Altre tecniche per il miglioramento delle prestazioni

Aniello Minutolo

Predizione dei salti

Sommario

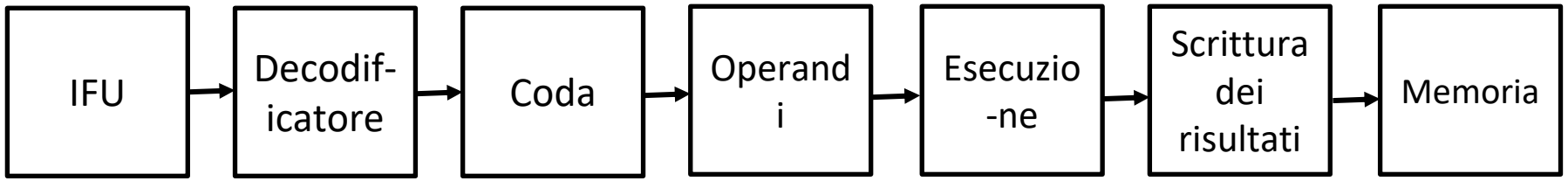
1. Predizione dei salti
2. Esecuzione fuori sequenza
3. Esecuzione speculativa

Pipeline nei calcolatori moderni

I calcolatori moderni sono organizzati a **pipeline** composte da 7 (come la pipeline di esempio riportata in figura), 10 o più stadi.

L'uso delle pipeline funziona in modo ottimale su codice lineare:

- l'unità di prelievo può semplicemente leggere dalla memoria parole consecutive e spedirle all'unità di decodifica prima ancora che siano effettivamente richieste.



Problemi con le istruzioni di salto

I programmi reali però non sono sequenze lineari di codice:

- sono in genere pieni d'istruzioni di salto che interrompono il flusso lineare del codice, creando difficoltà per le **pipeline**
- risulta spesso difficile prelevare le istruzioni a una velocità elevata in modo da alimentare la pipeline.

Consideriamo ad esempio la traduzione in linguaggio assemblativo delle semplici istruzioni seguenti

- la variabile i viene confrontata con 0 e, a seconda del risultato, si assegna un valore diverso alla variabile k

```
if (i == 0)
        k = 1;
else
        k = 2;
```

```
        CMP i,0     ; confronta i con 0
        BNE Else    ; salta a Else se diversi
Then:   MOV k,1     ; sposta 1 in k
        BR  Next    ; salta a Next
Else:   MOV k,2     ; sposta 2 in k
Next:
```

Problemi con le istruzioni di salto

BNE è un salto condizionale, ovvero un salto effettuato se e soltanto se è soddisfatta una particolare condizione

- in questo caso, il fatto che i due operandi della precedente istruzione, CMP, siano diversi

I salti incondizionati, come **BR Next**, effettuano un salto senza alcuna ambiguità sulla destinazione da raggiungere.

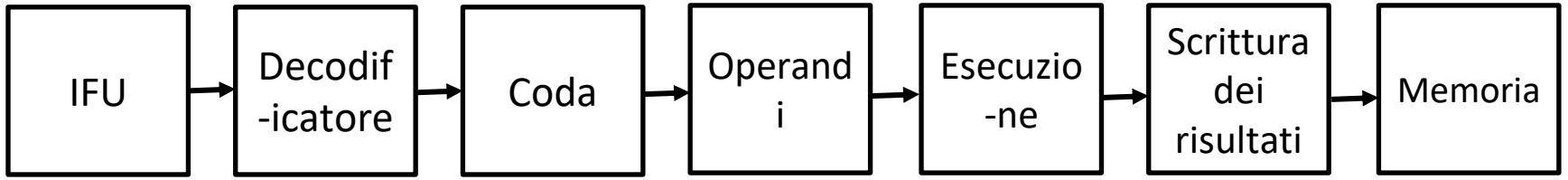
Sia i salti condizionali che que incondizionati causano ritardi nella pipeline a causa della natura stessa della pipeline.

```
        CMP i,0      ; confronta i con 0
        BNE Else     ; salta a Else se diversi
Then:   MOV k,1      ; sposta 1 in k
        BR  Next     ; salta a Next
Else:   MOV k,2      ; sposta 2 in k
Next:
```

Ritardi nella pipeline

Nella pipeline di esempio, la decodifica dell'istruzione si svolge nel secondo stadio, l'unità di prelievo deve quindi decidere dove prelevare le istruzioni successive ancor prima di conoscerne il tipo:

- i **salti incondizionati** vengono rilevati solo dopo un ciclo, quando l'unità ha già iniziato a prelevare l'istruzione successiva;
- i **salti condizionali** introducono ritardi ancora maggiori perché l'indirizzo di destinazione è noto solo in fasi avanzate della pipeline.



