Lecture
9_1.2

Pipelines

cini
Cybersecurity
National Lab

Paolo PRINETTO
Politecnico di Torino (Italy)
Univ. of Illinois at Chicago, IL (USA)
CINI Cybersecurity Nat. Lab. (Italy)
Paolo.Prinetto@polito.it
www.consorzio-cini.it
www.comitato-girotondo.org

License Information

This work is licensed under the
Creative Commons BY-NC
License

To view a copy of the license, visit:
http://creativecommons.org/licenses/by-nc/3.0/legalcode

Rel. 28.05.2019

© Prinetto - 2019

Disclaimer

- We disclaim any warranties or representations as to the accuracy or completeness of this material.
- Materials are provided “as is” without warranty of any kind, either express or implied, including without limitation, warranties of merchantability, fitness for a particular purpose, and non-infringement.
- Under no circumstances shall we be liable for any loss, damage, liability or expense incurred or suffered which is claimed to have resulted from use of this material.

Rel. 28.05.2019
© Prinetto - 2019

Goals

– The Lecture presents the basic concepts related to Pipelines organization and performance optimizations within processors

Rel. 28.05.2019
© Prinetto - 2019

Prerequisites

Rel. 28.05.2019

© Prinetto - 2019

Homework

– None

Rel. 28.05.2019

© Prinetto - 2019

Further readings

- **Students interested in making a reference to a text book on the arguments covered in this lecture can refer, for instance, to:**
  - ***G. Conte, A. Mazzeo, N. Mazzocca, P. Prinetto*: "Architettura dei calcolatori", *Città Studi, 2015* (Chapter 3 *Architettura interna del processore*)**

Outline

- Introduction
- Pipelines
- Dependency constraints
- Jump Instructions

Rel. 28.05.2019
© Prinetto - 2019

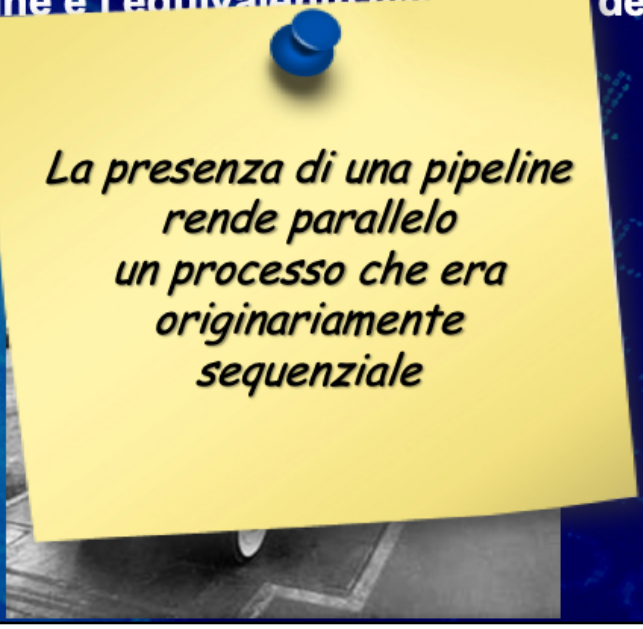
Pipeline

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

© Prinetto - 2019

Pipeline

- La pipeline è l'equivalente elettronico della catena



© Prinetto - 2019

Frederick Winslow Taylor

Tempi Moderni

Rel. 28.05.2019

© Prinetto - 2019

Pipeline

- Permette di eseguire in parallelo fasi diverse sullo stesso flusso di dati.

Rel. 28.05.2019
© Prinetto - 2019

Pipeline ideale

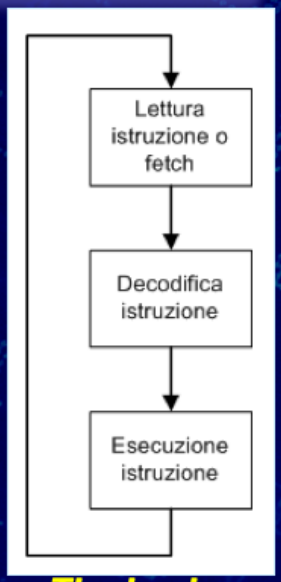
- Se:
  - tutti i dati dovessero attraversare gli stessi stadi
  - tutti gli stadi richiedessero sempre lo stesso tempo per eseguire la relativa elaborazione

  allora l'uso di una pipeline di n stadi potrebbe portare, a regime, un miglioramento di n volte nelle prestazioni.

Rel. 28.05.2019
© Prinetto - 2019

The basic Von-Neumann cycle

- La fase di esecuzione di ciascuna istruzione macchina vienesuddivisa in 2 sottofasi:
  - *Operate* (o elaborazione): esecuzione dell'operazione
  - *Write* (o scrittura): scrittura del risultato



