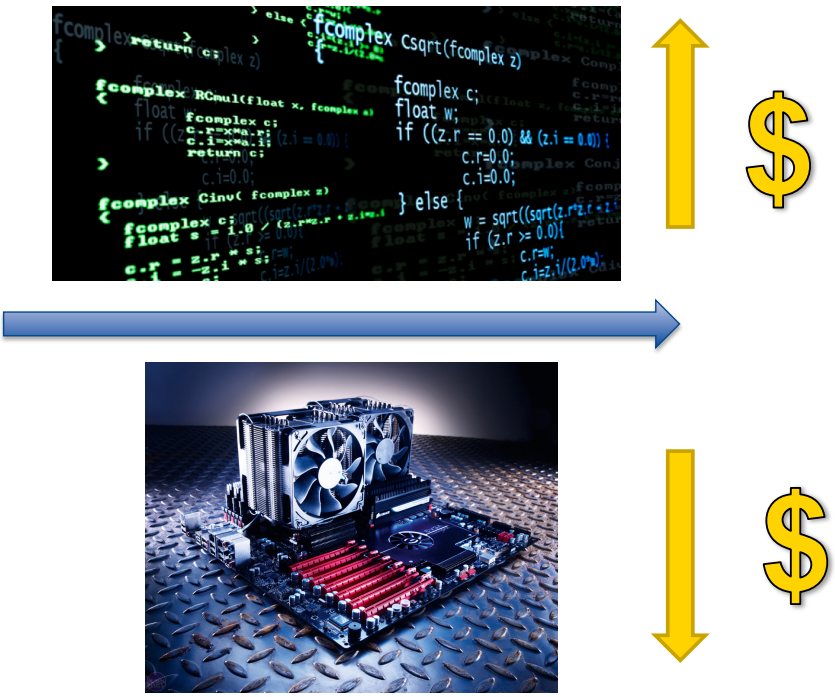
Architetture RISC

Reduced Instrucion Set Computer

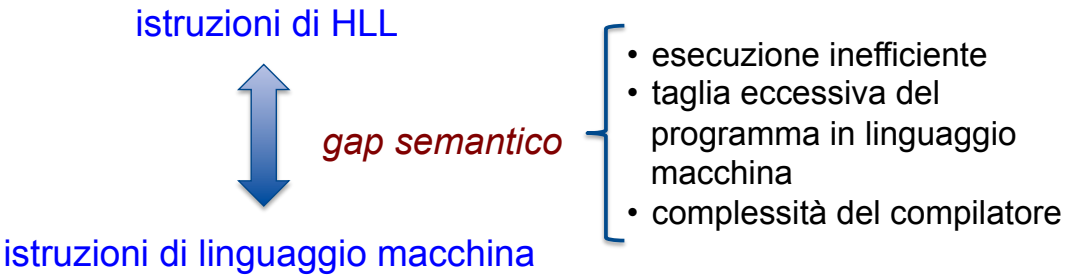
Evoluzione

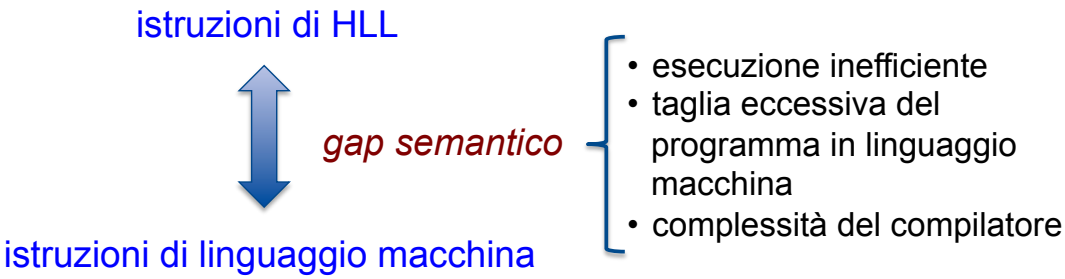
- Co-evoluzione tra hardware e linguaggi di programmazione



Evoluzione

- **High-level languages**
  - permettono di esprimere l'algoritmo risolutivo in modo più conciso: ***cosa***
  - lasciano al compilatore il compito di gestire i dettagli : ***come***
  - supportano costrutti di programmazione strutturata: paradigmi *imperativo, funzionale, logico, object-oriented*

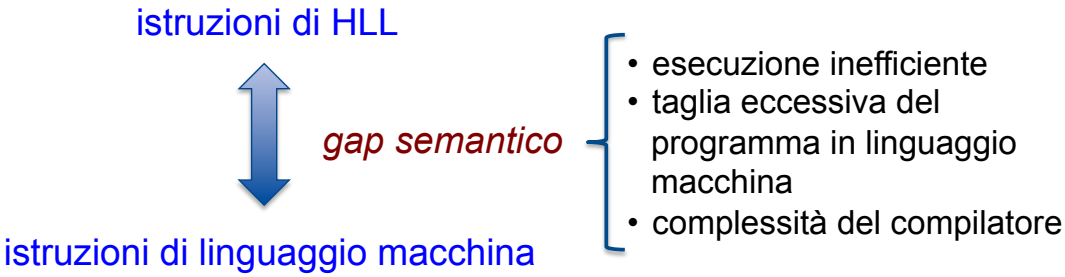




- Risposta dei progettisti hardware:
  - set di istruzioni più ampio
  - svariati modi di indirizzamento
  - implementazione hardware di costrutti di linguaggi ad alto livello (es. CASE (switch) su architettura VAX)