Predizione dei salti condizionali

Le prime macchine a pipeline rimanevano in **stallo** finché non venivano a sapere se occorresse o meno effettuare la diramazione

La maggior parte delle macchine moderne **predicono** se un salto condizionale verrà o meno effettuato per evitare stalli nella pipeline

Una semplice strategia di predizione

- i salti all'indietro vanno sempre effettuati, in quanto si assume facciano parte di un ciclo e quindi saranno eseguiti più volte;
- non tutti i salti in avanti vanno effettuati, in quanto associati a condizioni di errore e quindi ritenuti raramente eseguiti.

Conseguenze delle predizioni errate

Se una predizione di salto è corretta, l'esecuzione prosegue senza interruzioni dall'indirizzo di destinazione.

Le predizioni errate richiedono invece l'annullamento delle istruzioni già eseguite che non avrebbero dovuto esserlo.

Due metodi per gestire predizioni errate:

- salvare in registri temporanei i risultati che alterano lo stato della macchina, e copiare i valori nei registri reali solo dopo aver verificato che le predizioni di salto fossero corrette;
- salvataggio dello stato dei registri prima della modifica del loro valore, in modo da poter ripristinare lo stato della macchina in caso di predizioni di salto errate.

Complessità nella gestione delle predizioni

- La gestione delle predizioni di salto errate richiede in genere il tracciamento di grandi quantità di informazioni.
- Situazioni complesse sorgono quando un secondo salto condizionale viene incontrato prima di verificare la correttezza del primo.
- L'hardware deve gestire dipendenze e conflitti per garantire che lo stato della macchina sia coerente.

Esecuzione fuori sequenza

Architettura superscalare

La maggior parte delle CPU moderne funziona sia a pipeline sia in modo superscalare:

- ciò comporta la presenza di un'unità di fetch che preleva istruzioni dalla memoria prima che siano necessarie, in modo da alimentare un'unità di decodifica.

L'unità di decodifica attribuisce l'istruzione decodificata all'unità funzionale appropriata perché sia eseguita:

- in alcuni casi l'unità di decodifica può scomporre le singole istruzioni in micro-operazioni prima di distribuirle alle unità funzionali, secondo le capacità delle unità stesse.

Esecuzione in ordine e dipendenze

L'esecuzione in ordine garantisce risultati corretti ma può limitare le prestazioni a causa delle dipendenze tra istruzioni

Le dipendenze **RAW** (Read After Write) **bloccano** l'esecuzione finché un'istruzione precedente non produce un risultato:

- le istruzioni che richiedono un valore calcolato da un'istruzione precedente, **devono aspettare e non possono essere eseguite.**

Le dipendenze **WAR** (Write After Read) e **WAW** (Write After Write) sono meno gravi e **possono essere evitate** richiedendo che la seconda istruzione salvi il suo risultato da qualche altra parte (magari temporaneamente):

- in una dipendenza di tipo **WAR** (**WAW**) un'istruzione cerca di sovrascrivere un registro che un'istruzione precedente potrebbe non aver terminato di leggere (scrivere).

Esecuzione fuori sequenza

Nel tentativo di aggirare questi problemi e ottenere prestazioni migliori, alcune CPU utilizzano l'**esecuzione fuori sequenza** per saltare le istruzioni dipendenti ed eseguire le istruzioni indipendenti successive.

L'**esecuzione fuori sequenza** migliora le prestazioni ma deve garantire risultati identici all'esecuzione in sequenza

- usa uno strumento chiamato **scoreboard**, utilizzato per la prima volta nel CDC 6600, per decidere quali istruzioni possono essere lanciate.

Scoreboard e gestione delle dipendenze

Lo **scoreboard** utilizza dei contatori per monitorare ogni registro e tenere traccia delle istruzioni attualmente in esecuzione che richiedono quel particolare registro come sorgente di un operando

- monitora anche la disponibilità delle unità funzionali per evitare conflitti.