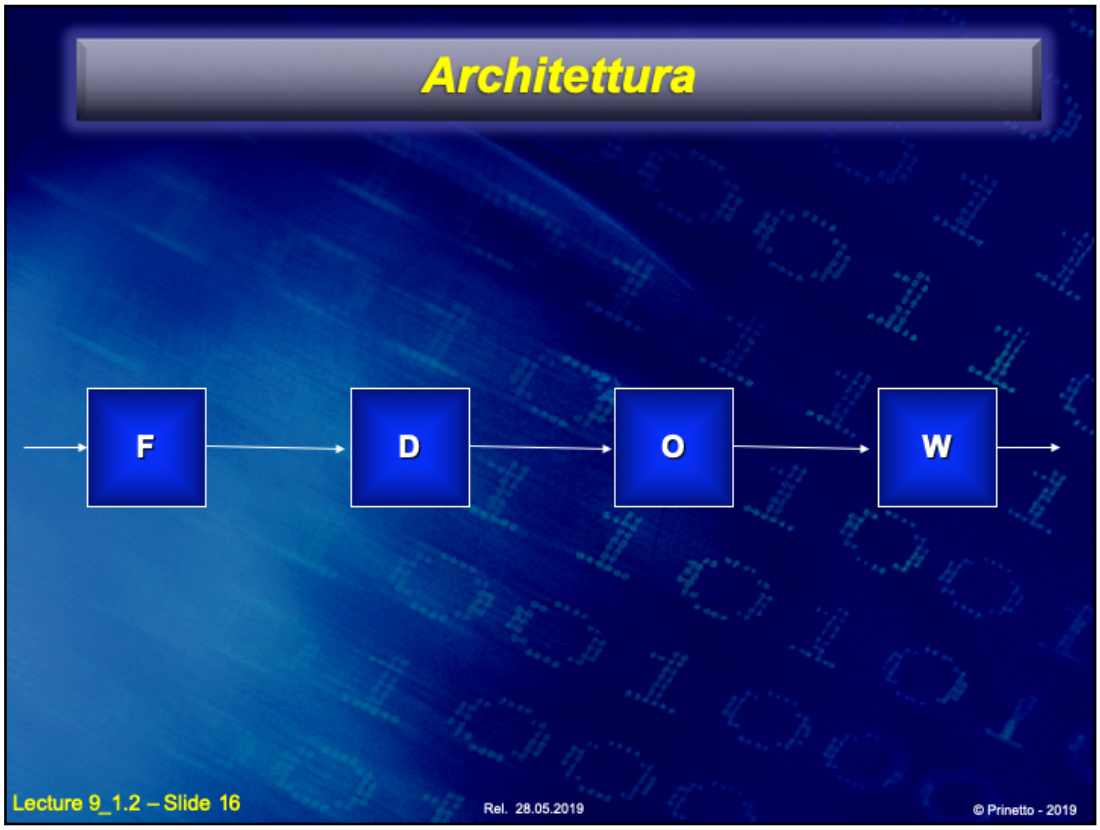
The basic Von-Neumann cycle

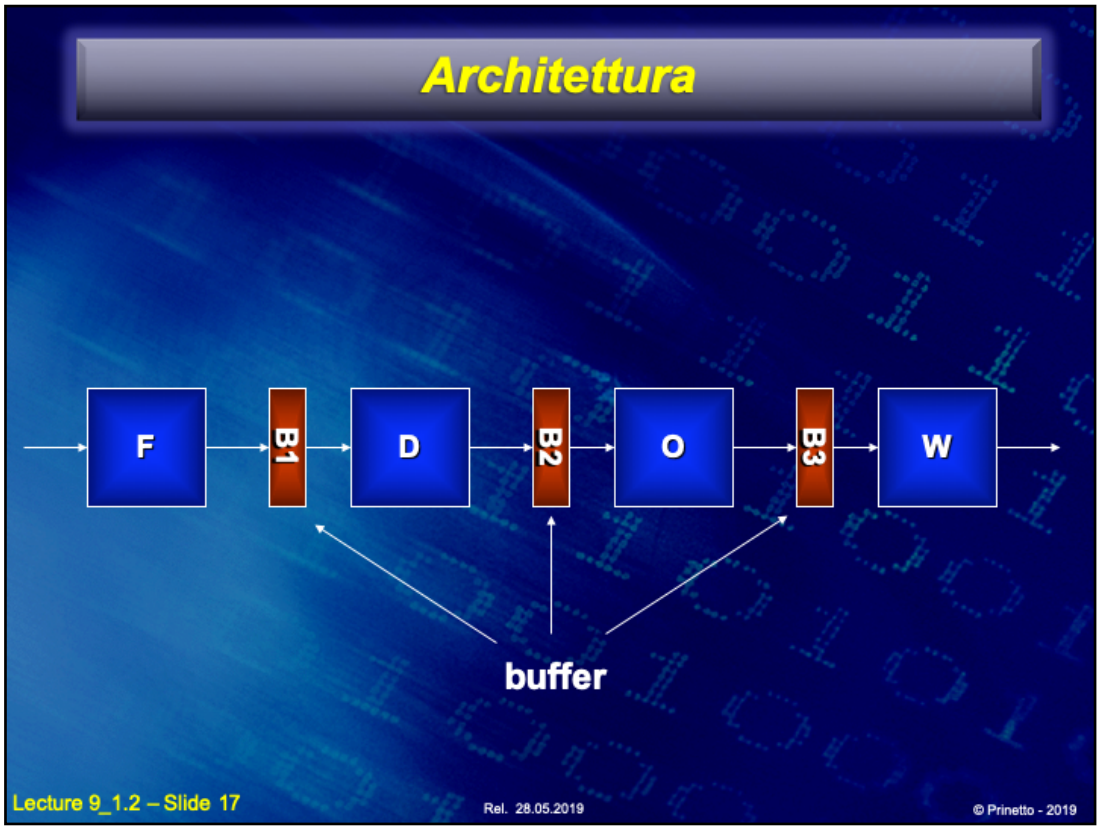
Rel. 28.05.2019
© Prinetto - 2019

La pipeline nei processori

- L'esecuzione di ciascuna istruzione risulta pertanto suddivisa in 4 fasi:
  - *Fetch*
  - *Decode*
  - *Operate*
  - *Write*

Rel. 28.05.2019
© Prinetto - 2019


Architettura
F
D
O
W
Lecture 9_1.2 – Slide 16
Rel. 28.05.2019
© Prinetto - 2019

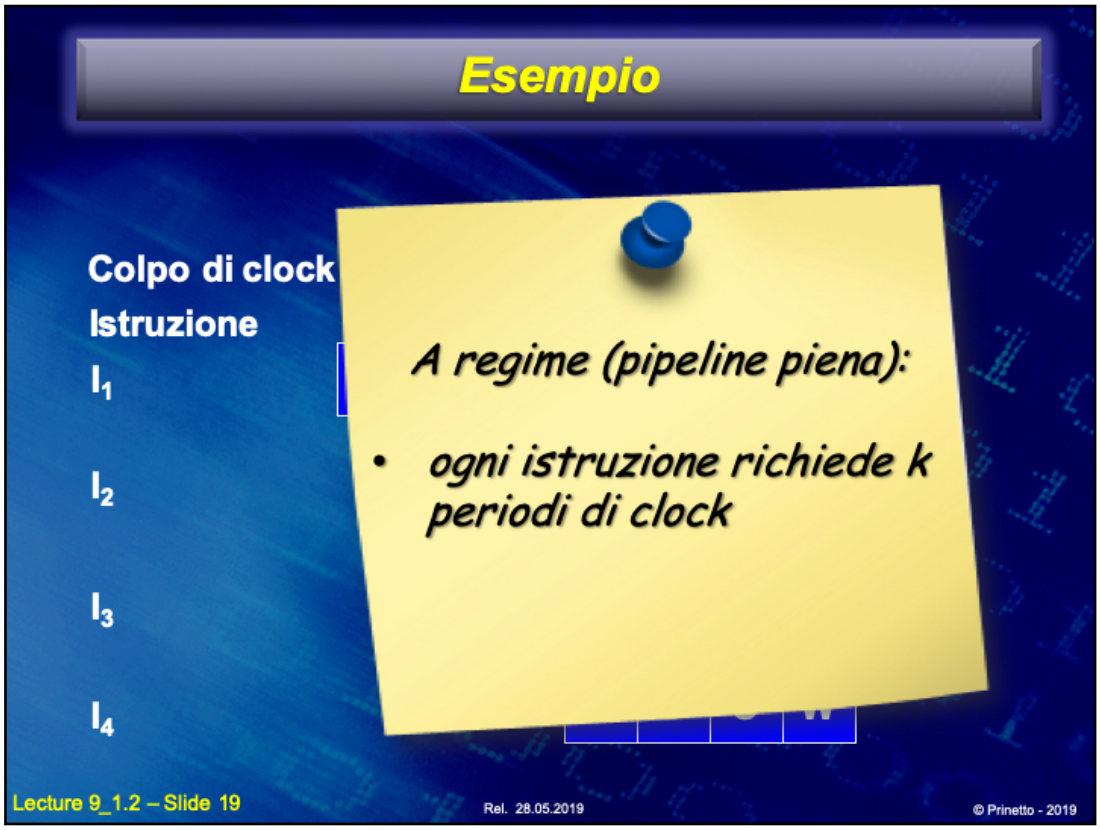

Architettura
F
B1
D
B2
O
B3
W
buffer
Lecture 9_1.2 – Slide 17
Rel. 28.05.2019
© Prinetto - 2019

