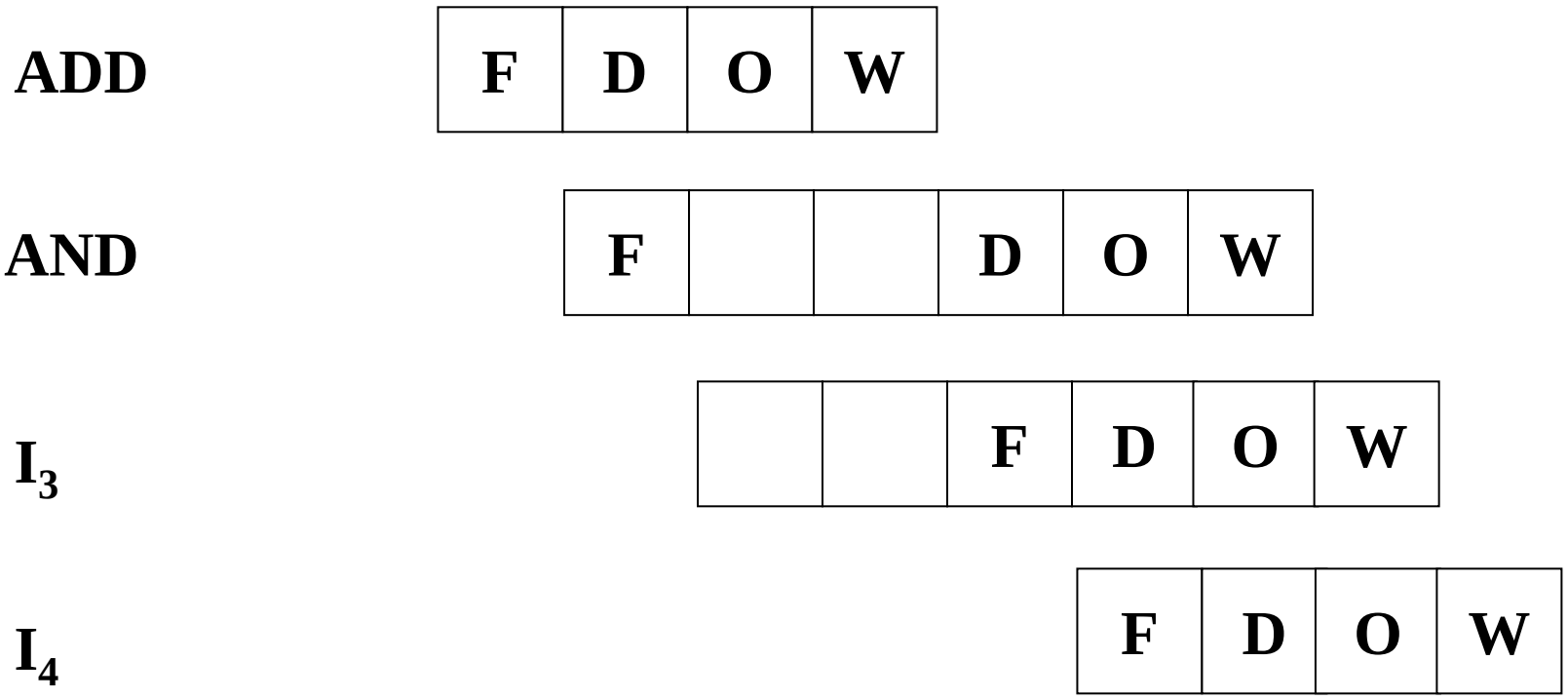
L'HW della pipeline è in grado di

- **riconoscere le situazioni di dipendenza dei dati**
- **inibire il funzionamento di specifici moduli della pipeline (introducendo degli stalli) per garantire la correttezza del risultato.**

Esempio

ADD R4, R2, R3

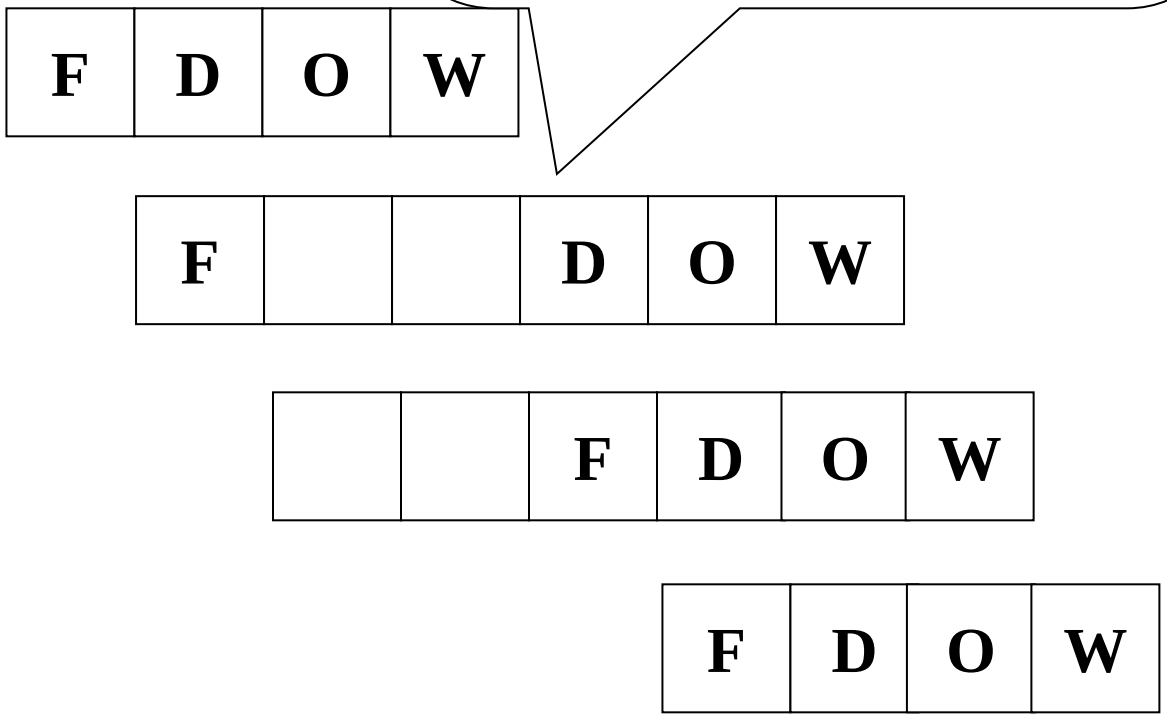
AND R5, R4, R6



Esem

```
ADD   R4, R2, R3
AND   R5, R4, R6
```

Periodo di clock	1	2	3



La fase di Decode (che legge gli operandi) non viene eseguita sino a che non è stata eseguita la fase di Write dell'istruzione precedente, che produce il valore aggiornato di R4.

Soluzione SW

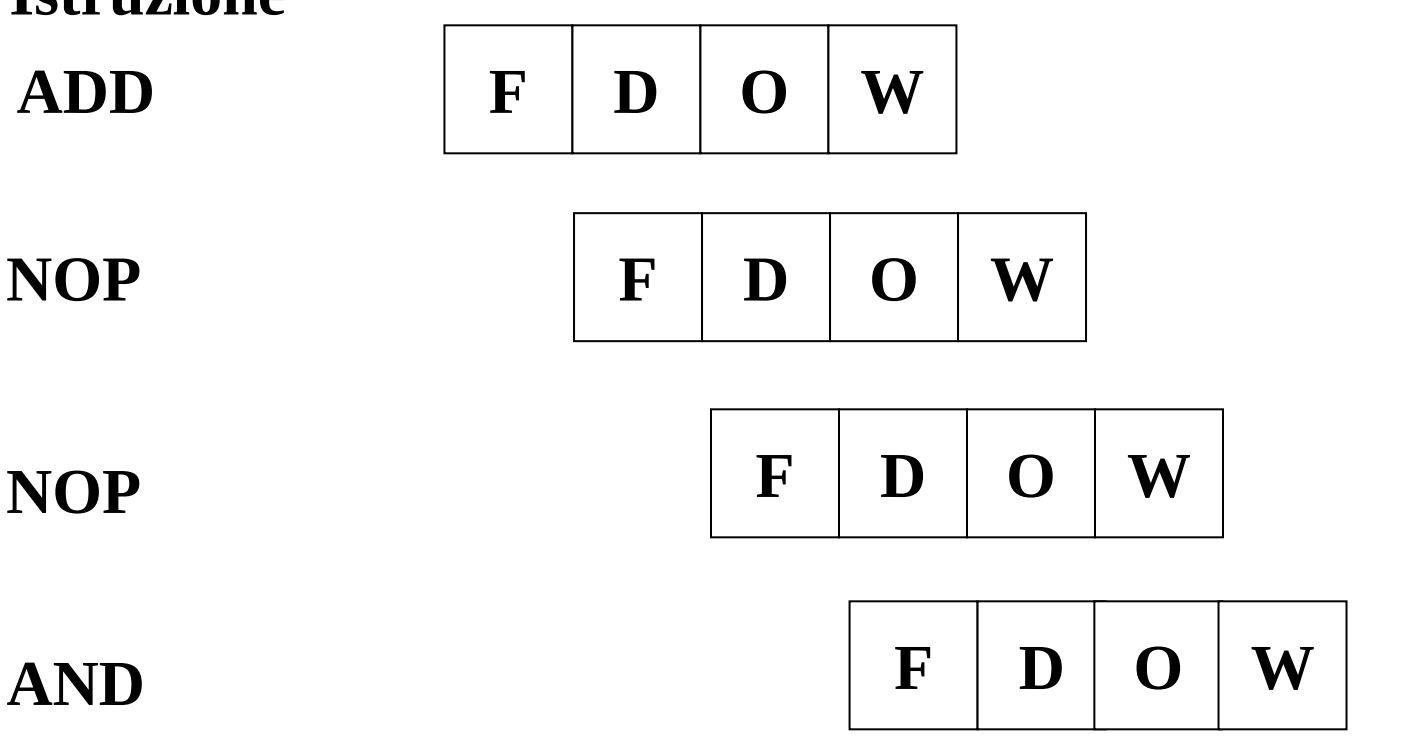
Il compilatore introduce delle istruzioni NOP, che permettono alla pipeline di produrre un risultato corretto.

Esempio

Codice originale	**codice modificato**
I_i	**I_i**
ADD R4, R2, R3	**ADD R4, R2, R3**
AND R5, R4, R6	**NOP**
I_j	**NOP**
I_k	**AND R5, R4, R6**
…	**I_j**

Esempio

```
ADD     R4, R2, R3
NOP
NOP
AND     R5, R4, R6
```



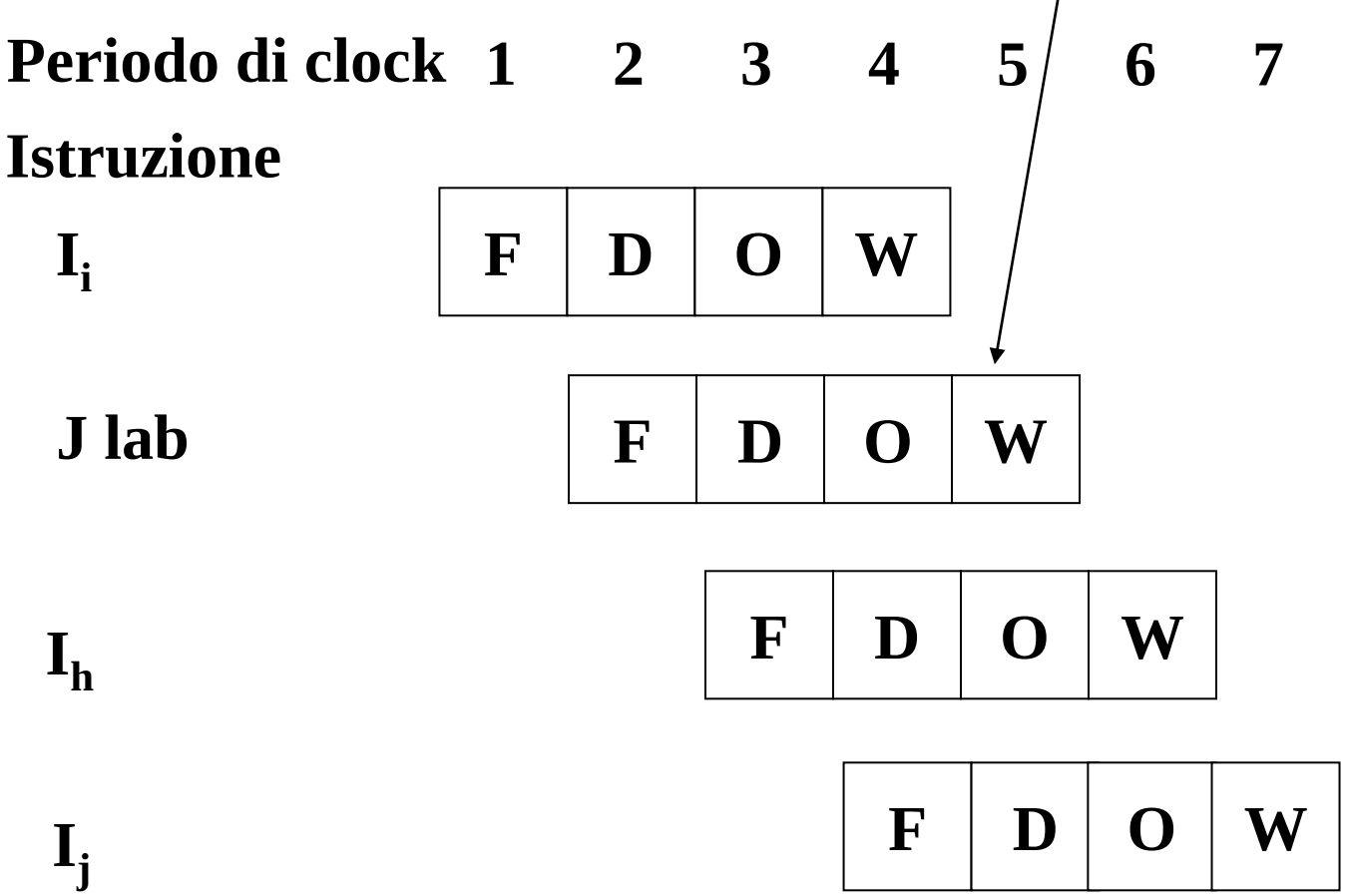
Istruzioni di salto

Le istruzioni di salto sono potenzialmente molto dannose per le prestazioni della pipeline, in quanto interrompono il normale flusso di esecuzione sequenziale.

