

Pipeline: concetti basilari

E' ovvio che si possa avere un incremento delle prestazioni se più unità funzionali operano in parallelo all'interno del processore. Ma **il successo delle moderne macchine programmabili è, per principio, basata sulla loro natura sequenziale**. E' solo grazie a questa caratteristica che sono in grado di eseguire qualsiasi algoritmo matematico. Qualunque tentativo di sconvolgere completamente la loro struttura sequenziale per puntare su un parallelismo esasperato espone, come vedremo, a seri rischi.

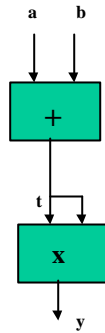
Il *pipelining* è un modo particolarmente efficace di organizzare in parallelo le attività all'interno di un calcolatore o di qualunque altro sistema. L'idea di base è molto semplice ed è applicata di frequente negli impianti di produzione, in cui il *pipelining* viene comunemente chiamato *catena di montaggio*.

Tutti sanno come funziona, in prima approssimazione, la catena di montaggio negli impianti di produzione delle automobili. I lavoratori di una catena di montaggio hanno tutti un compito limitato e ben determinato. Per esempio, nel settore carrozzeria, un operaio ha l'incarico di montare le portiere, un altro monta, invece, i sedili, e così via.

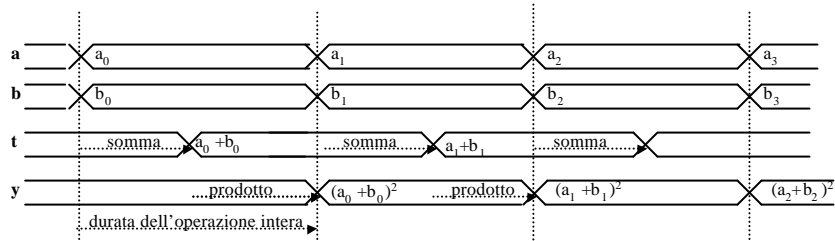
La catena di montaggio

- Il vantaggio è che molti operai cooperano al montaggio di una stessa automobile senza, interferire col lavoro degli altri e, globalmente, **producono molte automobili al giorno**.
- La qualifica professionale dell'operaio può essere ottimizzata in funzione del compito che gli è assegnato nella catena, così la parte operativa ed i controlli di una unità di pipelining possono essere ottimizzate per la specifica operazione logica che deve effettuare.
- la catena di montaggio non riduce il *tempo* di realizzazione di una singola automobile, anzi, molto spesso, lo allunga, ma aumenta il numero delle automobili su cui si lavora e quindi la *frequenza* con cui esse vengono completate.
- Un elemento importante è il cosiddetto *bilanciamento della catena*. I compiti assegnati ai singoli operai debbono richiedere lo stesso tempo di esecuzione. Se un'operazione è più complessa e richiede più tempo di altre, bisognerà rallentare il passo di avanzamento della catena e tutti gli operai che eseguono compiti più semplici rimarranno inoperosi per una frazione del loro tempo.

Pipeline per una funzione logica



il circuito esegue l'operazione
 $y = (a+b)(a+b) = (a+b)^2$

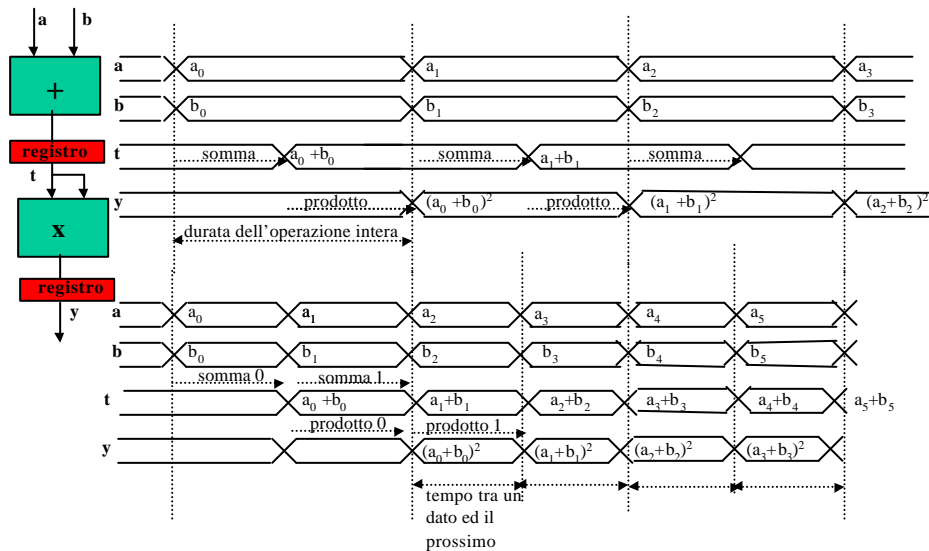


Maggio 2002

Architettura degli elaboratori, Mod. B - 7. Pipeline

3

Interposizione del registro di pipeline

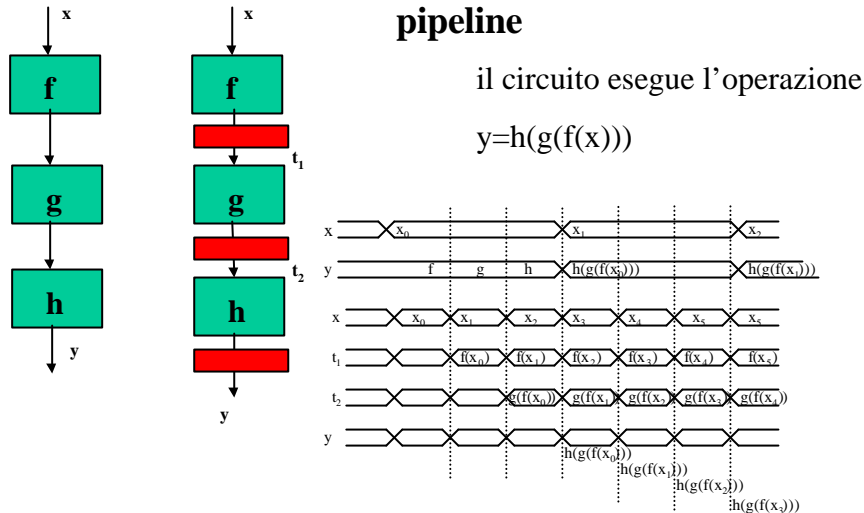


Maggio 2002

Architettura degli elaboratori, Mod. B - 7. Pipeline

4

Una funzione logica complessa: realizzazione pipeline



Maggio 2002

Architettura degli elaboratori, Mod. B - 7. Pipeline

5

Gli elementi di una *pipeline* (1/3)

Il processore esegue un programma prelevando ed eseguendo le istruzioni, una dopo l'altra, perché è **una macchina sequenziale**.