Esempio

Rel. 28.05.2019

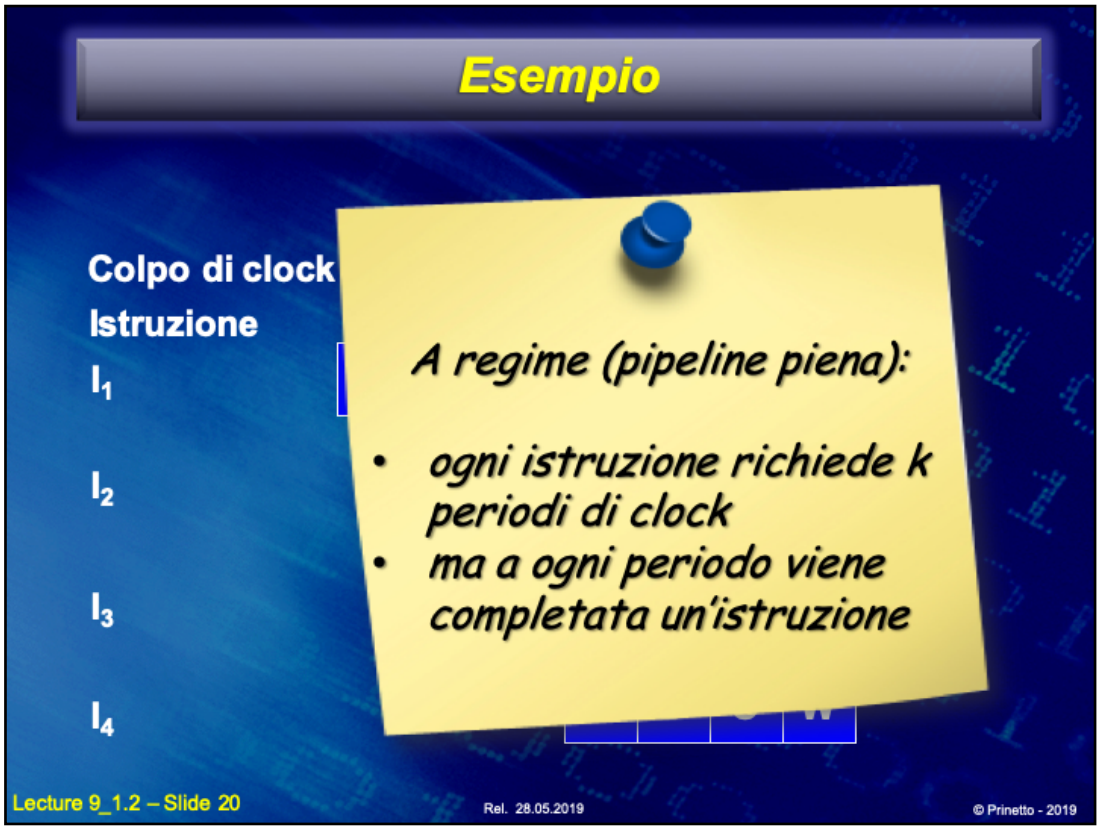
© Prinetto - 2019

Esempio



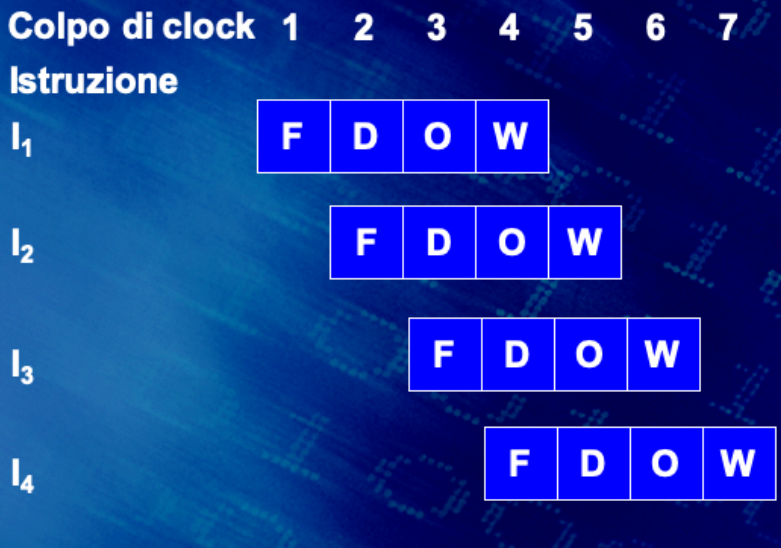
Rel. 28.05.2019
© Prinetto - 2019

Esempio



Rel. 28.05.2019
© Prinetto - 2019

Esempio



Rel. 28.05.2019

© Prinetto - 2019

Definizioni

Latenza = tempo di esecuzione di una istruzione

Throughput = frequenza di esecuzione delle istruzioni = 1 / Latenza

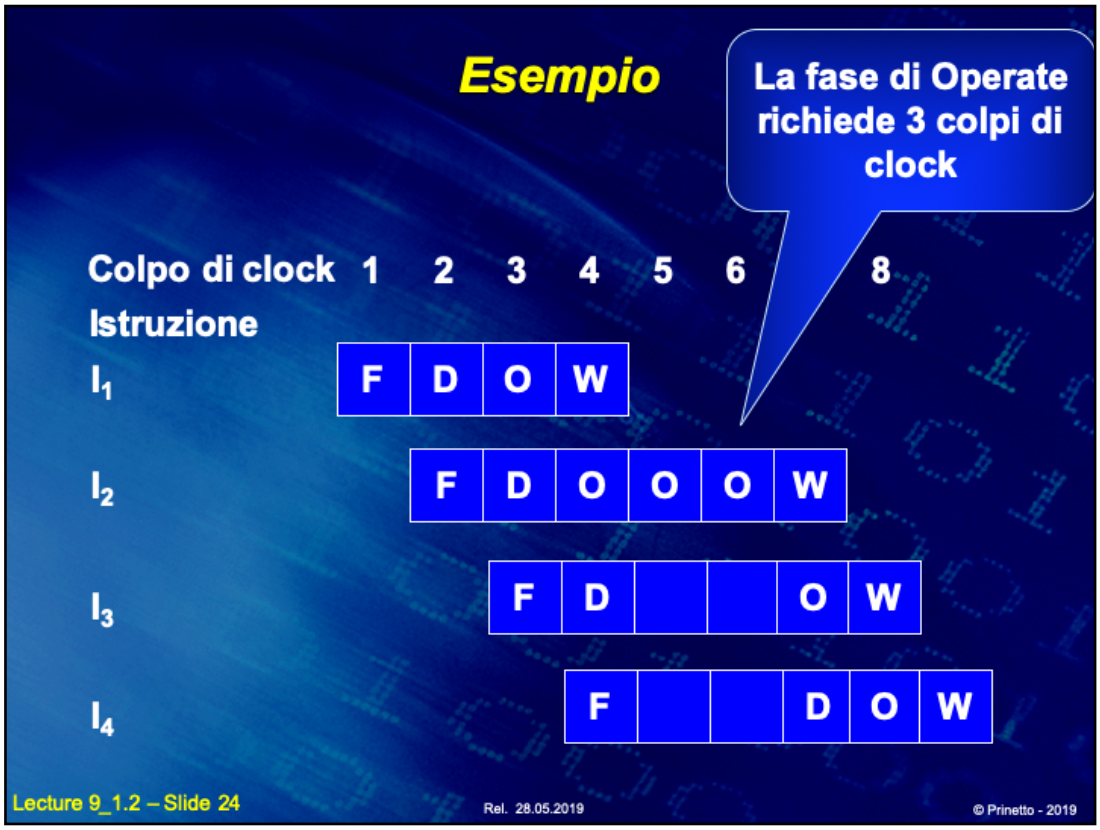
Rel. 28.05.2019

© Prinetto - 2019

Stallo

- In un processore reale, è possibile che a volte uno stadio non riesca a completare il proprio compito in un colpo di clock.
- In tal caso gli altri stadi (sia a monte sia a valle) devono arrestarsi, ossia andare in ***stallo***, in quanto non dispongono dei dati necessari, o i buffer in cui devono scrivere non sono liberi.
- In tal modo si limitano le prestazioni del processore.

Rel. 28.05.2019
© Prinetto - 2019


Esempio
La fase di Operate richiede 3 colpi di clock
Colpo di clock 1 2 3 4 5 6 8
Istruzione
I1 F D O W
I2 F D O O O W
I3 F D O W
I4 F D O W
Lecture 9_1.2 – Slide 24
Rel. 28.05.2019
© Prinetto - 2019

Cause di stallo

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

Rel. 28.05.2019
© Prinetto - 2019

Cache

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

Rel. 28.05.2019
© Prinetto - 2019

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 (ma non subito).

Rel. 28.05.2019

© Prinetto - 2019

Coda delle istruzioni

- Quando l'unità di fetch non trova una 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*

Coda delle istruzioni

Rel. 28.05.2019
© Prinetto - 2019

Coda delle istruzioni

- L'unità di fetch cerca di mantenere sempre piena tale coda, caricando anche più di un'istruzione per colpo di clock
- In tal modo si compensano eventuali ritardi dovuti a situazioni di miss nella cache.

Rel. 28.05.2019
© Prinetto - 2019

Vincoli di dipendenza

- 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 tutti gli operandi utilizzati in un'istruzione siano già stati prodotti dalle istruzioni precedenti.