Esempio

I_i
J lab
I_h
I_j
lab: I_k

L'istruzione di salto carica nel PC il nuovo valore in questo periodo di clock

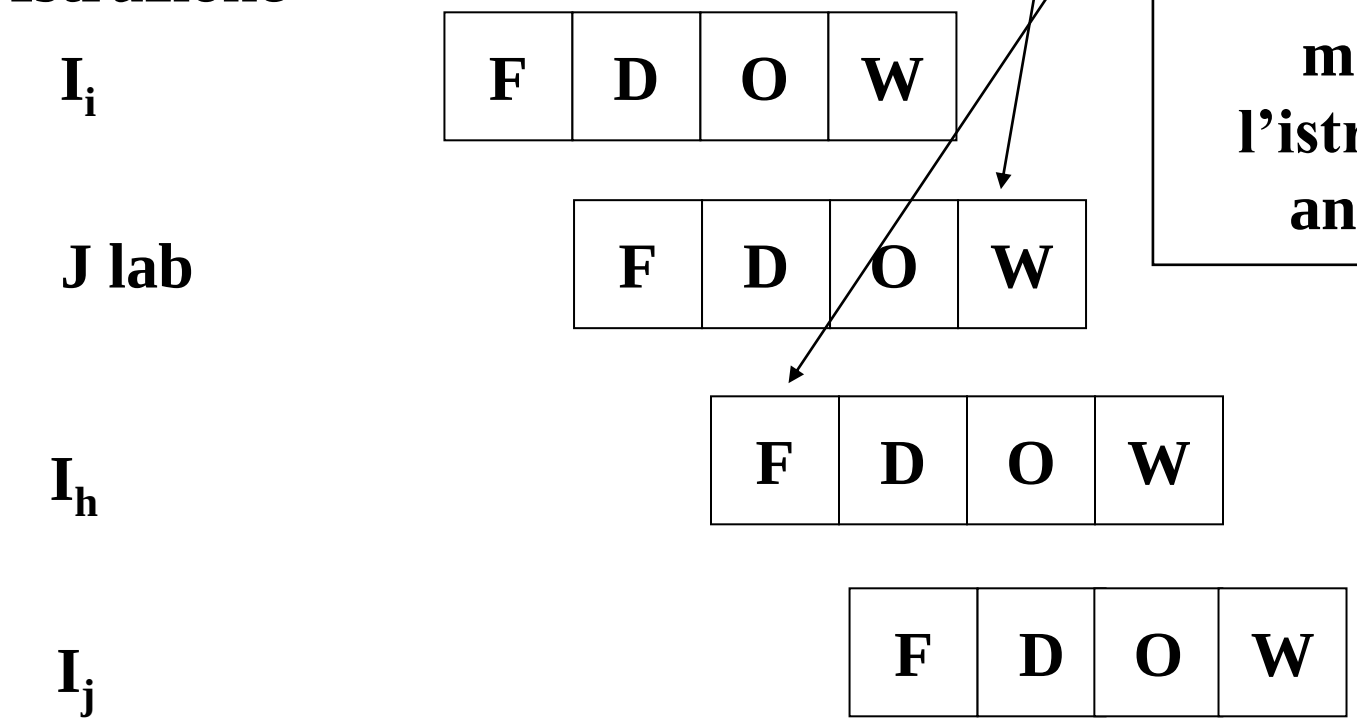


Esempio

I_i
J lab
I_h
I_j
lab: I_k

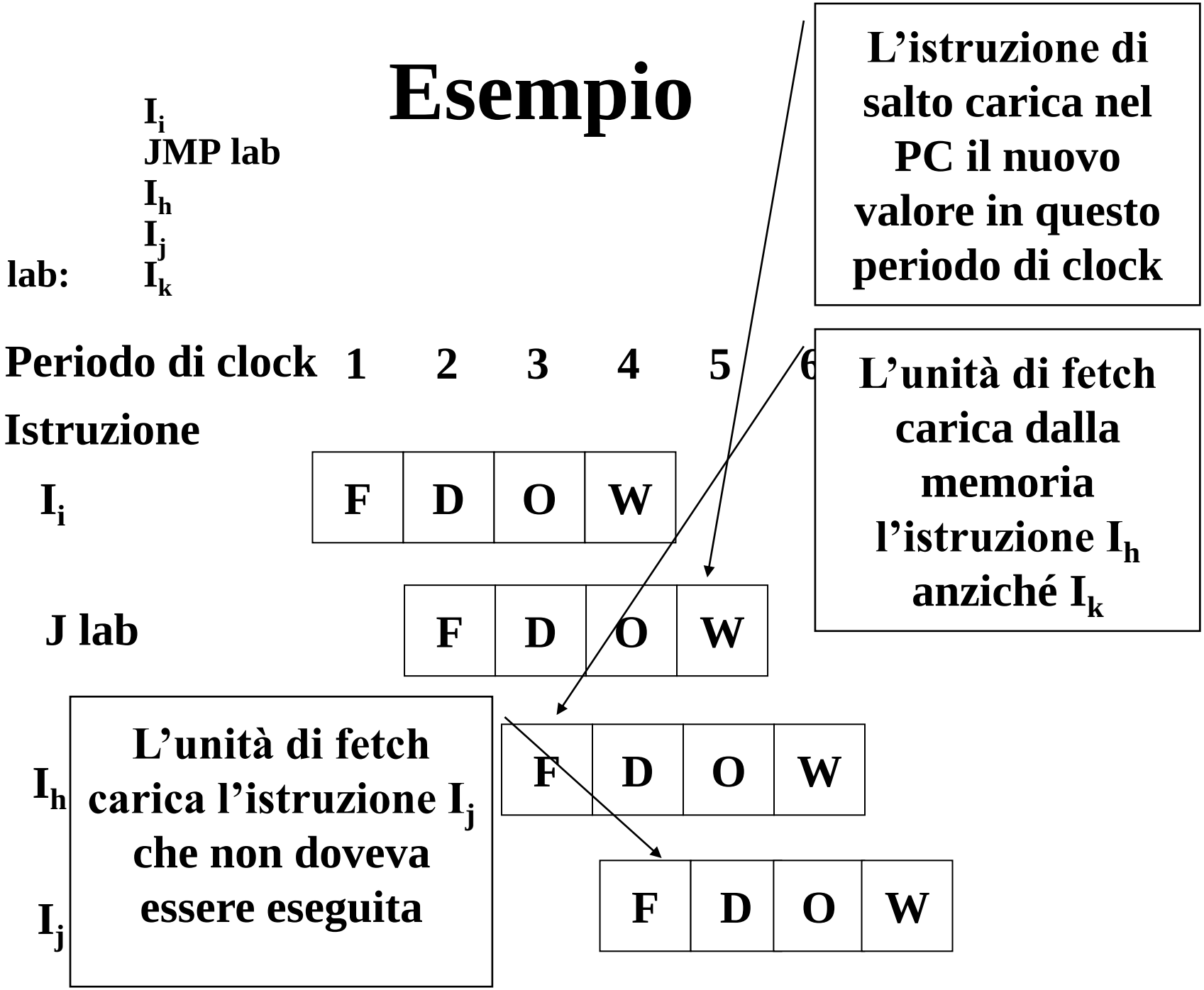
L'istruzione di salto carica nel PC il nuovo valore in questo periodo di clock

L'unità di fetch carica dalla memoria l'istruzione I_h anziché I_k



Esempio

```
        I_i
        JMP lab
        I_h
        I_j
lab:    I_k
```



Intervallo di ritardo del salto

Viene denominato intervallo di ritardo del salto (*branch delay slot*) il numero di periodi di clock successivi ad un'istruzione di salto, nei quali il processore esegue il fetch usando un valore del PC che non è necessariamente corretto.

Soluzioni

Per garantire la correttezza dei risultati prodotti dal processore anche in presenza di salti si possono anche qui utilizzare

- **tecniche HW**
- **tecniche SW.**

Tecniche HW

Possono

- **rilevare le istruzioni di salto, e svuotare conseguentemente la pipeline dalle istruzioni erroneamente caricate**

oppure

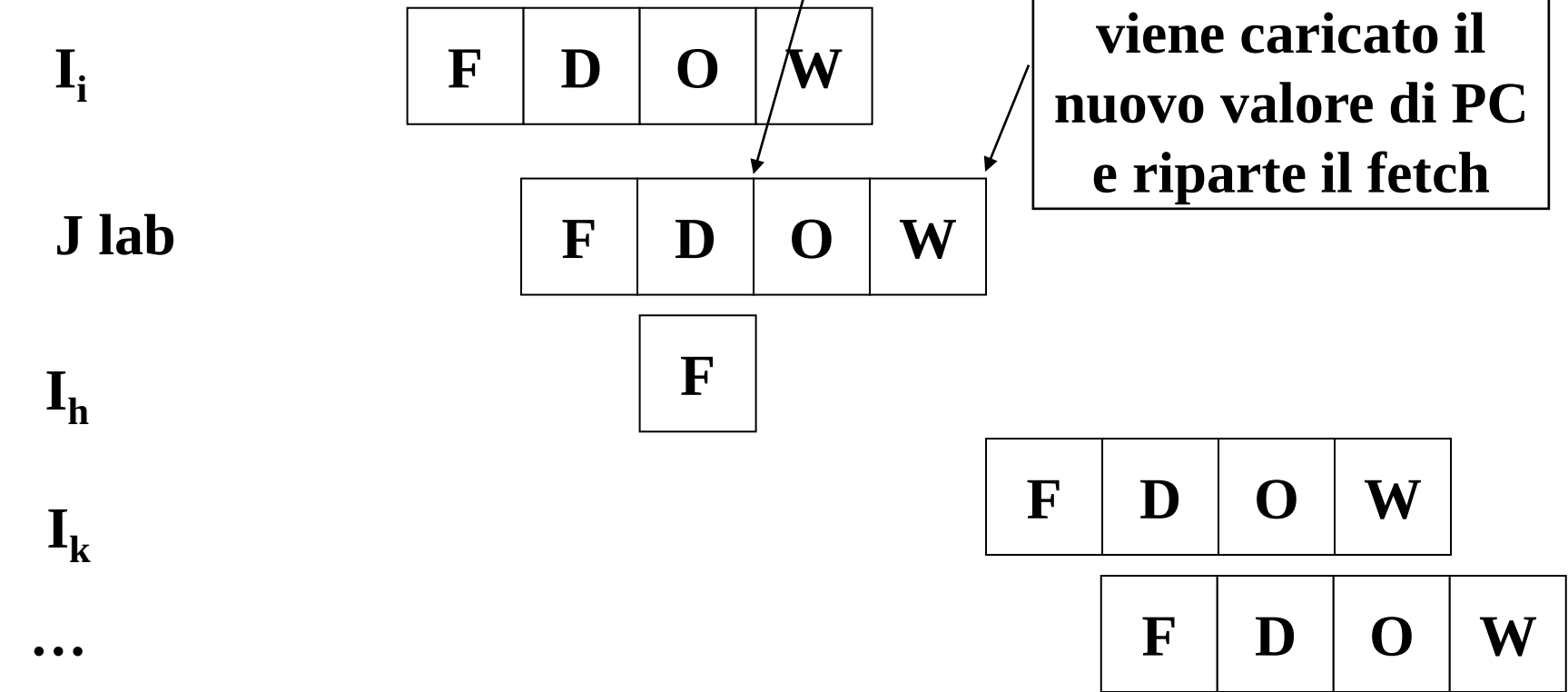
- **implementare tecniche più sofisticate, che dipendono dal fatto che il salto sia incondizionato o condizionato.**

Esempio

```
        Ii
        J lab
        Ih
        Ij
lab:    Ik
```

La CPU a questo istante capisce che l'istruzione corrente è di salto incondizionato e interrompe i fetch, inserendo stalli

A questo istante viene caricato il nuovo valore di PC e riparte il fetch



Tecniche SW

Sono basate su

- ***Salto ritardato***
  introduzione di istruzioni NOP da parte del compilatore (dopo ogni salto si inseriscono tante NOP quanto è il branch delay slot)

oppure

- **un'ottimizzazione del codice.**

Soluzione SW

Il compilatore introduce delle istruzioni NOP, che permettono alla pipeline di produrre un risultato corretto.

