Calcolatori Elettronici B
a.a. 2004/2005

Pipeline Superscalari
e Dinamiche

Massimiliano Giacomin

Eccezioni con pipeline

Tipologie di eccezioni

- Eccezioni interne
  - chiamata al sistema operativo da parte di un programma utente
  - uso di istruzioni indefinite, divisione per 0, overflow, ecc.
- Eccezioni esterne
  - Richiesta di un dispositivo di I/O
- Malfunzionamento dell'hardware [può essere interna o esterna]

Gestione delle eccezioni

Per ora, supponiamo basti soltanto fornire:

- Trasferimento del controllo ad una routine di servizio
- Indirizzo dell'istruzione "responsabile" dell'eccezione in EPC
  [Exception Program Counter] – in modo da poter poi riprendere il controllo
- Causa dell'eccezione:
  - modifica di un opportuno registro (routine unica per tutte le cause)
  - mediante interruzioni vettorizzate (una routine diversa per ogni causa)

Gestione delle eccezioni nel caso di controllo multiciclo:

- È sufficiente aggiungere al diagramma a stati finiti uno o più stati addizionali responsabili delle operazioni previste
- L'istruzione "responsabile" è quella in esecuzione al momento dell'occorrenza dell'eccezione, la causa dipende dall'evento occorso

⇒

- Per ogni causa uno stato specifico (cui si accede sulla base degli ingressi unità di controllo: opcode – segnali dal datapath – segnali esterni)
- A partire da questo stato:
  - scrittura del registro EPC con PC corrente [– "1 istruzione"]
  - eventuale scrittura del registro causa
  - aggiornamento PC con l'indirizzo della routine di servizio [unica o dipendente dalla causa]

In ogni caso, è importante notare che:

Il verificarsi di un'eccezione comporta l'intervento del sistema operativo, che deve riconoscere la causa dell'eccezione e prendere i relativi provvedimenti:

- Nel caso di istruzione indefinita, malfunzionamento HW, overflow aritmetico, normalmente il programma viene interrotto e viene segnalata l'eccezione
- Nel caso di richieste di I/O o chiamate al sistema operativo, viene salvato lo stato del programma per poi riprenderne l'esecuzione (dopo aver svolto il compito richiesto)
- Si vedrà poi come le eccezioni ed il sistema operativo abbiano entrambi un ruolo nella gestione della memoria virtuale

⇒ Interrelazione HW – SW [Sistema Operativo]

Gestione delle eccezioni nel caso di controllo con pipeline:

PROBLEMI:

- Nel momento in cui avviene un'eccezione, ci saranno più istruzioni nella pipeline...
- Quale istruzione è responsabile dell'eccezione?
- Cosa fare delle istruzioni precedenti ancora presenti in pipeline?
- Cosa fare delle istruzioni successive già presenti nella pipeline?

La gestione può essere diversa a seconda delle tipologie di causa:

1. Eccezioni interne [chiamata S.O. – Overflow o istruzione indefinita…]
2. Interruzioni esterne da parte dei sistemi I/O
3. Malfunzionamento HW

Vediamo la gestione delle eccezioni interne:

- in questo caso l'istruzione responsabile dell'eccezione è chiaramente definita
- l'eccezione avviene in uno stadio specifico S
  [p.es. overflow: stadio EX – istruzione indefinita: stadio D]
- l'istruzione responsabile dell'eccezione deve essere immediatamente interrotta [non deve modificare i registri] ed il controllo passato alla routine di servizio
- le istruzioni che seguono l'istruzione responsabile [negli stadi precedenti S] devono essere eliminate: completare la loro esecuzione sarebbe dannoso dopo il verificarsi di una eccezione
- le istruzioni che precedono l'istruzione responsabile [negli stadi seguenti S] devono essere completate: la routine di servizio deve conoscere lo stato che "precede" l'esecuzione dell'istruzione responsabile dell'eccezione

[vedi esempio nel lucido seguente]

addi $1, $1, 100

→ Valore di $s1 usato da add [via propagazione!]