Rel. 28.05.2019
© Prinetto - 2019

Esempio

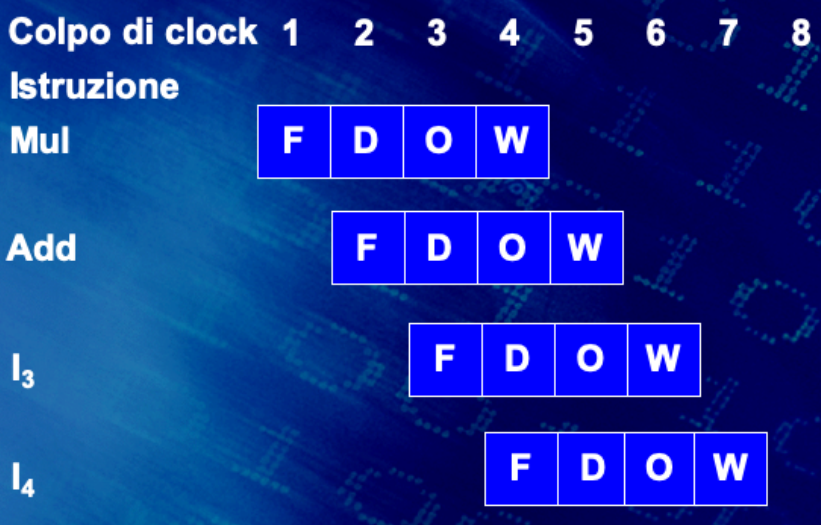
```
Mul   R2, R3, R4        ; R4 <= R2 * R3
Add   R5, R4, R6        ; R6 <= R5 + R4
```


Rel. 28.05.2019

© Prinetto - 2019

Mul R2, R3, R4
Add R5, R4, R6

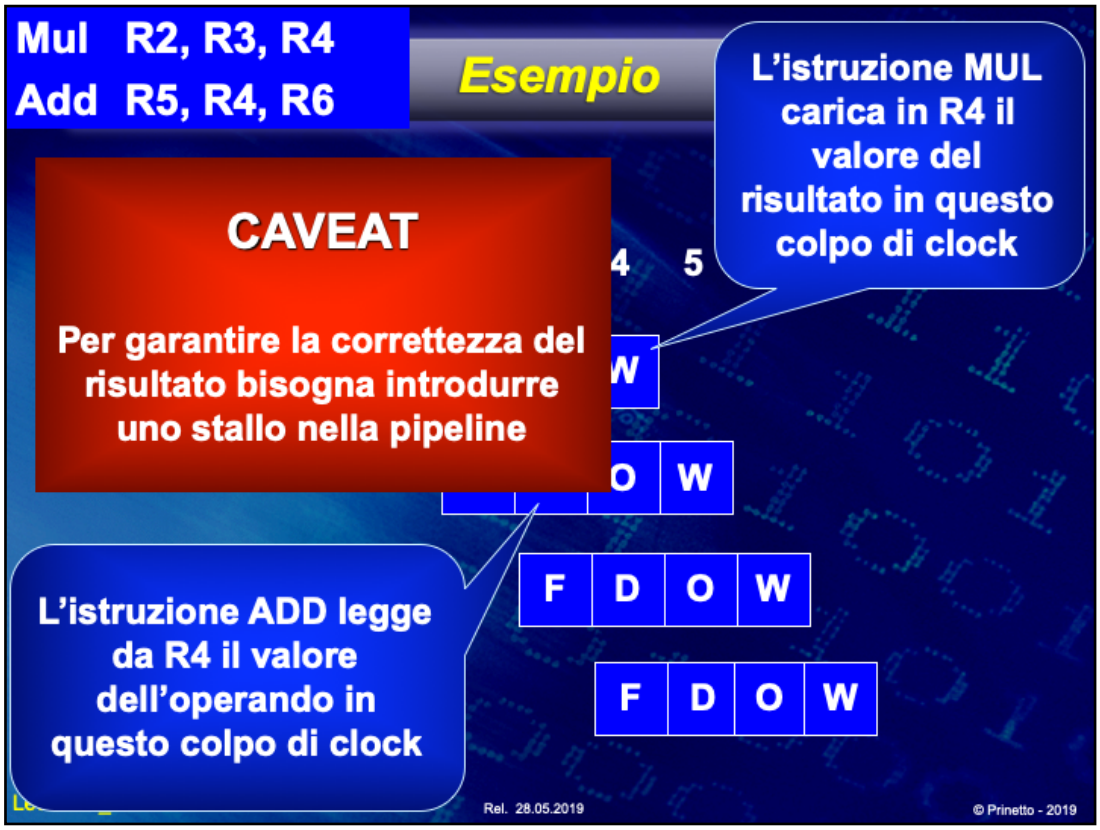
Esempio

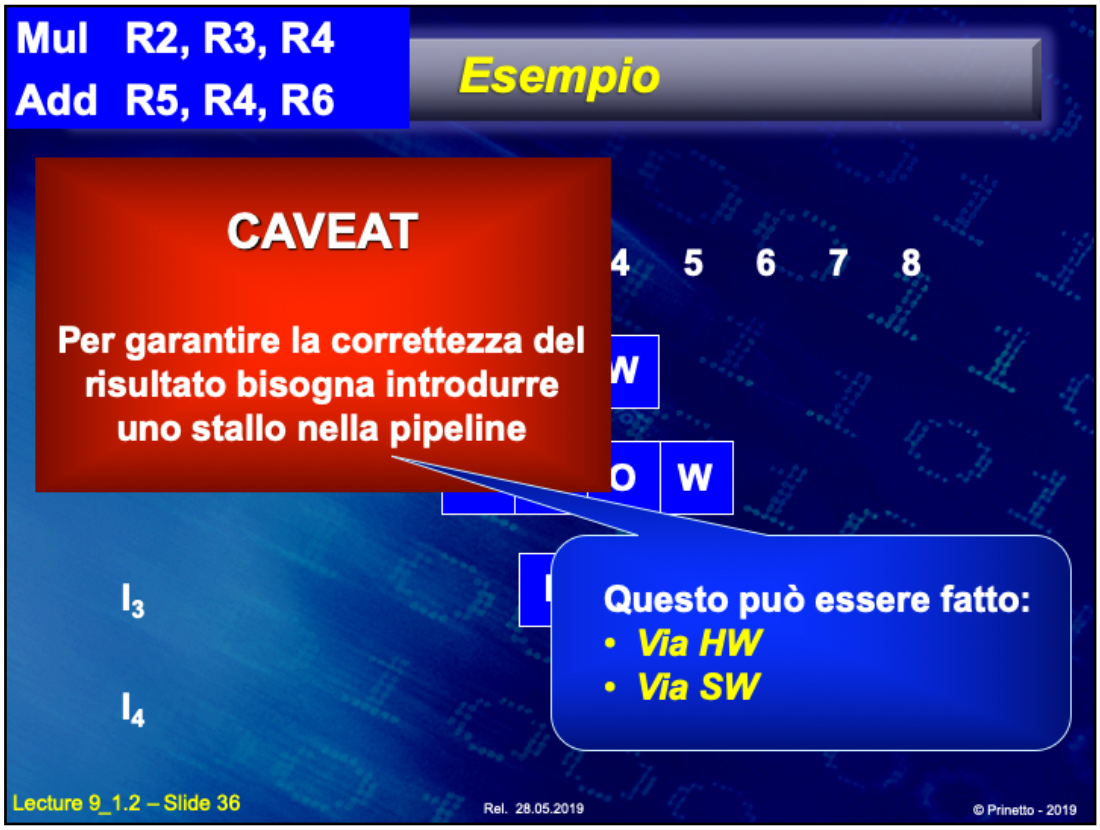


Rel. 28.05.2019
© Prinetto - 2019

Mul R2, R3, R4
Add R5, R4, R6
Esempio
L'istruzione MUL carica in R4 il valore del risultato in questo colpo di clock
Colpo di clock 1 2 3 4 5
Istruzione
Mul
F D O W
Add
F D O W
I3
F D O W
I4
F D O W
Lecture 9_1.2 – Slide 33
Rel. 28.05.2019
© Prinetto - 2019