Esempio

Codice originale

I_a
I_b
I_c
I_d
J lab_1
I_j
lab_1: I_k

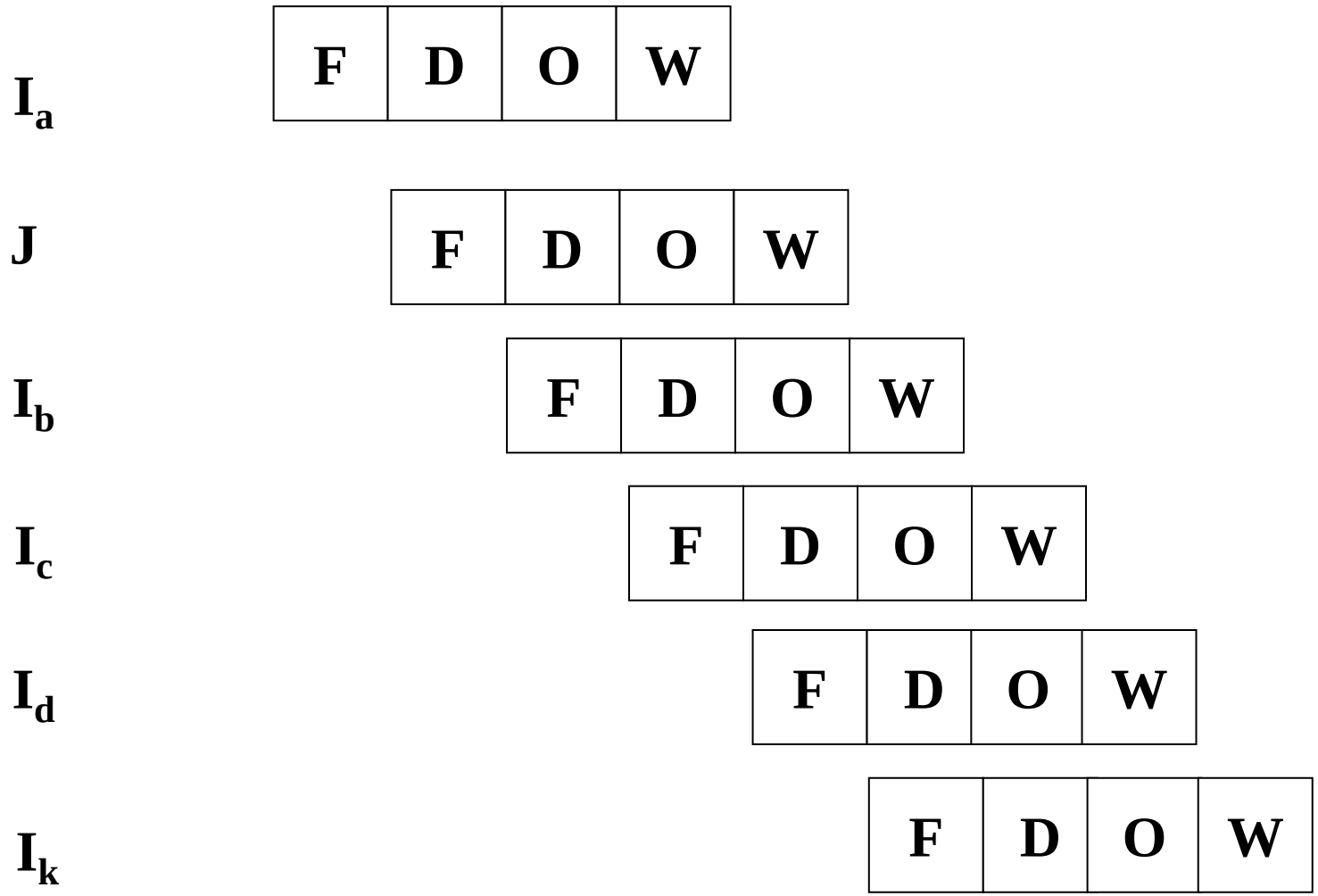
codice modificato

I_a
I_b
I_c
I_d
J lab_1
NOP
NOP
NOP
I_j
lab_1: I_k

Inserimento di NOP

Ottimizzazione

L'istruzione di salto viene anticipata ed eseguita prima di un numero istruzioni (se possibile) pari al *branch delay slot*, in modo da non far eseguire al processore le istruzioni NOP.



Salti condizionati

In questo caso è necessario attendere l'esecuzione dell'istruzione di salto per sapere se il salto deve essere eseguito o meno.

Nel frattempo però la coda delle istruzioni è stata riempita sequenzialmente.

Se il salto viene poi eseguito, vi saranno alcune istruzioni provenienti dalla coda che non dovranno essere eseguite (tante quante la dimensione del branch delay slot).

Se il salto non viene eseguito, si può procedere normalmente.

Predizione di salto

Se si usa una tecnica HW, la predizione permette di ridurre la penalizzazione introdotta da un'istruzione di salto.

Consiste nel far sì che l'unità di fetch disponga di una previsione sul fatto che il salto venga effettuato o meno.

Se la previsione è affermativa, essa carica le istruzioni a partire da quella destinazione; se è negativa, continua a caricare le istruzioni successive a quella di salto.

Successivamente, si verificherà se la previsione era corretta; in caso negativo si deve svuotare la pipeline.

I processori RISC

A partire dali primi anni ‘80 furono progettati e poi realizzati alcuni processori caratterizzati da

- **Elevata semplicità**
- **Numero ridotto di istruzioni (da cui la sigla RISC, che sta per *Reduced Instruction Set Computer*)**
- **Elevato numero di registri**
- **Architettura a pipeline.**

Negli anni ‘90 esistevano sul mercato varie famiglie di processori RISC, tra cui

- **ARM**
- **SPARC (Sun)**
- **MIPS**
- **PowerPC (IBM).**

Caratteristiche dei RISC: 1 istruzione = 1 periodo di clock

Ogni istruzione corrisponde a varie operazioni:

- **fetch degli operandi dai registri**
- **attivazione della ALU**
- **memorizzazione del risultato in un registro.**

Grazie all'adozione della pipeline, è possibile sovrapporre temporalmente l'esecuzione di più istruzioni, in modo che ad ogni periodo di clock si termini normalmente l'esecuzione di una istruzione.

Per poter utilizzare al meglio la pipeline è necessario che le istruzioni siano

- **Semplici**
- **Regolari (formato fisso).**

Caratteristiche dei RISC: unità di controllo

Le istruzioni RISC hanno la complessità delle microistruzioni CISC; per questa ragione l'unità di controllo dei RISC può essere realizzata con la tecnica cablata, anziché microprogrammata.

Caratteristiche dei RISC: LOAD & STORE

I processori RISC hanno normalmente due sole istruzioni che coinvolgono la memoria

- **LOAD (memoria ⇒ registro)**
- **STORE (registro ⇒ memoria).**

Questo permette di concentrare su 2 sole istruzioni gli sforzi per ridurre l'impatto negativo legato al calcolo dell'indirizzo e all'accesso in memoria.

Tali istruzioni sono le uniche a prevedere dei meccanismi per specificare l'indirizzo a cui accedere in memoria (modi di indirizzamento).

Caratteristiche dei RISC: formato delle istruzioni

I RISC hanno un formato delle istruzioni fisso o con poche alternative: il codice operativo ha di solito una lunghezza fissa.

Ne conseguono alcuni vantaggi:

- **la decodifica del codice operativo può avvenire in parallelo con il caricamento degli operandi dai registri**
- **l'unità di controllo è più semplice**
- **la fase di fetch è più ottimizzata.**

Caratteristiche dei RISC: modi di indirizzamento

I RISC possiedono un numero limitato di modi di indirizzamento, che comunque vengono utilizzati solo nelle istruzioni LOAD e STORE.

I registri

I registri sono la forma di memoria con minore tempo di accesso in quanto:

- **risiedono sullo stesso chip della CPU**
- **sono costruiti con la tecnologia più veloce**
- **sono accessibili con un meccanismo di indirizzamento semplice.**

Si può guadagnare in efficienza di esecuzione in 2 modi:

- **aumentando il numero di registri**
- **ottimizzando il loro uso.**

I registri sostituiscono lo stack (ad es. per la memorizzazione dei parametri per le procedure, delle variabili locali e dell'indirizzo di ritorno), riducendo il numero di accessi nello stack (ossia memoria).

Dimensione del codice eseguibile

- **I compilatori per RISC sono più complessi di quelli per i CISC (pur producendo codice composto da istruzioni più semplici) in quanto devono ottimizzare il codice per evitare gli stalli**
- **Il codice generato per un RISC ha dimensioni comparabili con quelle per un CISC in quanto:**
  - **il numero di istruzioni RISC generate è maggiore, ma**
  - **ogni istruzione occupa un numero inferiore di byte**
- **Codici più corti sono più efficienti perché:**
  - **si riduce il numero di fetch**
  - **si riduce il numero di page fault.**

Conclusioni sull'approccio RISC

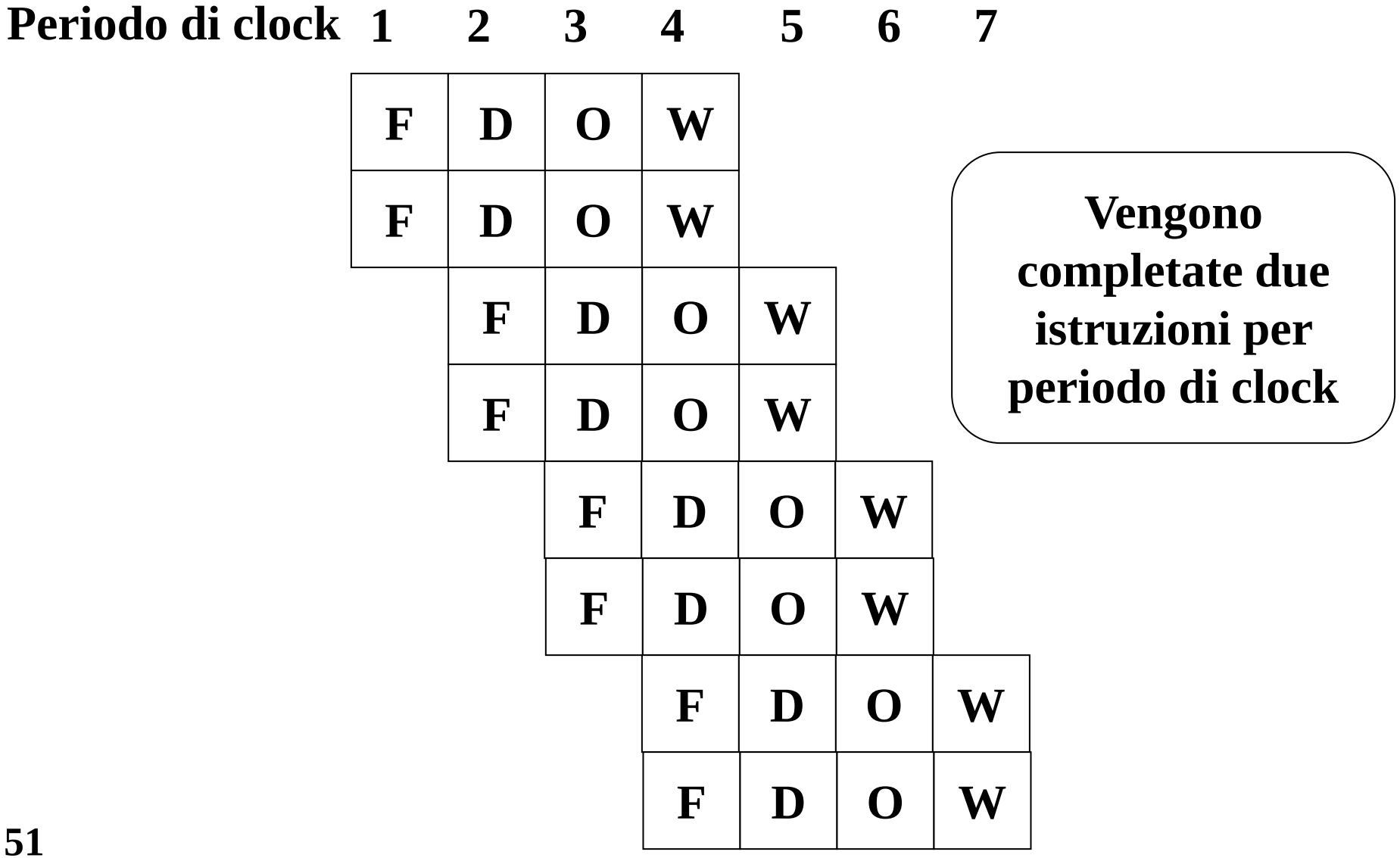
- **Le istruzioni CISC, per quanto più complesse, non sono più veloci, in quanto richiedono un hardware più complicato**
- **La gestione ottimizzata della pipeline delle istruzioni è più semplice con i RISC**
- **La disponibilità di un elevato numero di registri permette di ridurre significativamente il numero di accessi in memoria**
- **Nei RISC il tempo medio di attesa per il servizio di un interrupt (*latenza*) è inferiore rispetto ai CISC.**