✓ si semplifica il lavoro del compilatore

✓ migliora l'efficienza dell'esecuzione (sequenze di operazioni complesse implementate tramite microcodice)



Alternativa:

- individuare le caratteristiche e i **pattern di esecuzione** delle istruzioni macchina generate dai programmi in HHL
- per **semplificare** l'architettura sottostante ad HHL, non complicarla

Semplificare, ***cosa?***

- operazioni eseguite
  - semplificare le *funzionalità del processore* e la sua *interazione con la memoria*
- operandi
  - tipo e frequenza d'uso degli operandi determinano *l'organizzazione della memoria* e i *modi di indirizzamento*
- serializzazione dell'esecuzione
  - organizzazione della *pipeline* e del controllo

come?

- fare un' **analisi** delle istruzioni macchina generate dai programmi scritti in HLL
- misure **dinamiche**: raccolte eseguendo il programma e contando il *numero di occorrenze* di una certa proprietà o di una certa caratteristica. (le misure **statiche** si basano solo sul programma sorgente, che non dice quante volte è eseguita un'istruzione)

Operazioni

- predominanza di istruzioni di **assegnamento**
  - quindi il trasferimento dei dati deve essere efficiente
- molte istruzioni **condizionali** (IF, LOOP)
  - quindi il controllo delle dipendenze dai salti deve essere efficiente

- oltre a ***frequenza*** di istruzioni, quali istruzioni richiedono più ***tempo di esecuzione***?
  - quali istruzioni del HLL causano l'esecuzione della maggior parte delle istruzioni macchina, e in quanto tempo?

Frequenza relativa di istruzioni ad alto livello

[PATT82a]

	Occorrenza Dinamica	
	Pascal	C
ASSIGN	**45%**	**38%**
LOOP	5%	3%
CALL	15%	12%
IF	**29%**	**43%**
GOTO	—	3%
OTHER	6%	1%

Frequenza relativa di istruzioni ad alto livello

[PATT82a]

	Occorrenza Dinamica		Occorrenza ponderata sulle istruzioni	
	Pascal	C	Pascal	C
ASSIGN	45%	38%	13%	13%
LOOP	5%	3%	**42%**	**32%**
CALL	15%	12%	**31%**	**33%**
IF	29%	43%	11%	21%
GOTO	—	3%	—	—
OTHER	6%	1%	3%	1%

moltiplicato **per il numero di istruzioni macchina prodotte** dal compilatore (normalizato)

Frequenza relativa di istruzioni ad alto livello

[PATT82a]

	Occorrenza Dinamica		Occorrenza ponderata sulle istruzioni		Occorrenza ponderata sugli accessi a memoria	
	Pascal	C	Pascal	C	Pascal	C
ASSIGN	45%	38%	13%	13%	14%	15%
LOOP	5%	3%	42%	32%	33%	26%
CALL	15%	12%	31%	33%	**44%**	**45%**
IF	29%	43%	11%	21%	7%	13%
GOTO	—	3%	—	—	—	—
OTHER	6%	1%	3%	1%	2%	1%

Frequenza relativa di istruzioni ad alto livello

[PATT82a]

dipende da

- quale linguaggio HL
- quale tipo di applicazione
- quale architettura sottostante
- resta rappresentativa delle contemporanee architetture **CISC (Complex Instruction Set Computer)**

CALL	15%	12%	31%	33%	**44%**	**45%**
IF	29%	43%	11%	21%	7%	13%
GOTO	—	3%	—	—	—	—
OTHER	6%	1%	3%	1%	2%	1%

Operandi

- Principalmente variabili scalari *locali*
- L'ottimizzazione si deve concentrare sull'accesso alle variabili locali

	Pascal	**C**	**Media**
Costanti Intere	16%	23%	20%
Variabili scalari	58%	53%	***55%***
Array/ Strutture	26%	24%	25%

Chiamate di procedura

- sono le istruzioni la cui esecuzione consuma più tempo, va quindi trovata un'implementazione efficiente

- due aspetti significativi:
  - il numero di parametri e variabili gestite
  - il livello di annidamento (*nesting*)

- misurazioni:
  - meno di 6 parametri, meno di 6 variabili locali
  - la maggior parte degli operandi sono variabili locali
  - poco annidamento di chiamate di procedure

Implicazioni dell'analisi

Strategia migliore per supportare i linguaggi di alto livello:

- **non** rendere le istruzioni macchina più simili alle istruzioni di HLL
- ottimizzare le performance dei pattern più usati e più time-consuming

1. ampio numero di **registri** o loro uso ottimizzato dal compilatore
   - per ottimizzare gli accessi agli operandi (abbiamo visto che sono istruzioni molto frequenti, con operandi perlopiù ***scalari e locali***, quindi è utile ridurre gli accessi alla memoria aumentando gli accessi ai registri)
2. progettazione accurata della **pipeline**
   - gestione delle dipendenze dal controllo dovute a salti e chiamate di procedure evitando i prefetch errati
3. set di istruzioni **semplificato (ridotto)** e implementato in maniera efficiente.

architetture RISC