Calcolatori Elettronici B
a.a. 2004/2005

Pipeline e prestazioni:
Esercizi

Massimiliano Giacomin

Esercizio – confronto prestazioni pipeline vs. multiciclo

- Si consideri la seguente combinazione di istruzioni eseguite
  - 22% load, 11% store, 49% formato-R, 16% salti condizionati, 2% salti incondizionati
- Si consideri la pipeline MIPS a 5 stadi con i seguenti tempi di esecuzione:
  - Prelievo, esecuzione con ALU, accesso in memoria = 4 ns
  - Decodifica/lettura registri, scrittura registri = 2 ns

Determinare il tempo medio di esecuzione per istruzione per un processore con controllo multiciclo e per un processore con pipeline, considerando per la pipeline il **caso ideale**: situazione ***a regime*** e ***nessuna criticità.*** Confrontare le prestazioni delle sue soluzioni.

Soluzione

T_{clock} = 4 ns
(corrisponde al tempo di esecuzione dell'unità funzionale più lenta)

Tempo medio di CPU nel caso di controllo multi-ciclo

- Numero di cicli per ciascuna classe di istruzioni
  Load = 5
  Store = 4
  Formato-R = 4
  Salti cond. = 3
  Salti incond. = 3

- CPI = 0,22×5 + 0,11×4 + 0,49×4 + 0,16×3 + 0,02×3 = 4,04

- Tempo medio per istruzione = CPI × Ciclo di clock

- Tempo medio per istruzione = 4,04 × 4 ns = 16,16 ns

Pipeline ideale e confronto prestazioni

- Per una pipeline ideale si considera CPI = 1
- Un ciclo di clock è pari a 4 ns
- Il tempo medio di esecuzione è pari a 1×4 ns
- Confrontiamo ora i tempi medi nei due schemi considerati

$$\text{Speedup} = \frac{\text{Tempo}_{\text{multiciclo}}}{\text{Tempo}_{\text{pipeline}}} = \frac{16{,}16}{4} = 4{,}04$$

Le prestazioni migliorano di 4,04 volte nel caso di controllo pipeline

Influenza delle criticità sulle prestazioni

- Tipi di criticità:
  - **criticità strutturale**: ad esempio, viene usata un'unica memoria per i dati e per le istruzioni; oppure caso di fallimento in cache
  - **criticità sui dati**: quando ci sono dipendenze di dati fra istruzioni consecutive
  - **criticità sul controllo**: dovuta ai salti

- Vediamo come questi tipi di criticità influenzino le prestazioni di un processore con pipeline

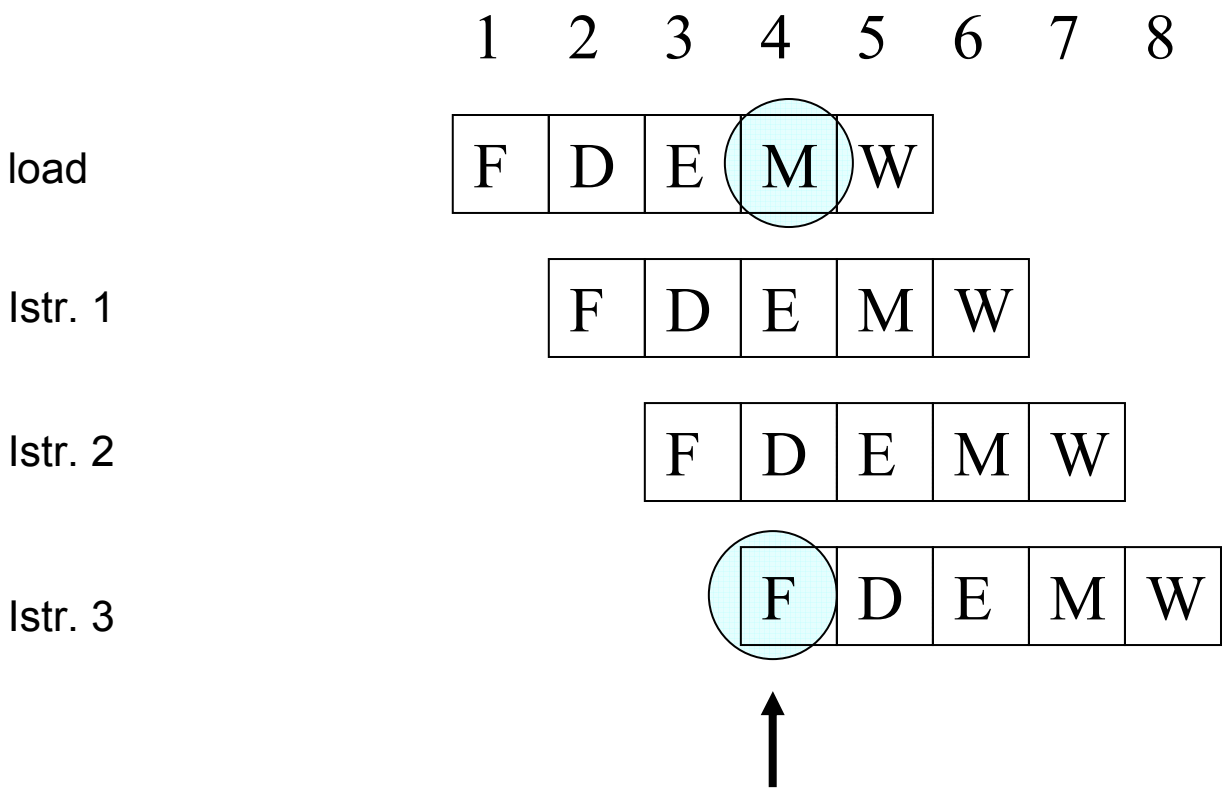
- Vedremo vari esercizi in cui le precedenti criticità (nell'ordine) influenzano le prestazioni

Esercizio – esempio di criticità strutturale

- Si consideri un processore dotato di pipeline (5 stadi) che disponga di una sola memoria cache per i dati e per le istruzioni:

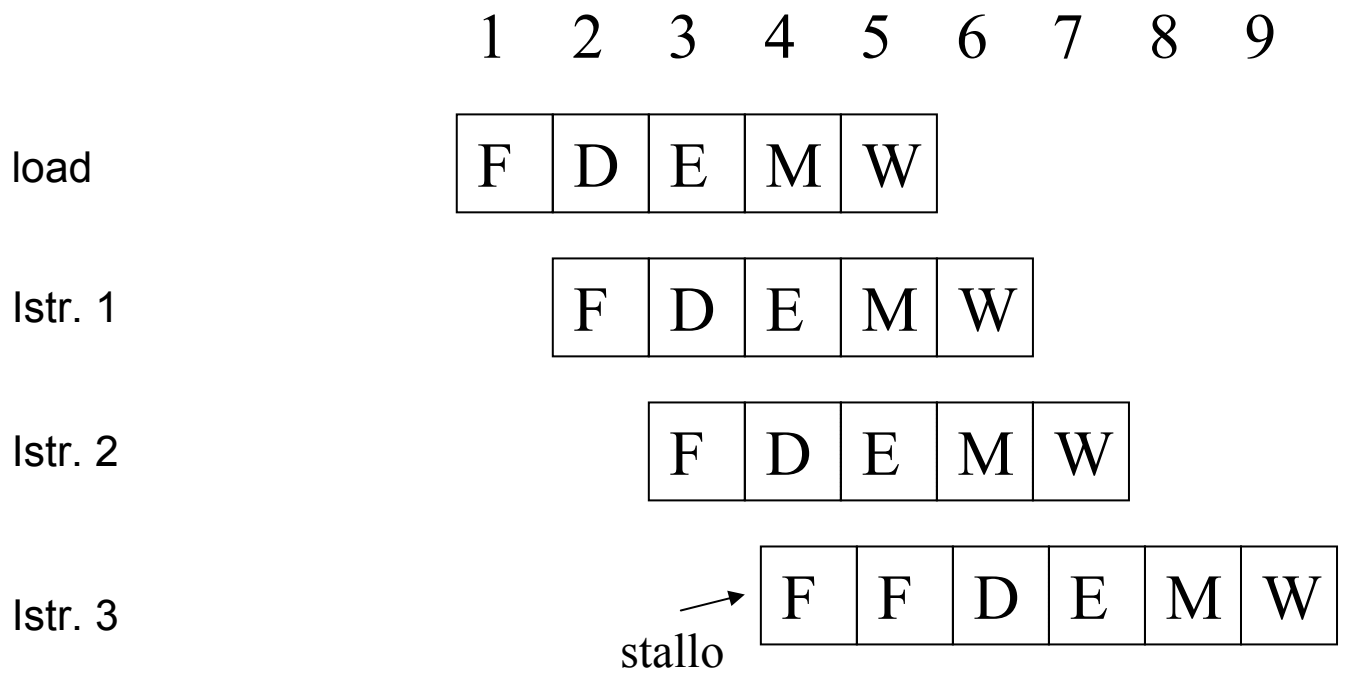
1) Si proponga una soluzione al problema basata sullo stallo della pipeline, descrivendo mediante diagramma temporale ciò che avviene in caso di conflitto sulla memoria.

2) Si supponga che il 40% delle istruzioni eseguite facciano riferimento ai dati e che non esistano altri tipi di stalli: calcolare il CPI medio ed il throughput delle istruzioni.



- Ogni volta che un'istruzione richiede un accesso ai dati, c'è un conflitto con un'istruzione successiva che prevede, nello stadio di fetch, un accesso alla memoria per il prelevamento dell'istruzione.
- Per risolvere il conflitto, la pipeline deve essere messa in stallo per 1 ciclo di clock.

Una possibile soluzione: unità di criticità che pone in stallo la pipeline per un ciclo di clock



L'istruzione 3 viene completata alla fine del ciclo 9, nessuna istruzione viene completata nel ciclo 8

- Ricordando la formula per il CPI nel caso di stalli:

CPI = CPI ideale + Cicli di stallo per istruzione

Nel nostro caso: una sola memoria per dati e per istruzioni;
il 40% delle istruzioni eseguite fanno riferimento ai dati
e quindi subiscono una penalità ciascuna di un ciclo di clock

$\Rightarrow$ Supponendo che non esistano altri tipi di stalli si ha

$$\text{CPI} = 1 + \underbrace{0{,}40 \times 1}_{\text{Cicli di stallo}} = 1{,}4$$

Throughput = 1/CPI = 1/1.4 = 0.71 istruzioni/ciclo

Esercizio – esempio di criticità strutturale (miss di cache)

Si consideri un sistema dotato solamente di **cache primaria**
(senza cache secondaria) distinta per i dati e le istruzioni.
Per la cache si suppone:

- una penalità di fallimento di 10 cicli di clock
- una percentuale di successo del 95% per le istruzioni e del 90% per i dati

Si suppone inoltre che il 40% delle istruzioni faccia accesso a dati in memoria.
Si supponga che il carico sia tale che il CPI medio nel caso multiciclo risulti
(come nel precedente esercizio) 4,04

Si calcolino il CPI effettivo ed il throughput per un processore multiciclo
e per un processore con pipeline (in tal caso trascurando altre cause di stallo).
Si confrontino le prestazioni delle due soluzioni progettuali.

