Filosofia RISC

Tra le principali innovazioni dei computer, esiste l'architettura per processori RISC (*Reduced Instruction Set Computer*), che è un tipo di architettura di microprocessore che utilizza un piccolo insieme di istruzioni altamente ottimizzate, anzichè un insieme di istruzioni altamente specializzate come quelle tipiche di altre architetture. Il RISC è un'alternativa all'architettura CISC (*Complex Instruction Set Computer*) ed è spesso considerato la tecnologia di architettura CPU più efficiente disponibile.

Una caratteristica fondamentale del RISC è che consente agli sviluppatori di aumentare il set di registri e di incrementare il parallelismo interno, aumentando il numero di thread paralleli eseguiti dalla CPU e aumentando la velocità di esecuzione delle istruzioni della CPU. Il RISC è il risultato dell'evoluzione tra hardware e i linguaggi di programmazione, arrivando a linguaggi di alto livello, che permettono di esprimere algoritmi in modo più sintetico, lasciando al compilatore il compito di gestire i dettagli. Esiste però una differenza semantica tra liguaggi HLL (*High Level Languages*) e il linguaggio macchina, portando ad un'esecuzione inefficiente.

In progettisti hardware hanno aumentato il set di istruzioni, creato vari modi di indirizzamento e cercato un'implementazione hardware di costrutti di alto livello. In questo modo:

- Viene semplificato il lavoro del compilatore
- Si migliora l'efficienza dell'esecuzione
- Si supportano HLL più complessi

Inoltre, si studiano le caratterestiche e i pattern dei linguaggi ad alto livello per semplificare:

- Le funzionalità del processore e la sua interazione con la memoria
- Il tipo e la frequenza d'uso degli operandi che determinano l'organizzazione della memoria e i modi di indirizzamento
- L'organizzazione della pipeline e il suo controllo

Per far ciò occorre fare un'analisi delle istruzioni macchine generate dalla compilazione di programmi in linguaggi HL. Svolgere misure dinamiche raccolte tramite l'esecuzione del programma e contando il numero di occorrenze di una certa proprietà o di una certa caratteristica.

Nei linguaggi prevalgono le istruzioni di assegnazione e le istruzioni condizionionali. Si deve inoltre comprendere quali sono le istruzioni che rubano più tempo: normalmente si tratta dei cicli, chiamate e condizioni.
La frequenza di istruzioni ad alto livello dipende da:

- Linguaggio HL
- Tipo di applicazione
- Architettura sottostante

Operandi

Per gli **operandi**, costituiti principalmente da variabili scalari locali, l'ottimizzazione si concetra sull'accesso alle variabili scalari locali.

Chiamate di procedura

Le **chiamate di procedura** sono le istruzioni che consumano più tempo, è quindi necessario trovare un'implementazione efficiente. Si considerano due aspetti:

- Il numero di parametri e variabili gestite
- Il livello di annidamento

Le misure da prendere sono:

- meno di 6 parametri, meno di 6 variabili locali
- la maggior parte dei parametri sono variabili locali
- poco annidamento di chiamate di procedure

Le migliori strategie per supportare i linguaggi di alto livello sono: **non** rendere le istruzioni macchina simili a quelle dei HLL, ottimizzare le performance dei pattern più usati e che consumano più tempo. Tramite:

1. Ampio numero di registri oppure ottimizzato loro utilizzo dal compilatore
    - per ottimizzare gli accessi agli operandi (come abbiamo visto sono le istruzioni più comuni)
2. Progettazione accurata della pipeline
    - gestione delle dipendenze dal controllo dovute a salti e chiamate evitando prefetch errati
3. Set di istruzioni semplificato (ridotto) e implementato in maniera efficiente

Uso dei registri

Per i **registri**, che hanno indirizzi più brevi rispetto alla cache e alla memoria principale, bisogna assicurare che gli operandi usati siano il più possibile all'interno dei registri. In questo modo si minimizzano i trasferimenti memoria-registro, ci sono due soluzioni:

- **Hardware**: si tratta di aumentare il numeri di registri, in questo modo si mantengono più variabili per più tempo
- **Software**: in cui il compilatore massimizza l'utilizzo dei registri, inoltre le variabili più utilizzate in un intervallo di tempo sono memorizzate nei registri. Sono richieste sofisticate tecniche di analisi dei programmi

Generalmente, si cerca di memorizzare nei registri le variabili scalari locali e si hanno pochi registri per le variabili locali.