Mul R2, R3, R4
Add R5, R4, R6
Esempio
L'istruzione MUL carica in R4 il valore del risultato in questo colpo di clock
Colpo di clock 1 2 3 4 5
Istruzione
Mul
F D O W
Add
F D O W
F D O W
F D O W
L'istruzione ADD legge da R4 il valore dell'operando in questo colpo di clock
Rel. 28.05.2019
© Prinetto - 2019


Mul R2, R3, R4
Add R5, R4, R6
Esempio
L'istruzione MUL carica in R4 il valore del risultato in questo colpo di clock
CAVEAT
Per garantire la correttezza del risultato bisogna introdurre uno stallo nella pipeline
4 5
W
O W
F D O W
F D O W
L'istruzione ADD legge da R4 il valore dell'operando in questo colpo di clock
Rel. 28.05.2019
© Prinetto - 2019


Mul R2, R3, R4
Add R5, R4, R6
Esempio
CAVEAT
Per garantire la correttezza del risultato bisogna introdurre uno stallo nella pipeline
4 5 6 7 8
W
O W
I3
I4
Questo può essere fatto:
• Via HW
• Via SW
Lecture 9_1.2 – Slide 36
Rel. 28.05.2019
© Prinetto - 2019

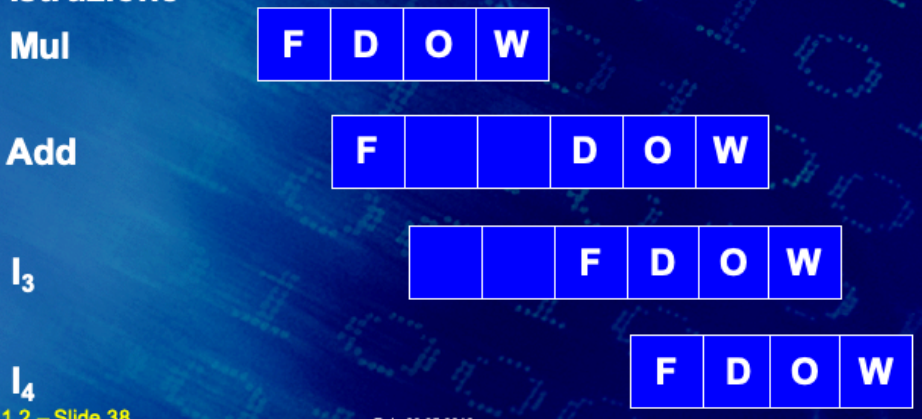
Soluzione HW

- L'HW della pipeline è in grado di:
  - Riconoscere le situazioni di dipendenza dei dati
  - Inibire il funzionamento di specifici moduli della pipeline per garantire la correttezza del risultato.

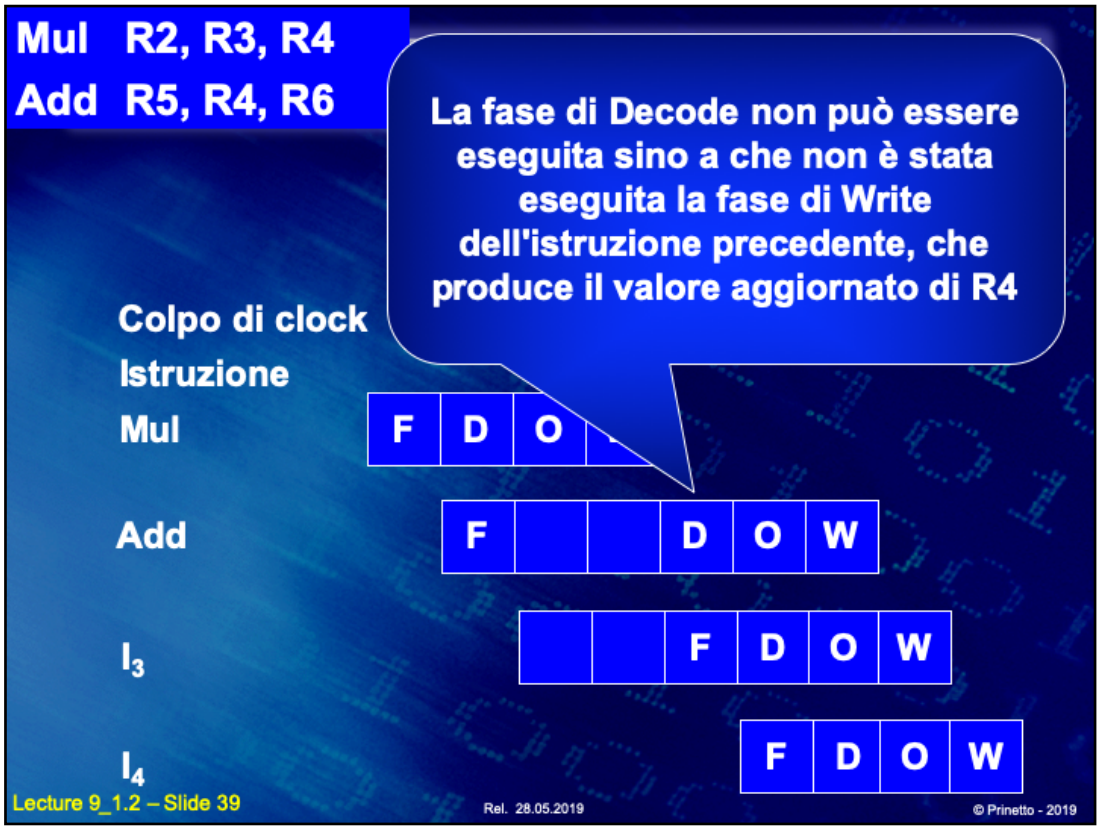
Rel. 28.05.2019
© Prinetto - 2019

Esempio

Mul R2, R3, R4
Add R5, R4, R6



Rel. 28.05.2019
© Prinetto - 2019


Mul R2, R3, R4
Add R5, R4, R6
La fase di Decode non può essere eseguita sino a che non è stata eseguita la fase di Write dell'istruzione precedente, che produce il valore aggiornato di R4
Colpo di clock
Istruzione
Mul
Add
I_3
I_4
F D O
F D O W
F D O W
F D O W
Lecture 9_1.2 – Slide 39
Rel. 28.05.2019
© Prinetto - 2019