add $1, $2, $1

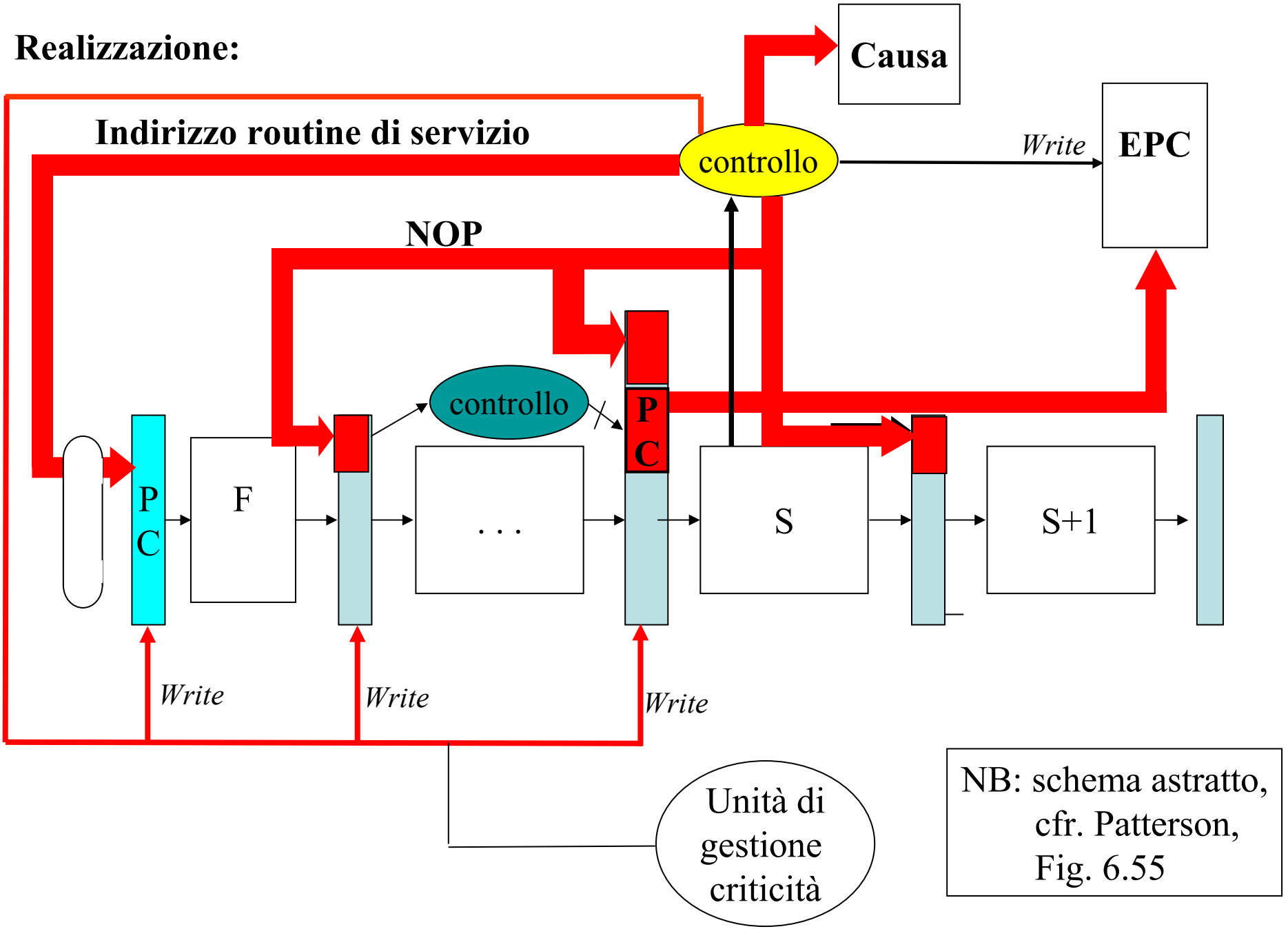
Overflow aritmetico: interrotta nello stadio EX

↓

Dipende anche dal valore di $1: la routine di servizio potrebbe dover conoscere il valore. Ma questo, quando add è nello stato EX, è stato propagato dalla addi ma non è ancora stato scritto nel register file dalla addi [che si trova in M].

⇨ E' necessario allora lasciare che le istruzioni precedenti vengano eseguite;
ciò non significa ritardare la chiamata alla routine di servizio: le istruzioni della routine di servizio possono essere caricate in pipeline normalmente!

Realizzazione:



NOTE AL LUCIDO PRECEDENTE:

- il concetto fondamentale è che le istruzioni fino a S sono scartate, quelle in stadi successivi continuano e i registri EPC, Causa, PC sono aggiornati opportunamente
- ovviamente i segnali Write ai registri interstadio sono in OR con tutti quelli che provengono da unità che necessitino di scrivere i registri interstadio (ovvero, l'unità di gestione delle criticità)
- l'introduzione di un valore "NOP" in un registro interstadio si intende ottenuta comandando un opportuno MUX [vedi p.es. i dettagli nella specifica realizzazione proposta dal Patterson]
- nell'esempio si usa un registro di causa, ma è facile immaginare la modifica per interruzioni vettorizzate (p.es. basta usare un MUX opportuno verso PC, i diversi ingressi corrispondono ai diversi indirizzi delle routine di servizio)

PROBLEMA:

In uno stesso ciclo di clock possono accadere più eccezioni,
con cause diverse (da diversi stadi)

⇒ - Si assegna una priorità alle eccezioni: servita quella a priorità più alta (di solito si serve l'istruzione più vecchia, ovvero lo stadio più avanzato)
- Il registro Causa può mantenere tutte le eccezioni che si verificano in un ciclo di clock: quando è stata servita la prima, sono disponibili le informazioni sulle altre…

Per le interruzioni esterne:

- È possibile "attribuire" l'interruzione all'istruzione più conveniente, ad esempio completando tutte le istruzioni già entrate nella pipeline…
- Per i malfunzionamenti hardware è opportuno interrompere subito l'esecuzione delle istruzioni

Eccezioni precise e eccezioni imprecise

- La gestione delle eccezioni quando si hanno più istruzioni nella pipeline è in ogni caso complicata
- Con pipeline viste fino ad ora, è possibile attribuire a EPC l'indirizzo di una istruzione e fare in modo che si abbia uno stato coerente
  [istruzioni seguenti scartate, precedenti completate]
  $\Rightarrow$ **eccezioni precise**
- Tuttavia, alcune soluzioni progettuali rilassano i requisiti ed ammettono che lo stato risultante dall'eccezione possa essere incoerente:
  $\Rightarrow$ si parla in tal caso di **eccezioni imprecise**

Esempi:

- un'istruzione successiva a quella che ha provocato l'eccezione arriva alla fase di completamento modificando i registri in modo incoerente; oppure
- EPC viene comunque scritto con il valore di PC, che corrisponde all'istruzione in fase di fetch a prescindere da quella che ha provocato l'eccezione.