La successione temporale di una CPU tradizionale non è altro che una successione continua di operazioni di prelievo in memoria di un'istruzione, seguite dalle operazioni necessarie per la sua esecuzione.

In una ipotetica CPU di tipo rigidamente sequenziale ma con un set d'istruzioni impostato sul modello RISC, una istruzione di **Load Word (LW)** richiederà, ad esempio:

- 10 ns per il prelievo dell'istruzione,
- 5 ns per la lettura del registro (o dei registri) della CPU
- 10 ns per l'operazione di calcolo dell'indirizzo effettivo,
- 10 ns per il prelievo del dato in memoria
- 5 ns per la scrittura del dato in un registro della CPU.

Il tempo totale di esecuzione è di 40 ns.

Maggio 2002

Architettura degli elaboratori, Mod. B - 7. Pipeline

6

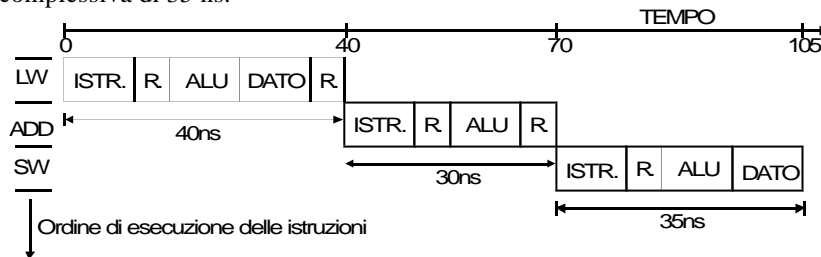
Gli elementi di una pipeline (2/3)

Una istruzione di ADD (somma aritmetica in virgola fissa) richiederà:

- 10 ns per la lettura in memoria dell'istruzione,
- 5ns per la lettura dei registri dove si trovano gli operandi,
- 10 ns per l'operazione della ALU
- 5 ns per la registrazione del risultato in un registro.

Tempo totale 30 ns.

Una operazione di Store Word (SW) richiederà gli stessi tempi di quella di Load, ma senza l'operazione di scrittura nel registro con una durata complessiva di 35 ns.



Maggio 2002

Architettura degli elaboratori, Mod. B - 7. Pipeline

7

Gli elementi di una pipeline (3/3)

Un sistema in grado di eseguire le tre operazioni sarebbe molto più semplice **con un ciclo unico fisso di 40ns**. Avrebbe una struttura adatta a svolgere l'operazione più articolata, nel nostro caso la LW, e le altre verrebbero ottenute non effettuando le operazioni che non servono.

In una architettura semplice, del tipo **“macchina di Mano”** non ci sono alternative ad una evoluzione delle operazioni sequenziale, perché le istruzioni che si susseguono usano, in tempi successivi, una sola logica di controllo con una sola struttura operativa; ma se supponiamo di spezzare l'architettura della CPU in reti sequenziali logicamente autonome, in grado di eseguire una delle 4 o 5 operazioni di cui si compone ciascuna istruzione, allora il discorso cambia sostanzialmente.

Ciascuna unità opera sui valori istantanei delle variabili di stato e sui risultati intermedi registrati in **buffer del tipo “master/slave”** che, **mentre sono pronti a ricevere in ingresso i valori che l'unità precedente sta trattando, sostengono in uscita i valori che i parametri avevano nel periodo di clock precedente**, in modo che lo stadio successivo possa eseguire l'operazione che gli è affidata a partire da essi.

Maggio 2002

Architettura degli elaboratori, Mod. B - 7. Pipeline

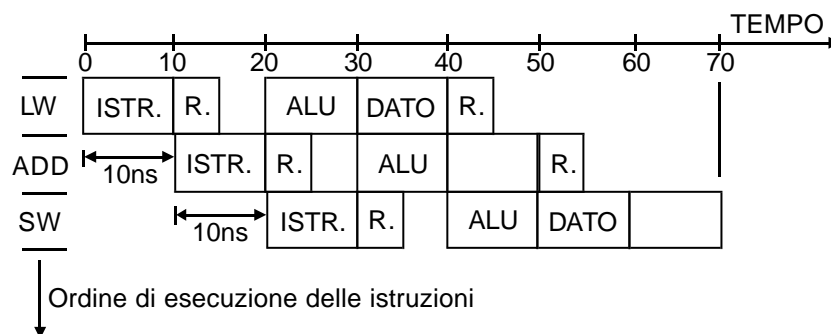
8

Standardizzazione delle operazioni

- Per definire la *catena di montaggio*, a questo punto, manca un solo elemento, il **passo di avanzamento**, che nelle reti sequenziali sincrone, è il **periodo di clock**. Poiché abbiamo operazioni da 5ns ed altre da 10ns, non ci può essere alternativa ad un clock da 10ns di periodo (100 MHz) che permette di eseguire tutte le operazioni in un periodo di clock.
- Una catena di montaggio può produrre solo oggetti che richiedono un **numero uguale di operazioni** e ciò, nel nostro caso, non è vero, perché mentre l'istruzione LW si compone di 5 fasi le altre due ne hanno solo 4. E' chiaro che anche in questo caso bisognerà trovare una soluzione che allinei le istruzioni più semplici a quelle più complesse. L'unità la cui funzione non è richiesta passerà allo stadio successivo gli elementi necessari senza intervenire.

L'esecuzione in pipeline

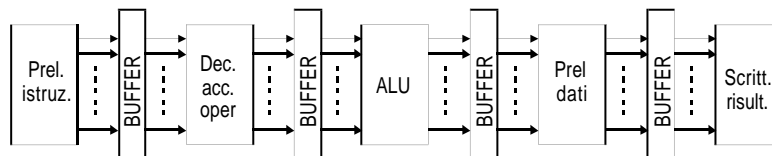
Il tempo va confrontato con 105 ns, 120 ns o 150 ns?



