Informatica

Esplicazione del manuale e delle slide

Marco

Premessa

Chi siamo

Appunti Luiss è un progetto nato per rendere meno difficoltosa e più soddisfacente la vita universitaria.

Questo è stato possibile perché il team di appunti Luiss ha fatto una scoperta tanto banale quanto geniale: la collaborazione tra studenti tramite la condivisione di esperienze universitarie facilita il superamento degli esami. Tale collaborazione e condivisione, molto spesso, si concretizza nella produzione, anche involontaria, di lavori come appunti, compendi o esplicazioni.

Ora, dato che la diffusione di questo tipo di lavori aiuta lo studio e il superamento degli esami, il **favorire** tale diffusione è il primo obbiettivo che Appunti Luiss si propone.

Il secondo obbiettivo che ci proponiamo è quello di **valorizzare** questo tipo di lavori. Tale valorizzazione, per natura, produce un doppio effetto: favorisce la **diffusione,** incentivando gli studenti a produrne sempre di più, e costituisce la giusta **ricompensa** per gli studenti che li hanno prodotti agevolando anche il sostentamento dello studente stesso.

Insomma, quello che Appunti Luiss vuole fare è **aiutare** gli studenti e **premiare** coloro che hanno reso questo possibile.

Appunti Luiss Team

Indice

INFORMAZIONE E LA SUA CODIFICA

L'informatica è la scienza della rappresentazione e dell'elaborazione dell'informazione.
Il mezzo dell'informazione è il **supporto**, l'operazione con cui essa viene scritta su tale mezzo fisico è la *codifica* mentre la *decodifica* permette di leggere tale informazione. Condizione necessaria perché un supporto sia in grado di portare informazioni è perciò che esso possa assumere **configurazioni differenti**, a ognuna delle quali venga associata una differente entità di informazione. Tale interpretazione richiede un codice, ossia un'insieme di regole che identificano in modo non ambiguo l'insieme delle possibili configurazioni del supporto e le associ all'insieme delle possibili entità di informazione.
Affinché l'entità di informazione sia trasmessa, essa deve assumere, in ordine, tre livelli:
- **Informazione sintattica**: ossia una sufficienza delle configurazioni del supporto a rappresentare un determinato messaggio.
- **Informazione semantica**: ossia che ad ogni segno sia associato un significato.
- **Informazione pragmatica**: ossia che i segni e significati abbiano un valore concreto.

CODIFICA BINARIA

Discutendo del livello sintattico dell'informazione, l'alfabeto più semplice che si possa utilizzare è quello costituito da due soli simboli, per questo detto binario. In questo modo i calcolatori hanno bisogno per la memorizzazione di dati ed istruzioni di semplici dispositivi bistabili, in grado di assumere stabilmente una configurazione scelta tra due differenti (per esempio in corrispondenza alla presenza o assenza di tensione elettrica ai capi del dispositivo).
L'unità elementare di informazione è il **bit**, da binary digit, che può essere 1 o 0.
Il codice adottato per l'alfabeto binario è basato sulla posizione delle cifre, per tradurre ogni elemento di un qualsiasi insieme di informazioni si deve adottare un procedimento dicotomico: si divide il sistema in due, le informazioni della prima parte avranno come primo simbolo un bit mentre le informazioni della seconda parte avranno come primo simbolo l'altro bit. Questa operazione si ripete finché tutti gli elementi avranno una simbologia univoca, ovviamente maggiore è il numero degli elementi dell'insieme maggiori saranno i bit da utilizzare per definire il singolo elemento.
Se dunque una successione di k bit consente di identificare $n = 2^k$ elementi diversi il problema inverso è: dato il numero di elementi da identificare dell'insieme S (#S), qual è la lunghezza (k) di bit per ogni per definire ogni singolo elemento? $k = \log_2 \#S$. Per esempio con 8 bit (1 byte) si possono rappresentare 256 dati diversi (2^8). I primi sistemi di codifica in linguaggio binario per rappresentare le lettere dell'alfabeto sono l'ASCII ($128 = 2^7$) e l'UNICODE ($65535 = 2^{16}$).
Per **convertire** un numero in base due in un numero in base dieci:
$101100_{due} = 1 \cdot 2^5 + 0 \cdot 2^4 + 1 \cdot 2^3 + 1 \cdot 2^2 + 0 \cdot 2^1 + 0 \cdot 2^0 = 44$
Per **convertire** un numero in base dieci in un numero in base due:
573_{dieci} = 573 : 2 → resto 1 (cifra binaria meno significativa)
286 : 2 → resto 0
143 : 2 → resto 1
71: 2 → resto 1
35 : 2 → resto 1
17 : 2 → resto 1
8 : 2 → resto 0
4 : 2 → resto 0
2 : 2 → resto 0
1 : 2 → resto 1 (cifra binaria più significativa)
In entrambi i casi risulta come un numero in notazione binaria richiede (a causa del minor numero di simboli) più cifre rispetto a quelle impiegate in notazione decimale.