Ha importanti riflessi sul sistema operativo!

Pipeline superscalari

• Replicano i componenti interni per lanciare più istruzioni all'interno della pipeline:
CPI < 1 o throughput > 1 istruzione/ciclo!!!

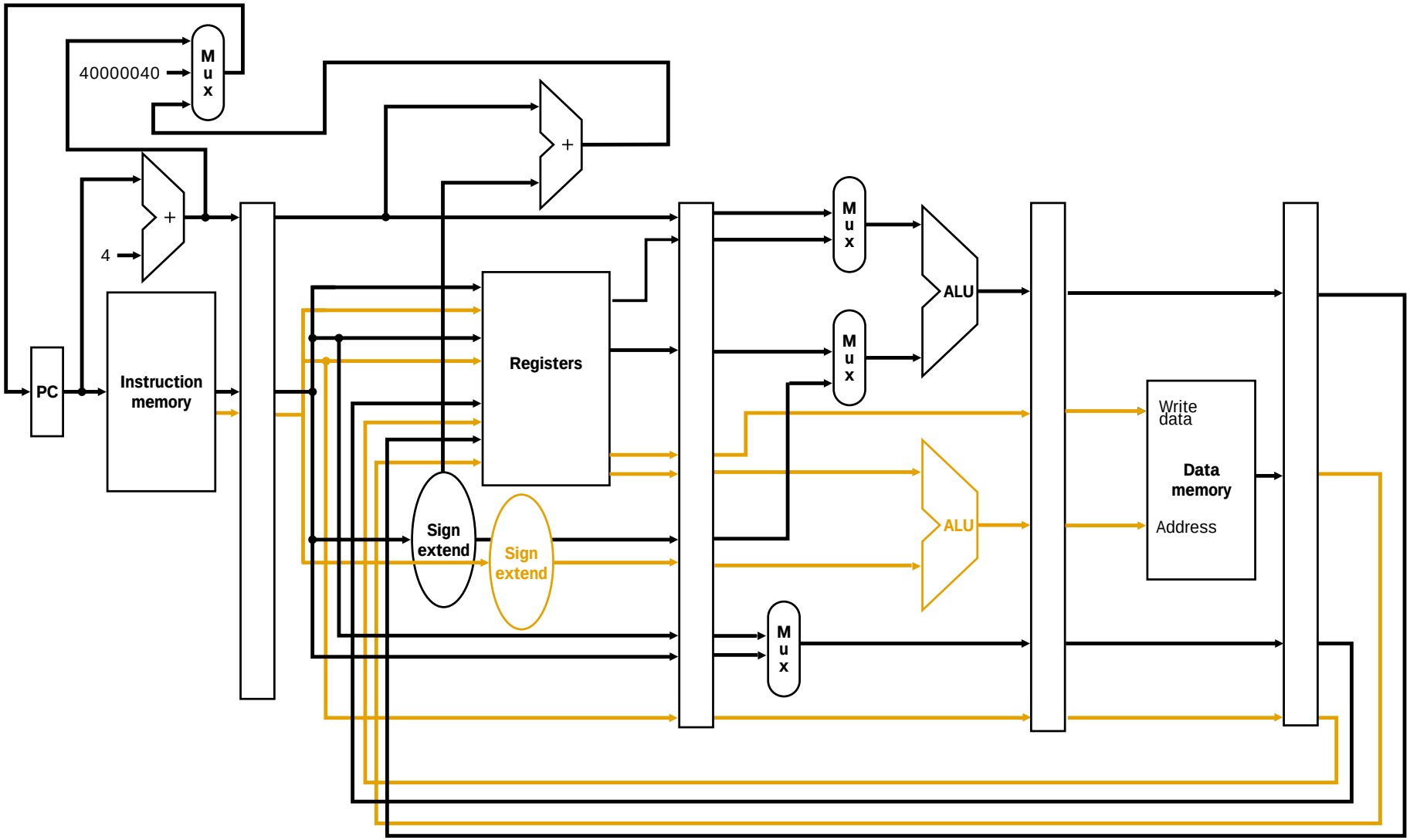
Esempio di riferimento

• Si vogliono eseguire in parallelo per ogni ciclo di clock:
- un'istruzione aritmetico/logica o un salto condizionato
- un'istruzione di trasferimento dati (lw o sw)
• Per semplicità, supponiamo le istruzioni allineate e raggruppate due a due

⇒ Richiede modifiche al data path di "basso livello"
(p.es. porte addizionali nel register file per leggere 4 registri e scrivere due registri in un ciclo di clock).