I processori superscalari

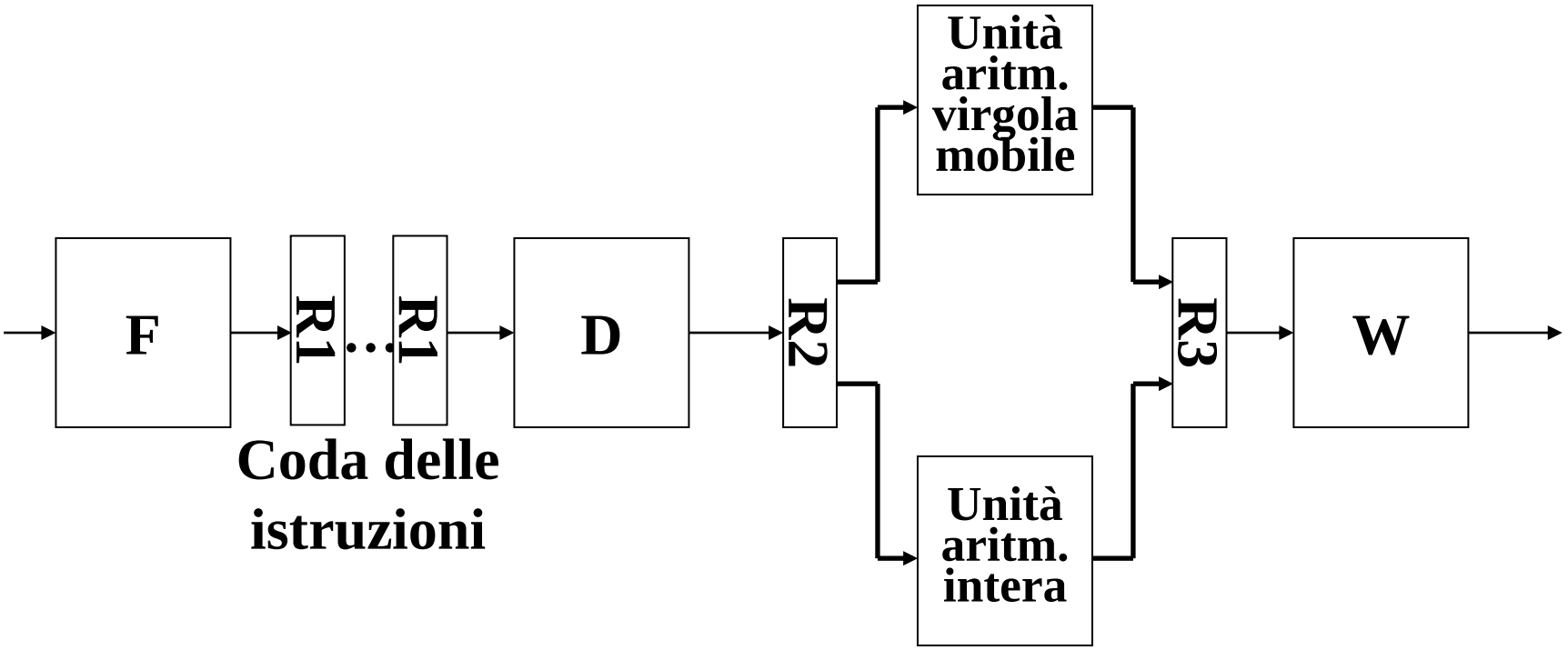
I processori superscalari possono completare l'esecuzione di più di un'istruzione per ciascun periodo di clock.

Internamente, contengono una pipeline in cui a ciascuno stadio corrisponde più di una unità.

Comportamento ideale

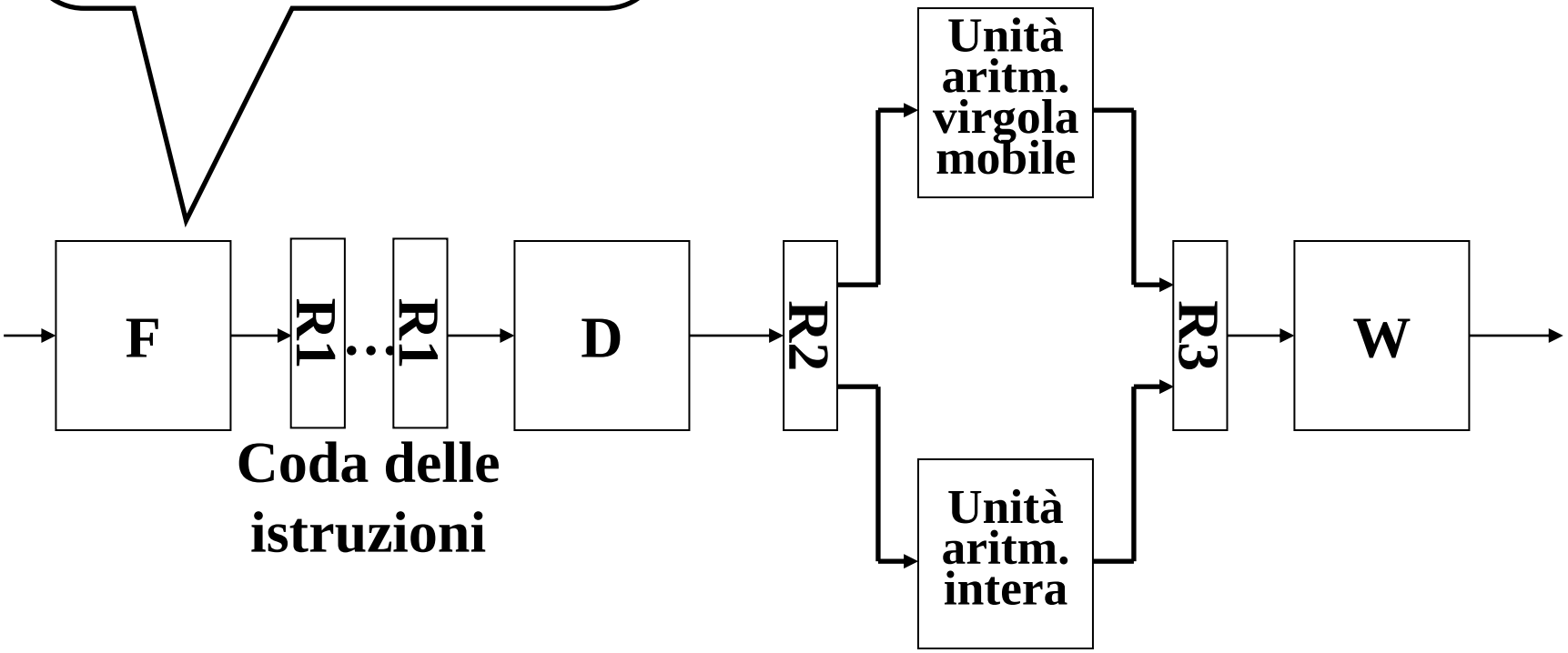


Esempio


F
R1
R1
D
R2
Unità aritm. virgola mobile
Unità aritm. intera
R3
W
Coda delle istruzioni

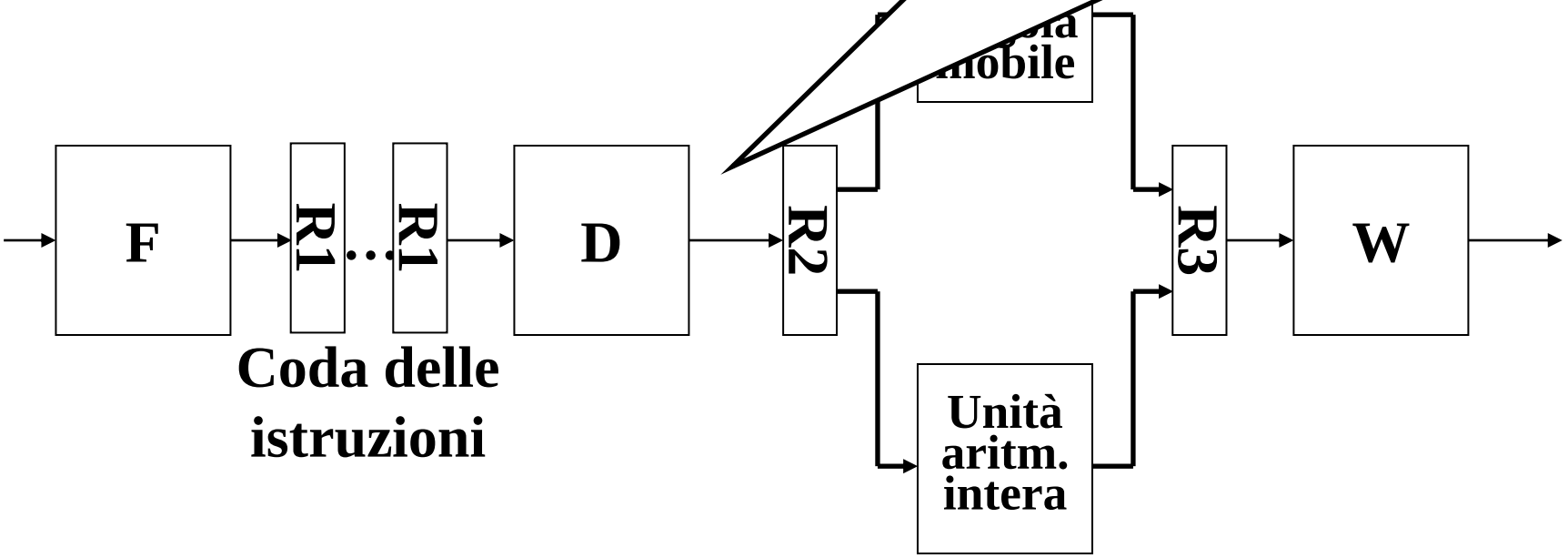
Esempio

L’unità di fetch può caricare 4 istruzioni per periodo di clock

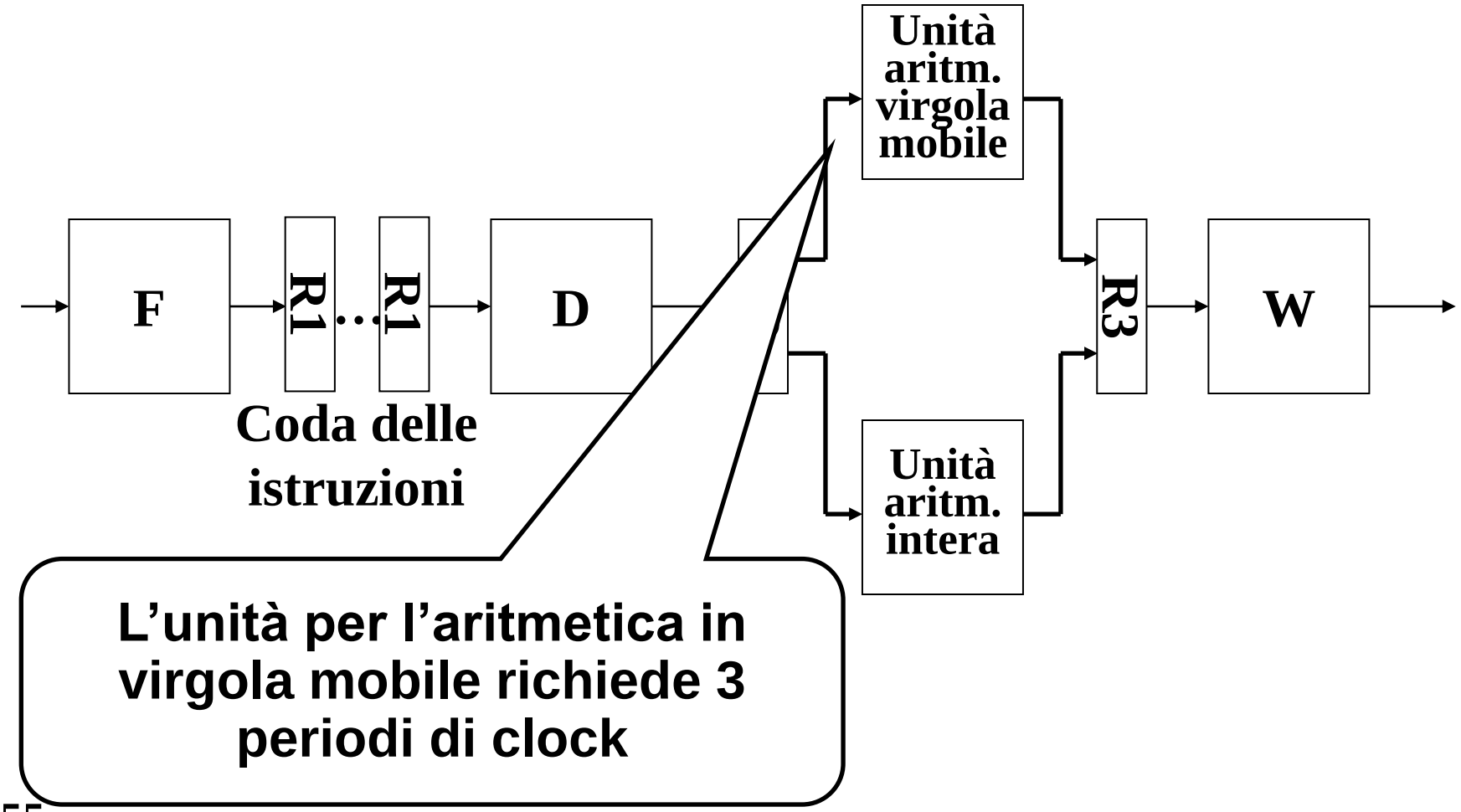


Ese

L'unità di decodifica può decodificare 2 istruzioni per periodo di clock e, se sono adatte, smistarle alle due unità aritmetiche dopo aver caricato i relativi operandi.