Consideriamo il processore con pipeline

- Se avviene un fallimento di accesso alla cache si ha un ritardo dell'istruzione in cui si è verificato il fallimento: si ha un incremento di CPI pari alla penalità di fallimento di accesso alla cache (10 cicli)
- Dobbiamo considerare l'incremento medio di CPI dovuto ai fallimenti di accesso alla cache, chiamiamo δ_{miss} questo incremento
- Supponiamo 95% percentuale di successo per le istruzioni e 90% percentuale di successo per i dati
- Supponiamo che il 40% delle istruzioni faccia accesso a dati in memoria

$$\delta_{miss} = \underbrace{0{,}05 \times 10}_{\substack{\text{Cicli di stallo} \\ \text{per le istruzioni}}} + \underbrace{0{,}4 \times 0{,}1 \times 10}_{\substack{\text{Cicli di stallo} \\ \text{per i dati}}} = 0{,}9 \text{ cicli}$$

$\Rightarrow$ CPI effettivo (in assenza di altre cause di stallo):

$$CPI = 1 + \delta_{miss} = 1 + 0{,}9 = 1{,}9$$

$\Rightarrow$ $Throughput_{pipeline} = 1/CPI = 1/1{,}9 = 0{,}53$ istruzioni/ciclo

Consideriamo il processore multiciclo:

In assenza di miss di cache avrei CPI = 4,04

Ma per le istruzioni che danno miss (fetch è accesso ai dati) ho una penalità pari a quella del caso con pipeline:

$$\delta_{miss} = 0{,}05 \times 10 \;+\; 0{,}4 \times 0{,}1 \times 10 \;= 0{,}9 \text{ cicli}$$

$$\Rightarrow CPI_{multiciclo} = 4{,}04 + 0{,}9 = 4{,}94$$

$$\Rightarrow Throughput_{multiciclo} = 1/CPI = 1/4{,}94 = 0{,}20 \text{ istruzioni/ciclo}$$

Quindi l'incremento delle prestazioni nel caso con pipeline è

0,53/0,20 = 2,65 [Posso confrontare i throughput o, ugualmente, CPI – o, ugualmente, i tempi di esecuzione: il risultato non cambia!]

Esercizio – esempio di criticità strutturale (miss di cache)

Un altro caso: Supponiamo di avere sia **cache primaria** che **cache secondaria**

- Per la cache primaria, come prima:
  - 95% percentuale di successo per istruzioni, 90% per i dati
  - Si supponga inoltre che per trasferire blocchi dalla cache secondaria alla cache primaria occorrano 4 cicli.
  - Penalità di fallimento: 10 cicli.
- Per la cache secondaria:
  - 94% la percentuale di successo per le istruzioni, 92% per i dati.
- Come prima, supponiamo che il 40% delle istruzioni faccia accesso ai dati in memoria

Calcolare CPI e Throughput per un processore con pipeline (trascurando altri stalli)

Soluzione

$$\delta_{miss} = \overbrace{0{,}05 \times 0{,}94 \times 4 + 0{,}05 \times 0{,}06 \times 10}^{\text{Cicli di stallo per le istruzioni}} + \overbrace{0{,}4 \times (0{,}1 \times 0{,}92 \times 4 + 0{,}1 \times 0{,}08 \times 10)}^{\text{Cicli di stallo per i dati}} =$$

$$= 0{,}4 \text{ cicli}$$

Da cui, **CPI** = 1 + 0,4 = 1,4 **Throughput** = 1/1,4 = 0,71 istruzioni/ciclo

Nota all'esercizio precedente:

un fallimento alla cache secondaria (che segue un fallimento alla cache primaria) ha una penalità di fallimento pari a 10 cicli, ovvero quella della cache primaria: è il tempo necessario per portare il blocco mancante nella memoria cache primaria! [la cache secondaria viene caricata indipendentemente]

Per esercizio:

svolgere il calcolo per un processore multiciclo e confrontare le prestazioni in questo secondo caso (cache primaria + cache secondaria)

Esercizio – criticità sui dati

- Si supponga di avere il seguente codice assembler

```
add $s0, $t0, $t1
sub $t2, $s0, $t3
```

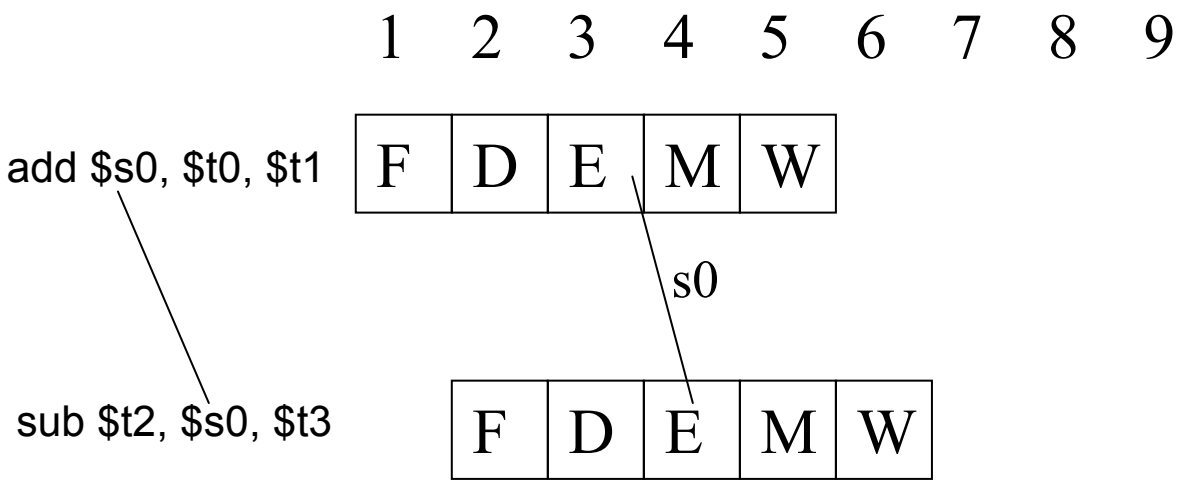
Si consideri la pipeline a 5 stadi del MIPS:

- si individuino le eventuali dipendenze tra i dati
- supponendo di non disporre dell'unità di propagazione, tracciare il diagramma temporale e individuare il numero di stalli necessari
- supponendo di disporre dell'unità di propagazione verso lo stadio E, tracciare il diagramma temporale

Senza propagazione:

- L'operazione sub, per essere eseguita, deve aspettare che sia disponibile il nuovo valore di $s0.
- Con una pipeline a 5 stadi (prelievo, decodifica, esecuzione, accesso in memoria, scrittura risultato) la add scrive il risultato nel quinto stadio e quindi si creano ***tre bolle*** nella pipeline (= stallo di 3 cicli di clock).

Con propagazione:



Nessuno stallo!

Esercizio – criticità sui dati

- Si consideri il codice assembler:

```
lw      $s0, 10($t1)
sub     $t2, $s0, $t3
```

  Supponendo di disporre dell'unità di propagazione verso lo stadio E, tracciare il diagramma temporale ed individuare il numero di stalli necessari.

- Supponendo che il 30% delle istruzioni eseguite siano delle load e che, nel 50% dei casi, a un'istruzione di load segua un'istruzione che dipende dal risultato della load:

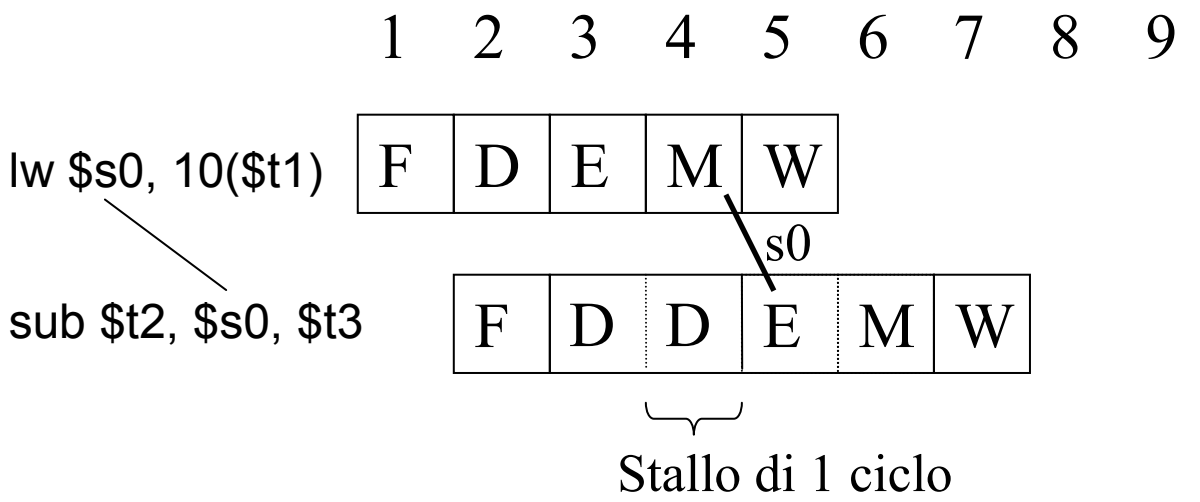
  calcolare il CPI effettivo

  [senza considerare altri tipi di stallo]

Diagramma temporale:

Codice assembler:

```
lw      $s0, 10($t1)
sub     $t2, $s0, $t3
```



⇒ Ogni volta che una load è seguita da un'istruzione che ne utilizza il dato, ho un ciclo di clock addizionale

- Ricordiamo che:

  1) Nel caso ideale il CPI di un processore con pipeline è pari a 1

  2) In presenza di stalli:

CPI = CPI ideale + Cicli di stallo per istruzione (1)

Quindi, supponendo che il 30% delle istruzioni eseguite siano delle load e che nel 50% dei casi, a un'istruzione di load, segua un'istruzione che dipende dal risultato della load

Ogni stallo creato in questo modo è di 1 ciclo di clock (come abbiamo visto usando la tecnica di propagazione)

Supponiamo che non esistano altri tipi di stalli

Da cui

$$\text{CPI} = 1 + \underbrace{0{,}30 \times 0{,}50 \times 1}_{\text{Cicli di stallo}} = 1{,}15$$

Esercizio – criticità sui dati

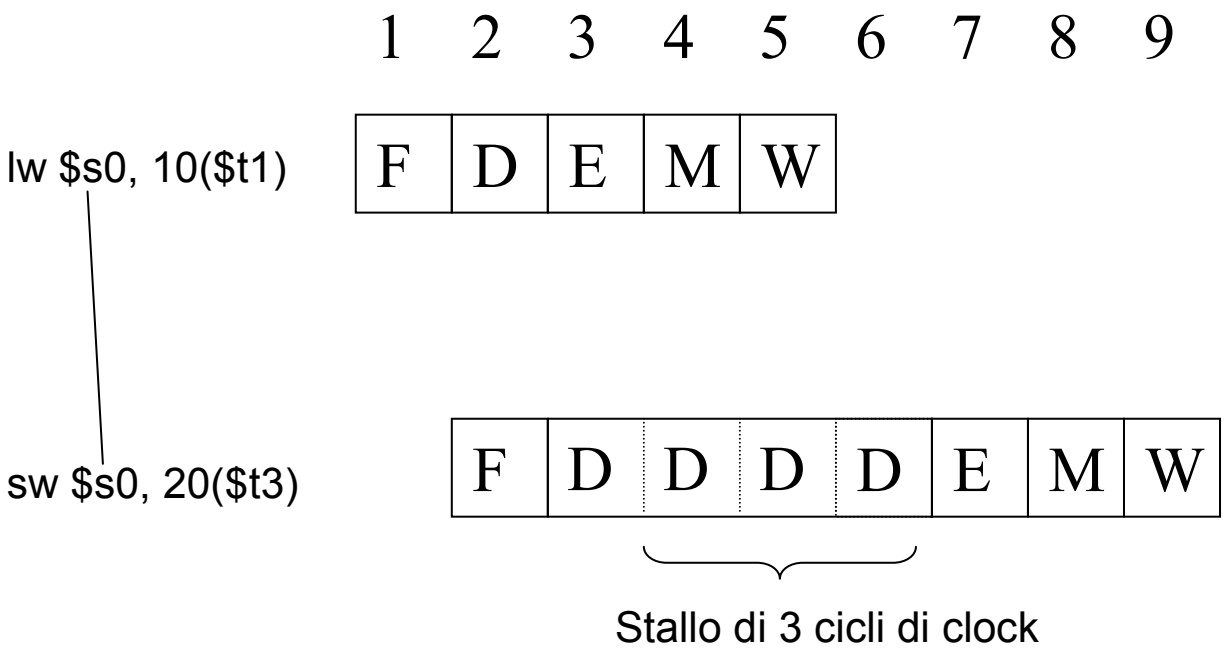
- Si consideri il codice assembler:

```
lw      $s0, 10($t1)
sw      $s0, 20($t3)
```