Considerazioni generali

- Per organizzare la struttura a pipeline del processore è spesso necessario allungare il tempo di esecuzione della singola istruzione ma si possono raggiungere tempi di esecuzione di un programma ridotti **di un fattore strettamente correlato al numero di stadi**.
- Nell'esempio, il tempo di esecuzione dell'istruzione è passato per l'istruzione ADD da un tempo minimo teorico di 30ns a 5 periodi di clock (50ns) ma il tempo di esecuzione del segmento di programma si è quasi dimezzato.
- Di solito si dice che il vantaggio del metodo si traduce in un aumento della *frequenza* di completamento delle istruzioni (*throughput*)

Organizzazione hardware



- Ad ogni transizione attiva del clock ciascun buffer trasferisce i livelli logici dei parametri d'uscita dell'unità di sinistra a quella di destra e così via.
- I buffer registrano **comandi, segnali di controllo in genere e risultati parziali**.
- Non tutte le unità di elaborazione operano su tutti i parametri, anzi molto spesso, comandi, operandi e dati vengono semplicemente passati all'unità successiva.

Accelerazione dell'elaborazione (1/2)

Ogni istruzione viene eseguita in 5 periodi di clock (in generale k periodi, essendo k il numero di stadi); in uno schema rigidamente sequenziale l'esecuzione di N istruzioni richiederebbe un tempo

$$t_1 = N k \tau$$

dove τ è il periodo del clock di sistema che nell'esempio era di 10 ns).

Abbiamo appena visto che in una struttura pipeline a 5 stadi, 3 istruzioni vengono eseguite in 7 periodi di clock, cioè in $5 + 3 - 1$, ovvero, in generale, il tempo necessario per eseguire N istruzioni in una pipeline di k stadi è:

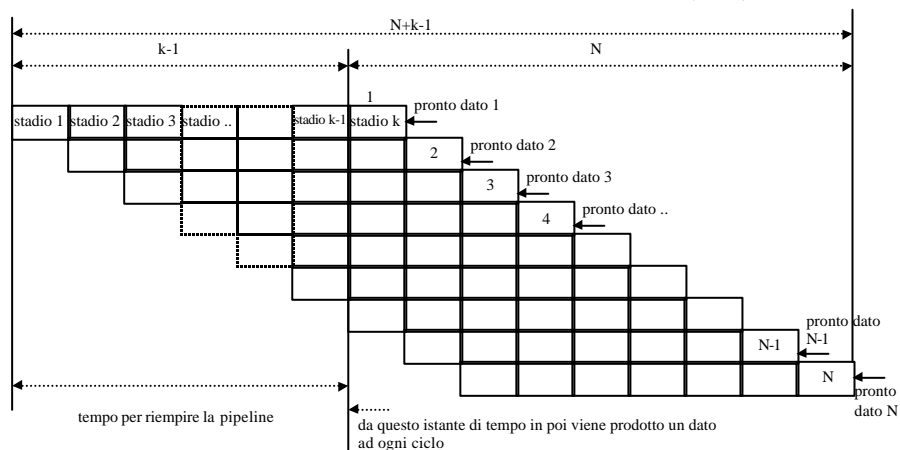
$$t_2 = (k + N - 1) \tau$$

L'accelerazione dell'elaborazione è il rapporto tra i due tempi:

$$s = \frac{t_1}{t_2} = \frac{kN\tau}{(k + N - 1)\tau} \approx k$$

Poiché il numero di stadi k può al massimo avvicinarsi a 10 mentre N per programmi professionali è dell'ordine delle migliaia, al denominatore il parametro $k - 1$ può essere trascurato rispetto a N .

Accelerazione dell'elaborazione (2/2)



nell'esempio riportato sopra $k=7$ $N=9$ n. di cicli = $7-1+9=15$

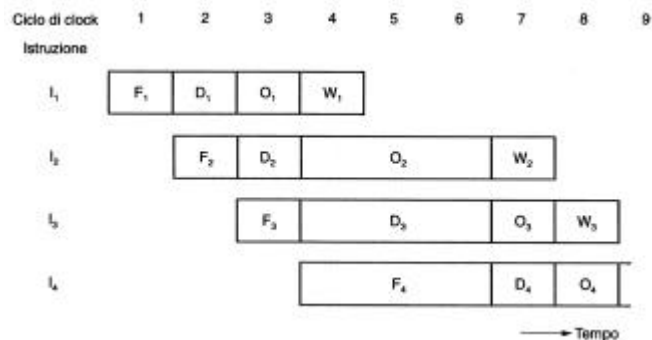
Si possono standardizzare tutte le istruzioni?

- Il segmento di programma preso ad esempio per illustrare la struttura pipeline contiene tre istruzioni diverse ma facilmente riconducibili ad **una struttura hardware in grado di eseguire ciascuna operazione elementare in un periodo di clock**.
- Le piccole difficoltà di sincronismo sono state facilmente superate ipotizzando che i segnali di comando dedotti dalla decodifica dell'istruzione **impongano** di tanto in tanto, ad uno o più degli stadi di **non eseguire la propria funzione**. Per esempio l'unità di prelievo dati, durante l'esecuzione di una istruzione aritmetica non deve prelevare niente, essendo gli operandi già nei registri della CPU.
- Quando però entrano in campo istruzioni molto più complesse, come può essere una divisione in virgola mobile, allora ipotizzare di **allungare i tempi di esecuzione** di tutte le istruzioni per equipararli a quelli dell'istruzione che richiede il massimo tempo, **diventa improponibile** e bisogna rassegnarsi ad accettare che non tutte le operazioni siano concluse in un periodo di clock.

La pipeline entra in fase di *stallo*

- Quando, per esempio, bisogna eseguire una istruzione che impegna l'unità aritmetica in virgola mobile per alcuni periodi di clock, i dati non possono essere trasferiti al buffer d'uscita prima che l'operazione sia conclusa.
- Ciò, non solo blocca l'unità successiva, ma anche tutte le altre che lavorano in parallelo. perchè sia il buffer d'ingresso che quello d'uscita dello stadio di esecuzione non potranno essere aggiornati, fintanto che l'operazione matematica non è conclusa. Quando, infatti, il clock di sistema avvia l'operazione di uno stadio viene contemporaneamente inibito l'arrivo del segnale di clock ai buffer a valle ed a monte del blocco logico e solo la fine dell'operazione può ripristinarlo.