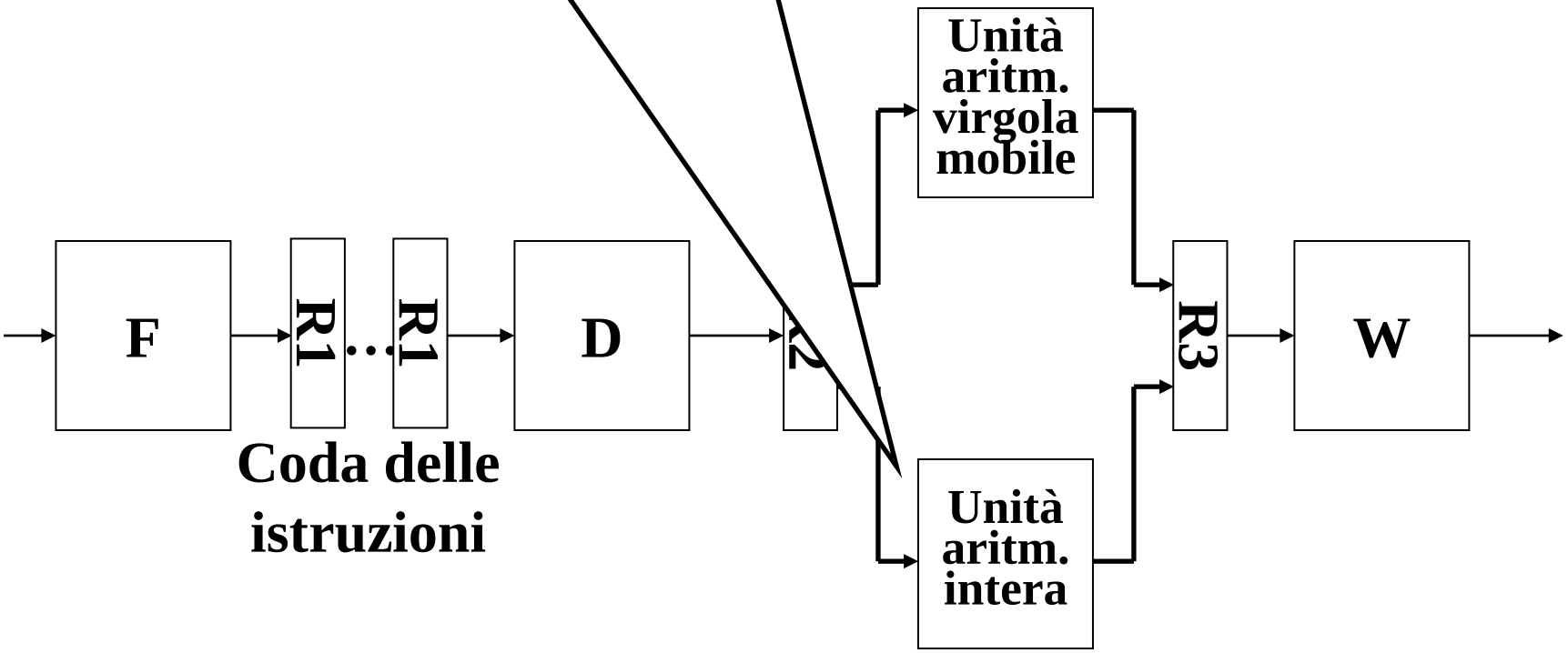


Esempio

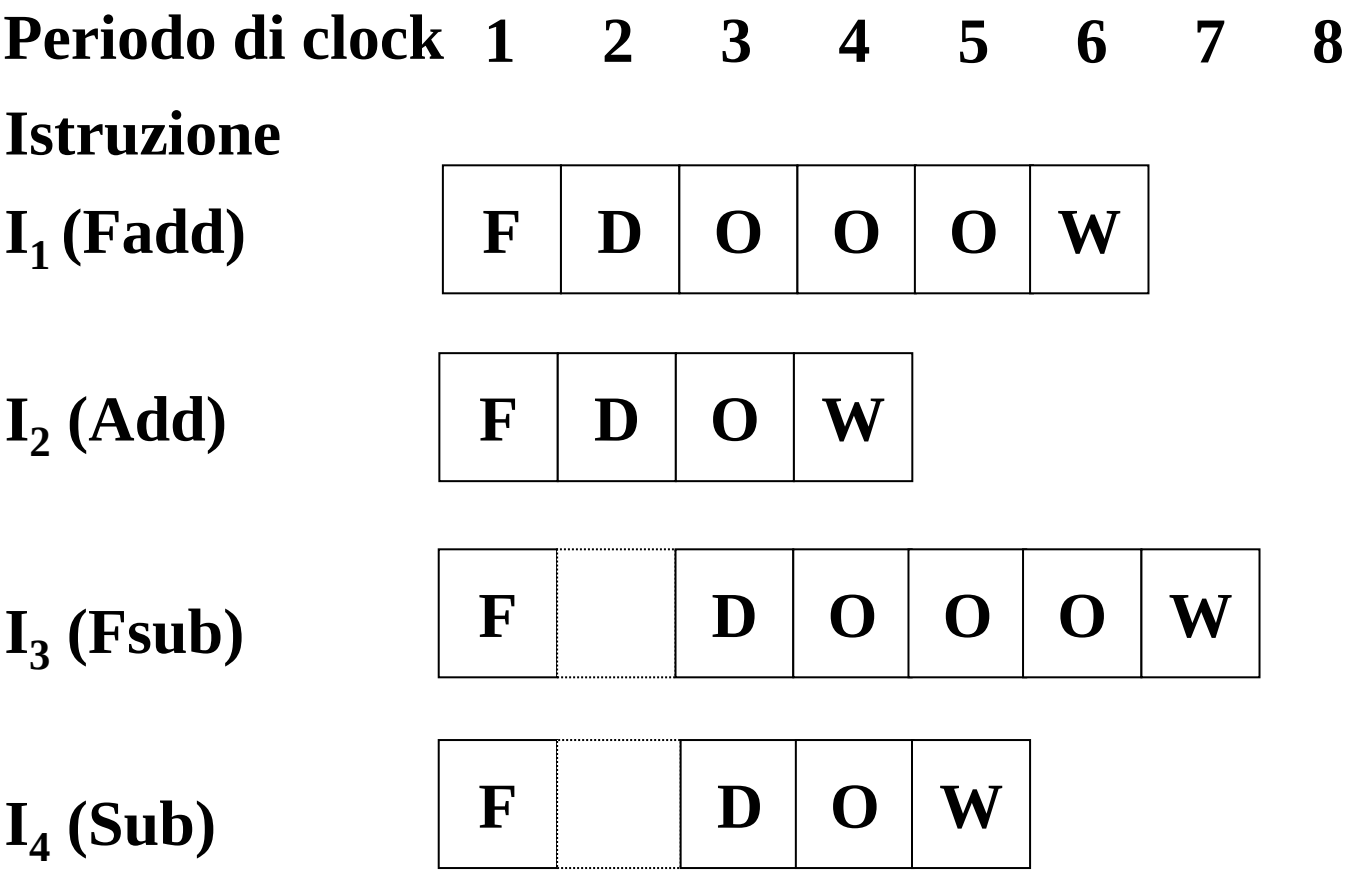

Unità aritm. virgola mobile
F
R1
R1
D
R3
W
Coda delle istruzioni
Unità aritm. intera
L'unità per l'aritmetica in virgola mobile richiede 3 periodi di clock

bio

L'unità per l'aritmetica intera richiede 1 periodo di clock



Flusso di esecuzione



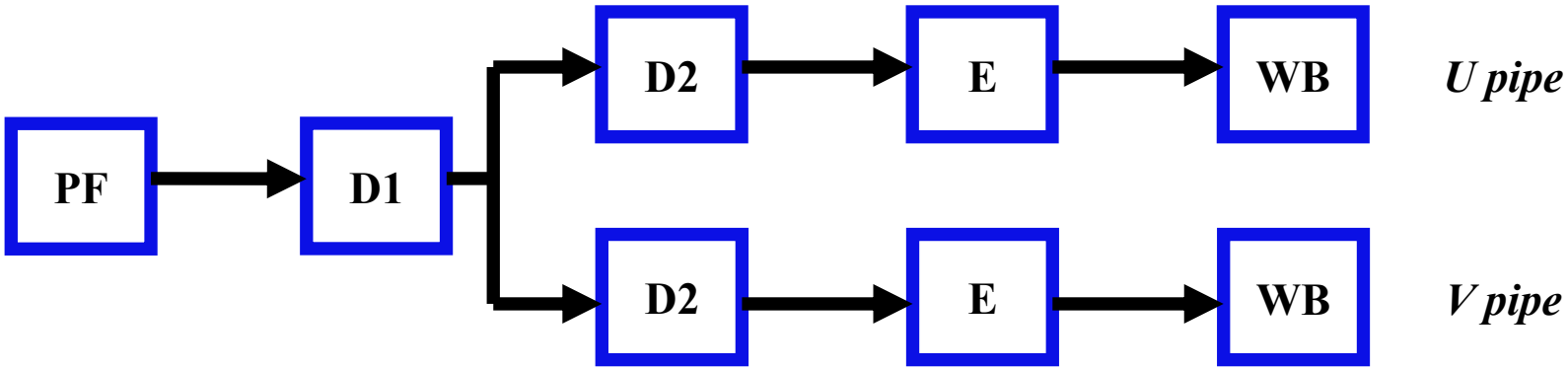
Note

- **Le unità appartenenti allo stesso stadio possono anche essere funzionalmente uguali**
- **La gestione della pipeline deve garantire la correttezza delle operazioni svolte, tenendo conto delle dipendenze tra dati**
- **L'esempio riporta un caso di *completamento non-in-ordine* delle istruzioni.**

Esempio: Intel Pentium

Processore Intel, della famiglia x86, prodotto dal 1993:

- **possedeva 2 pipeline che gli permettevano di completare più di una operazione per ciclo di clock:**
  - **"*pipeline U*" poteva eseguire qualunque istruzione,**
  - **"*pipeline V*" era in grado di eseguire solo quelle più semplici e comuni (logica cablata).**