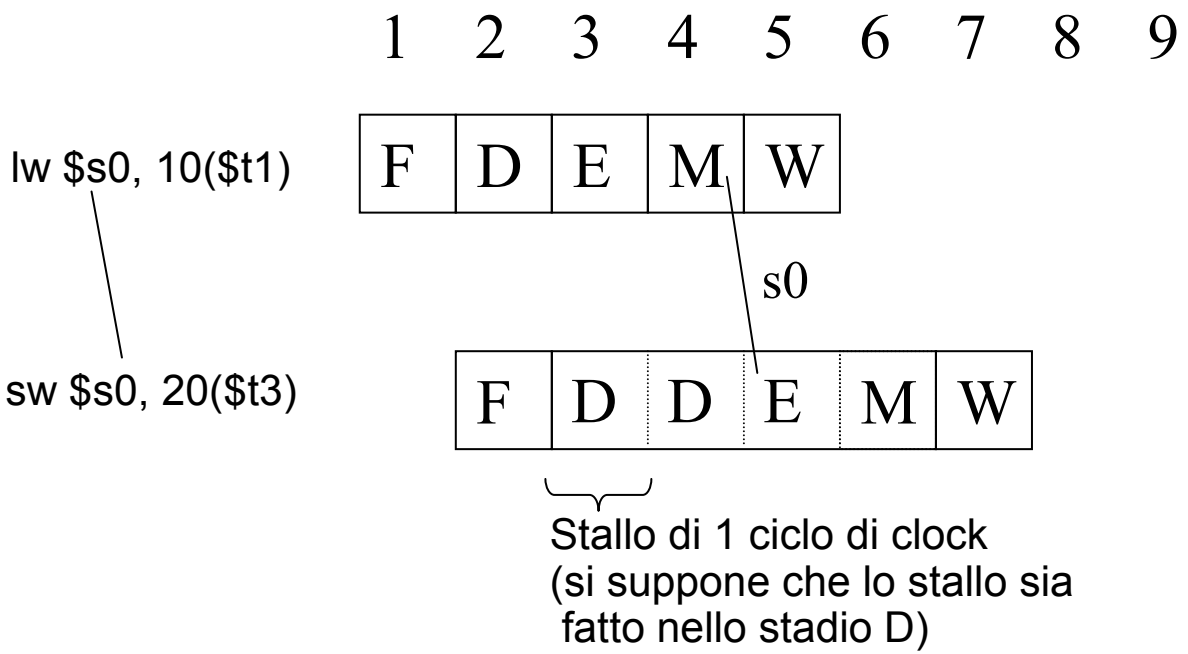
Tracciare il diagramma temporale ed individuare il numero di stalli necessari, per ciascuno dei seguenti tre casi:

- non è presente alcuna unità di propagazione
- è presente un unità di propagazione verso lo stadio E
- è presente l'unità di propagazione verso lo stadio E e lo stadio M

Senza propagazione:



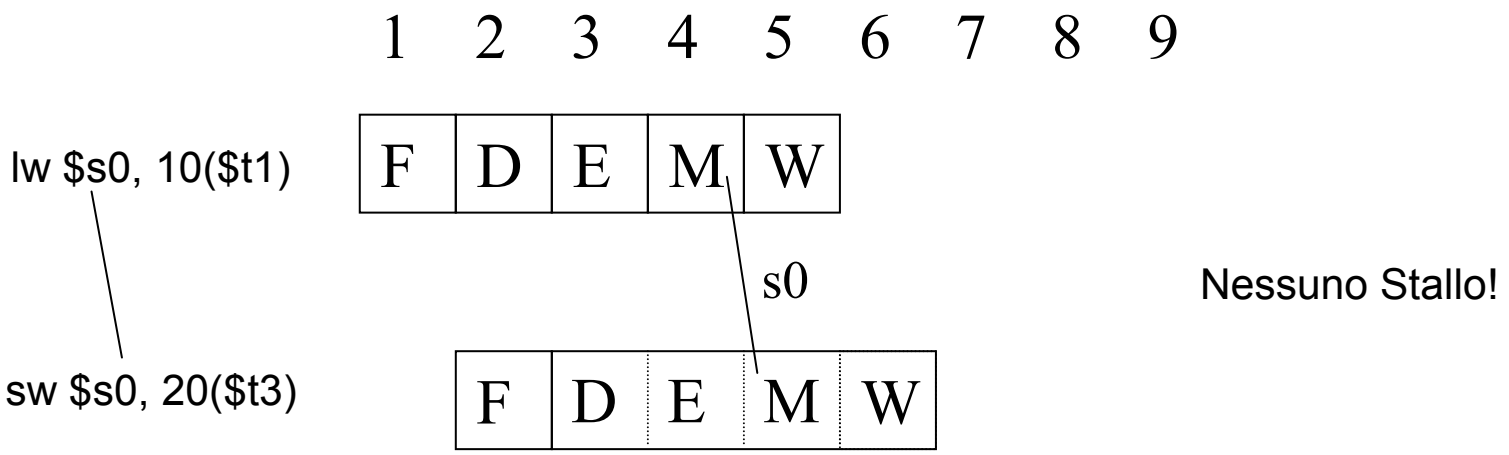
Propagazione verso E:



NB: s0 prodotto dalla lw nello stadio M [caricato dalla memoria] e propagato verso lo stadio E di sw: sw deve comunque "aspettare 1 ciclo"

NB2: Il dato letto viene propagato come ingresso di E che lo trasferisce in EX/MEM (sarà usato da sw nello stadio M)

Propagazione verso E e verso M:



[Nel ciclo di clock 4, lw accede alla memoria per prelevare s0.
Il dato s0 è quindi disponibile (nel registro interstadio M/W)
nel ciclo di clock 5, quando può essere propagato verso M di sw
che lo pone in memoria nel suo stadio M.]

NB: l'implementazione del MIPS richiede un'estensione per supportare la propagazione verso M [nella versione base supporta solo propagazione verso E]

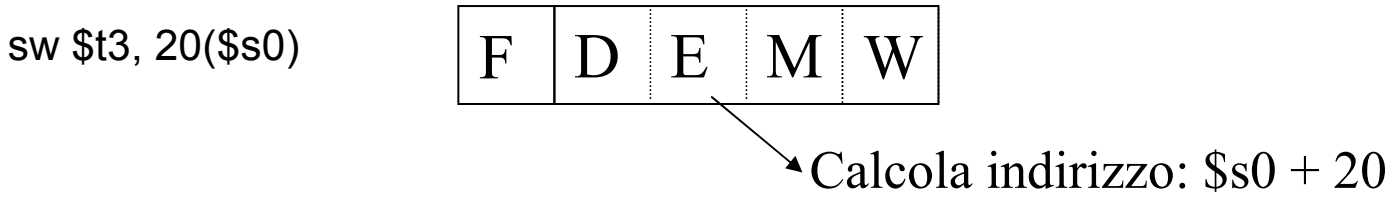
Esercizio – criticità sui dati: attenzione alla lieve differenza…

- Si consideri il codice assembler:

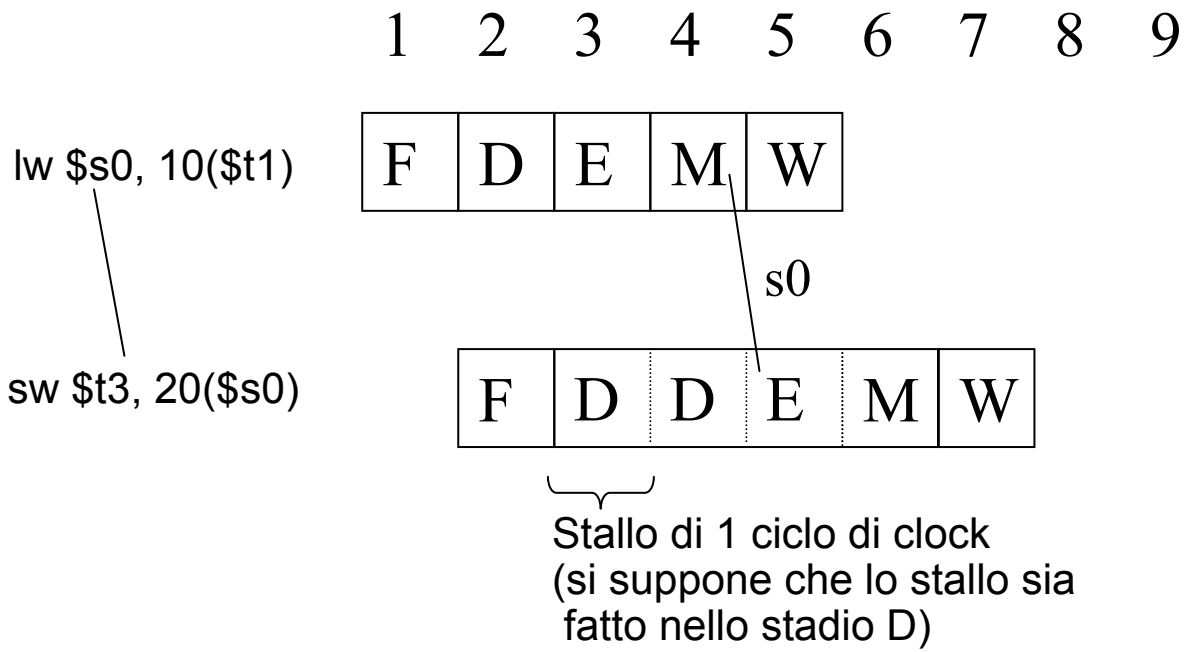
```
lw      $s0, 10($t1)
sw      $t3, 20($s0)
```

Tracciare il diagramma temporale ed individuare il numero di stalli necessari, supponendo di poter fare la propagazione verso qualunque stadio.

Attenzione: occorre sempre analizzare in che stadio serve il dato!!!



Quindi in questo caso s0 va propagato necessariamente verso E: 1 stallo



Proposta di esercizi:

ES. 1

Si consideri il seguente codice assembler MIPS:

```
lw   $s0, 10($s1)
add  $t0, $s0, $s1
add  $t1, $t0, $s0
```

Individuare le dipendenze e, supponendo di disporre solo dell'unità di propagazione verso E, disegnare il diagramma temporale.