Stallo da ritardo di esecuzione



Se la fase di esecuzione dell'istruzione I_2 dura 3 periodi di clock, le operazioni della pipeline sono in *stallo* per due cicli di clock. Le normali operazioni della pipeline riprendono nel ciclo di clock 7.

Il ruolo della cache nella pipeline (1/3)

- Ogni volta che uno degli stadi della pipeline non può completare l'operazione in un ciclo di clock, **la pipeline entra in stallo**.
- Tale situazione può essere causata da un'operazione aritmetica particolarmente lunga, come nell'esempio precedente, o da un accesso alla memoria principale **in seguito a un fallimento nell'accesso alla cache** (*cache miss*).
- **Quando si verifica una situazione di stallo nella pipeline, si determina un degrado delle prestazioni.** Un obiettivo importante nel progetto di un processore dotato di pipeline è quindi l'identificazione di queste situazioni e delle possibili soluzioni che ne minimizzino l'impatto sulle prestazioni.
- **La pipeline funziona come un registro a scorrimento.** Al termine di ogni ciclo di clock, le informazioni contenute in tutti gli stadi della pipeline si spostano in avanti di uno stadio. Ogni qualvolta un qualunque stadio della pipeline richieda più tempo per completare la sua attività, l'intera pipeline entra in stallo finché tale unità non risulta nuovamente pronta ad accettare una nuova istruzione.

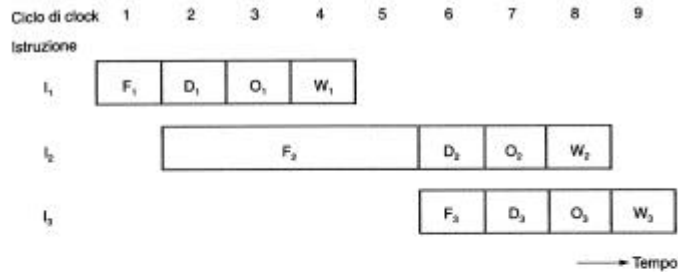
Il ruolo della cache nella pipeline (2/3)

- Lo stallo della pipeline deve essere un **evento raro** e deve verificarsi solo in corrispondenza di situazioni non altrimenti gestibili.
- La struttura hardware della pipeline deve spezzare tutte le istruzioni, che non richiedano tempi di esecuzione molto più lunghi della media o non siano, per loro natura, a rischio stallo, in stadi operativi il più possibile uguali (**bilanciamento della pipeline**): E' indispensabile che tutte gli stadi operino con certezza in un periodo di clock.
- Questa considerazione è particolarmente importante per il passo, di **prelievo delle istruzioni**. Tale organizzazione implica che il ciclo di clock debba avere lunghezza pari o maggiore del tempo richiesto per completare l'operazione di prelievo di un'istruzione. D'altra parte, il tempo di accesso alla memoria principale può essere anche dieci volte maggiore del tempo necessario a svolgere le operazioni interne al processore che costituiscono gli stadi base della pipeline, come la somma di due numeri. Quindi, se ogni fase di prelievo delle istruzioni richiedesse l'accesso alla memoria principale, il pipelining non sarebbe di grande utilità.

Il ruolo della cache nella pipeline (3/3)

- L'utilizzo delle memorie cache risolve il problema dell'accesso alla memoria principale. In particolare, quando una cache primaria è inclusa all'interno dello stesso chip del processore, **il tempo di accesso alla memoria cache è dell'ordine dei tempi necessari a svolgere una qualsiasi altra operazione interna al processore.**
- Questo rende possibile, come abbiamo visto nell'esempio, la suddivisione della fase di prelievo e di esecuzione dell'istruzione in passi di durata più o meno uguale. Ognuno di questi passi viene svolto da un diverso stadio della pipeline, e il periodo di clock viene scelto in modo da corrispondere al passo che dura più a lungo.
- In alcuni casi, una richiesta di accesso alla memoria può dar luogo a un **fallimento di accesso alla cache**. Ciò comporta un allungamento del tempo di esecuzione dello stadio di pipeline che ha effettuato la richiesta di accesso alla memoria. In questa situazione, **la pipeline entra in stallo.**

Stallo da read miss nella memoria cache



a) Passi di esecuzione di un'istruzione in funzione del tempo.

Ciclo di clock	1	2	3	4	5	6	7	8	9
Stadio									
F: Preflievo	F ₁	F ₂	F ₂	F ₂	F ₂	F ₃			
D: Decodifica		D ₁	inattivo	inattivo	inattivo	D ₂	D ₂		
O: Esecuzione			O ₁	inattivo	inattivo	inattivo	O ₂	O ₂	
W: Scrittura				W ₁	inattivo	inattivo	inattivo	W ₂	W ₃

b) Attività svolte da ogni stadio in funzione del tempo.

Maggio 2002

Architettura degli elaboratori, Mod. B - 7. Pipeline

21

Affioramento delle bolle

L'unità di decodifica è inattiva a partire **dal ciclo di clock 3 fino al 5**, l'unità di esecuzione è inattiva **dal ciclo di clock 4 al 6** e lo stadio di scrittura rimane inattivo **dal ciclo di clock 5 al 7**. Tali periodi di inattività sono talvolta denominati **bolle (bubbles)** nella pipeline.

La rappresentazione alternativa delle operazioni di una pipeline è particolarmente efficace per osservare come il periodo di inattività di tre cicli di clock, o **bolla**, creato dal fallimento di accesso alla cache, si muova lungo la pipeline fino all'uscita.

Le operazioni di lettura e scrittura in memoria sono necessarie per fornire l'accesso alle istruzioni e ai dati del programma. La capacità di trasferimento dei dati della cache deve essere tale da non portare la pipeline in stallo fintanto che le istruzioni e i dati richiesti vi risiedono.

Questo può essere facilitato fornendo **due cache separate** sul chip del processore, una per le istruzioni e una per i dati. L'esistenza di due cache separate, oltre all'ovvio vantaggio di maggiore capacità globale, **elimina la possibilità di un ritardo nell'operazione di prelievo di un'istruzione a causa di un accesso a un operando in corso.**

Maggio 2002

Architettura degli elaboratori, Mod. B - 7. Pipeline

22

Vincoli di dipendenza

Se due istruzioni, I_1 ed I_2 , costituiscono un programma eseguito da una pipeline, **l'esecuzione di I_2 può iniziare prima che l'esecuzione di I_1 sia completata**. Ciò significa che i risultati prodotti da I_1 possono non essere disponibili per l'utilizzo da parte di I_2 .