Idea per usare al meglio i registri *general-purpose*:

- Suddividere i registri in molti piccoli gruppi
- Ogni procedura ha il proprio gruppo di registri
- È sempre indirizzabile un solo gruppo, ossia quello corrispondente alla procedura attiva

- Chiamata di procedura
  - cambia automaticamente il numero di registri da usare
  - invece di provocare il salvataggio dei dati in memoria
  - al ritorno si seleziona il gruppo di registri assegnato in precedenza alla procedura chiamante
  - le finestre relative a procedure adiacenti sono parzialmente sovrapposte, in modo da facilitare il passaggio dei parametri

Finestre di registri

Ogni gruppo/finestra di registri è diviso in tre sottogruppi

- *Parameter registers*: contengono i parametri passati all'invocazione della procedura e il valore da restituire alle chiamate
- *Local registers*: memorizzano il contenuto delle variabili locali
- *Temporary registers*: usati per scambiare parametri e valore di ritorno con l'eventuale procedura chiamata (nested)

Quante finestre di registri ci sono?

- una per chiamata di procedura attivata
- c'è lo spazio per un numero limitato (solo le più recenti)
- le attivazioni precedenti vanno salvate in memoria e poi recuperate quando diminuisce il nesting

I registri sono organizzati a buffer **circolare**.