ES. 2

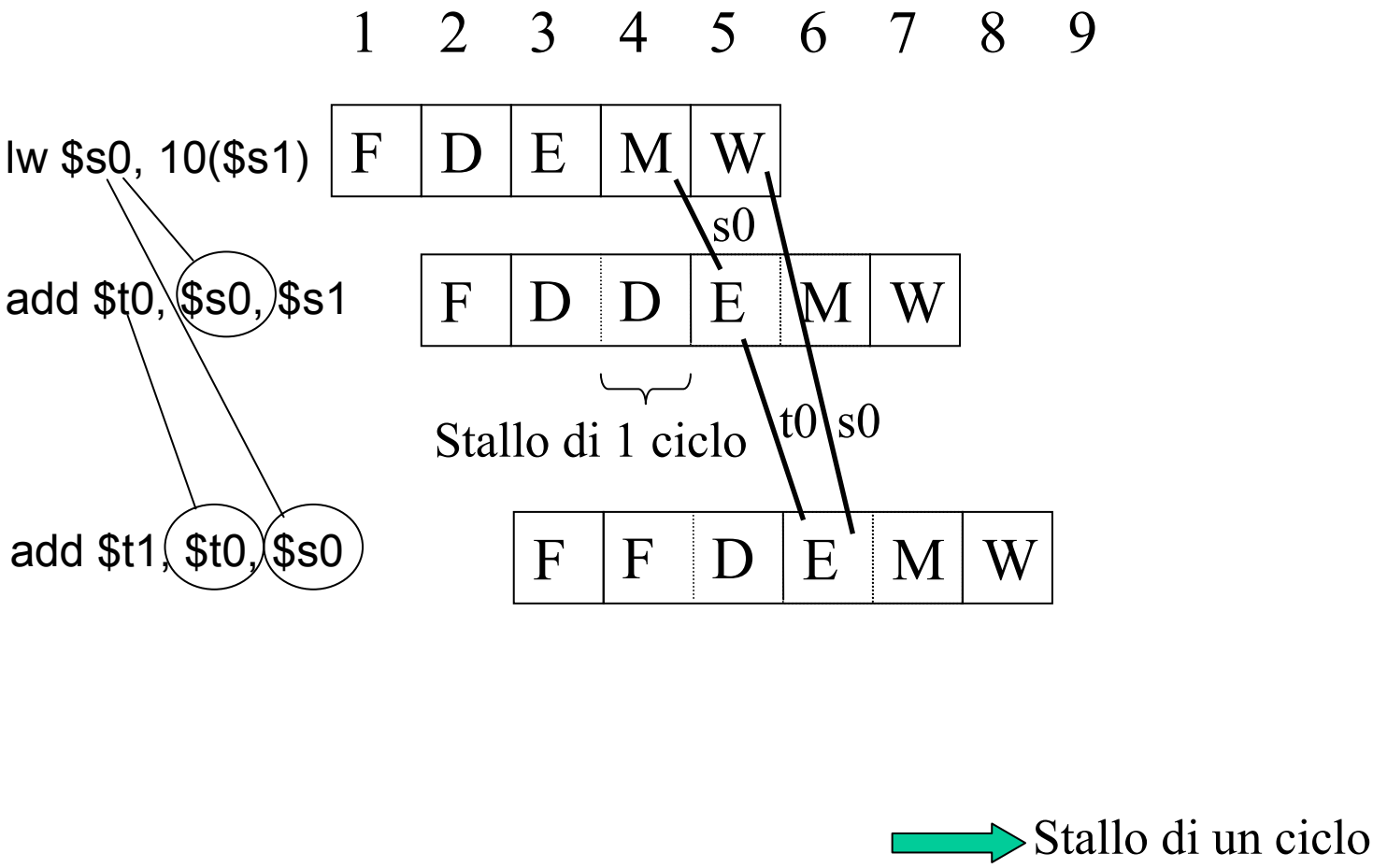
Si considerino i due seguenti codici assembler MIPS:

```
lw   $s0, 10($s1)
add  $s0, $t1, $t2
sub  $t3, $s0, $t4
```

```
lw   $s0, 10($s1)
add  $s1, $t1, $t2
sub  $t3, $s0, $t4
```

Per ciascuno di essi, individuare le dipendenze e, supponendo di poter propagare i dati verso qualunque stadio, disegnarne il diagramma temporale.

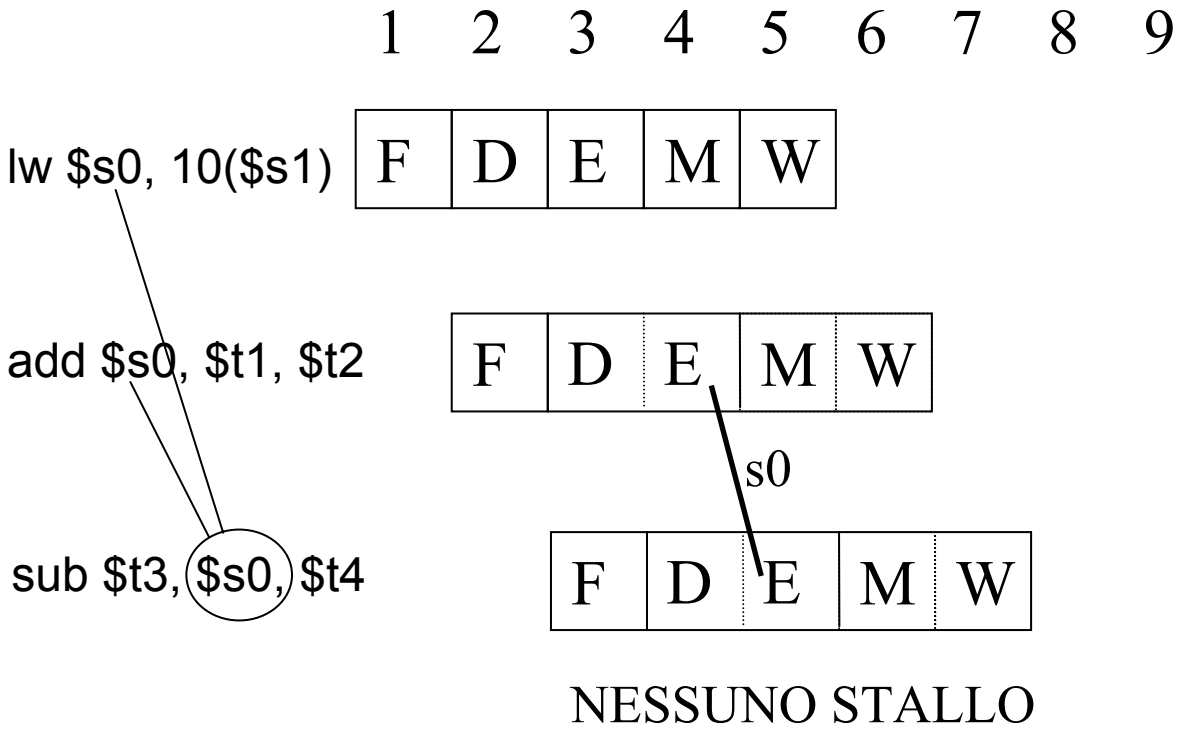
SOLUZIONE ESERCIZIO PROPOSTO N.1



Stallo di un ciclo

SOLUZIONE ESERCIZIO PROPOSTO N.2

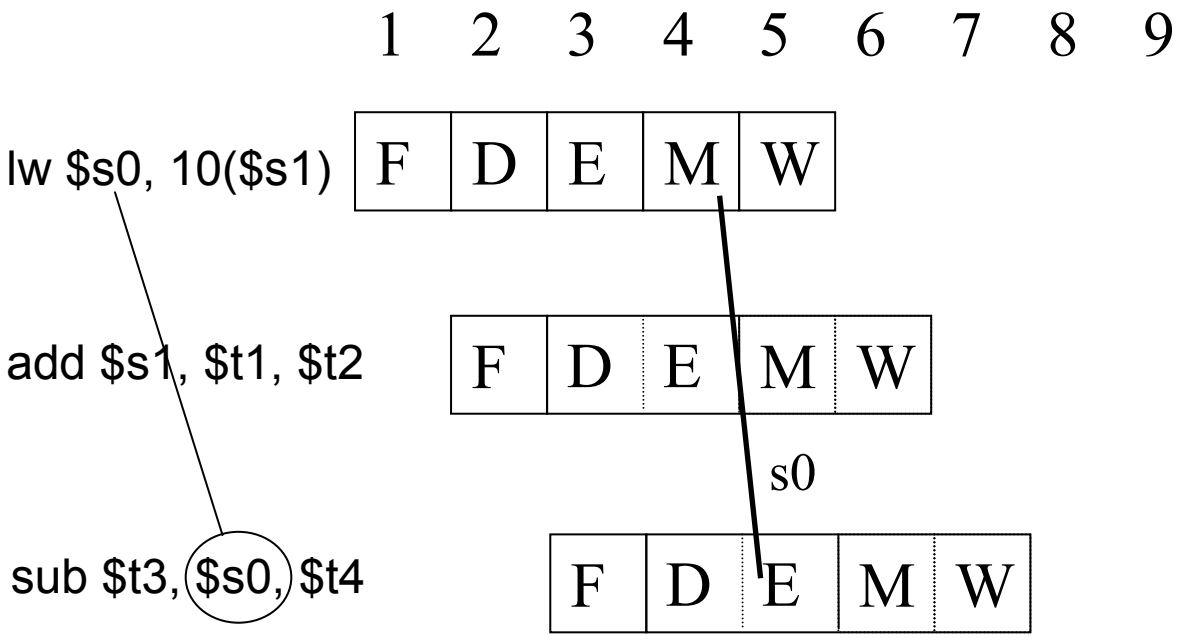
CODICE 1



NESSUNO STALLO

Importante: propago dall'istruzione add (la più recente!)

CODICE 2



NESSUNO STALLO

NB: in tal caso ovviamente propago dall'istruzione lw

Esercizio – Criticità sui dati e riordinamento delle operazioni

Codice di partenza (NB: è lo scambio M[$t1] ↔ M[$t1+4])

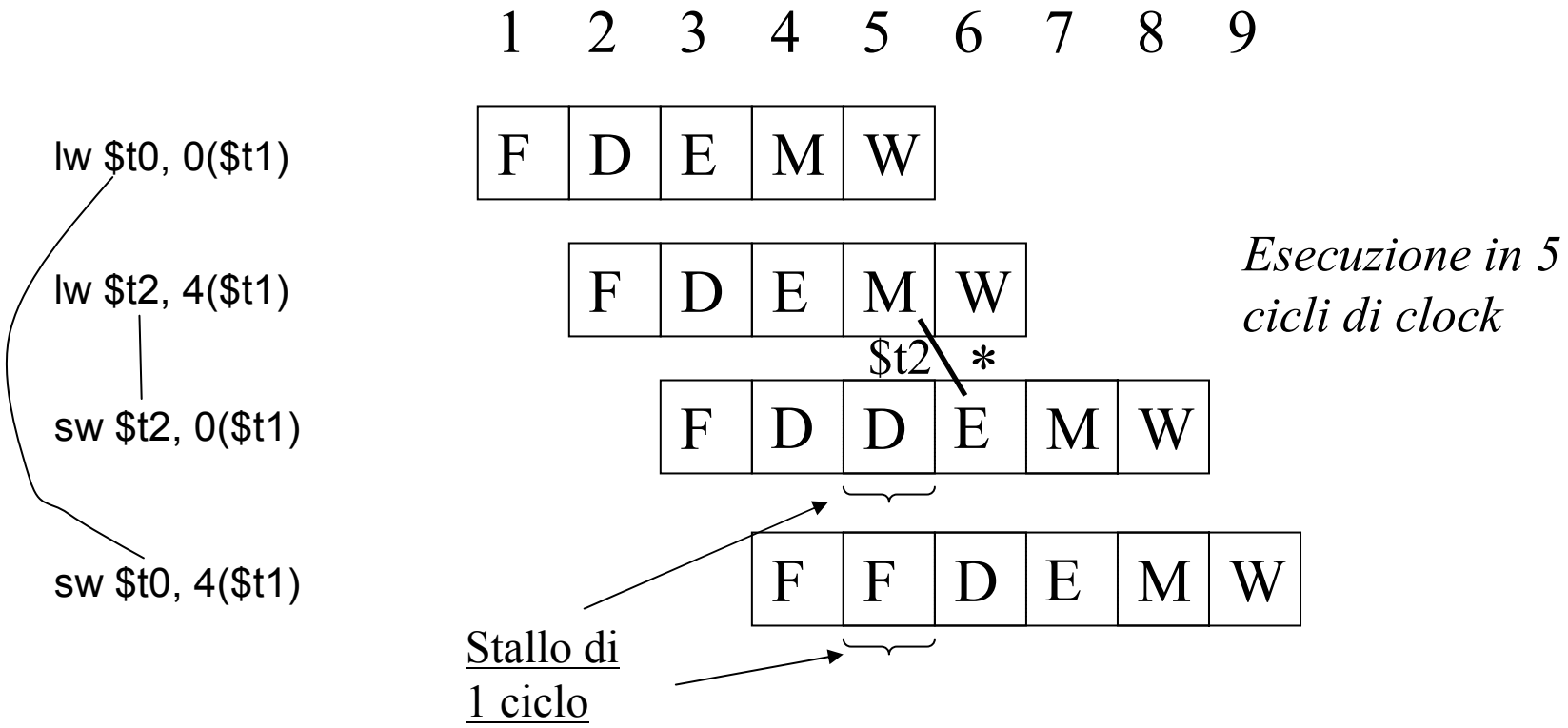
```
lw      $t0, 0($t1)
lw      $t2, 4($t1)
sw      $t2, 0($t1)
sw      $t0, 4($t1)
```