Richiede nello stadio di esecuzione sia ALU per operazioni logico/aritmetiche, sia ALU per il calcolo degli indirizzi

Risulta un datapath del tipo: …



Esegue in parallelo:

istruzione logico/aritmetica – salto || accesso in memoria

- Sono sempre presenti le unità di propagazione dei dati e di gestione delle criticità.
- La prima istruzione della coppia può essere eseguita solo se non ci sono condizioni critiche.
- Si assume inoltre che, per semplificare la gestione delle criticità:
  - la seconda istruzione non possa essere eseguita se non lo è la prima (istruzioni eseguite ordinatamente, senza riordino dinamico)
  - le due istruzioni della coppia siano indipendenti tra loro

⇒ Questo è un fattore che limita notevolmente le prestazioni

P.es:

```
lw    $t0, 20($t2)
addu  $t1, $t0, $t2
sub   $s4, $s4, $t3
slti  $t5, $s4, 20
```

lw causa stallo di addu per un ciclo di clock, bloccando anche sub e slti che potrebbero essere eseguite [non hanno dipendenze con lw e addu]

⇒ Per sfruttare i vantaggi servono:
- compilatori molto sofisticati, oppure
- tecniche di schedulazione hardware dinamica

Esempio di schedulazione superscalare del codice

[somma del valore in $2 agli elementi di un vettore]

```
Ciclo:  lw    $t0, 0($s1)
        addu  $t0, $t0, $s2
        sw    $t0, 0($s1)
        addi  $s1, $s1, -4
        bne   $s1, $zero, Ciclo
```

Stallo di un ciclo di clock

Per evitare il più possibile le condizioni di stallo, si può procedere così:

1: lw deve essere la prima
2: addu allora deve seguirla dopo 2 cicli di clock
3: sw deve venire dopo

	ALU - beq	lw, sw	Cicli di clock
Ciclo:		lw $t0, 0($s1)	1
			2
	addu $t0, $t0, $s2		3
		sw $t0, 0($s1)	4

⇒ Dopo le prime tre istruzioni, schedulo le altre…

Esempio di schedulazione superscalare del codice

[somma del valore in $2 agli elementi di un vettore]

```
Ciclo:  lw    $t0, 0($s1)
        addu  $t0, $t0, $s2
        sw    $t0, 0($s1)
        addi  $s1, $s1, -4
        bne   $s1, $zero, Ciclo
```

Stallo di un ciclo di clock

Per evitare il più possibile le condizioni di stallo, si può procedere così:

1: lw deve essere la prima
2: addu allora deve seguirla dopo 2 cicli di clock
3: sw deve venire dopo

	ALU - beq	lw, sw	Cicli di clock
Ciclo:		lw $t0, 0($s1)	1
	addi $s1, $s1, -4		2
	addu $t0, $t0, $s2		3
	bne $s1, $zero, Ciclo	sw $t0, 4($s1)	4

Si vede che:

- con due operazioni per ciclo di clock, idealmente CPI = 0.5
- invece, in questo caso si hanno 5 istruzioni in 4 cicli: CPI = 0.8

⇒ Occorrono tecniche sofisticate di compilazione, p.es. espansione dei cicli

```
Ciclo:  lw      $t0, 0($s1)
        addu    $t0, $t0, $s2
        sw      $t0, 0($s1)
        addi    $s1, $s1, -4
        bne     $s1, $zero, Ciclo
```

Assumendo che l'indice del ciclo sia multiplo di quattro:

- devo anticipare il più possibile la lw [creano lo stallo]: effettuo quattro lw come primo blocco di istruzioni, utilizzando 4 registri temporanei;
- pongo le quattro istruzioni di add separate da due cicli di clock
- sistemo le quattro istruzioni di sw subito sotto quelle di lw
- pongo negli slot liberi l'aggiornamento dell'indice [addi $s1, $s1, -16] e il salto bne

⇒ Ottengo…

	ALU - beq	lw, sw	Cicli di clock
Ciclo:	addi $s1, $s1, -16	lw $t0, 16($s1)	1
		lw $t1, 12($s1)	2
	addu $t0, $t0, $s2	lw $t2, 8($s1)	3
	addu $t1, $t1, $s2	lw $t3, 4($s1)	4
	addu $t2, $t2, $s2	sw $t0, 16($s1)	5
	addu $t3, $t3, $s2	sw $t1, 12($s1)	6
		sw $t2, 8($s1)	7
	bne $s1, $zero, Ciclo	sw $t3, 4($s1)	8

- in 8 cicli di clock 4 iterazioni:
  2 cicli di clock a iterazione vs. 4 ottenute prima
  [miglioramento di un fattore due]
- abbiamo però utilizzato 4 registri temporanei ($t0--$t3) al posto di uno ($t0)

Pipeline dinamiche

- Una soluzione alternativa per non perdere cicli di clock è permettere la schedulazione dinamica da parte dell'hardware

P.es:

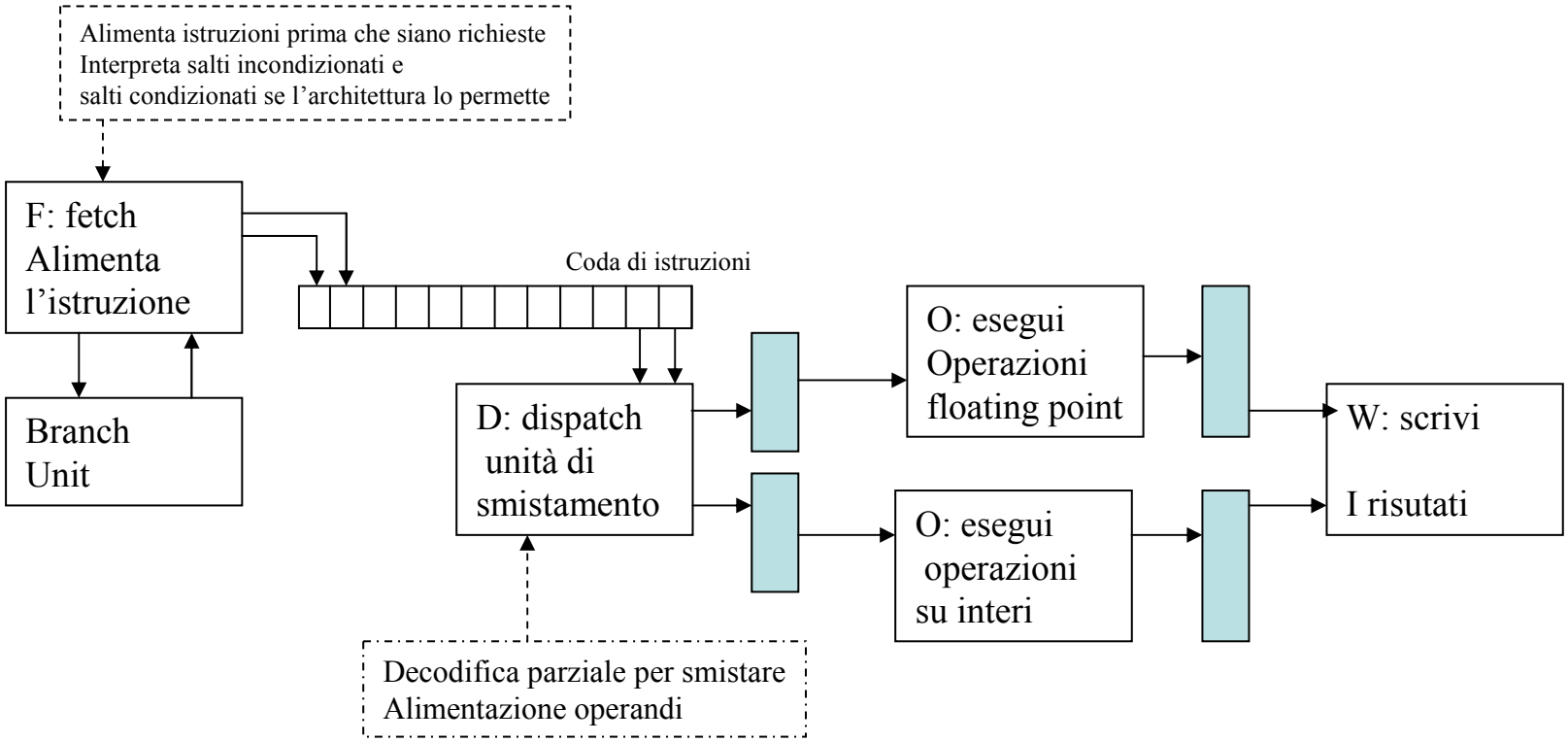
```
lw      $t0, 20($t2)
addu    $t1, $t0, $t2
sub     $s4, $s4, $t3
slti    $t5, $s4, 20
```

Stallo di un ciclo di clock

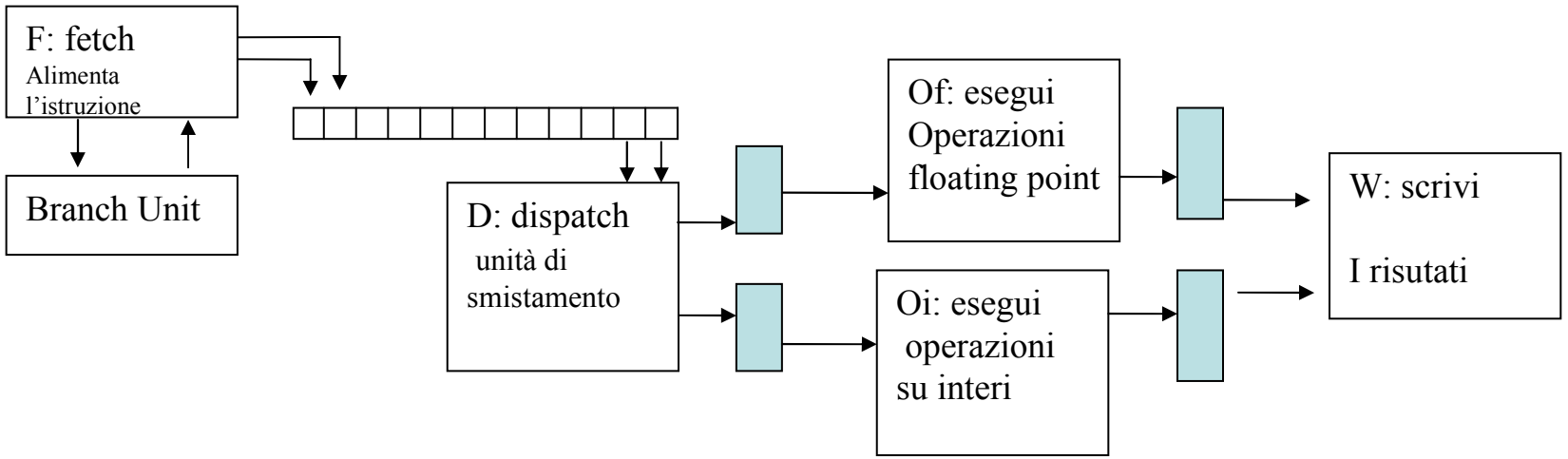
Le istruzioni successive alla coppia sub e slti dovrebbero procedere indipendentemente dalle prime due, perché non dipendono da esse!