E' possibile dimostrare attraverso un semplice esempio come **sia possibile ottenere risultati non corretti quando le operazioni vengono svolte in parallelo**.

Si ipotizzi che $A = 5$, e si considerino le due operazioni seguenti:

$$A \leftarrow 3 + A$$

$$B \leftarrow 4 \times A$$

Quando queste operazioni vengono svolte nell'ordine dato, il risultato è:

$$B = 32.$$

Ma se le due operazioni vengono effettuate in parallelo, il valore di A utilizzato per calcolare B è quello iniziale 5; ciò porta

$$B = 20.$$

Tipica dipendenza: istruzione di trasferimento

- **Due operazioni qualsiasi, che siano tra loro dipendenti, non possono mai essere effettuate in parallelo.**

Questa condizione, piuttosto ovvia, ha notevoli conseguenze.

Capire le sue implicazioni è la chiave per comprendere la varietà delle alternative di progetto e dei compromessi che si trovano nei calcolatori dotati di pipeline.

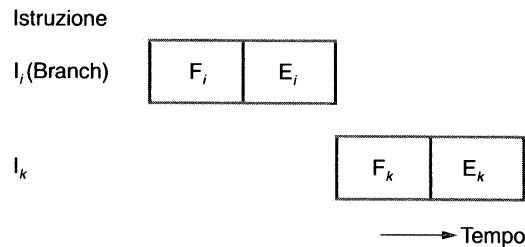
- **L'operazione di prelievo di un'istruzione deve essere indipendente dai risultati dell'esecuzione dell'istruzione precedente.**

Questa condizione è soddisfatta quando un processore preleva le istruzioni da locazioni con indirizzi successivi, ma non è necessariamente soddisfatta in presenza di istruzioni di salto (branch).

- **L'indirizzo da cui deve essere prelevata l'istruzione successiva dipende dal risultato dall'esecuzione dell'istruzione di salto.**

Senza un provvedimento specifico, la pipeline entrerebbe in stallo.

Stallo della pipeline da istruzione di *branch*



La figura si riferisce ad una pipeline a due soli stadi.
L'esecuzione dell'istruzione I_i porta ad eseguire l'istruzione I_k e non la I_{i+1} che segue ed è stata prelevata dall'unità di prelievo (Fetch) nel periodo di clock in cui l'unità di esecuzione stava eseguendo l'istruzione di branch

Stallo da dipendenza (1/2)

In una pipeline più lunga, le fasi di esecuzione di istruzioni successive si sovrappongono. (Con "*fase di esecuzione*" si indicano tutte le operazioni svolte dopo il prelievo di un'istruzione).

La dipendenza che si verifica quando la destinazione di un'istruzione viene utilizzata come sorgente in un'istruzione successiva produce uno stallo che azzerava i vantaggi della pipeline.

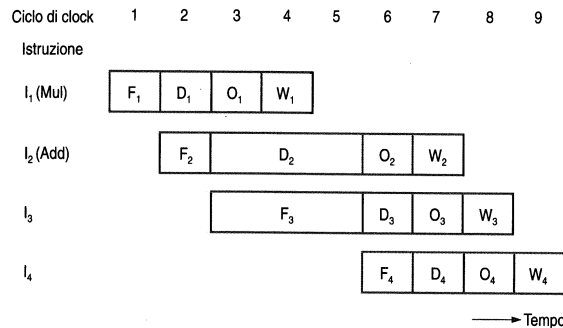
Ad esempio, l'esecuzione in una pipeline a quattro stadi delle due istruzioni:

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

in cui il risultato dell'istruzione di moltiplicazione viene posto nel registro R4, ed è utilizzato dalla seconda come operando, porta ad una situazione di stallo.

Il controllo di dipendenza, indispensabile per evitare di stravolgere l'elaborazione, durante la fase di decodifica, rilevando l'uso di R4, non concluderà l'operazione prima che l'istruzione precedente sia stata perfezionata.

Stallo da dipendenza (2/2)



L'istruzione I_3 viene prelevata nel ciclo di clock 3 mentre I_2 è in fase di decodifica. Se essa è indipendente da I_1 e I_2 , non esiste alcuna ragione per ritardarne l'esecuzione, ma l'unità di decodifica non sarà in grado di accettare una nuova operazione fintanto che è in corso D_2 . Solo quando (alla fine del periodo di clock 5) il buffer d'uscita avrà registrato i parametri di D_2 , quelli di D_3 potranno propagarsi attraverso lo stadio di decodifica.

Maggio 2002

Architettura degli elaboratori, Mod. B - 7. Pipeline

27

Considerazioni generali

- I processori con prestazioni elevate sono spesso caratterizzati da unità di esecuzione multiple per evitare di bloccare un'istruzione in grado di proseguire nell'esecuzione, come accadeva per la I_3 dell'esempio appena visto.

Le ultime CPU Intel hanno 5 unità di esecuzione che possono lavorare in parallelo su istruzioni indipendenti.

- Le cause che possono determinare *bolle* nella pipeline sono molteplici e per ciascuna di esse sono state proposte delle tecniche per evitare lo stallo o per limitarne i danni sulle prestazioni della CPU.
- In un certo senso, come abbiamo visto, la cache memory può considerarsi un provvedimento per evitare, quando possibile, lo stallo nella fase di fetch.
- **Per problemi di tempo non possiamo analizzare tutte le cause di stallo e tutte le possibili soluzioni. Ci limiteremo a considerare alcune delle soluzioni più efficaci per alcuni problemi.**

Maggio 2002

Architettura degli elaboratori, Mod. B - 7. Pipeline

28

Una strategia globale contro lo stallo della pipeline

Le principali cause di stallo della pipeline sono:

- **Fallimento di accesso alla cache**,
- **interdipendenza tra istruzioni consecutive**,
- **trasferimenti incondizionati**,
- **trasferimenti condizionati**, che richiedono delle scelte strategiche.

Queste situazioni comportano la possibilità di frequenti interruzioni nella sequenza di istruzioni generata dall'unità di prelievo.

Per ridurre l'effetto di tali interruzioni, la maggior parte dei processori utilizza sofisticate *unità di prelievo* in grado di prelevare le istruzioni prima che siano effettivamente richieste e di inserirle in una *coda*. Tipicamente la *coda di istruzioni* può memorizzare un certo numero di istruzioni. Una *unità di smistamento (dispatch unit)* preleva le istruzioni dalla testa della coda che funziona, ovviamente secondo una logica (**FIFO**, First In First Out) e le invia alle unità di esecuzione nel momento in cui si liberano.