Scoreboard e gestione delle dipendenze

Le regole dello **scoreboard** impediscono il lancio di istruzioni in caso di dipendenze **RAW**, **WAR** o **WAW**

1. se un operando è in fase di scrittura, non lanciare (dipendenza RAW);
2. se il registro del risultato è in lettura, non lanciare (dipendenza WAR);
3. se il registro del risultato è in scrittura, non lanciare (dipendenza WAW).

Un'istruzione può essere lanciata, se non si verifica nessuna delle tre dipendenze e l'unità funzionale di cui ha bisogno è disponibile.

Interrupt precisi e imprecisi

- Un **interrupt preciso** permette di salvare lo stato della macchina in modo coerente, essenziale per il debugging.
- L'esecuzione fuori sequenza può rendere gli **interrupt imprecisi**, complicando il ripristino dello stato della macchina.
- Il ritiro delle istruzioni fuori sequenza rende gli interrupt imprecisi, ed è questo il motivo per cui alcune macchine completano le istruzioni in ordine.

Esecuzione speculativa

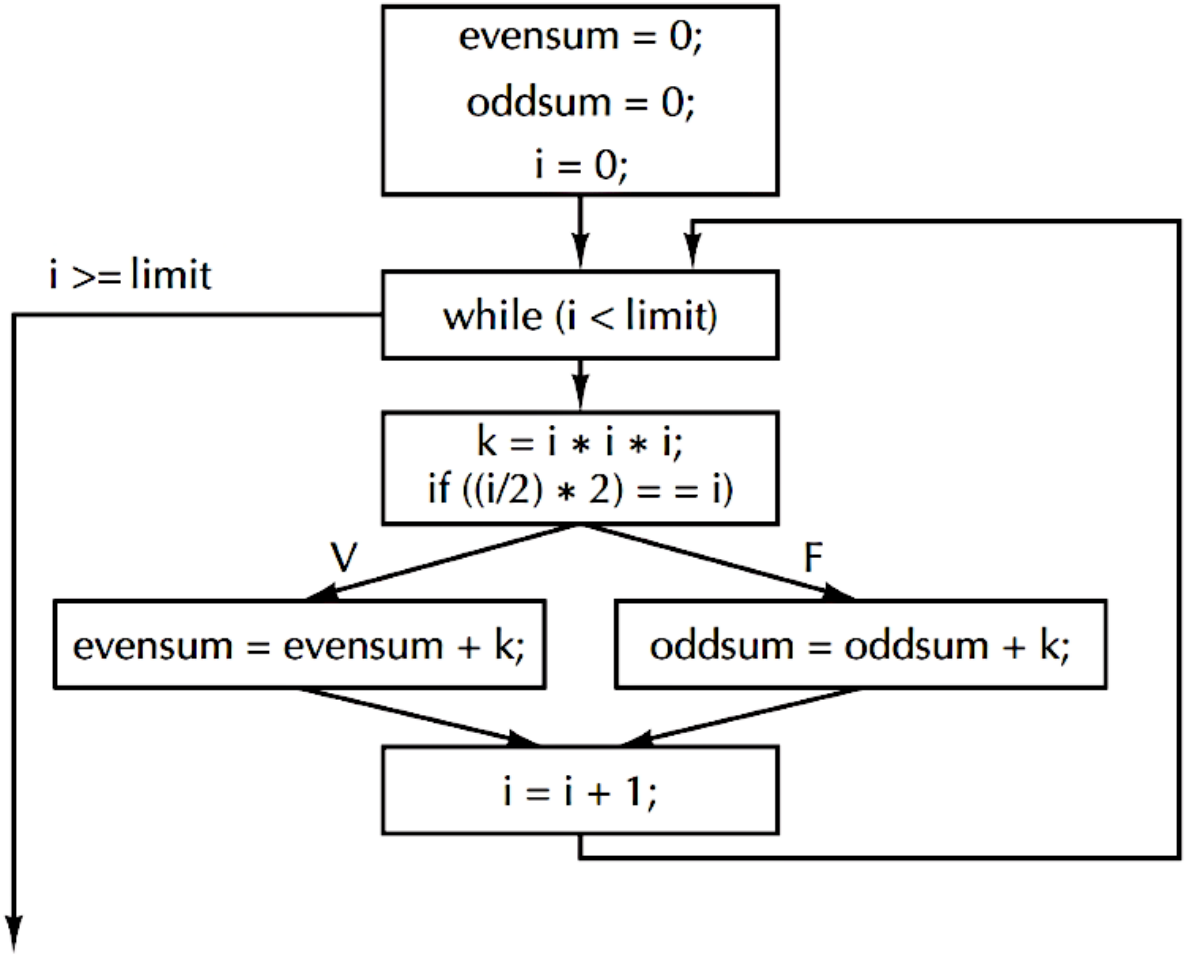
Blocchi elementari e riordinamento

I programmi sono suddivisi in **blocchi elementari**, sequenze lineari senza diramazioni interne (come i costrutti if o while) con un punto d'ingresso all'inizio e un punto di uscita alla fine:

- la traduzione di un blocco elementare in linguaggio macchina non presenta nessuna diramazione;
- il collegamento fra diversi blocchi elementari è costituito dalle strutture di controllo.