Somma di numeri binari

Scrivendo i due numeri in colonna si somma cifra per cifra ricordando che 0 + 0 = 0; 1 + 0 = 1; 0 + 1 = 1; 1 + 1 = 10. In quest'ultimo caso scrivo 0 e riporto l'1 a sinistra, se le operazioni di riporto vanno ad aumentare il numero di bit, tutti quelli in più a sinistra si cancellano.

Moltiplicazione di numeri binari

$10011001_2 \times (153_{10})$
$1011_2 = (11_{10})$
$10011001_2 +$
$10011001_2 +$
$00000000_2 +$
$10011001_2 =$
$11010010011_2 \quad (1683_{10})$

Bisogna moltiplicare ogni cifra del moltiplicatore (che può essere 0, caso a, o 1, caso b) per ogni cifra del moltiplicando: in tutti i casi a si ottiene una sequenza di tanti 0 quante sono le cifre del moltiplicando, in tutti i casi b si il risultato è uguale al moltiplicando. Tutti i risultati vengono poi sommati tenendo conto della posizione iniziale della cifra usata (il secondo risultato sarà spostato di un posto a sinistra, il terzo di due ecc.).

Codifica binaria dei numeri razionali

Poiché la rappresentazione che può essere utilizzata da un esecutore automatico deve prevedere solo successioni di bit di lunghezza finita si utilizzano le cifre significative con una notazione scientifica nel caso di numeri molto grandi o moto piccoli: $0,m \cdot 10^e$ (m = mantissa). Lo standard internazionale IEEE-754 stabilisce la lunghezza di mantissa ed esponente in modo che con 32 bit si possono rappresentare i numeri compresi in $[-10^{38}, -10^{-38}] \cup [10^{-38}, 10^{38}]$ oppure con 64 bit i numeri compresi in $[-10^{308}, -10^{-308}] \cup [10^{-308}, 10^{308}]$ stabilendo quanti bit indicano la mantissa e quanti l'esponente. Rimangono comunque dei numeri che non si possono rappresentare (a meno che non siano aumentati i bit) e che si possono trovare nell'area di **overflow negativo** o **positivo** oppure nell'area di **underflow**.

CODIFICA ANALOGICA E CODIFICA DIGITALE

Abbiamo visto che il concetto formale di informazione è fondato sulla possibilità di distinzione, tuttavia in molti casi l'informazione che si tratta è più che classificatoria dato che consente non solo di riconoscere distinzioni ma anche, per esempio, di stabilire una relazione di ordine (questo è maggiore di quest'altro) o di metrica (questo è distante un certo valore da quest'altro). Si apporta in questo modo una **meta-informazione** in presenza della quale l'insieme delle entità di informazione diventa un sistema. Si pone allora il problema di come fare in modo che le configurazioni del supporto su cui viene codificata l'informazione siano in grado di portare anche questa meta-informazione, per questo sono possibili due strategie di codifica:

Analogica: la meta-informazione è implicita, di conseguenza questa modalità è applicabile sono nei casi in cui la struttura dispone di questa meta-informazione, non è necessaria nessuna ridefinizione della codifica al variare dell'insieme delle entità d'informazione ed è applicabile anche per un numero non finito di elementi informativi.

Digitale: la meta-informazione deve essere esplicitata, di conseguenza questa modalità è applicabile in ogni caso, al variare dell'insieme delle entità d'informazione bisogna estendere la meta informazione (elencazione delle corrispondenze valide tra configurazioni ed entità di informazione) ed è per questo applicabile solo per un numero finito di elementi informativi.

