Appunti di Architetture degli elaboratori
A.A. 2022/2023

1 Fondamenti

1.1 Calcolatore

Un computer digitale é una macchina in grado di risolvere problemi eseguendo istruzioni appositamente specificate.

In un calcolatore possono quindi essere individuati:

- *hardware*: componenti **fisiche** del calcolatore (es. circuiti integrati, periferiche, ecc.)
- *sofware*: insieme di **istruzioni** e **informazioni** necessarie al sistema per risolvere i problemi che gli vengono forniti.
- *firmware*: software integrato direttamente in un dispositivo, necessario per avviare il componente stesso e farlo interagire con altri componenti (es. scheda di rete))

Un *programma* é una sequenza di istruzioni scritte in un linguaggio direttamente comprensibile da un computer.

1.2 Architettura a livelli

L'insieme delle istruzioni eseguite direttamente dall'hardware di un calcolatore é detto **linguaggio macchina**, é binario ed é detto L_0. Con linguaggio macchina si puó (erroneamente) indicare il linguaggio **assembly**, un linguaggio, sempre di basso livello (ad esempio si interagisce direttamente con i registri), costituito da un ristretto insieme di istruzioni (es. somma due numeri, invio di un segnale ecc.) piú comprensibili a un umano. L'assembly é detto linguaggio L_1 perché é il linguaggio immediatamente sopra a quello binario direttamente comprensibile al computer.
Si puó estendere questo concetto dell'architettura a livelli arrivando a n linguaggi: per far eseguire al computer le istruzioni in linguaggio L_n, é necessaria la presenza di un *traduttore* che traduce le istruzioni da L_n a L_0 o direttamente (piú complesso), o passando per i livelli intermedi. Generalmente, piú un linguaggio é di alto livello, piú facilmente é comprensibile ad un umano.
Il traduttore é detto:

- **compilatore**: legge il programma in L_i e produce un programma in un linguaggio inferiore. Se L_i é L_1, il compilatore é detto **disassemblatore** (generalmente c'é una corrispondenza 1:1)
- **interprete**: legge il programma in L_i, traducendo ed eseguendo di volta in volta le istruzioni in L_0

In generale, i calcolatori moderni si basano su architetture a piú livelli:

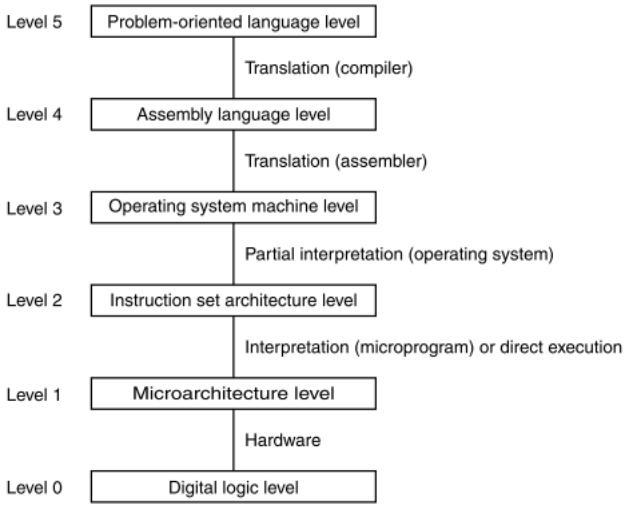

Figure 1-2. A six-level computer. The support method for each level is indicated below it (along with the name of the supporting program).

- applicativo: linguaggi di alto livello utilizzati dai programmatori (es. C, C++, Python)
- sistema operativo: fornisce un'insieme di funzionalitá al livello applicativo (es. syscall, gestione della memoria)
- ISA (Instruction Set Architecture): insieme di istruzioni (instruction set) di uno specifico processore
- microarchitettura: interpretazione delle istruzioni ISA in microistruzioni (semplici istruzioni come fare la somma di due numeri)
- logico-digitale: hardware del calcolatore e le sue componenti elementari

1.2.1 Traduzione dei programmi C, C++

Generalmente, C e C++ vengono compilati (nulla vieta di utilizzare un interprete) e questo é generalmente il processo che porta da un file sorgente a un eseguibile:

1. compilazione: i singoli file sorgenti sono compilati in file oggetto (estensione .obj o .o)

2. linking: vengono risolti i riferimenti dei singoli file

A questo punto é disponibile un eseguibile, che, se invocato, viene caricato in memoria, i riferimenti vengono trasformati in indirizzi di memoria e il programma viene eseguito.