Esempio: PowerPC

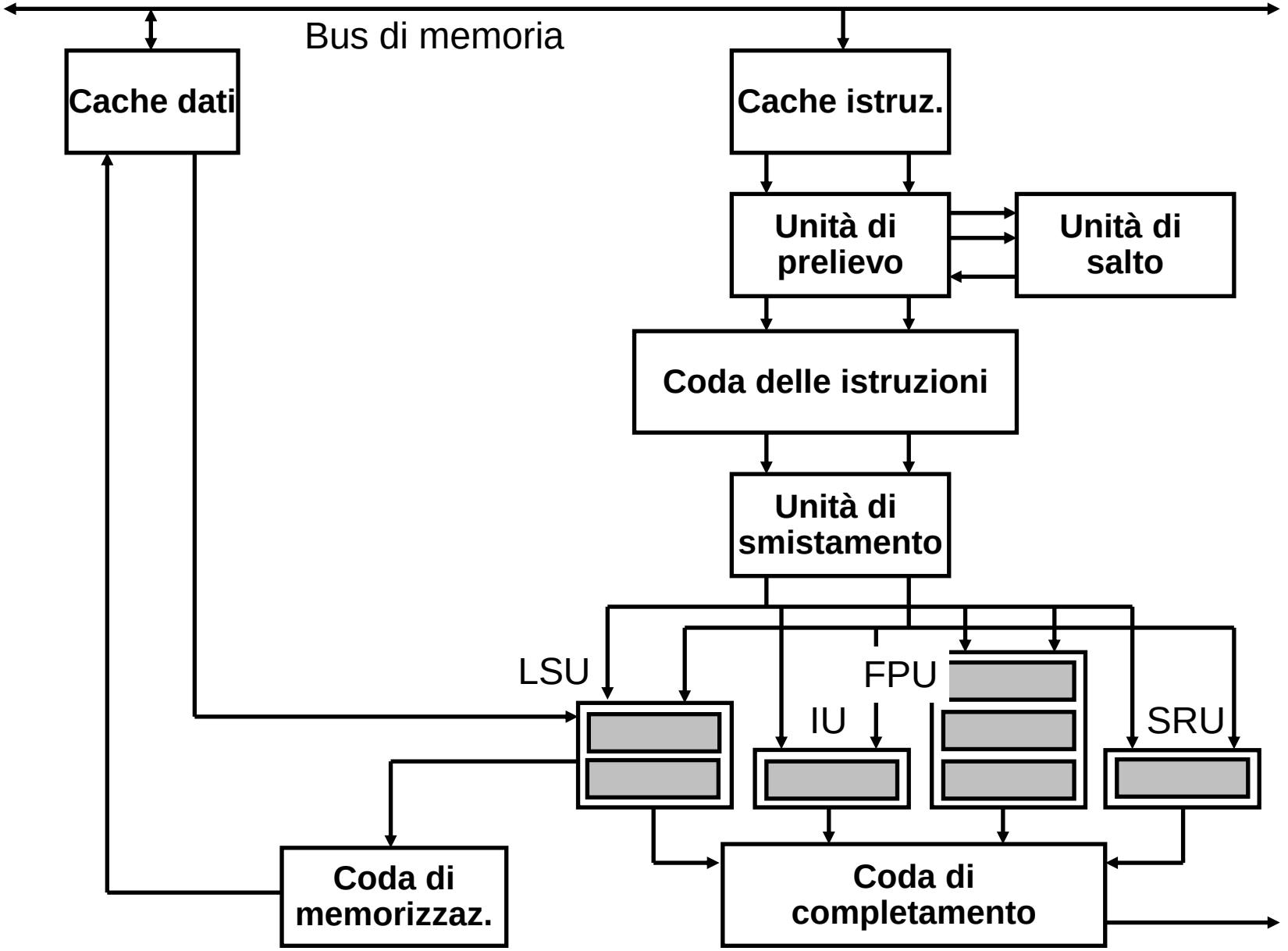
I processori PowerPC furono progettati e prodotti negli anni 80 da un'alleanza tra Apple, IBM e Motorola.

Il PowerPC fu progettato con principi RISC, e consentiva un'implementazione superscalare.

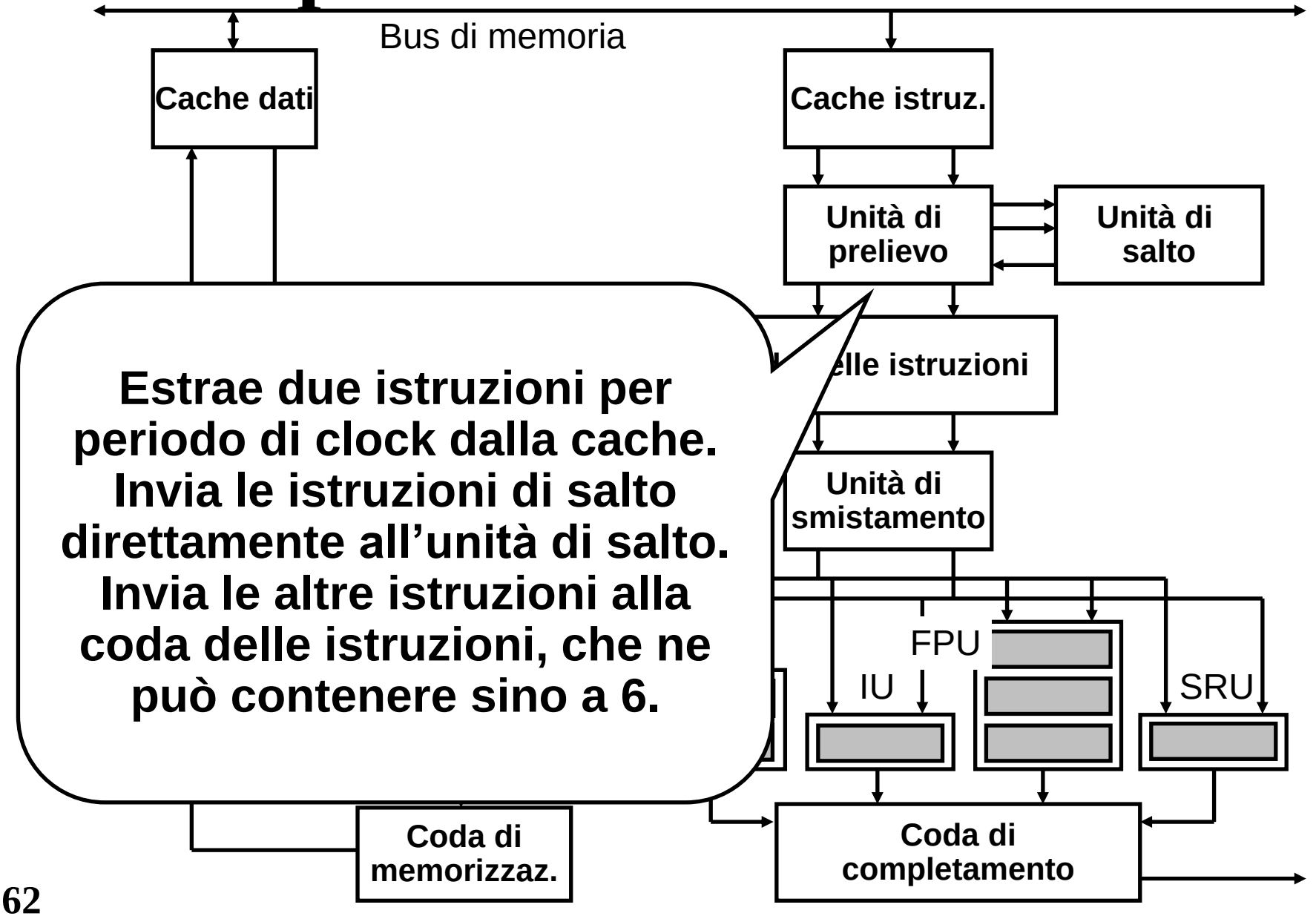
Furono prodotte versioni sia con implementazione a 32 bit che a 64 bit.

I processori PowerPC furono utilizzati inizialmente per applicazioni general-purpose. Ebbero poi grandissimo successo in campo special-purpose.

Architettura del PowerPC 603

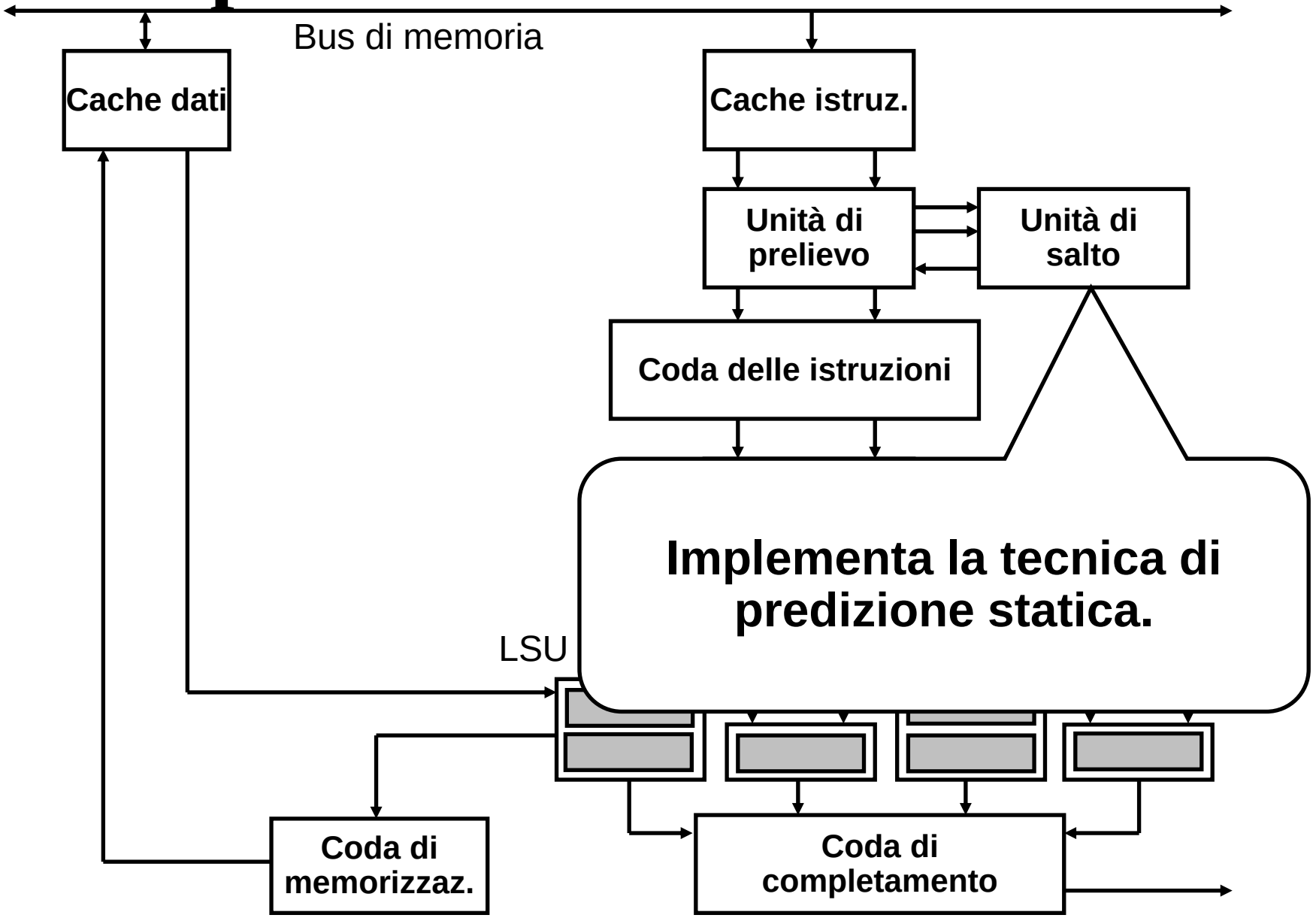


Pipeline del PowerPC 603

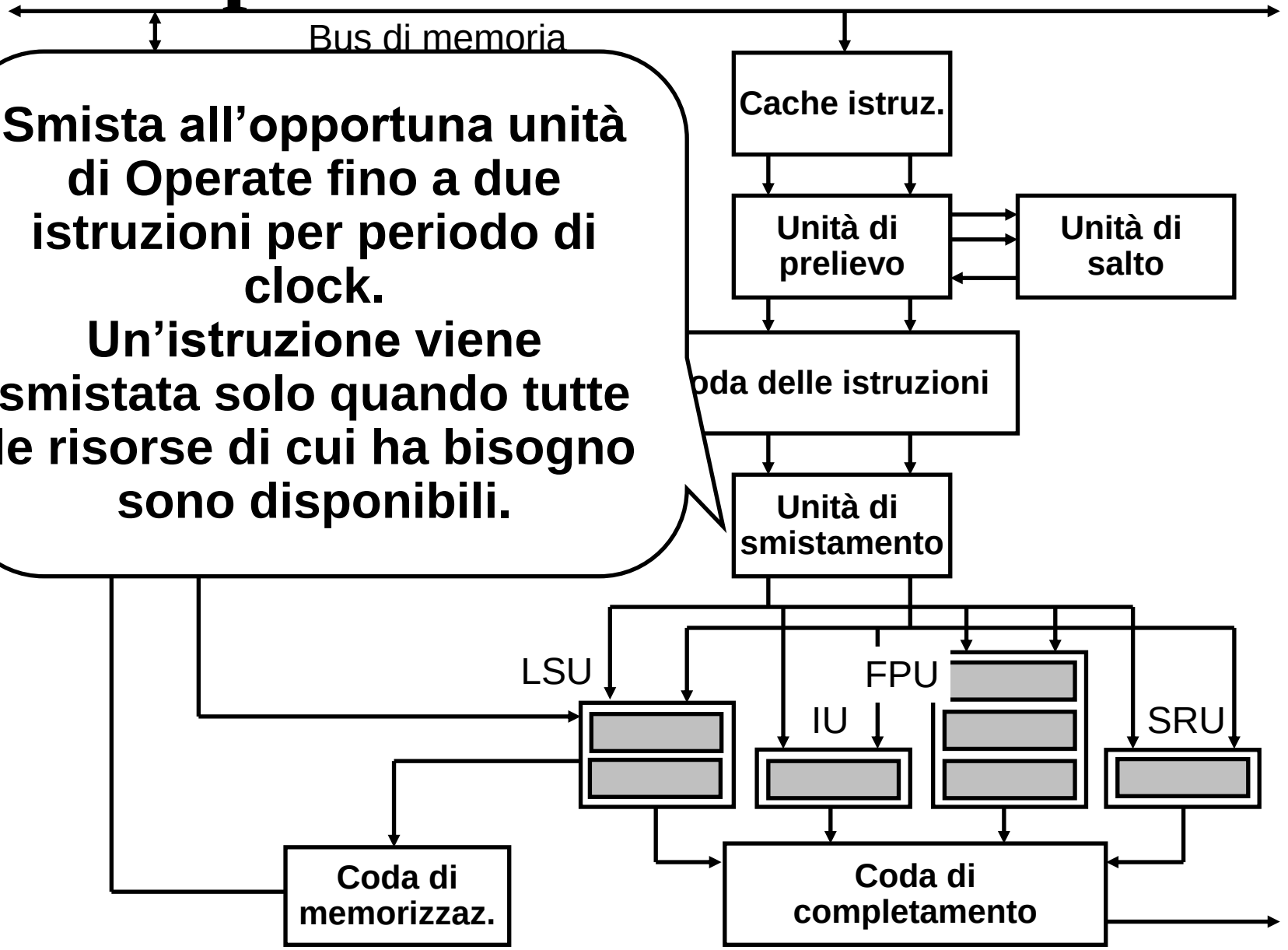


Estrae due istruzioni per periodo di clock dalla cache. Invia le istruzioni di salto direttamente all'unità di salto. Invia le altre istruzioni alla coda delle istruzioni, che ne può contenere sino a 6.