Soluzione SW

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

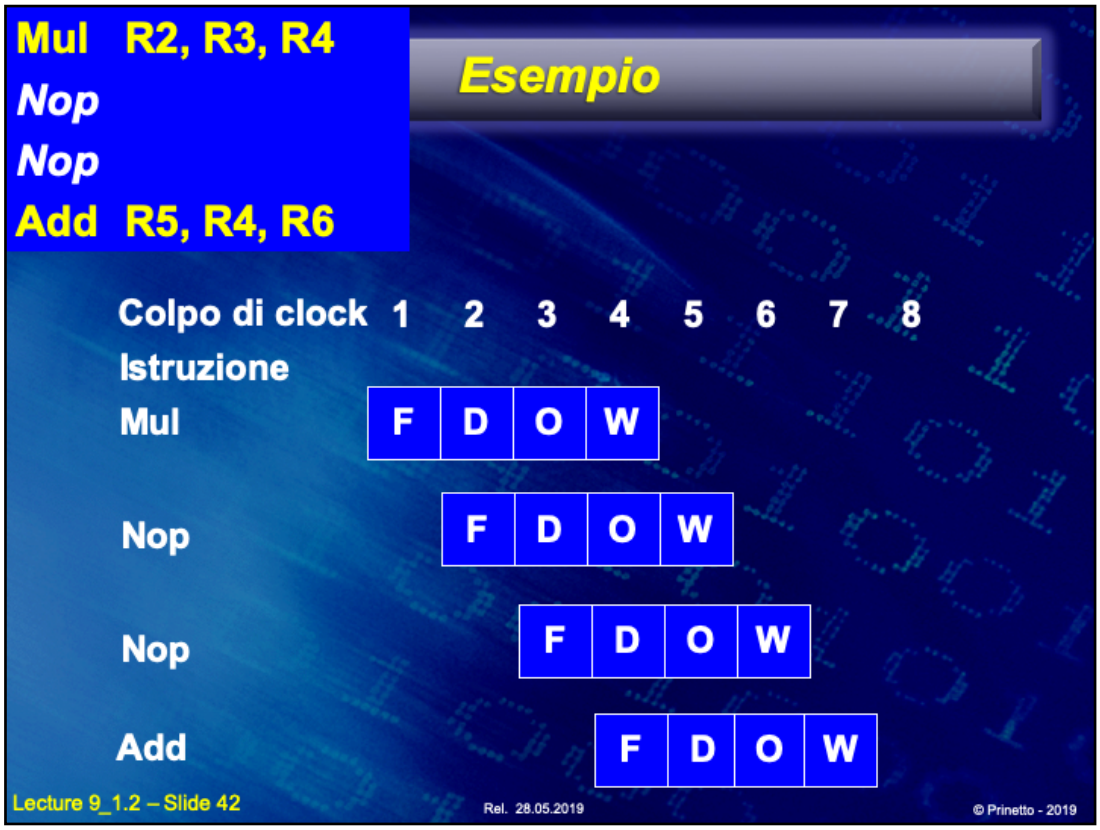
Rel. 28.05.2019
© Prinetto - 2019

Esempio

Codice originale	Codice modificato
I_i	I_i
Mul R2, R3, R4	Mul R2, R3, R4
Add R5, R4, R6	*Nop*
I_j	*Nop*
I_k	Add R5, R4, R6
	I_j
	I_k

Rel. 28.05.2019

© Prinetto - 2019


Mul R2, R3, R4
Nop
Nop
Add R5, R4, R6
Esempio
Colpo di clock 1 2 3 4 5 6 7 8
Istruzione
Mul
F D O W
Nop
F D O W
Nop
F D O W
Add
F D O W
Lecture 9_1.2 – Slide 42
Rel. 28.05.2019
© Prinetto - 2019

Delayed Load

- In alcuni casi il compilatore individua delle istruzioni, precedenti a quelle tra cui esiste la dipendenza, che vengono spostate in avanti al posto delle Nop, aumentando così l'efficienza.

Rel. 28.05.2019
© Prinetto - 2019

Esempio

Codice originale

I_i
Mul R2, R3, R4
Add R5, R4, R6
I_j
I_k

Codice modificato

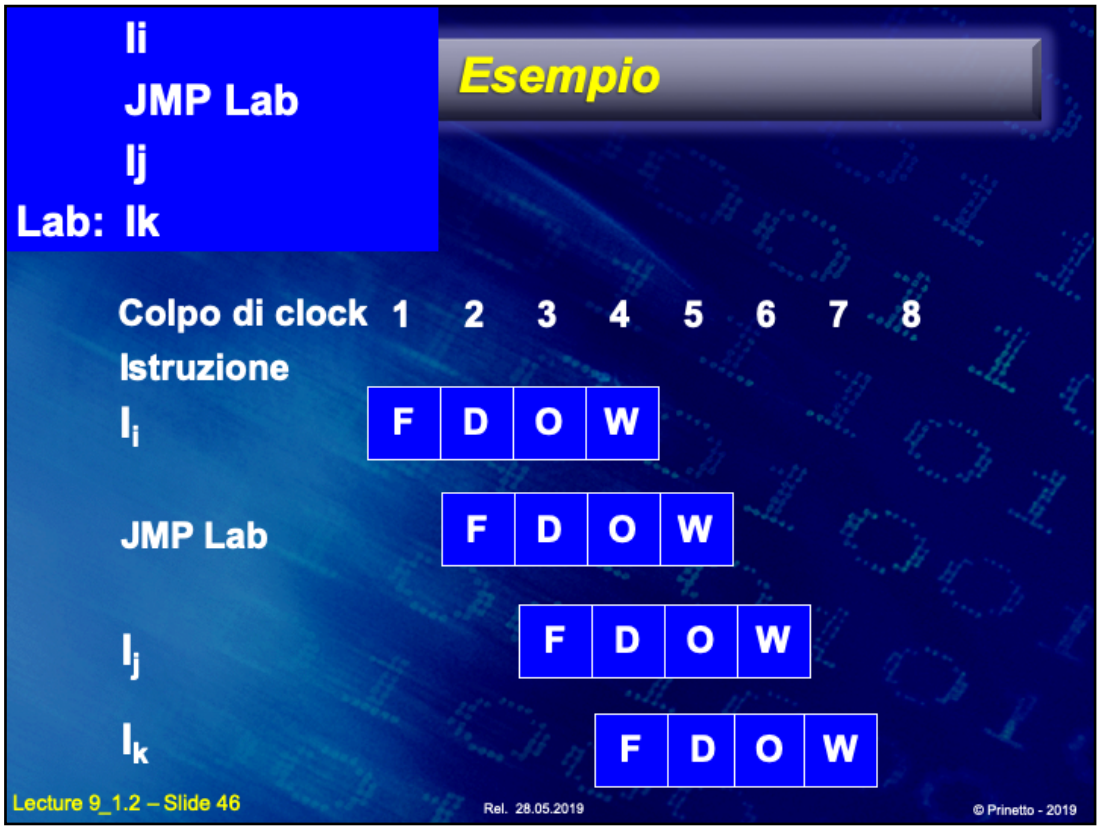
Mul R2, R3, R4
I_i
Nop
Add R5, R4, R6
I_j
I_k