1.2.2 Macchine virtuali

La *virtualizzazione* é un meccanismo che permette di astrarre un determinato componente di un sistema. Oggi, é utilizzata in diversi modi:

- hardware: vengono virtualizzate le componenti fisiche di un computer, riuscendo ad esempio ad ospitare sistemi operativi diversi contemporaneamente, emulandoli

- software: alcuni linguaggi, come Java, vengono compilati in un *linguaggio macchina immaginario* (in questo caso chiamato bytecode). Il linguaggio intermedio viene interpretato da una macchina viruale (JVM=Java Virtual Machine), che traduce queste istruzioni nello specifiche istruzioni ISA. La macchina virtuale é un interprete, quindi generalmente piú lento di un compilatore, anche se utilizza tecniche di **compilazione just-in-time** (compila delle istruzioni a runtime, appena prima che vengano eseguite) e **caching** (non ritraduce codice giá tradotto)

1.3 Rappresentazione digitale delle informazioni

1.3.1 Precisione finita

I calcolatori moderni utilizzano segnali *digitali* (lat. digitus, dito).
A causa dela limitazione della quantitá di memoria, il calcolatore utilizza numeri a *precisione finita*, ovvero rappresentati con un numero finito di cifre. A causa di questo, é possibile, in operazioni aritmetiche, ottenere errori: **underflow** (il risultato dell'operazione é minore del piú piccolo valore rappresentabile), **overflow** (il risultato dell'operazione é maggiore del piú grande valore rappresentabile) e **non appartenenza** (il risultato non é nè troppo grande, nè troppo piccolo ma non appartiene all'insieme dei valori rappresentabili). Per questo, non sempre valgono la proprietá associativa $(a+(b+c) = (a+b)+c)$ e distributiva $(a(b+c) = ab+ac)$.

1.3.2 Notazione posizionale base 2

Per rappresentare i numeri, si utilizza la notazione posizionale in **base 2**: $\sum_{i=-k}^{n} d_i \cdot 2^i$ ($-k$ per indicare anche i numeri decimali).
Un *byte* indica un insieme di 8 bit; sequenze piú lunghe di 1 byte sono dette

word (hanno lunghezza variabile). Il numero di configurazioni dato un numero n di bit é 2^n.

1.3.3 Conversioni

Le principali conversioni sono:

- binario $\rightarrow$ decimale: si moltiplica la cifra i-esima per 2^i (es. $11010_2 = 1 \cdot 2^4 + 1 \cdot 2^3 + 1 \cdot 2^1 = 16 + 8 + 1 = 25_{10}$)
- binario $\rightarrow$ ottale: si raggruppano, a partire dalla virgola, i bit in gruppi di 3 e si convertono in una cifra del sistema ottale (perché $8 = 2^3$)
- binario $\rightarrow$ esadecimale: stessa cosa dell'ottale ma con gruppi di 4 cifre (perché $16 = 2^4$)

1.3.4 Codifica dei numeri interi

I numeri interi vengono rappresentati con due principali metodi:

- complemento a 2
- eccesso 2^{m-1}

Nel complemento a 2, dato un numero n, $-n$ é ottenibile invertendo ciascun bit di n e aggiungendo 1. I numeri positivi hanno come primo bit 0, i negativi 1. Per ottenere il valore assoluto di un numero negativo, basta quindi applicare il complemento a 2.

Nella codifica eccesso 2^{m-1}, un numero n é rappresentato da $n + 2^{m-1}$, dove m é il numero di bit utilizzati per rappresentare n. É identico al complemento a 2 con il bit di segno invertito: i numeri da -2^{m-1} a $+2^{m-1} - 1$ sono mappati da 0 a $2^m - 1$ (es. con 8 bit, i numeri da -127 a 128 sono mappati da 0 a 255).

Con entrambe queste codifiche si ha una sola rappresentazione per ciascun numero (compreso lo 0), oltre ai seguenti vantaggi:

- possibilitá di utilizzare un unico circuito sia per la somma che per la sottrazione, perchè $a - b = a + (-b)$
- per verificare se é avvenuto un overflow/underflow, é sufficiente controllare che gli ultimi 2 bit del riporto siano diversi