} Dipendenza dei dati

Trovare:

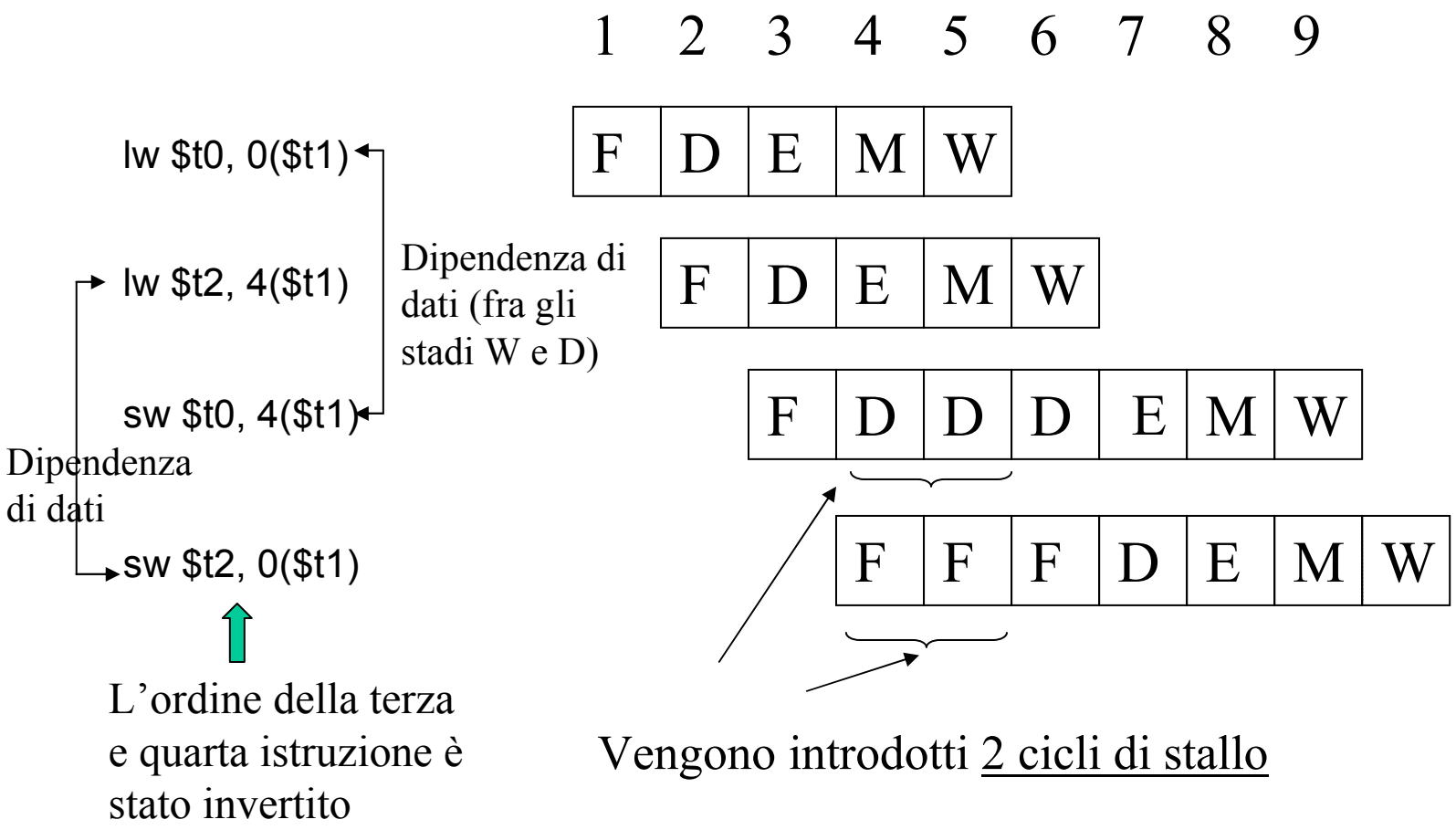
- diagramma temporale supponendo propagazione verso E
- supponendo di non disporre di unità di propagazione, riordinamento delle istruzioni per evitare il più possibile stalli della pipeline
- supponendo di disporre di unità di propagazione verso E, riordinamento delle istruzioni per evitare il più possibile stalli della pipeline

Propagazione verso E

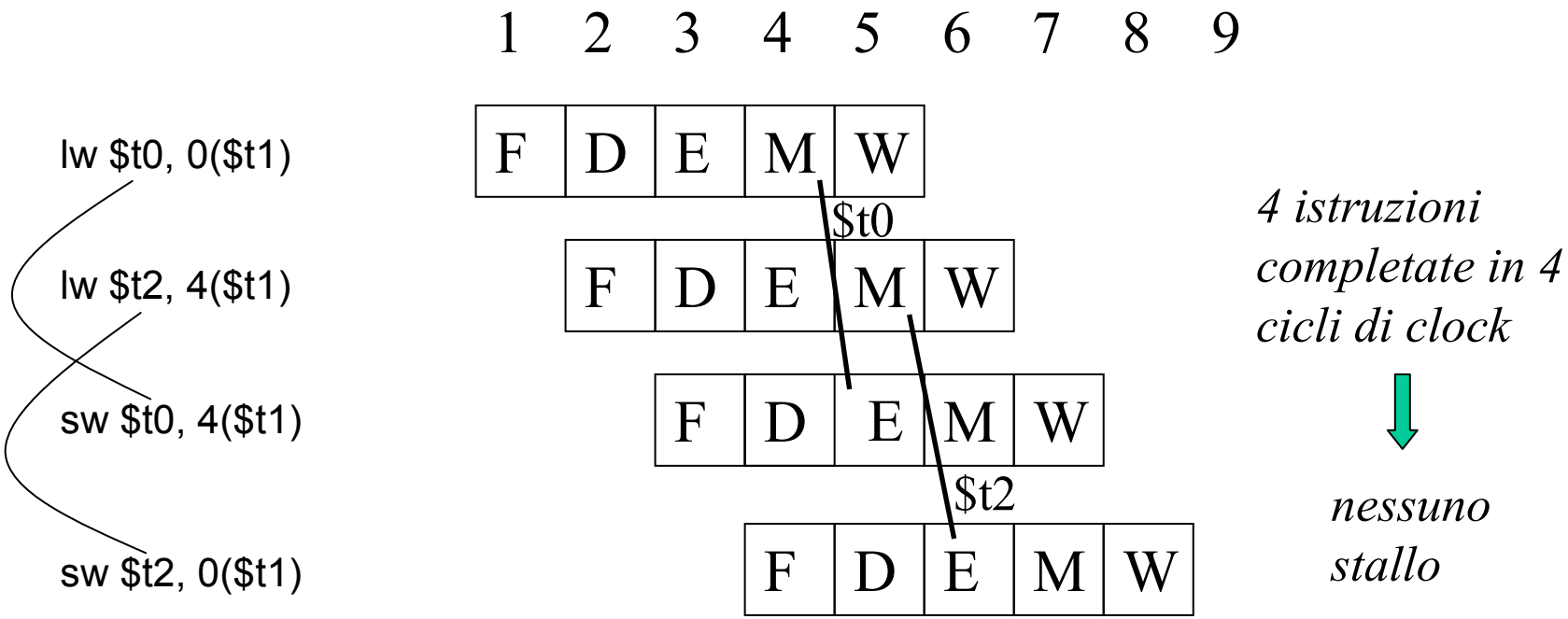


* Il dato letto viene propagato come ingresso di E che lo trasferisce in EX/MEM

Uso del riordino (senza propagazione)



Uso del riordino e della propagazione (verso E)

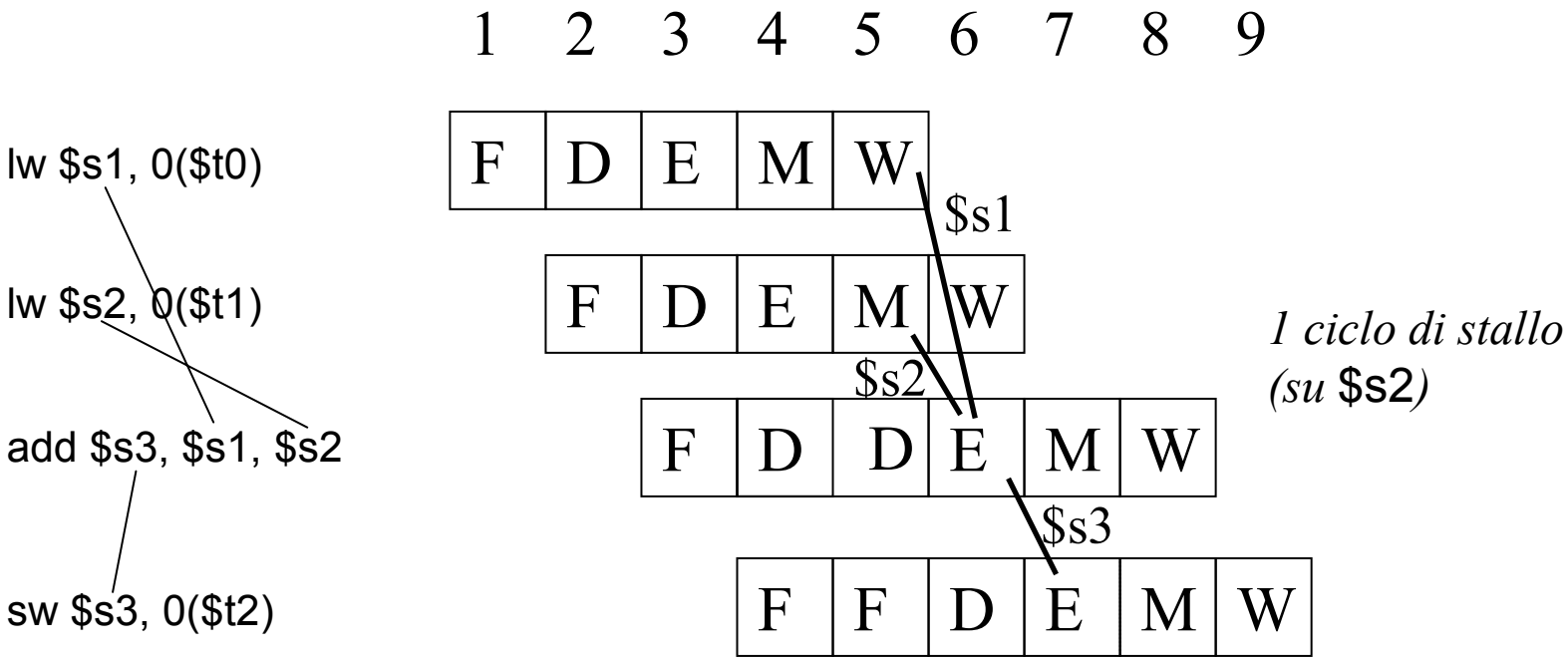


Esercizio/Esempio – Criticità sui dati e riordinamento delle operazioni

La tipica configurazione di codice che deriva da istruzioni del tipo

A = B + C

genera **uno stallo** sul valore del secondo dato



Ruolo del compilatore:

- Il compilatore può **modificare la sequenza** di codice per eliminare il conflitto
- Il compilatore cerca di evitare di fare eseguire il **caricamento** e immediatamente dopo **usare il registro destinazione** del caricamento

VEDIAMO COME PARTENDO DA UN ESEMPIO…

- Codice di partenza

```
a = b + c;
d = e – f;
```

$\Rightarrow$ stallo su c nell'addizione
[se si carica prima b, altrimenti stallo su b]

$\Rightarrow$ stallo su f nella sottrazione

Supponiamo che

- b, c, e ed f siano nelle locazioni di memoria il cui indirizzo è rispettivamente in $t0, $t1, $t3, $t4
- a e d vadano memorizzati nelle locazioni il cui indirizzo è rispettivamente in $t2 e $t5

Esecuzione senza intervento del compilatore

```
lw      $s1, 0($t0)       # carica b
lw      $s2, 0($t1)       # carica c
add     $s3, $s1, $s2     # somma con stallo
sw      $s3, 0($t2)       # memorizza a
lw      $s4, 0($t3)       # carica e
lw      $s5, 0($t4)       # carica f
sub     $s6, $s4, $s5     # sottrazione con stallo
sw      $s6, 0($t5)       # memorizza d
```

…intervento del compilatore…

```
lw      $s1, 0($t0)      # carica b
lw      $s2, 0($t1)      # carica c
add     $s3, $s1, $s2    # somma con stallo
sw      $s3, 0($t2)      # memorizza a
lw      $s4, 0($t3)      # carica e
lw      $s5, 0($t4)      # carica f
sub     $s6, $s4, $s5    # sottrazione con stallo
sw      $s6, 0($t5)      # memorizza d
```

Esecuzione con intervento del compilatore

```
lw      $s1, 0($t0)        # carica b
lw      $s2, 0($t1)        # carica c
lw      $s4, 0($t3)        # carica e
add     $s3, $s1, $s2      # (scambiate lw e add)
lw      $s5, 0($t4)        # carica f
sw      $s3, 0($t2)        # scambiate lw e sw
sub     $s6, $s4, $s5      # sottrazione
sw      $s6, 0($t5)        # memorizza d
```

ORA PASSIAMO AD ESERCIZI IN CUI

COMPARE LA

CRITICITA' SUL CONTROLLO

Esercizio

Parte 1

Si consideri il codice assembler:

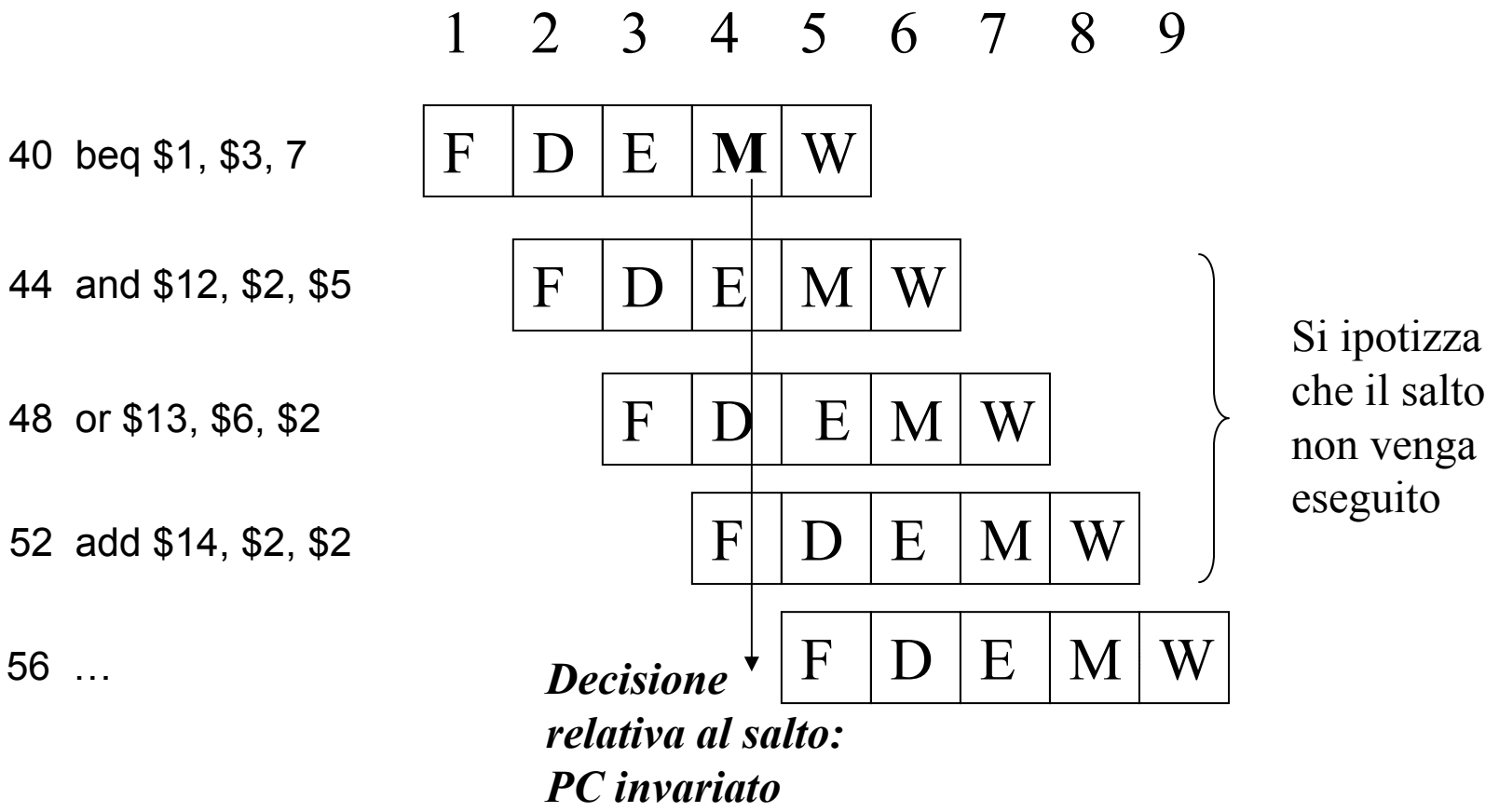
```
40      beq     $1, $3, 7          #  salto a: 40 + 4 + 7*4 = 72
44      and     $12, $2, $5
48      or      $13, $2, $6
52      add     $14, $4, $2
…
72      lw      $4, 50($7)
```

- Si consideri la tecnica di previsione statica "ipotesi di salto non eseguito".
- Si consideri una pipeline a 5 stadi (F, D, E, M, W) in cui la decisione del salto (e l'aggiornamento del PC) si effettuano nello stadio M:

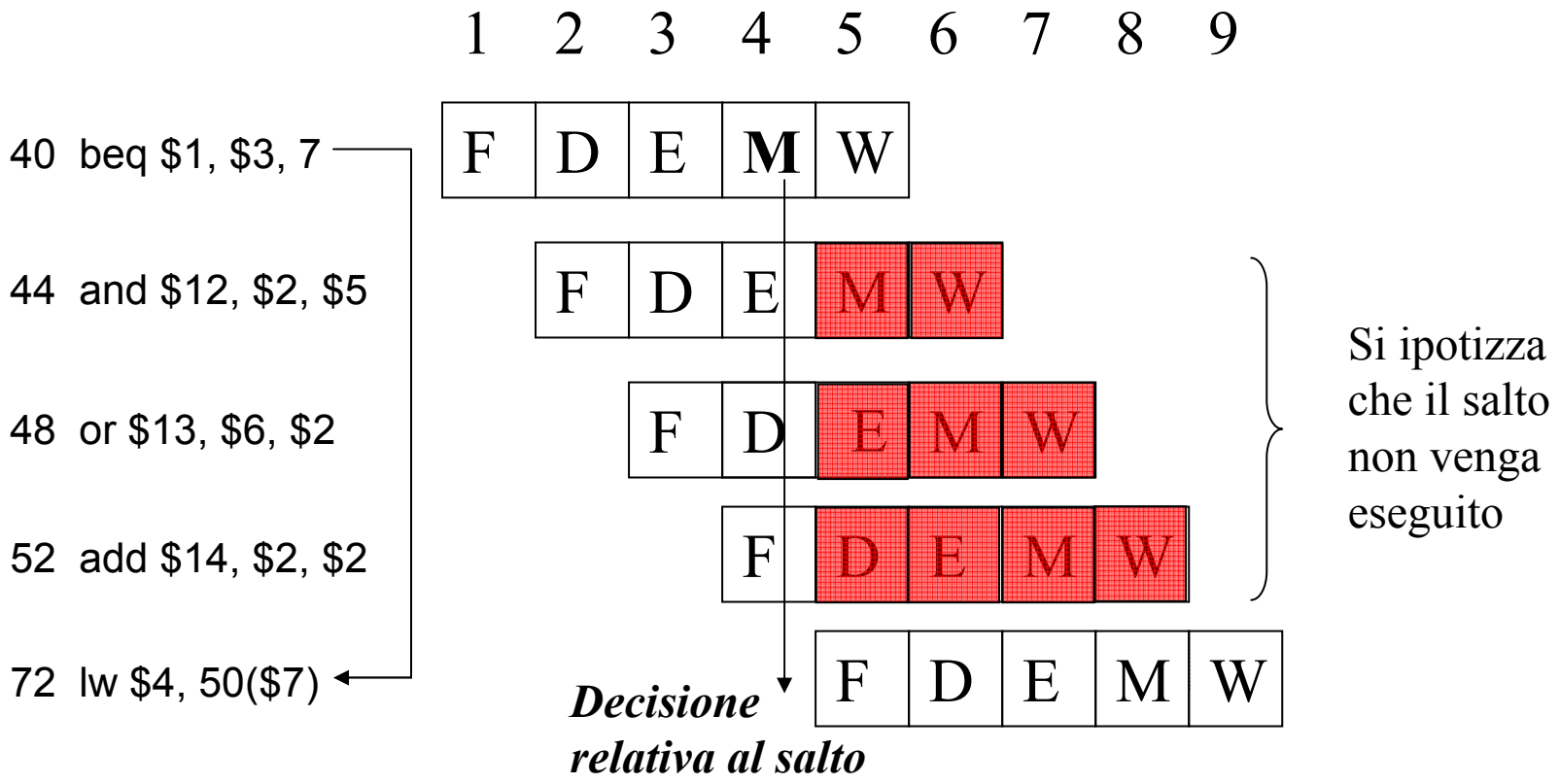
  ⇒ Tracciare il diagramma temporale nell'ipotesi di previsione corretta e nell'ipotesi di previsione errata; indicare in questo caso il numero di cicli di ritardo.