⇒ Si introducono **pipeline dinamiche**, capaci di schedulare dinamicamente le istruzioni: uno stallo non blocca tutta la pipeline, ma vengono eseguite le istruzioni successive in attesa che lo stallo sia risolto.

Realizzazione

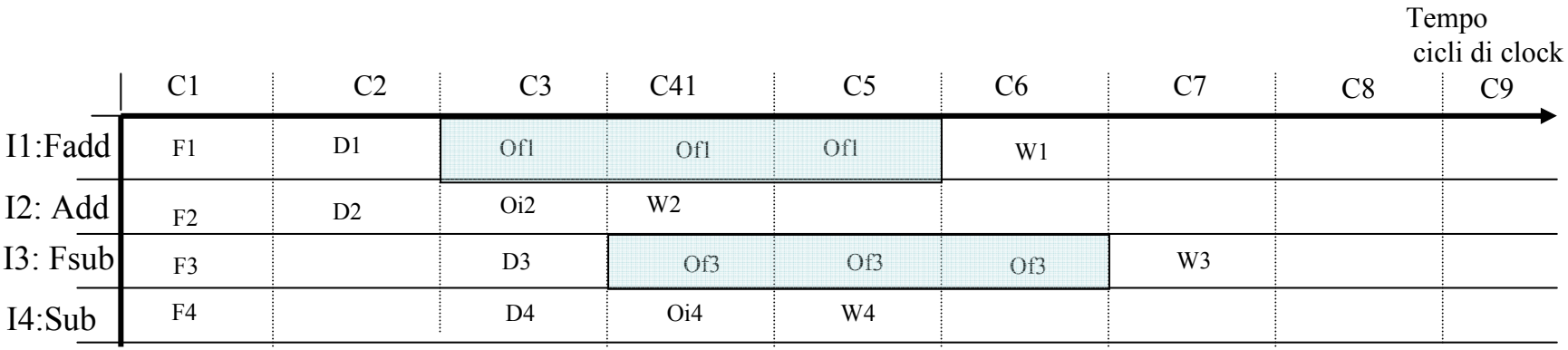


- Operazioni indipendenti possono procedere in parallelo [unità di smistamento può smistare più istruzioni in un ciclo di clock]
- Ciascuna unità O, a seconda della complessità dell'operazione, può essere costituita da più stadi e impiegare quindi diversi cicli di clock [p.es. aritmetica intera vs. virgola mobile]



P.es.

- 4 istruzioni per ciclo di clock prelevate (fetch) dalla cache istruzioni
- Fadd e Fsub sono in virgola mobile [3 stadi per O], add e sub intere [1 stadio per O]
- 2 istruzioni per ciclo di clock smistate da unità D



NB: istruzioni <u>smistate</u> **in ordine** [secondo la sequenza del programma]
istruzioni **<u>completate</u> fuori ordine**

• Ovviamente, vi saranno in pratica diverse (più di due) unità di esecuzione, sia per le varie tipologie di istruzioni sia per eseguire più istruzioni dello stesso tipo in uno stesso ciclo di clock [p.es. più unità per aritmetica intera]

Difficoltà per il completamento fuori ordine

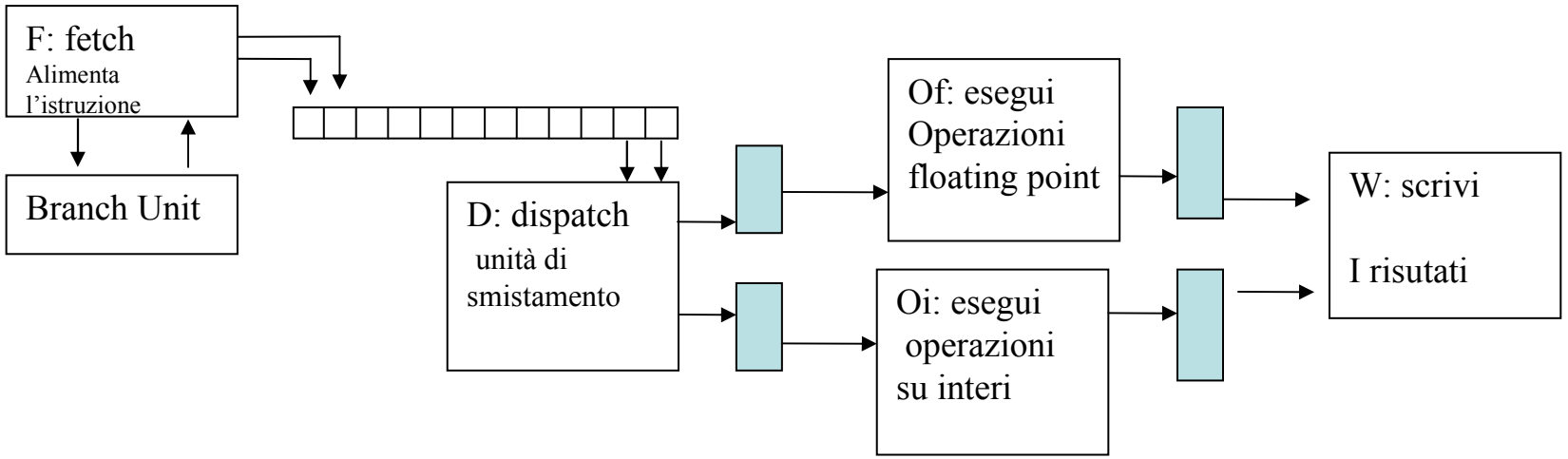
- E' necessario rispettare le dipendenze (tramite propagazione e stallo)
- E' necessario garantire che i valori scritti nei registri corrispondano comunque alla semantica del programma [istruzione precedente non deve "sovrascrivere" un'istruzione seguente!]
- Il problema più grosso si ha con le **eccezioni…**