Blocchi elementari e riordinamento

Un programma può essere rappresentato attraverso un grafo orientato per mettere in evidenza i collegamenti tra blocchi elementari attraverso strutture di controllo.



Blocchi elementari e riordinamento

Il riordinamento delle istruzioni funziona bene all'interno di un blocco elementare ma è limitato dalla loro breve lunghezza

- in quanto non vi è in genere al loro interno sufficiente parallelismo da poter sfruttare in modo efficace.

Il passo successivo per cercare di utilizzare tutti i cicli disponibili consiste nel permettere al riordinamento di superare i limiti che separano i blocchi elementari.

Blocchi elementari e riordinamento

Il guadagno maggiore si ottiene quando un'istruzione potenzialmente lenta può essere spostata più in alto nel grafo di modo che la sua esecuzione possa iniziare in modo anticipato

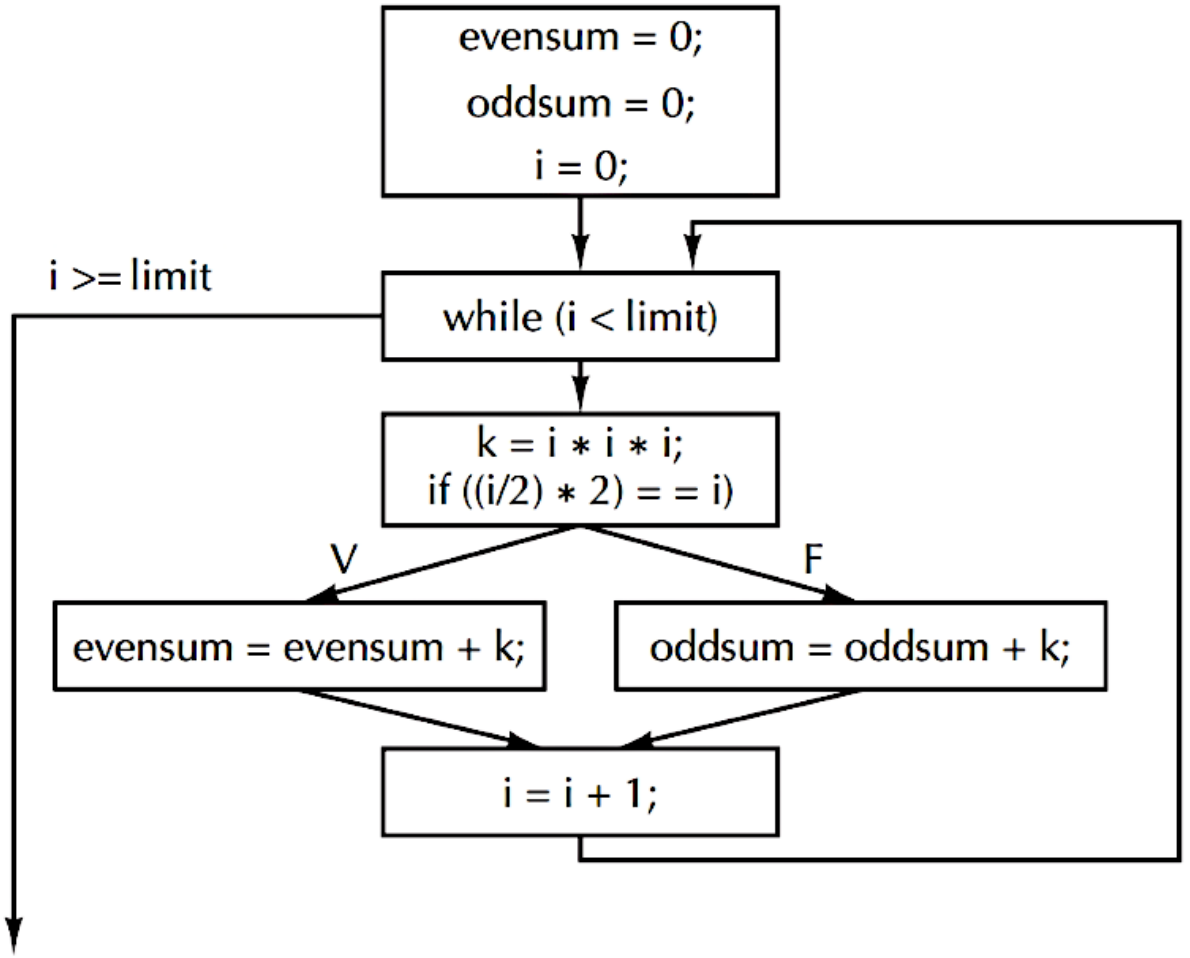
- questa tecnica è utile per istruzioni lente come **LOAD**, per operazioni in virgola mobile, oppure per l'inizio di una lunga catena di dipendenze.