  ⇒ Ripetere esercizio considerando salto in stadio D (al posto di M)

Se il salto non viene effettuato [previsione corretta] tutto procede normalmente:



Se il salto viene effettuato [previsione errata] nel ciclo di clock 4 è aggiornato PC [caricamento nuova istruzione di destinazione al ciclo di clock successivo]

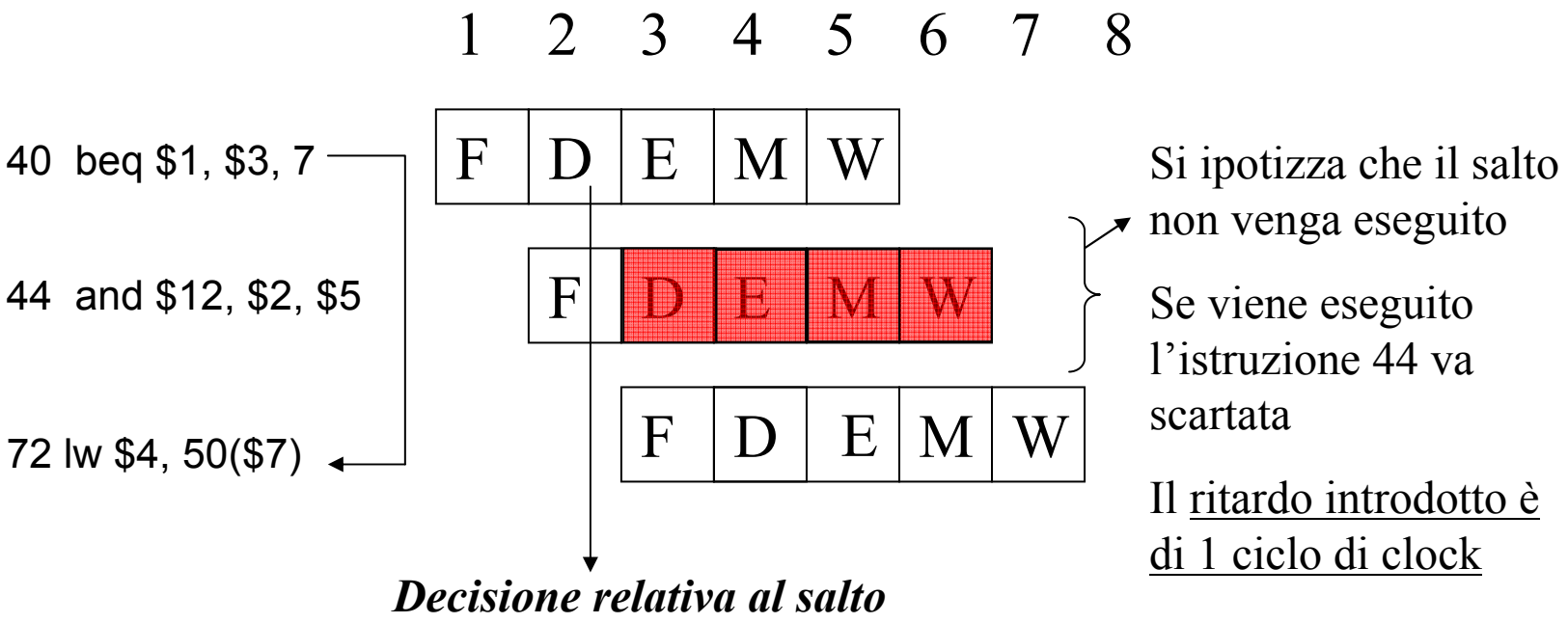


Se il salto viene effettuato [previsione errata], le istruzioni 44, 48, 52 vanno scartate, si introduce un ritardo di 3 cicli di clock.

In rosso sono indicate le bolle create nel ciclo di clock 4

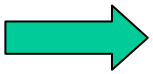
- Spostando l'esecuzione dei salti dallo stadio M allo stadio D si riduce il ritardo a un solo ciclo:

  nell'esempio se si verifica il salto, solamente l'istruzione 44 (di cui è stato fatto il fetch) va scartata



Parte 2

- Si consideri la seguente combinazione di istruzioni
  - 22% load, 11% store, 49% formato-R, 16% salti condizionati, 2% salti incondizionati
- Per esecuzione con pipeline [per branch, previsione di "salto non eseguito"]:
  a) metà delle istruzioni di load seguita da un'istruzione che ne utilizza il risultato (dipendenza dati, senza riordino)
  b) un quarto dei salti si suppone siano oggetto di previsione errata
  c) ritardo causato dai salti incondizionati pari a 1 ciclo di clock (in questo caso si suppone che il salto venga effettuato nello stadio di decodifica)
- Tempo di ciclo: 2 ns (per l'unità funzionale più lenta)



Confrontare le prestazioni relative ad uno schema multiciclo e ad uno schema con pipeline [considerando le due ipotesi di esecuzione dei salti, ovvero in M o D]

Al solito, per il multiciclo:

Numero di cicli per ciascuna classe di istruzioni (fig. 5.42)

- Load = 5
- Store = 4
- Formato-R = 4
- Salti cond. = 3
- Salti incond. = 3

CPI = 0,22×5 + 0,11×4 + 0,49×4 + 0,16×3 + 0,02×3 = 4,04

Tempo medio per istruzione = 4,04 × 2 ns = **8,08 ns**

Per la pipeline (esecuzione salto nello stadio M: ritardo 3 cicli di clock):

Calcoliamo prima il numero di cicli di ritardo per ciascuna classe di istruzioni (nel caso di pipeline a regime):

Load = 0,22×0,50×1 = 0,11 (vedi (a))

Salti cond. = 0,16×0,25×n = 0,04×n (vedi (b))

Salti incond. = 0,02×1 = 0,02 (vedi (c))

dove n = 3 [esecuzione salti cond. in stadio M] o n=1 [in stadio D]

$\delta_M = 0,11 + 0.12 + 0,02 = 0,25$

$\delta_D = 0,11 + 0.04 + 0,02 = 0,17$

Quindi:

$CPI_M = 1 + \delta_M = 1,25$ Tempo medio esecuz. 1,25×2ns = 2,5ns

$CPI_D = 1 + \delta_D = 1,17$ Tempo medio esecuz. 1,17×2ns = 2,34ns

Confronto prestazioni pipeline vs. multiciclo: speedup

NB: facciamo riferimento alla soluzione migliore – esecuzione salto in stadio D
[per l'altro caso il calcoli sono analoghi]

L'incremento delle prestazioni (o speedup) del processore con pipeline rispetto al processore multi-ciclo è dato da:

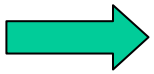
$$\text{Speedup} = \frac{\text{Tempo}_{\text{multiciclo}}}{\text{Tempo}_{\text{pipeline}}} = \frac{8{,}08}{2{,}34} = 3{,}45$$

Il tempo con pipeline ideale sarebbe 2 ns e quindi lo speedup nel caso ideale sarebbe

$$\text{Speedup} = \frac{\text{Tempo}_{\text{multiciclo}}}{\text{Tempo}_{\text{pipeline-ideale}}} = \frac{8{,}08}{2} = 4{,}04 \ \ (> 3{,}45)$$

Esercizio (sulla coda delle istruzioni)

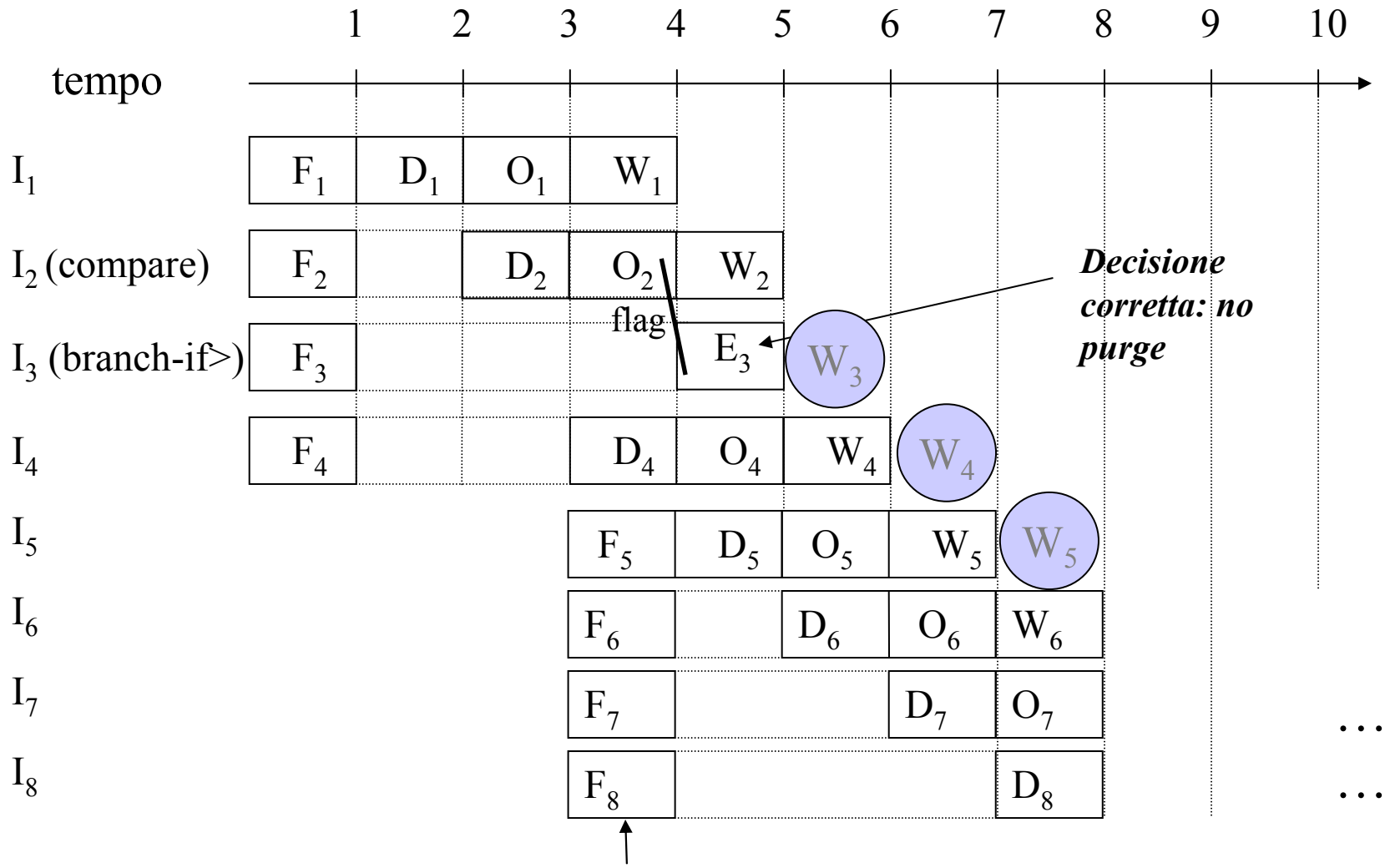
- Si consideri uno schema che preveda una coda delle istruzioni con pipeline a 4 stadi e capace di caricare 4 istruzioni per ciclo di clock (ipotesi usuali).
- Si assuma una semplice predizione sui salti condizionati di tipo "salto non effettuato".
- Si assuma che la condizione di salto sia valutata sulla base di un'istruzione "compare" immediatamente precedente al salto che fa una sottrazione di due registri ed aggiorna di conseguenza i codici di condizione.
- Si assuma che l'hardware renda possibile la propagazione dei dati verso l'unità di prelievo delle istruzioni.