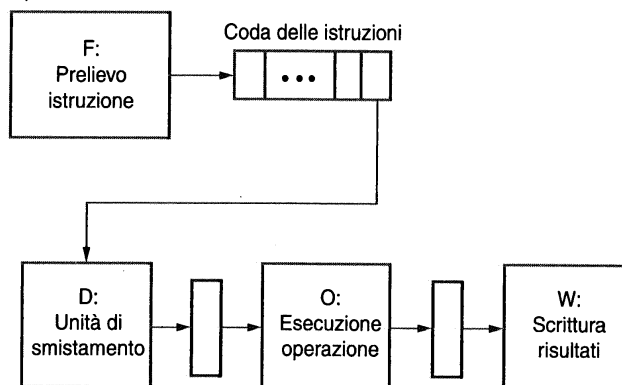
Maggio 2002

Architettura degli elaboratori, Mod. B - 7. Pipeline

29

La nuova struttura hardware della pipeline

Unità di prelievo delle istruzioni



Per essere efficace, l'unità di prelievo deve possedere caratteristiche di decodifica ed elaborazione tali da permettere il riconoscimento e l'esecuzione delle istruzioni a rischio, come le istruzioni di salto.

Maggio 2002

Architettura degli elaboratori, Mod. B - 7. Pipeline

30

Caratteristiche dell'unità di prelievo

Un'istruzione di salto incondizionato, ad esempio, non comporterà ritardi eccessivi, dato che l'unità di prelievo **calcola l'indirizzo di destinazione del salto e continua a prelevare istruzioni partendo da tale indirizzo**. Lo stesso vale per un'istruzione di salto condizionato per cui la condizione di salto è già stata valutata da altre unità.

Il compito dell'unità di prelievo è di mantenere la coda di istruzioni sempre piena. Questo riduce l'impatto di ritardi occasionali durante la fase di prelievo delle istruzioni. Per esempio, in caso di fallimento di accesso alla cache, l'unità di smistamento continua a inviare istruzioni per l'esecuzione finché la coda delle istruzioni non è vuota. Nel frattempo, il blocco corrispondente della cache viene letto dalla memoria principale o dalla cache secondaria. Quando le operazioni di prelievo vengono riprese, la coda delle istruzioni viene riempita nuovamente. Finché la coda non si svuota, un fallimento di accesso alla cache non ha alcun effetto sulla velocità di esecuzione delle istruzioni. **La frequenza con cui le istruzioni devono essere lette dalla cache deve essere sufficientemente elevata per permettere all'unità di prelievo di riempire la coda.**

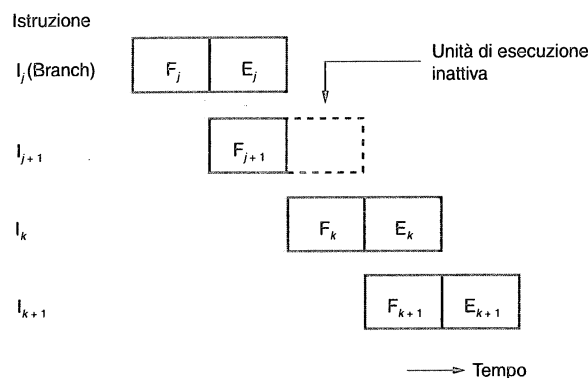
Maggio 2002

Architettura degli elaboratori, Mod. B - 7. Pipeline

31

Penalità di salto

Consideriamo il caso in cui la coda delle istruzioni non sia utilizzata e l'esecuzione proceda con una pipeline a due stadi.



I_j è un'istruzione di trasferimento ad I_k . Il tempo di E_{j+1} è la penalità.

Maggio 2002

Architettura degli elaboratori, Mod. B - 7. Pipeline

32

Branch folding (1/2)

L'utilizzo di una coda delle istruzioni e di una cache veloce rappresenta un modo molto efficace di gestire i salti incondizionati, poiché il prelievo delle istruzioni viene realizzato prima che esse siano in realtà necessarie.

Supponiamo che l'unità di prelievo possa prelevare le istruzioni a quattro a quattro dalla relativa cache. Quando le quattro istruzioni da I_1 ad I_4 vengono inserite nella coda, esse vengono esaminate dalla logica di distribuzione e se I_3 è un'istruzione di trasferimento, ne viene immediatamente avviata l'esecuzione (passo E_3). Nel ciclo di clock successivo, l'unità preleva altre quattro istruzioni, da I_k a I_{k+3} , partendo dall'indirizzo di destinazione del salto. L'istruzione I_4 viene scartata.

Nel frattempo, l'istruzione I_1 procede nella fase di esecuzione, seguita da I_2 . Quest'ultima avanza nella pipeline seguita da I_k .

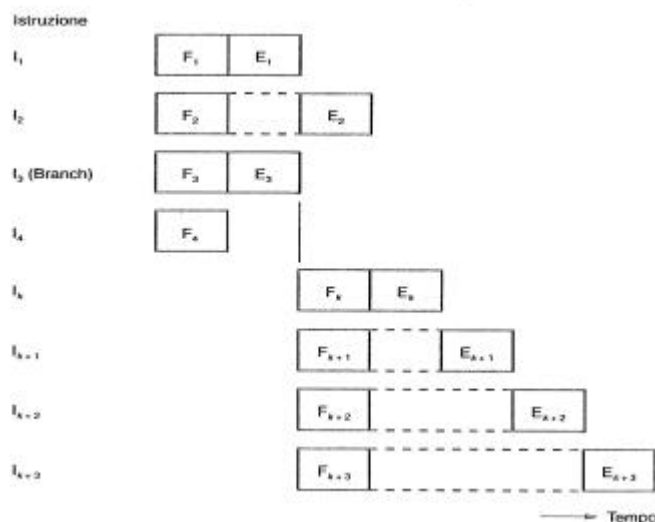
L'istruzione di salto non comporta, quindi, alcun ritardo, dato che viene eseguita dall'unità di prelievo in parallelo con l'esecuzione di I_1 e I_2 . Questa tecnica è nota come *branch folding*.