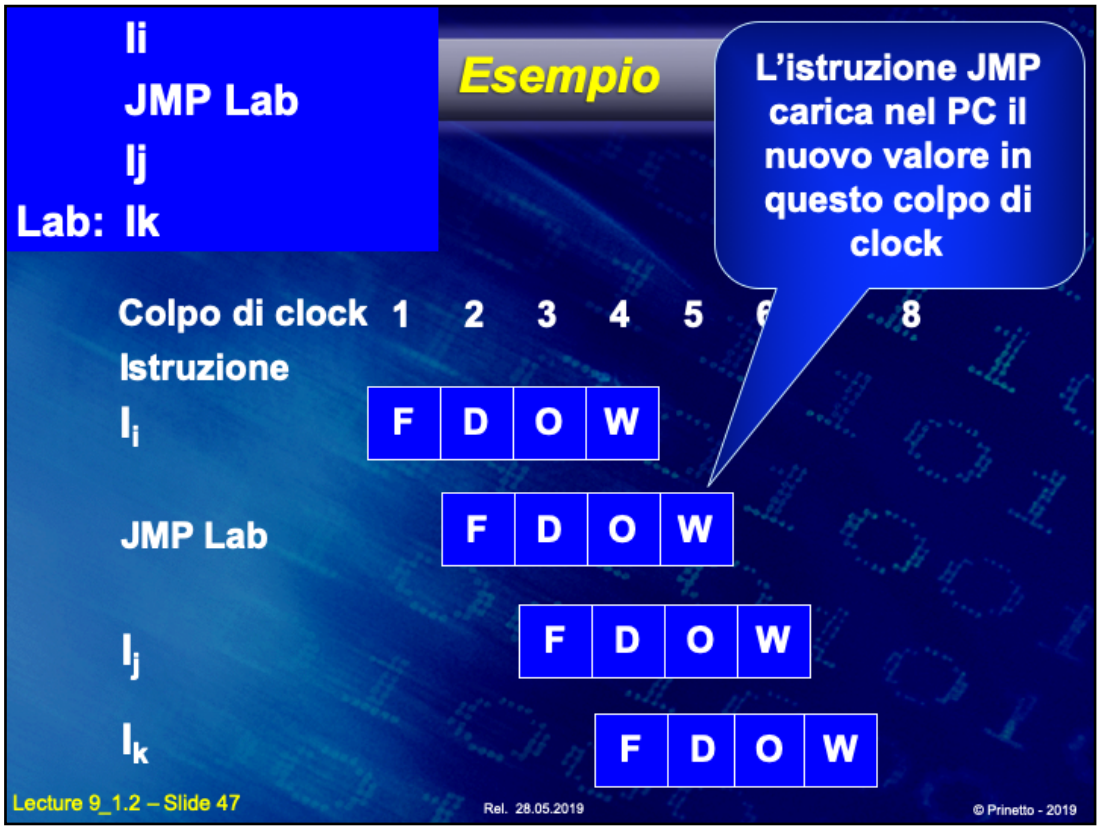
Rel. 28.05.2019
© Prinetto - 2019

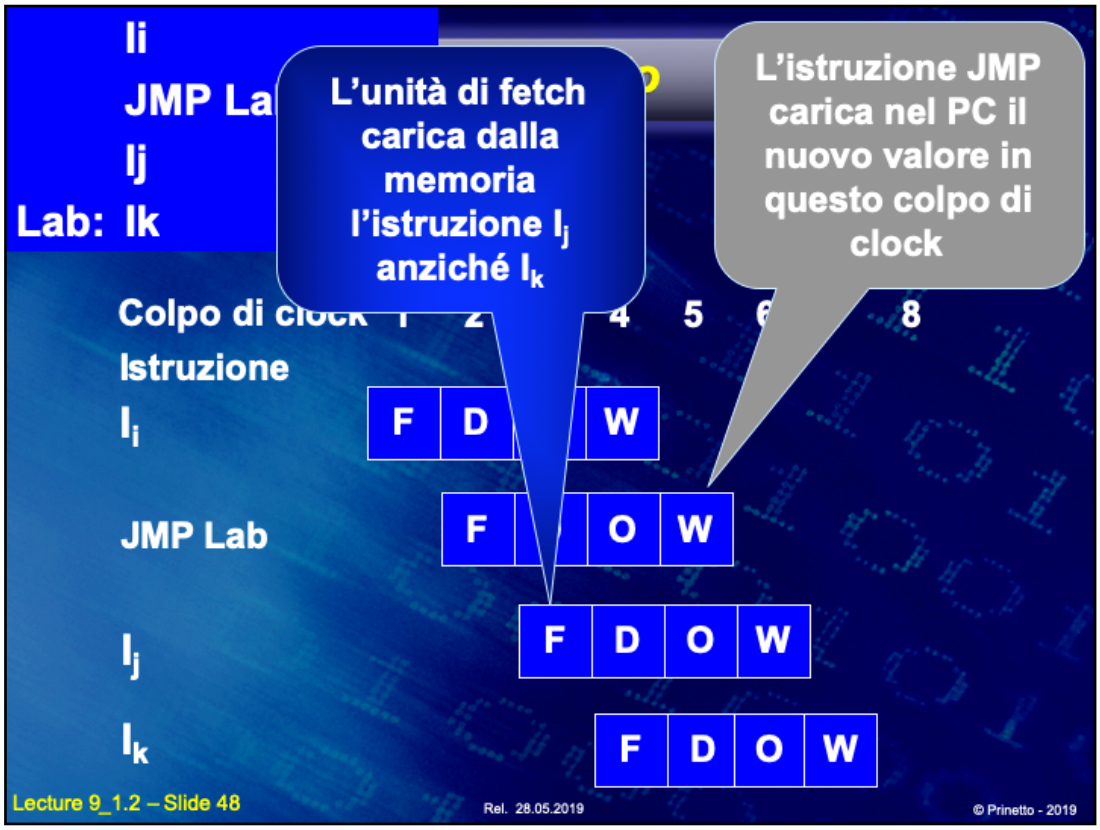
Problemi posti dalle istruzioni di salto

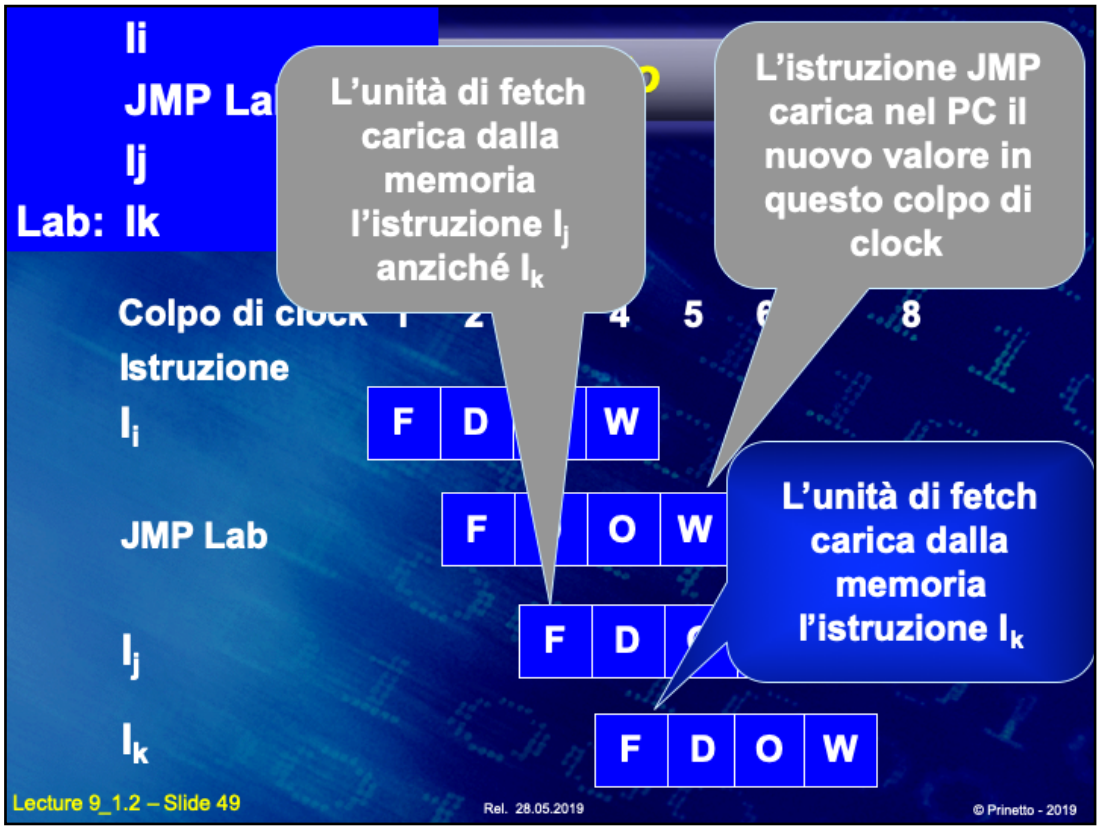
- Le istruzioni di salto sono potenzialmente molto dannose per la pipeline, in quanto interrompono il normale flusso di esecuzione sequenziale.

Rel. 28.05.2019
© Prinetto - 2019


Esempio
Ii
JMP Lab
Ij
Lab: Ik
Colpo di clock 1 2 3 4 5 6 7 8
Istruzione
I_i
F D O W
JMP Lab
F D O W
I_j
F D O W
I_k
F D O W
Lecture 9_1.2 – Slide 46
Rel. 28.05.2019
© Prinetto - 2019


Ii
JMP Lab
Ij
Lab: Ik
Esempio
L'istruzione JMP carica nel PC il nuovo valore in questo colpo di clock
Colpo di clock 1 2 3 4 5 6 8
Istruzione
I_i
F D O W
JMP Lab
F D O W
I_j
F D O W
I_k
F D O W
Lecture 9_1.2 – Slide 47
Rel. 28.05.2019
© Prinetto - 2019


Ii
JMP La
Ij
Lab: Ik
L'unità di fetch carica dalla memoria l'istruzione I_j anziché I_k
L'istruzione JMP carica nel PC il nuovo valore in questo colpo di clock
Colpo di clock 1 2 4 5 8
Istruzione
I_i
F D W
JMP Lab
F O W
I_j
F D O W
I_k
F D O W
Lecture 9_1.2 – Slide 48
Rel. 28.05.2019
© Prinetto - 2019


li
JMP La
lj
Lab: lk
L'unità di fetch carica dalla memoria l'istruzione l_j anziché l_k
L'istruzione JMP carica nel PC il nuovo valore in questo colpo di clock
Colpo di clock 1 2 4 5 8
Istruzione
l_i
F D W
JMP Lab
F O W
l_j
F D
l_k
F D O W
L'unità di fetch carica dalla memoria l'istruzione l_k
Lecture 9_1.2 – Slide 49
Rel. 28.05.2019
© Prinetto - 2019

Soluzioni

- Per attenuare gli effetti negativi che derivano alla pipeline dai salti, si possono anche qui utilizzare
  - *tecniche SW*
  - *tecniche HW*

Rel. 28.05.2019
© Prinetto - 2019

Tecniche SW

- Sono basate su:
  - introduzione di istruzioni NOP da parte del compilatore
  - riorganizzazione del codice.