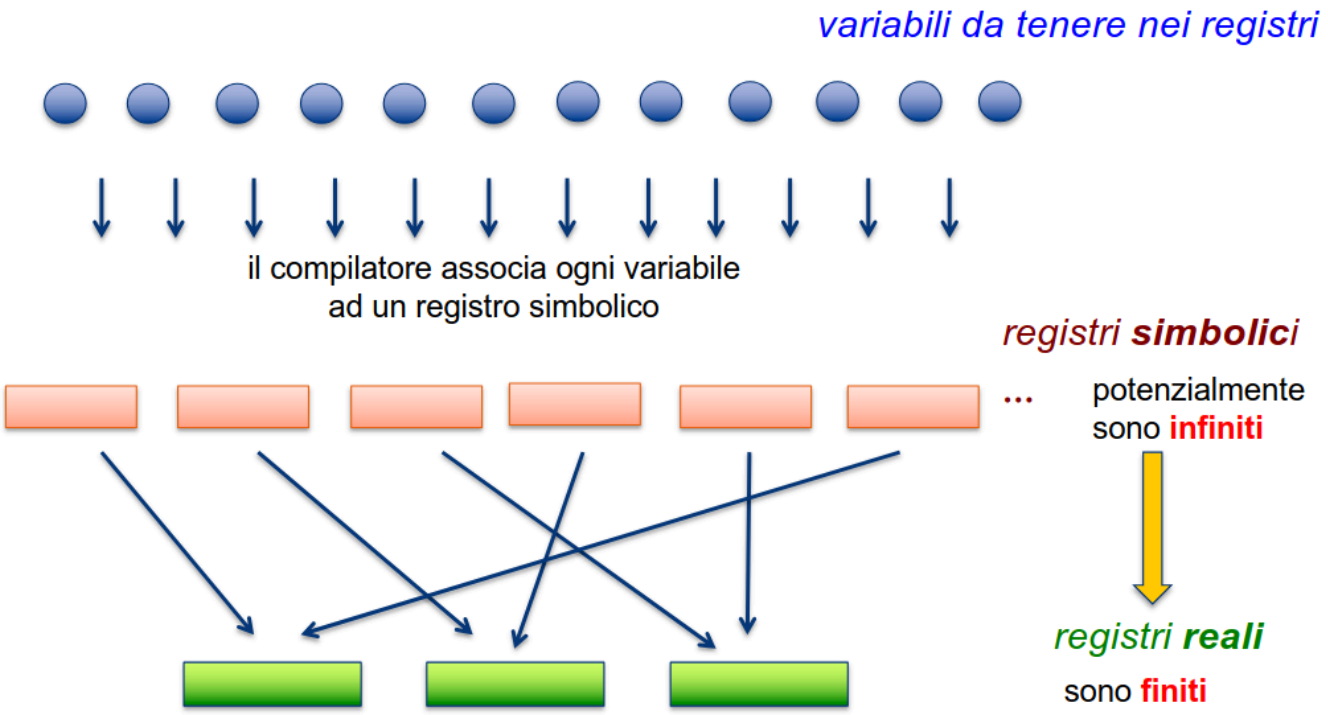
Variabili globali

Sono variabili accessibili da qualunque procedura. Vengono memorizzate in:

- Il compilatore le alloca in memoria. metodo poco efficiente se usato spesso
- gruppo di registri ad hoc, disponibili a tutte le procedure. Lo scopo è di trovare gli operandi il più possibile nei registri, diminuendo le operazioni di load/store. L'architettura RISC può avere pochi

registri (16-32) il cui uso viene ottimizzato dal compilatore

Continuando con l'ottimizzazione dei registri, bisogna decidere quale registro simbolico assegnare a quale registro reale. Si hanno m compiti da eseguire, n risorse, con m >> n.



- il compilatore mappa ogni registro simbolico ad un registro reale
- se due registri simbolici si usano in momenti diversi, possono essere mappati sullo stesso registro reale
- se in un certo intervallo di tempo i registri reali non sono in numero sufficiente per contenere tutte le variabili riferite in quell'intervallo, alcune variabili vengono mantenute nella memoria principale

CISC

È caratterizzato da un ampio set di istruzioni e più complesse. In questo modo si semplifica il compilatore che deve generare sequenze di istruzioni brevi e semplici da eseguire, migliorando le performance.
Un'istruzione complessa può essere eseguita più velocemente di tante semplici, ma:

- Richiesta CU più complessa
- Il controllo microprogrammato richiede più spazio
- Si rallenta l'esecuzione delle istruzioni più semplici, che sono comunque le più frequenti

Caratteristiche di CISC

1. Un'istruzione per ciclo di clock
    - *instruction cycle*: tempo impiegato per fare fetch-decode-execute-wrtie di un'istruzione base
    - Istruzioni CISC richiedono più di un ciclo

    - RISC: hanno un ciclo esecutivo che dura un solo machine cycle, quindi se la pipeline è piena, ad ogni ciclo termina un'istruzione
2. Operazioni da registro a registro, tranne *load* e *store*
    - CISC attuali hanno anche operazioni *memory-memory* e *register-memory*
    - considerando il frequente uso di scalari locali, aumentando o ottimizzando i registri la maggior parte degli operandi sta a lungo nei registri
3. Pochi e semplici modi di indirizzamento
    - indirizzo di registro, spiazzamento (in base al PC)
    - si semplifica l'istruzione e la CU
4. Pochi e semplici formati fissi per le istruzioni
    - campi e opcode a dimensione fissa, in questo modo la decodifica dell'opcode e l'accesso ai registri operandi può avvenire in contemporanea
    - istruzioni a lunghezza fissa sono allineate con la lunghezza delle parole, quindi il fetch è ottimizzato per prelevare una parola e multipli
    - la regolarità facilita le ottimizzazioni del compilatore
    - più responsivo agli interrupt, controllati tra due istruzioni più semplici
5. Unità di controllo cablata
    - se cablata (hardware) è meno flessibile ma più veloce
    - se microprogrammata è più flessibile ma meno veoloce