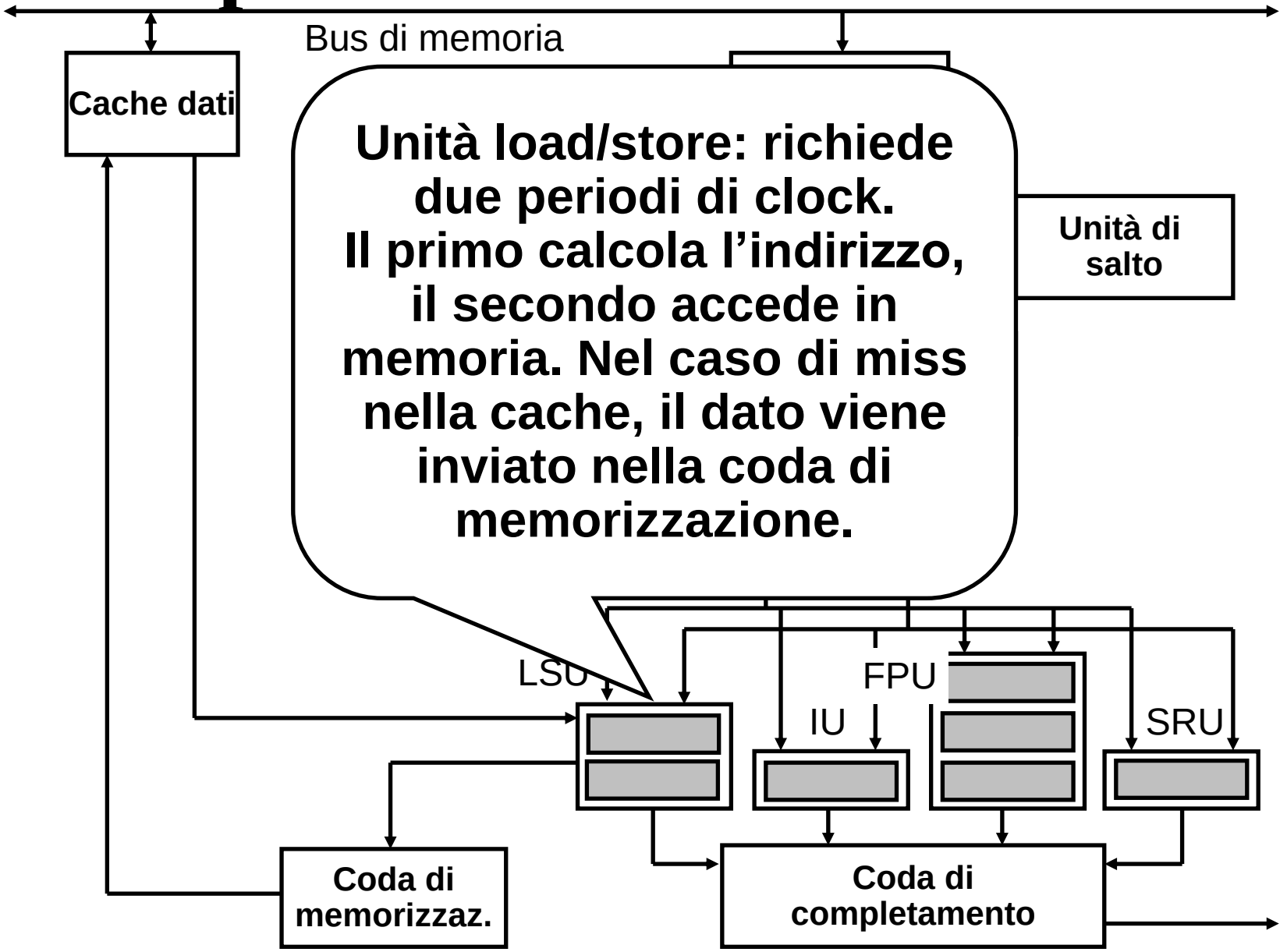
Pipeline del PowerPC 603



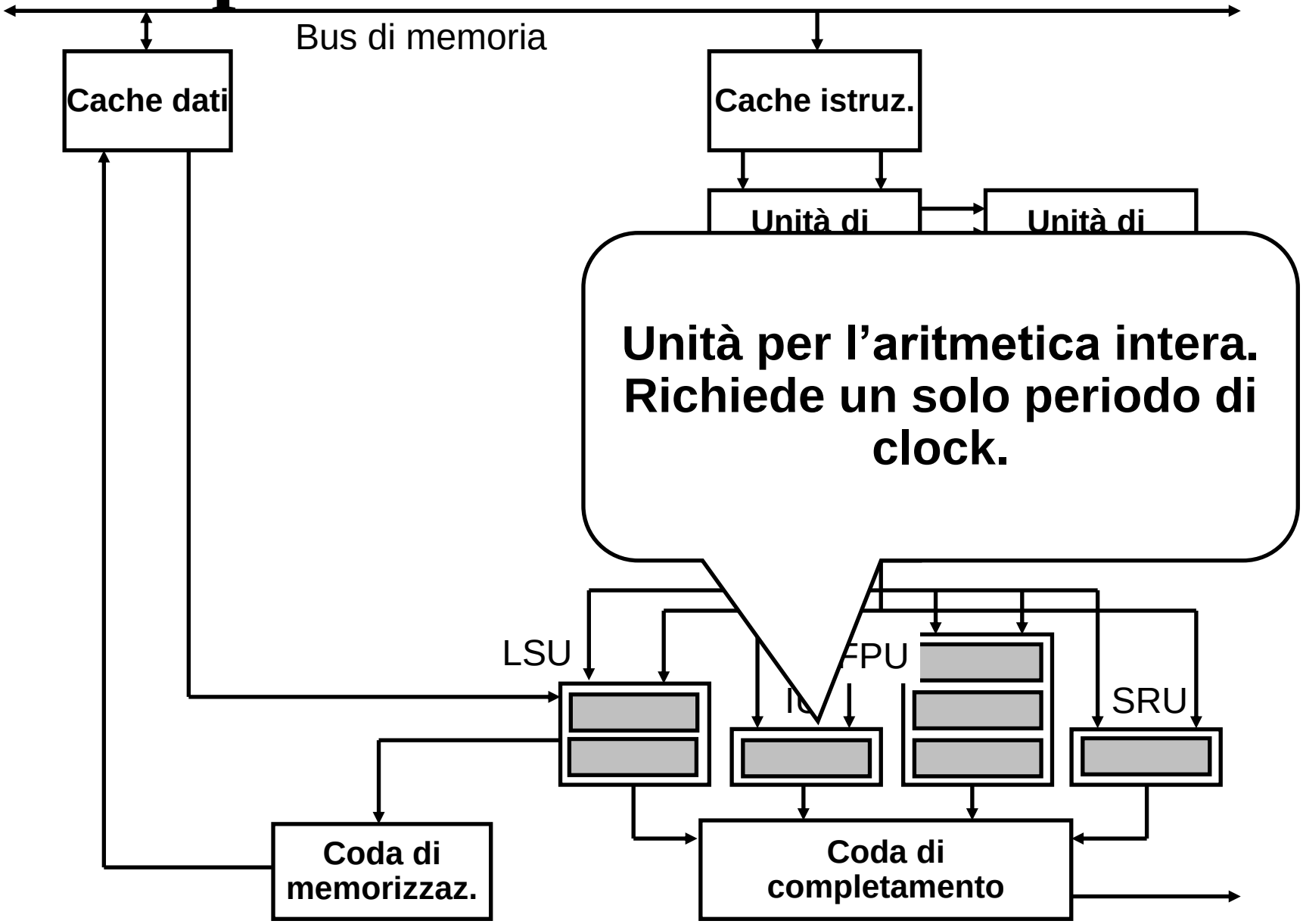
Pipeline del PowerPC 603



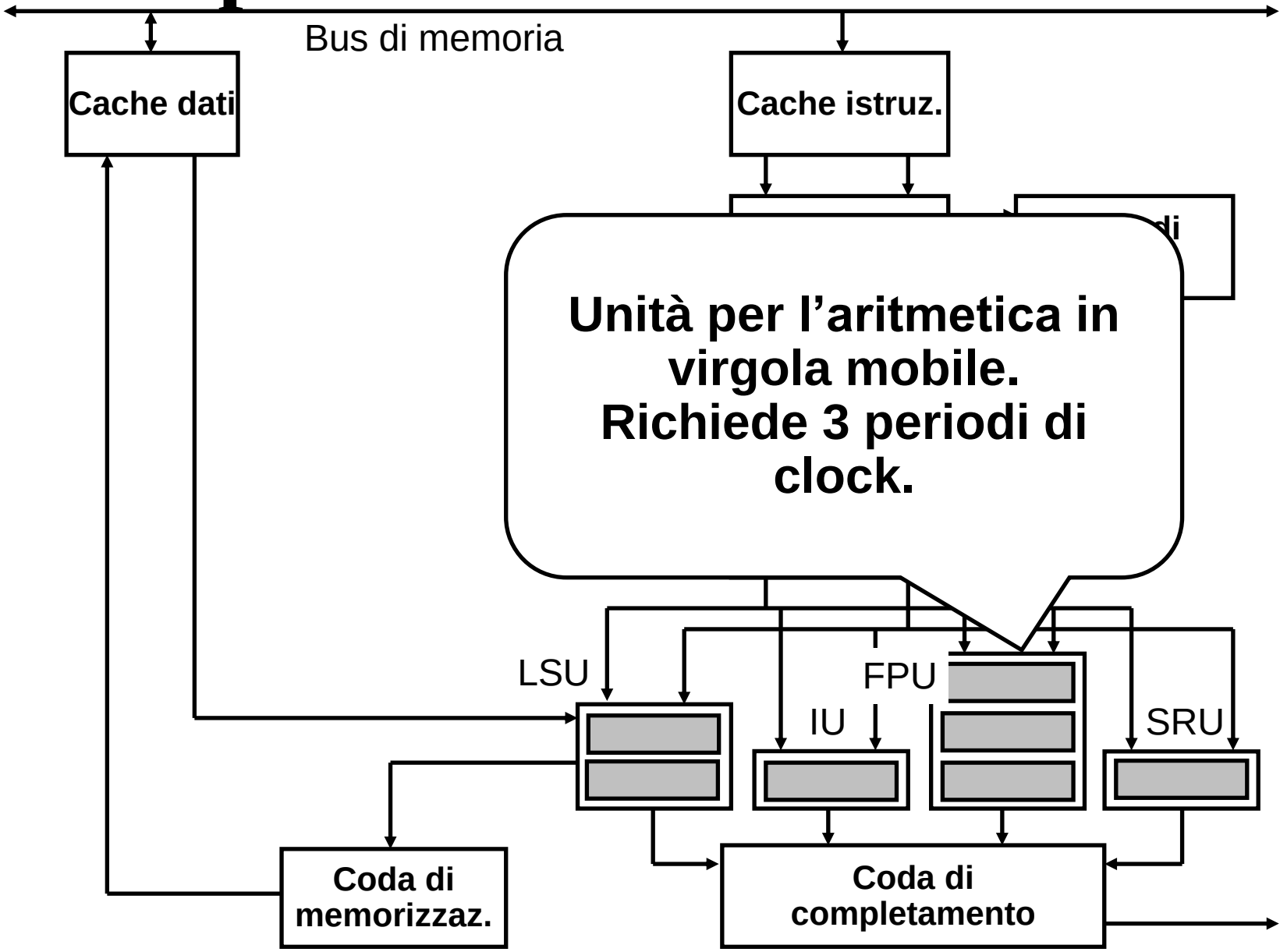
Pipeline del PowerPC 603



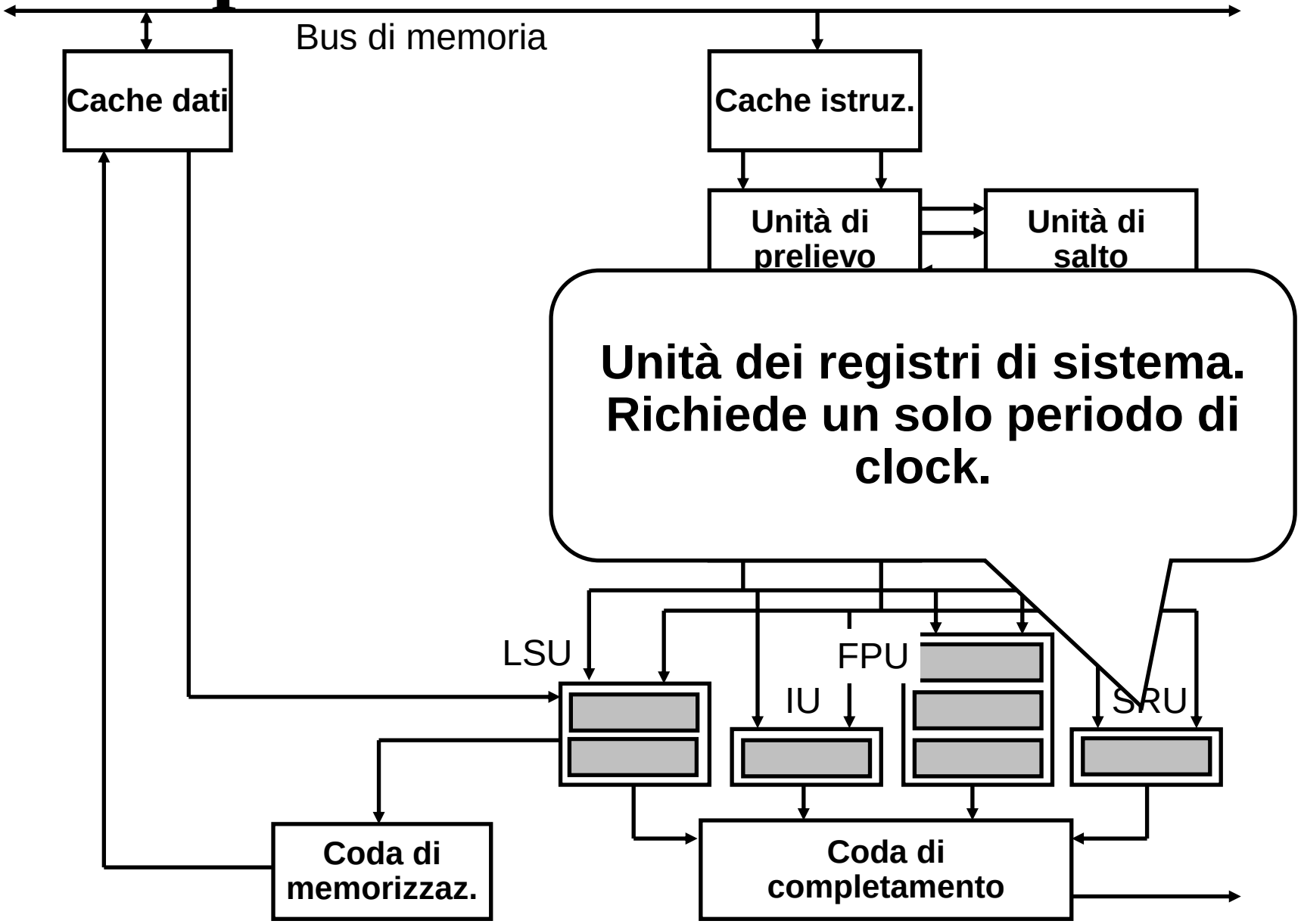
Pipeline del PowerPC 603



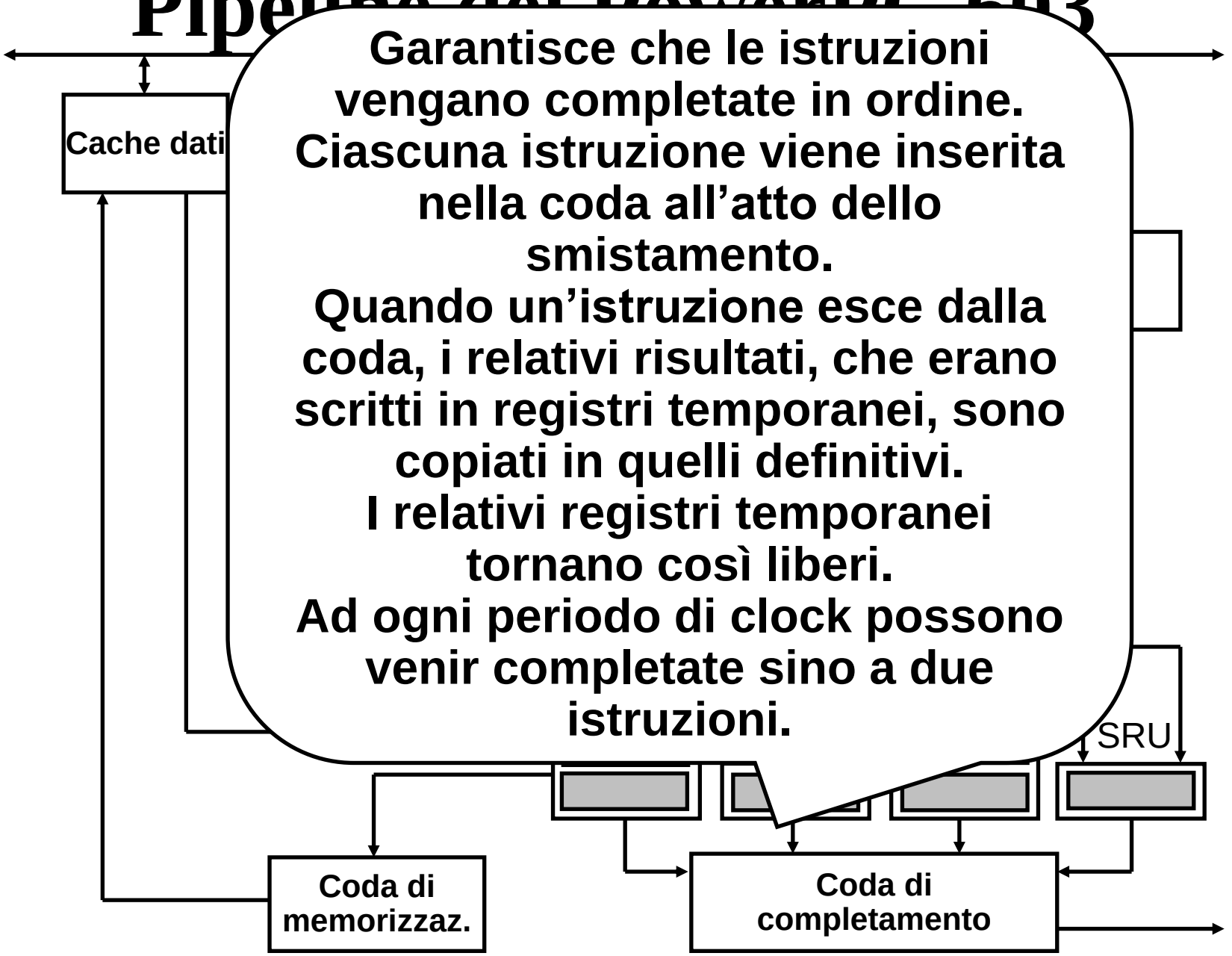
Pipeline del PowerPC 603



Pipeline del PowerPC 603



Pipeline del PowerPC 603



Numero di stadi della pipeline

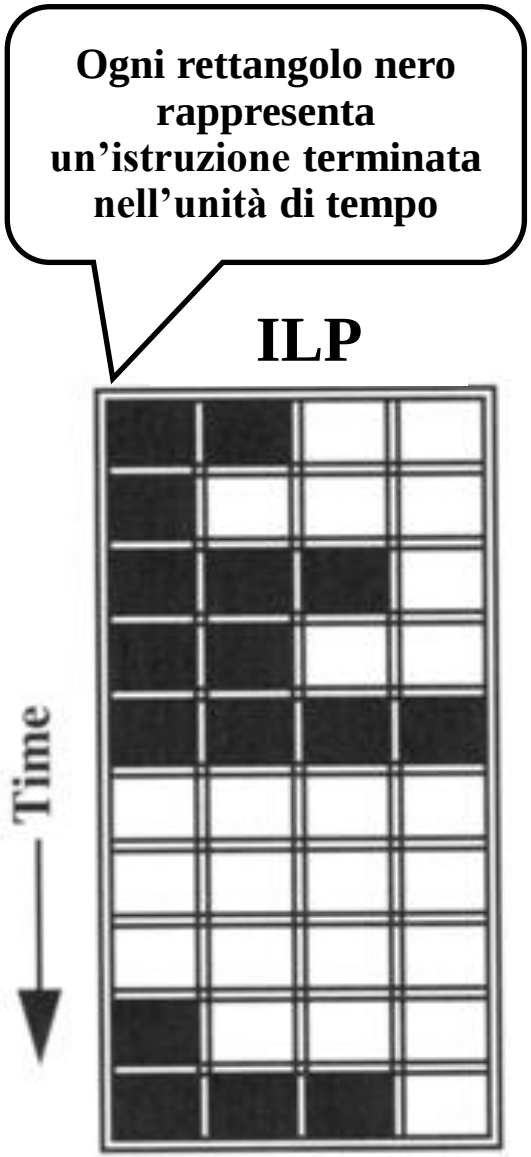
Se cresce, permette teoricamente di aumentare le prestazioni del processore.

Ma crescendo il numero di stadi, cresce la probabilità di dipendenze, e quindi che la pipeline debba andare in stallo.

Instruction Level Parallelism

I processori con pipeline sfruttano il fatto che le istruzioni di un programma possono in molti casi essere eseguite in parziale sovrapposizione (*Instruction Level Parallelism*, o ILP).

Le architetture dei processori superscalari sono in grado di sfruttare al meglio tale forma di parallelismo.

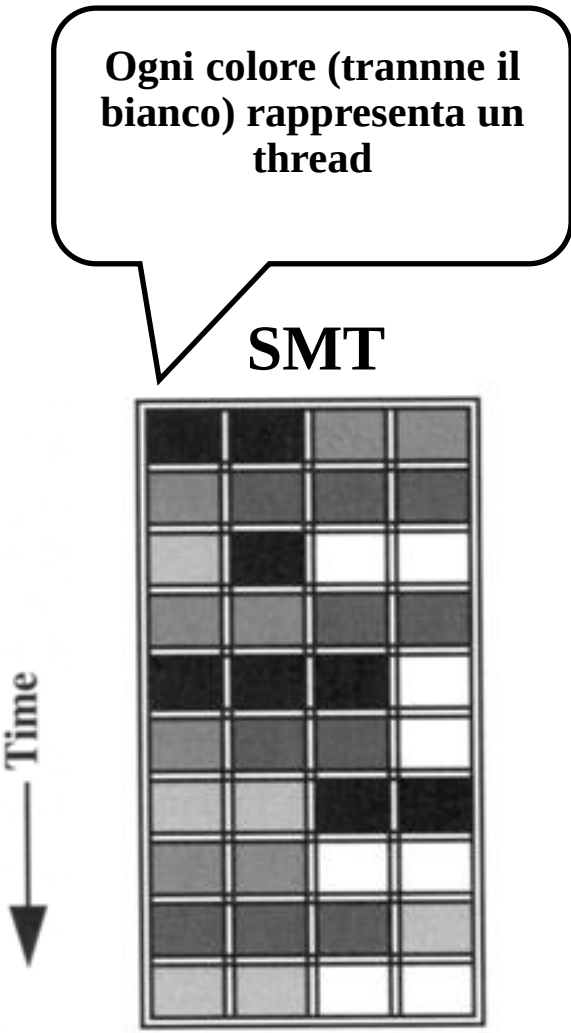


Processori multithread

Per ottenere ulteriori incrementi di prestazioni è necessario individuare e sfruttare forme diverse di parallelismo, in particolare quello a livello di thread (*Thread Level Parallelism*, o TLP).

Un thread corrisponde ad una porzione di programma con i suoi dati e un programma può essere partizionato su più thread.

I processori multithread (*Simultaneous Multi-Threading*, SMT) eseguono in parallelo più thread.



Processori multicore

Un’altra tendenza degli ultimi anni è quella ad integrare nello stesso dispositivo più processori.

Ciascun processore ha una complessità non troppo elevata, ed è in grado di eseguire più thread.

In tal modo (se si riescono a sfruttare tutti i core) si ottengono prestazioni elevate con bassi consumi.

Esempio: Intel Sandy Bridge

L'architettura Intel Sandy Bridge fu sviluppata a partire dal 2005. Il primo prodotto basato su quest'architettura fu commercializzato a partire dal 2011.

L'architettura permette di combinare fino a 8 core.

Altre caratteristiche dell'architettura:

- **Cache L1: 32 kB data + 32 kB instruction per core**
- **Cache L2: 256 kB per core**
- **Cache L3 cache condivisa (fino a 20MB).**

Esempio: ARM Cortex-A9

ARM produce IP core di processori per applicazioni embedded.

Il Cortex-A9 può essere utilizzato da solo o in configurazione multicore.

Rappresenta una soluzione per applicazioni in cui sono richieste alte prestazioni e basso consumo.