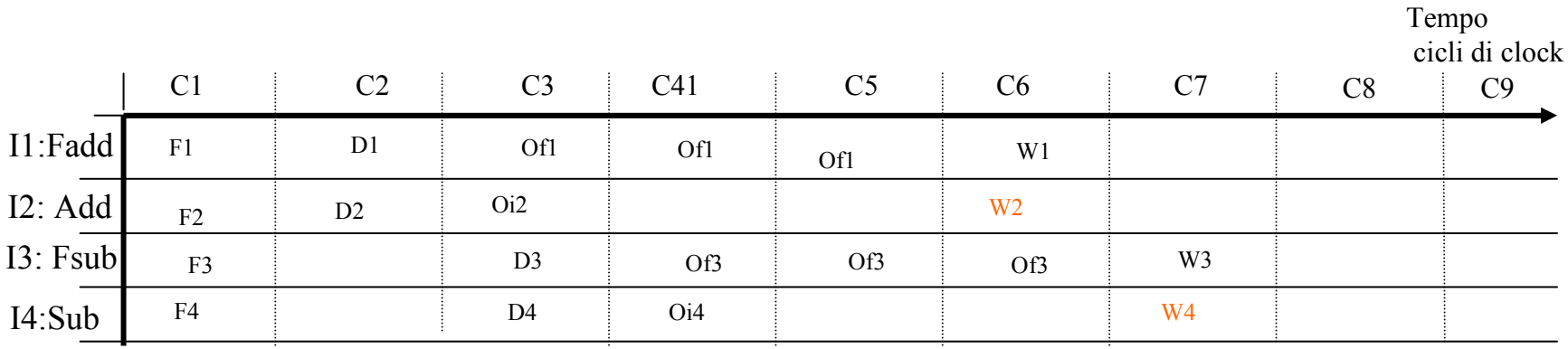
→ Nell'esempio precedente: se un'eccezione accade in Of1 nel ciclo 5, è *eccezione imprecisa* (W2 già avvenuto)

⇨ Lo stato non è coerente, perché deriva da modifiche da parte di istruzioni seguenti quella interrotta!

⇨ RIMEDIO: **completamento in ordine**

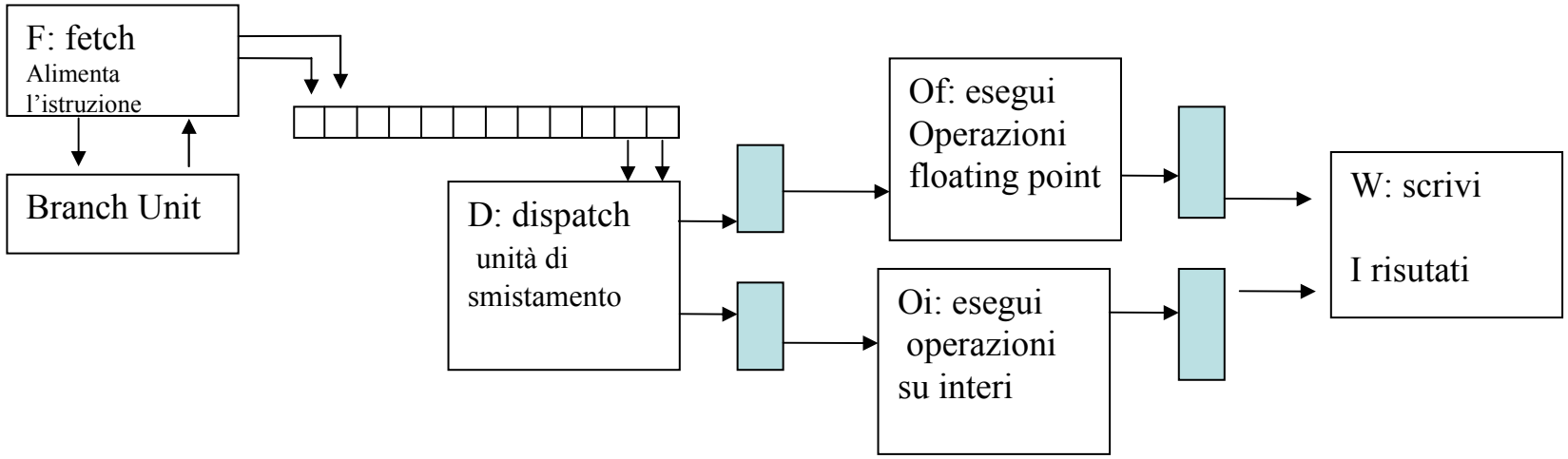
Macchine con più di una unità di esecuzione:
Rimedio al problema del 'fuori ordine' in una macchina con coda , alimentazione di 4 istruzioni contemporaneamente dalla cache: ritardare le scritture (aggiungendo registri temporanei e usando cache non bloccanti)





Se un'eccezione accade in Of1 nel ciclo 5, è *eccezione precisa* poiché gli unici registri aggiornati sono quelli scritti da istruzioni precedenti I1: è quindi possibile scartare le istruzioni logicamente successive a I1.