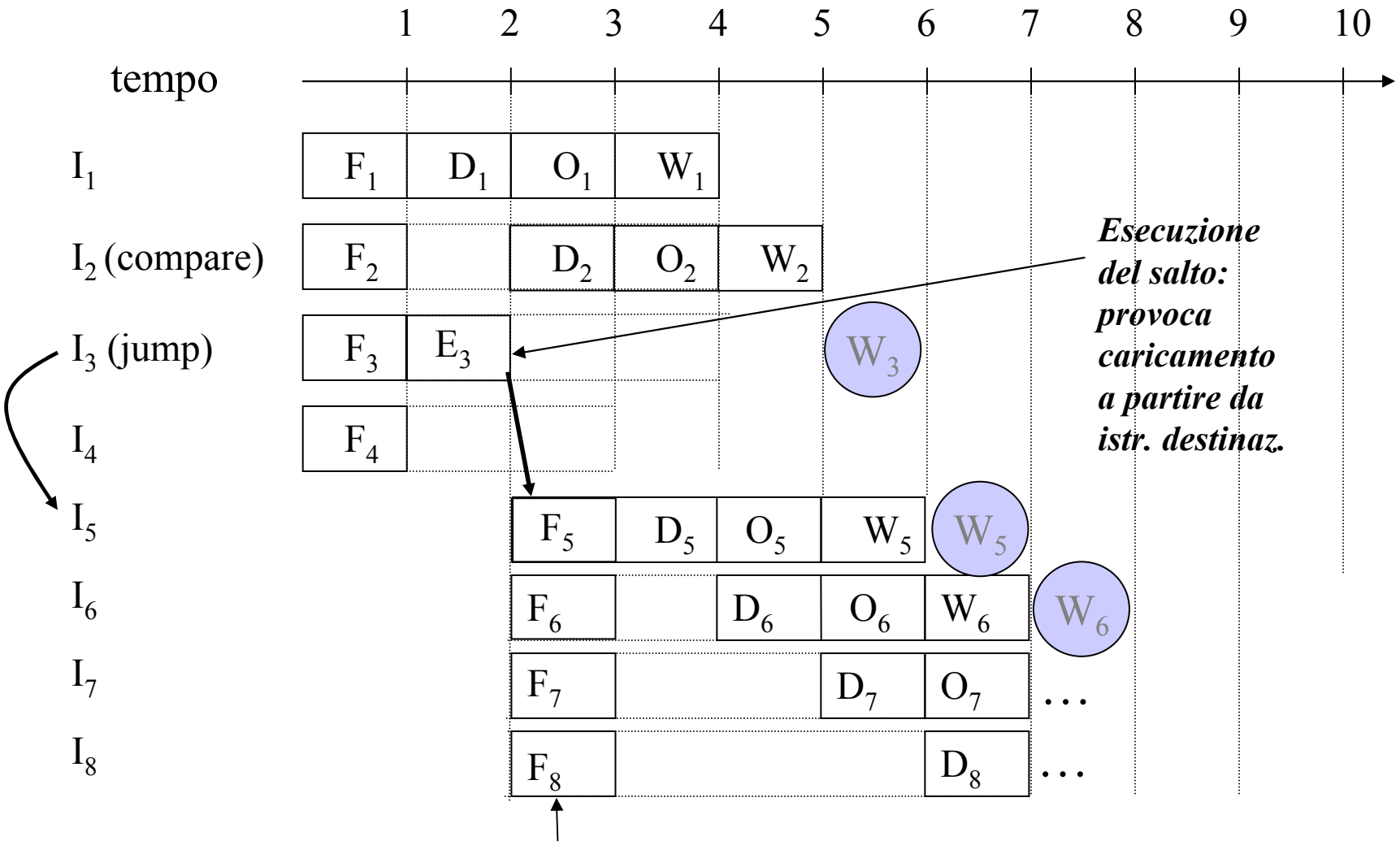
Tracciare i diagrammi temporali nel caso di salto incondizionato, salto condizionato con previsione corretta e salto condizionato con previsione errata [assumere che la coppia compare - branch appaia in seconda e terza posizione tra le quattro caricate]

Determinare le penalità di salto nei tre casi.

Caso di salto condizionato con predizione corretta



Caso di salto incondizionato

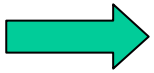


NB: caricamento non appena stadio D_2 impegnato!

PENALITA' = -1

Note sui lucidi precedenti:

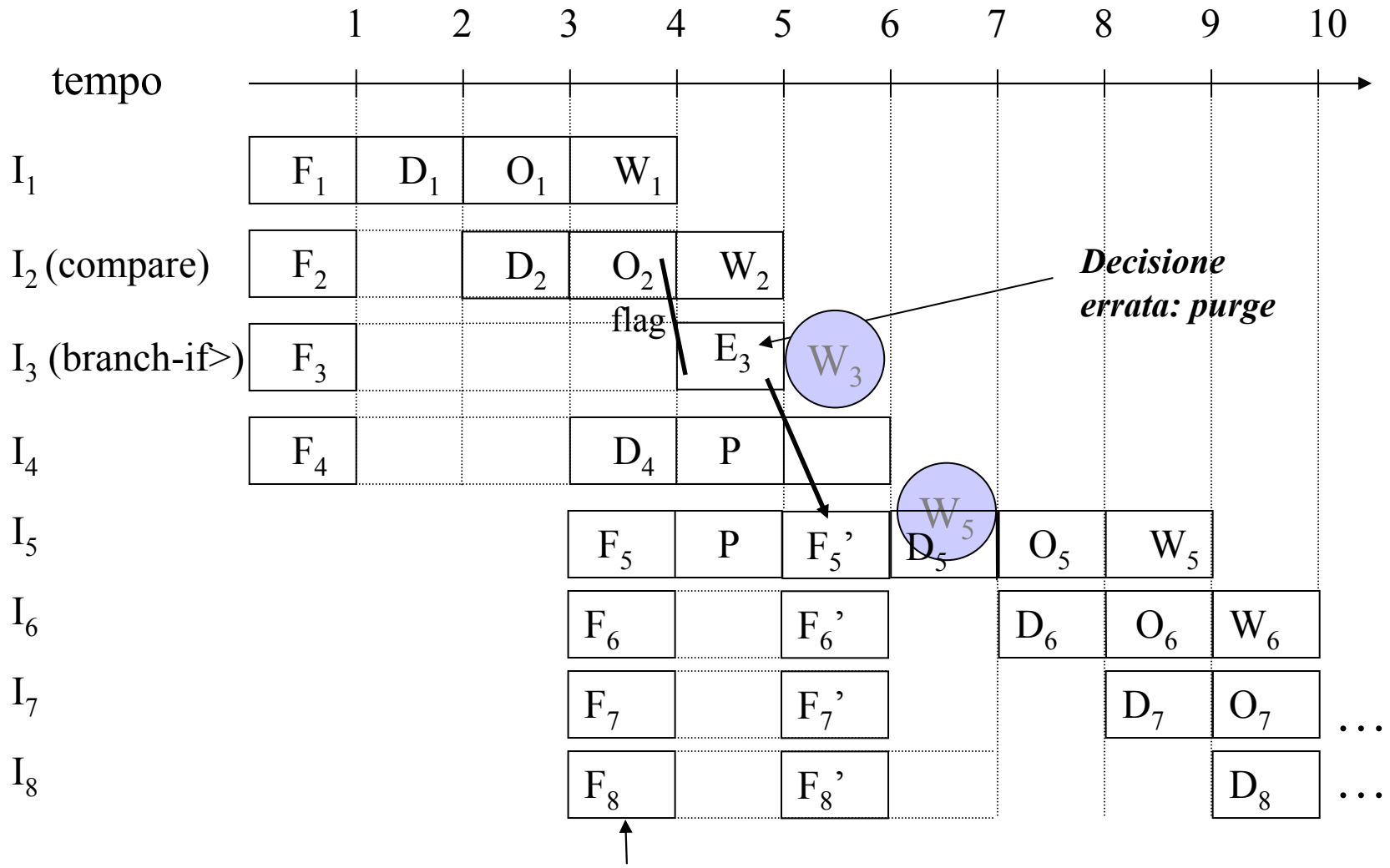
- L'istruzione di salto viene decodificata ed eseguita direttamente dall'unità di prelievo: non impegna stadi della macchina
- Il caricamento delle quattro istruzioni successive avviene non appena la coda è libera – ovvero nel ciclo di clock in cui l'ultima istruzione [salti esclusi] utilizza lo stadio di decode



In tal modo si riesce a conseguire una penalità -1 per i salti [come se i salti non esistessero]

- Nel caso di previsione corretta, il prelievo avviene come descritto sopra e consegue quindi una penalità -1; nel caso di previsione errata, occorre ricaricare le nuove istruzioni a partire dal ciclo successivo a quello di esecuzione del salto da parte dell'unità di prelievo…

Caso di salto condizionato con predizione errata



Seconda parte dell'esercizio

Supponendo che:

- il 30% delle istruzioni eseguite siano salti
- l'80% si questi siano incondizionati|condizionati con condizione nota|
  condizionati con previsione corretta

⇒ Calcolare il CPI supponendo nessun altro tipo di stalli

Soluzione

Dei salti, l'80% dà penalità -1, il rimanente 20% dà penalità 2

$\Rightarrow$ CPI = 1 + 0,3×0,8×(-1) + 0,3×0,2×2 = 1 – 0,24 + 0,12 = 0,88

$\Rightarrow$ Throughput = 1/CPI = 1,14 istruzioni/ciclo

Nota che è > 1 grazie alla coda delle istruzioni

Coda delle istruzioni e cache

- La presenza della coda delle istruzioni fa sì che un fallimento nella cache delle istruzioni non incida (o incida meno) sulla velocità di esecuzione: mentre c'è il prelevamento dell'istruzione dalla memoria principale, vengono "consumate" le istruzioni presenti in coda
- Questo si traduce in una diminuzione della percentuale di fallimento nell'accesso alla cache
- Supponiamo che il sistema visto nell'ultimo esercizio disponga di sola cache primaria e che 2% sia la percentuale di fallimento per le istruzioni e 10% la percentuale di fallimento per i dati
- Sia la penalità di fallimento di 6 cicli
- Supponiamo che il 30% delle istruzioni faccia accesso a dati in memoria
- La penalità dovuta ai fallimenti in cache è

$$\delta_{miss} = 0{,}02 \times 6 + 0{,}3 \times 0{,}1 \times 6 = 0{,}30 \text{ cicli}$$

Prestazioni in caso di cache miss e salti: esempio

- Calcoliamo l'effetto combinato delle penalità relative ai fallimenti di accesso alla cache e alle istruzioni di salto
- Sia $\delta_{miss} = 0{,}30$ (penalità di miss appena calcolata)
- Sia $\delta_{salti} = 0{,}30 \times (0{,}8 \times (-1) + 0{,}2 \times 2) = -\ 0{,}12$ (vedi esercizio precedente)
- In assenza di altri stalli si avrà

$$\mathbf{CPI} = 1 + \delta_{\mathbf{miss}} + \delta_{\mathbf{salti}} = 1 + 0{,}30 - 0{,}12 = 1{,}18$$

$$\mathbf{Throughput} = 1 / 1{,}18 = 0{,}85 \text{ istruzioni/ciclo}$$