Maggio 2002

Architettura degli elaboratori, Mod. B - 7. Pipeline

33

Branch folding (2/2)



Maggio 2002

Architettura degli elaboratori, Mod. B - 7. Pipeline

34

Un problema cruciale: i salti condizionati

Anche inserendo una coda delle istruzioni, **i salti condizionati incorrono, sempre in una penalità**, perché la condizione di salto spesso dipende dal risultato di un'istruzione precedente e, pertanto, l'eventuale prelievo delle istruzioni dall'indirizzo di destinazione non può essere attivato se l'esecuzione dell'istruzione non è stata completata.

Le istruzioni di salto sono molto frequenti.

Rappresentano circa il 20% del *conteggio dinamico delle istruzioni* della maggior parte dei programmi (il conteggio dinamico rappresenta il numero di istruzioni eseguite, tenendo conto del fatto che alcune istruzioni del programma vengono ripetute più volte a causa dei cicli).

L'elevata probabilità dei salti ed una pesante penalità temporale potrebbero vanificare completamente l'utilità della pipeline.

Fortunatamente, le istruzioni di salto possono essere gestite in molti modi per ridurre l'impatto negativo sulla frequenza di esecuzione delle istruzioni.

Intervallo di ritardo del salto

Abbiamo visto che in una struttura pipeline senza coda istruzioni, il processore inizia con il prelievo dell'istruzione I_{j+1} prima di determinare se l'istruzione corrente, I_j rappresenta un'istruzione di salto.

Quando l'esecuzione di I_j è stata completata e deve essere eseguito un salto, il processore deve scartare l'istruzione che era stata già prelevata e prelevarne un'altra a partire dall'indirizzo di destinazione del salto.

La locazione che segue un'istruzione di salto è chiamata *intervallo di ritardo del salto (branch delay slot)*.

Possono essere presenti più intervalli di ritardo del salto, in base al tempo necessario per eseguire un'istruzione di salto. Per esempio, se la fase di esecuzione di un'istruzione di salto occupa due passi della pipeline, il processore subirà due intervalli di ritardo. Le istruzioni negli intervalli di ritardo vengono comunque prelevate e parzialmente eseguite prima che sia presa la decisione relativa al salto e venga calcolato l'indirizzo di destinazione del salto.

La tecnica del *delayed branch*

Una tecnica, denominata *salto ritardato*, può minimizzare la penalizzazione dovuta alla presenza delle istruzioni di salto condizionato. L'idea è semplice: dato che le istruzioni durante gli intervalli di ritardo vengono comunque prelevate, è possibile prevedere che vengano sempre eseguite completamente, qualunque sia il risultato della valutazione della condizione di salto. Naturalmente il programmatore od il compilatore debbono strutturare il programma in modo che le istruzioni che seguono quella di salto siano istruzioni utili in ogni caso.

Quando la tecnica ha successo, l'operazione della pipeline non viene mai interrotta, e non si verificano cicli di attesa.

Concettualmente, il programma viene scritto anticipando la posizione dell'istruzione di salto rispetto al punto in cui si vuole sia eseguita.

In un ciclo, ad esempio, l'istruzione di salto condizionato può essere premessa all'operazione di aggiornamento della variabile indice, che va eseguita in ogni caso. Mentre l'istruzione spostata è in esecuzione, viene risolto il quesito sul salto e si può proseguire o saltare senza penalità.

Un'istruzione utilissima: NOP

Una macchina che adotti l'approccio del salto ritardato deve usare un compilatore in grado di inserire le istruzioni appropriate negli intervalli di ritardo del salto.

Se non è possibile eseguire nulla di utile in questi intervalli, è necessario inserire istruzioni NOP.

Si tenga presente che alcuni processori, come per esempio il 68.000, hanno in repertorio istruzioni NOP esplicite, mentre altri processori, come il PowerPC, non le hanno.

In questo caso, l'effetto di un'istruzione NOP può essere ottenuto mediante una istruzione che riscrive in un registro il contenuto precedente in modo da non modificare i contenuti di alcun registro del processore, eccetto il contatore di programma.

La sua esecuzione introduce solo un ritardo pari a un'istruzione nella pipeline di esecuzione delle istruzioni.

L'efficacia dell'approccio del *salto ritardato* dipende dal numero di volte in cui è effettivamente possibile riordinare opportunamente le istruzioni.

Limiti della tecnica

- Dati sperimentali ricavati da molti programmi indicano che tecniche di compilazione sofisticate **sfruttano un intervallo di ritardo del salto nell'85% dei casi.**
- Per una macchina con due intervalli di ritardo di salto, il compilatore tenta di identificare due istruzioni precedenti l'istruzione di salto che possano essere spostate negli intervalli di ritardo senza introdurre errori logici.
- **La probabilità di identificare due istruzioni con queste caratteristiche è considerevolmente inferiore rispetto alla possibilità di trovarne una sola.** Di conseguenza, se aumentare il numero di stadi della pipeline significa anche incrementare il numero di intervalli di ritardo del salto, si potrebbe non ottenere completamente l'incremento di prestazioni che ci si aspetta.

Predizione del salto

Si basa su valutazioni statistiche. Un'istruzione di salto che si trova alla **fine di un ciclo** provoca un trasferimento all'inizio del ciclo, ad ogni iterazione delle operazioni, eccetto l'ultima. Solo dopo l'ultima esecuzione del corpo del ciclo il salto non viene effettuato.

Quando durante l'esecuzione si incontra un'istruzione di salto, può essere vantaggioso per il processore ipotizzare che il salto verrà effettuato.

L'unità di prelievo delle istruzioni preleverà le istruzioni a partire dall'indirizzo di destinazione del salto e le caricherà nella coda delle istruzioni. Se, dopo aver valutato la condizione di salto, si verifica che in effetti il salto doveva essere effettuato, l'esecuzione può continuare normalmente e l'istruzione di salto non introduce alcun ritardo. Viceversa, se il salto non doveva essere eseguito, le istruzioni prelevate in precedenza devono essere scartate e sostituite con le istruzioni corrette.

Un ragionamento analogo può essere fatto anche per le istruzioni di salto posizionate all'inizio di un ciclo. **In questo caso, è più vantaggioso ipotizzare che il salto non venga effettuato.**

Il problema della dipendenza fra dati

In precedenza è stato introdotto il concetto di dipendenza tra i dati, che si verifica quando **un operando sorgente per un'istruzione dipende dal risultato dell'esecuzione di un'istruzione precedente**.

Se i risultati dell'esecuzione dell'istruzione precedente non sono ancora stati memorizzati nei rispettivi registri, la pipeline entra in stallo.