Completamento in ordine richiede:



- Le istruzioni smistate nelle unità di esecuzione vengono eseguite indipendentemente, ma scrivono risultati in registri temporanei;
- l'istruzione viene completata trasferendo il contenuto dei registri temporanei nei relativi registri permanenti; non appena liberati, registri temporanei disponibili per nuove istruzioni;
- il completamento delle istruzioni (= scrittura dei registri permanenti) avviene **in ordine**

⇒ Possibile scartare le istruzioni seguenti non ancora completate nel caso di eccezione ⇒ **eccezioni precise**

Ciò richiede una **coda di completamento delle istruzioni** che memorizza l'ordine delle istruzioni [completamento segue questa coda]

In particolare

- L'unità di smistamento quando smista un'istruzione prenota:
  - registri temporanei usati dall'istruzione
  - l'unità di esecuzione appropriata
  - posizione nella coda di completamento (segue l'ordine definito nel programma)

  L'esecuzione procede solo se sono disponibili tutte queste risorse
- L'unità di scrittura dei risultati procede a trasferire i risultati da registri temporanei a registri permanenti secondo l'ordine fissato dalla coda, liberando registri temporanei e posizione nella coda di completamento per altre istruzioni

NB: lo smistamento non in ordine può generare blocchi critici…

P.es:

- I_4 in attesa perché l'unità di esecuzione [p.es. virgola mobile] non è libera [p.es. occupata da una istruzione per miss di cache dei dati]
- I_5 e successive vengono smistate, "occupando" i registri temporanei
- I_4 non può procedere perché non ci sono registri temporanei disponibili
- se I_5 e successive dipendono da I_4, ho stallo per attendere I_4 che non può procedere…

⇒ Difficile da gestire, a volte anche lo smistamento in ordine

ESEMPIO:
POWERPC 603

NB: cfr. anche Patterson,
par. 6.9

LSU: Unità load/store
IU: Unità Aritmetica Intera
FPU: Unità Floating Point
SRU: Unità dei registri di
Sistema

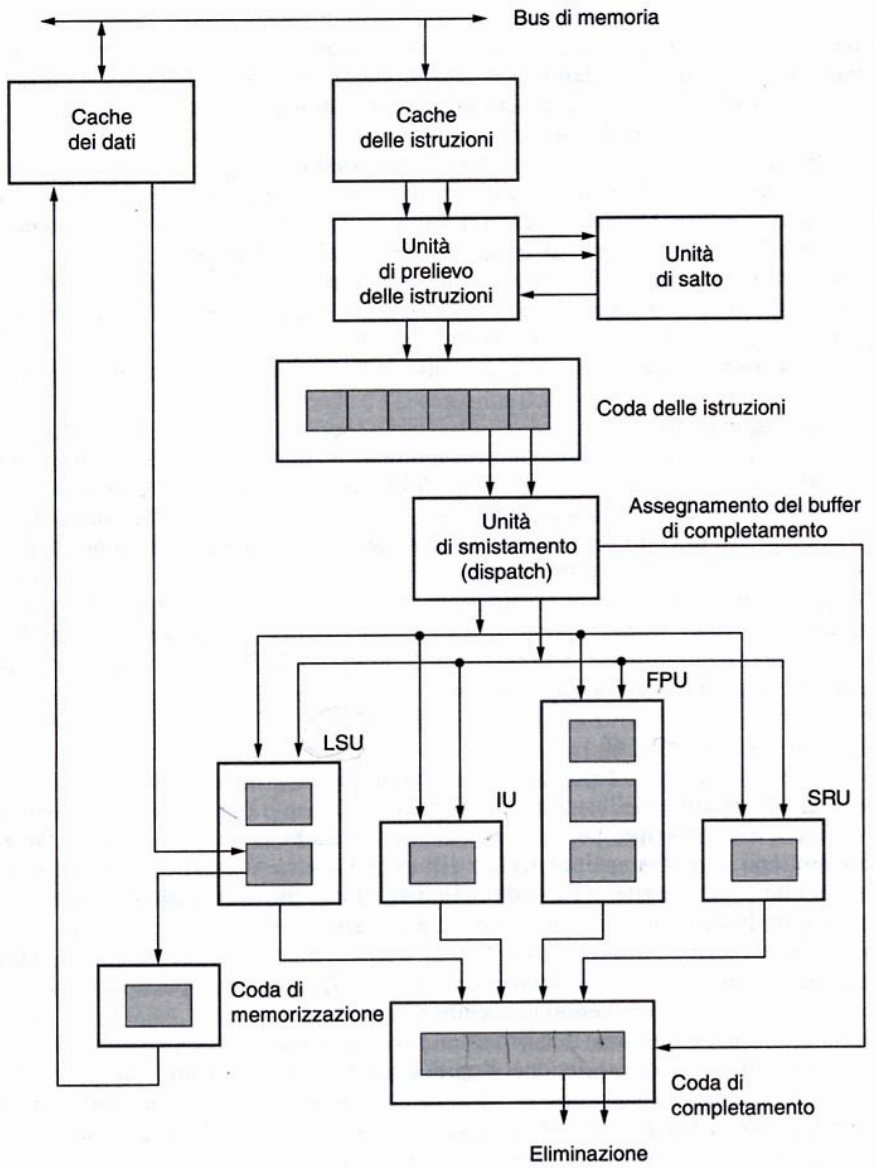


Figura 7.22 Flusso di esecuzione delle istruzioni nel PowerPC 603.