Rel. 28.05.2019
© Prinetto - 2019

Salto ritardato

- Consiste in una riorganizzazione del codice, mirante a garantire che nell'intervallo di ritardo del salto vi siano
  - delle istruzioni NOP, oppure
  - delle istruzioni, originariamente poste prima di quella di salto, che vanno comunque eseguite.

Rel. 28.05.2019
© Prinetto - 2019

Salto ritardato *(cont'd)*

- Qualora non sia possibile trovare un numero sufficiente di istruzioni (tra quelle antecedenti) con cui riempire l'intervallo di ritardo di salto, si inseriscono delle NOP
- Dati sperimentali indicano che nell'85% dei casi il compilatore è in grado di evitare l'uso delle NOP.

Rel. 28.05.2019
© Prinetto - 2019

Tecniche HW

- Possono:
  - Limitarsi a rilevare le istruzioni di salto, e a svuotare conseguentemente la pipeline dalle istruzioni erroneamente caricate
  - Implementare tecniche più sofisticate, che dipendono dal fatto che il salto sia incondizionato o condizionato (*Branch Prediction Unit*).

Rel. 28.05.2019
© Prinetto - 2019

Salti Incondizionati

- L'unità di fetch è in grado di riconoscere le istruzioni di salto incondizionato e di procedere a una decodifica dell'istruzione, al fine di caricare nella coda delle istruzioni direttamente l'istruzione destinazione del salto.
- Questa tecnica è nota come ***branch folding***.

Rel. 28.05.2019
© Prinetto - 2019

Salti condizionati

- In questo caso è necessario attendere l'esecuzione dell'istruzione precedente a quella 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 eseguito, vi saranno alcune istruzioni provenienti dalla coda che non dovranno essere eseguite: queste costituiscono l'intervallo di ritardo del salto (*branch delay slot*).

Rel. 28.05.2019
© Prinetto - 2019

Predizione di salto

- Le tecniche HW permettono di ridurre la penalizzazione introdotta da un'istruzione di salto.
- Consistono nel dotare l'unità di fetch di una capacità di previsione sul fatto che il salto venga effettuato o meno.

Rel. 28.05.2019
© Prinetto - 2019

Predizione di salto

- 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.

Rel. 28.05.2019
© Prinetto - 2019

Esecuzione speculativa

- Quando si segue il suggerimento fornito dalla previsione, che è suscettibile di errore, bisogna che il processore sia in grado, allorché ci si accorge dell'errore, di annullare gli effetti di un certo numero di istruzioni all'indietro.

Rel. 28.05.2019
© Prinetto - 2019

Esecuzione speculativa (cont'd)

- In pratica, da quando entra in una fase di esecuzione speculativa, il processore non modifica più né memoria né registri, ma memorizza i nuovi valori in appositi registri temporanei
- Se poi si conferma la correttezza di quanto fatto, allora avverrà l'aggiornamento di memoria e registri
- L'incorrere di eccezioni può creare grossi problemi all'esecuzione speculativa.

Rel. 28.05.2019
© Prinetto - 2019

Tipi di Previsione

- Sono possibili 2 tipi di previsione:
  - Statica
  - Dinamica

Rel. 28.05.2019
© Prinetto - 2019

Predizione statica

- Il compilatore codifica nelle istruzioni di salto stesse la relativa predizione, elaborata sulla base di analisi del codice.

Rel. 28.05.2019
© Prinetto - 2019

Predizione dinamica

- A ogni istruzione di salto condizionato sono associati uno o due flag, che indicano la previsione.
- I flag vengono aggiornati dal processore ogni volta che l'istruzione viene eseguita.

Rel. 28.05.2019
© Prinetto - 2019

Predizione dinamica *(cont'd)*

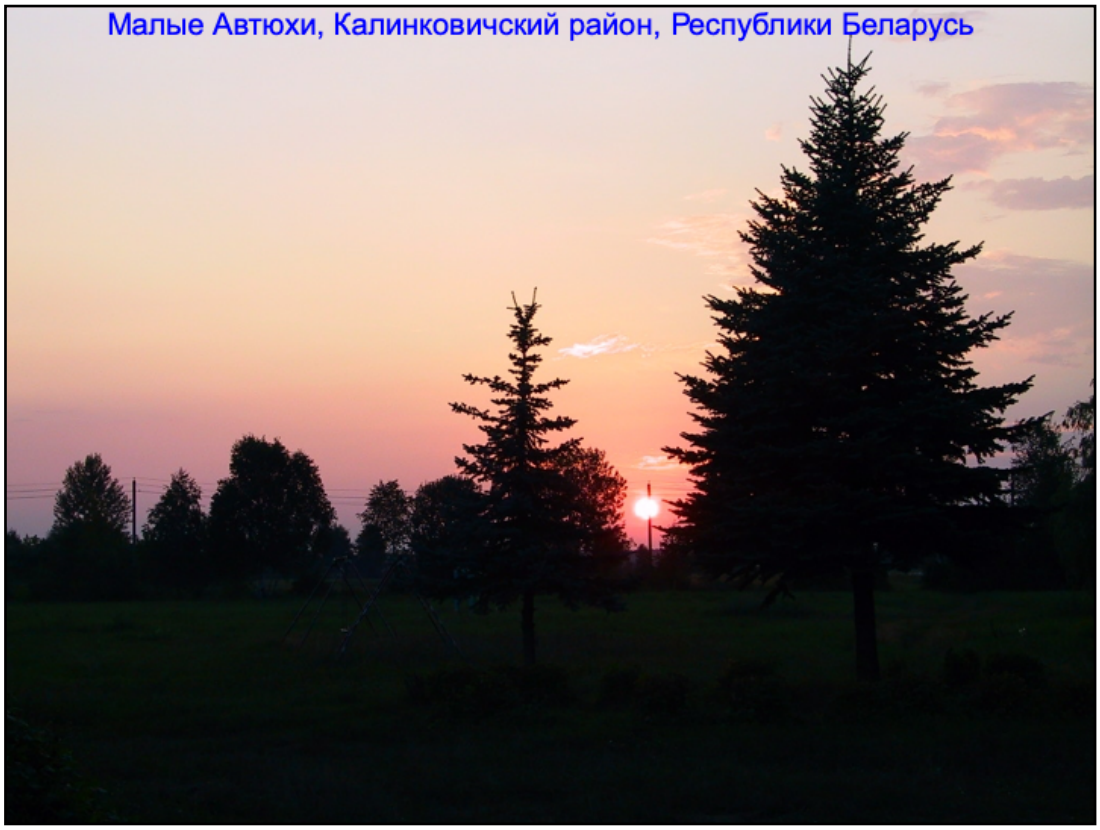
- Nel caso più semplice, il flag memorizza se, all'ultima esecuzione, il salto era stato eseguito o meno.
- Con due flag è possibile avere una predizione più accurata, basata su una storia precedente codificata in 4 possibili stati.

Rel. 28.05.2019
© Prinetto - 2019

RISC e superscalari

- I processori tradizionali richiedevano un certo numero di colpi di clock per l'esecuzione di ogni istruzione.
- Grazie all'introduzione delle pipeline, in ogni colpo di clock:
  - I processori ***RISC*** eseguono normalmente una istruzione
  - I processori ***superscalari*** eseguono più di una istruzione.

Rel. 28.05.2019
© Prinetto - 2019


Малые Автюхи, Калинковичский район, Республики Беларусь