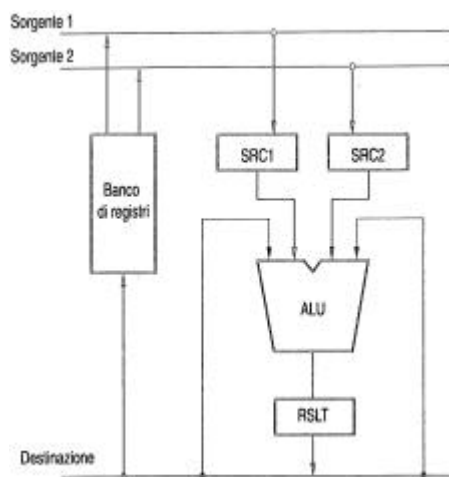
Si consideri un processore che utilizzi una pipeline a quattro stadi.

- Il primo stadio preleva un'istruzione dalla cache.
- Il secondo stadio decodifica l'istruzione e legge gli operandi sorgente dal banco di registri.
- Il terzo stadio esegue un'operazione della ALU
- Il quarto memorizza il risultato nella locazione di destinazione.

Si ipotizzi che alcune istruzioni del processore abbiano tre operandi: due operandi sorgente e un operando destinazione.

Un'organizzazione hardware che supporti queste caratteristiche è mostrata nella prossima trasparenza

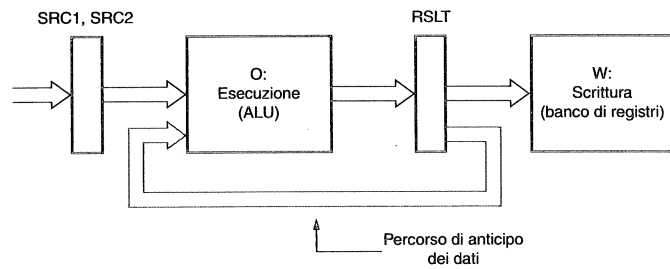
L'hardware operativo



Il banco di registri permette di accedere simultaneamente a tre diverse locazioni durante ogni ciclo di clock; due locazioni forniscono i due operandi sorgente, SRC1 e SRC2, che vengono trasferiti in parallelo ai registri in ingresso alla ALU. Contemporaneamente, i contenuti di RSLT, il registro risultato, sono memorizzati nella terza locazione del banco di registri. Quindi, al termine di ogni singolo ciclo di clock, vengono caricati in SRC1 e SRC2 due nuovi operandi, e un nuovo risultato generato dalla ALU viene memorizzato nel registro RSLT, per essere trasferito al banco di registri nel successivo ciclo di clock.

L'hardware per l'anticipo degli operandi

Lo schema mostra la posizione dei tre registri, SRC1, SRC2 e RSLT, nella pipeline. I tre registri fanno parte dei buffer interstadio, utilizzati per passare i dati da uno stadio al successivo durante le operazioni della pipeline.



Supponiamo, ora, si debba eseguire il seguente segmento di programma:

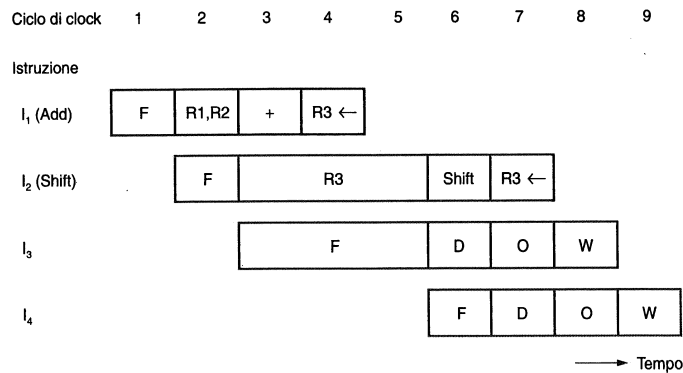
I₁: ADD R1,R2,R3
I₂: SHIFT_LEFT R3

Maggio 2002

Architettura degli elaboratori, Mod. B - 7. Pipeline

43

Temporizzazione normale delle operazioni



I contenuti di R1 ed R2 sono caricati nei registri SRC1 e SRC2. La loro somma viene generata dalla ALU e caricata nel registro di uscita della ALU, durante il ciclo di clock 3. Da qui, viene trasferita nel registro R3 del banco di registri durante il ciclo di clock 4.

Maggio 2002

Architettura degli elaboratori, Mod. B - 7. Pipeline

44

E' possibile evitare lo stallo da dipendenza?

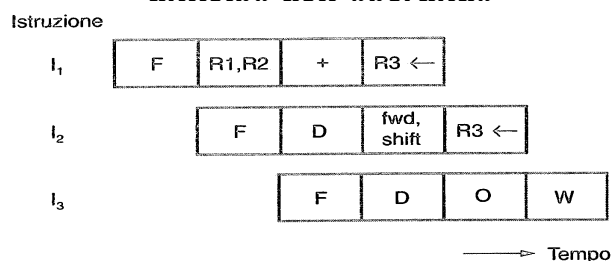
La struttura hardware del processore rileva che l'operando sorgente di questa istruzione è l'operando destinazione dell'istruzione precedente, e deve assicurare che venga utilizzato il valore aggiornato. Di conseguenza, lo stadio di decodifica, responsabile del prelievo dell'operando sorgente da R3, non può completare tale compito fino al ciclo di clock 5; quindi la pipeline rimane in stallo per due cicli di clock.

Ovviamente la condizione di stallo dell'unità di prelievo provoca quella delle altre unità.

Quella delineata è l'evoluzione di una normale pipeline a quattro stadi, quando due istruzioni consecutive, operano sul contenuto di uno stesso registro.

E' possibile evitare lo stallo della pipeline?

Temporizzazione della pipeline con *anticipo dell'operando*



Dopo la decodifica dell'istruzione I_2 , i circuiti di controllo rilevano che l'operando sorgente di I_2 è l'operando destinazione di I_1 . Quindi, la lettura di R3 viene inibita. Nel successivo ciclo di clock, i circuiti di controllo hardware dello stadio di elaborazione *Operate* fanno in modo che l'operando sorgente provenga direttamente dal registro RSLT, come indicato dal termine "fwd" nella Figura.

Di conseguenza, l'esecuzione di I_2 procede senza interruzioni.