La tecnica di anticipare l'esecuzione del codice prima di una diramazione è nota come **slittamento**.

Esecuzione spaeculativa

Ad esempio, potrebbe aver senso spostare le istruzioni di LOAD di *evensum* e *oddsum* prima della diramazione dell'istruzione IF

- per farle cominciare anticipatamente di modo che i loro valori siano già disponibili nel momento in cui vengano richiesti.



Esecuzione spaeculativa

Ovviamente, durante ogni iterazione del ciclo di esempio, solo uno dei valori di *evensum* e *oddsum* sarà effettivamente necessario, mentre l'altra operazione di LOAD verrà sprecata:

- se però sia la cache sia la memoria sono strutturate a pipeline e se vi sono cicli disponibili, potrebbe valerne la pena.

L'esecuzione di parti del codice prima ancora di sapere se saranno effettivamente richieste è chiamata **esecuzione speculativa:**

- per poter usare questa tecnica sono necessari sia il supporto da parte del compilatore e dell'hardware sia alcune estensioni dell'architettura;
- è compito del compilatore spostare esplicitamente le istruzioni, dato che un loro riordinamento che scavalchi i limiti dei blocchi elementari è generalmente al di là delle possibilità dell'hardware.

Problemi dell'esecuzione speculativa

Le istruzioni speculative non devono produrre risultati irrevocabili fino a quando non sono confermate, dato che in un secondo momento potrebbe risultare che non dovevano essere eseguite.

Problemi dell'esecuzione speculativa

Un modo comune per evitare scritture irrevocabili nei registri, consiste nella **rinomina dei registri** di destinazione utilizzati dal codice speculativo:

- in questo modo vengono modificati solo i registri di lavoro, così che non sorgano problemi se in ultima istanza il codice risulta non richiesto;
- se al contrario il codice è necessario, allora i registri di lavoro vengono copiati nei registri di destinazione corretti.

Lo **scoreboard** diventa complesso da gestire con l'esecuzione speculativa a causa dei registri temporanei, ma può tuttavia essere realizzato disponendo di hardware sufficiente.

Fallimenti nella cache

Un **fallimento nella cache** durante un'istruzione **LOAD** speculativa può rallentare la CPU se il dato non è necessario:

- porre la macchina in attesa per vari cicli durante il fetch di una parola che successivamente non risulterà necessaria è controproducente.

Queste “ottimizzazioni” potrebbero rendere la CPU più lenta di quanto non lo sarebbe se non si facesse ricorso alle stesse.

Istruzioni speculative di LOAD

Una soluzione utilizzata in alcune macchine moderne consiste nell'avere una speciale istruzione speculativa di LOAD, **SPECULATIVE-LOAD**, che cerchi di prelevare la parola dalla cache, ma che non faccia altro se non la trova:

- se, nel momento in cui viene richiesto, il valore è presente, allora può essere utilizzato;
- altrimenti l'hardware deve andare a recuperarlo sul momento.

In questo modo il fallimento della cache non produce alcuna penalità se alla fine il valore risulta essere non necessario.

Bibliografia

Libro di testo

- Andrew S. Tanenbaum e Todd Austin. Architettura dei calcolatori. Un approccio strutturale. 6/ED. Anno 2013. Pearson Italia spa. (Disponibile nella sezione “Biblioteca”)

Fonte argomenti e immagini

- Capitolo 4, Paragrafi da 4.5.2 a 4.5.4, pp. 321-334