Digitalizzazione

Per codificare in modo digitale una grandezza fisica i cui valori assumono variabili in un intervallo continuo è necessario suddividere tale intervallo in una serie finita di sottoinsiemi e per ognuno di essi si assegna uno dei valori presenti (solitamente quello medio).

Questo modo di operare comporta sempre necessariamente una perdita di informazione, perché tutti i valori compresi nello stesso sottoinsieme sono codificati nello stesso modo e diventano indistinguibili, ovviamente questo fenomeno si riduce riducendo la dimensione dei sottoinsiemi.

La quantizzazione di una grandezza comporta quindi un numero di bit di **quantizzazione** (più sono bit minore è la perdita di dati) mentre se si vuole codificare una grandezza fisica che varia nel tempo quest'ultimo viene sottoposto a **campionamento** (sottointervalli di tempo indicati da un istante di riferimento) che comporta una frequenza di campionamento (Hz).

Compressione dei dati

La qualità della compressione si basa sul *rapporto di compressione* (dimensione del file originale diviso quella del file compresso) migliore con le tecniche lossy.

Tecniche lossless (senza perdita di dati)

Sono degli algoritmi che sfruttano le ridondanze di dati in modo da poter comprimere l'informazione senza perderne parti, pena l'inutilizzabilità della stessa (es. programmi o documenti).

- *Tecniche statistiche*: l'informazione viene analizzata dal punto di vista statistico al fine di assegnare ai simboli una nuova rappresentazione caratterizzata da una lunghezza di bit variabile, che utilizzi successioni di bit più brevi per i simboli più frequenti (es. l'identificazione 010 della lettera A diventa 0).
- *Tecniche sostitutive*: si sostituisce un simbolo elementare ad una successione di simboli ripetuta frequentemente nell'informazione (es. casa = 1, il=3).

Tecniche lossy (con perdita di dati)

Il principio è che si può accettare di perdere durante la compressione una parte dell'informazione originale se il risultato che si ottiene dalla decompressione consente ancora un uso appropriato dell'informazione stessa (immagini, audio).

INFRASTRUTTURE HARDWARE

L'architettura di base di un calcolatore è la **macchina di Von Neumann** costituita dall'unità di elaborazione centrale (Central Processing Unit, **CPU**), dalla **memoria centrale** (vista come un insieme di celle adiacenti ciascuna caratterizzata da un indirizzo univoco), dalle periferiche di Input-Output (**I/O**) e dal **bus** che collega i precedenti tre elementi.

Sulla scheda madre (**motherboard**) del calcolatore, una superficie di materiale plastico, sono integrati il bus, la CPU, la memoria centrale e alcune interfacce I/O (mouse e tastiera), le altre sono realizzate su schede separate che vengono inserite in appositi alloggi montati sulla scheda madre (**connettori**) per consentire il collegamento elettrico con il bus.

BUS

Il bus è una linea alla quale sono contemporaneamente connesse le unità del calcolatore e che consente il trasferimento di dati tra tali unità. Il sistema di trasferimento si basa sul principio *master-slave*: la CPU (master) gestisce i collegamenti del bus (slave). Quest'ultimo è suddiviso in:

- **Bus dati**: utilizzato per trasferire i dati;
- **Bus indirizzi**: identifica la posizione delle celle di memoria in cui la CPU deve scrivere o leggere;
- **Bus di controllo**: transitano i segnali di controllo che consentono di volta in volta di selezionare le unità coinvolte in un trasferimento di dati, definire la direzione dello scambio e in generale di coordinare il sistema.

Il bus nel suo complesso può poi essere *di memoria* se collega il processore alla sua cache esterna, *di sistema* se collega questa cache alla memoria principale e alle periferiche veloci, o *periferico* se collega quello di sistema alla periferiche lente e agli altri sottosistemi (**architettura a bus gerarchici**).

CPU

La CPU si divide in:

Unità di controllo: coordina le operazioni di tutto il processore, contiene i registri PC (*Program Counter*), IR (*Instruction Register*) e PSW (*Program Status Word*).

Data path: comprende dispositivi in grado di elaborare i dati (Arithmetic Logic Unit, **ALU**) e alcune unità di memorizzazione temporanea, i **registri**.

Le operazioni che il processore ripete ciclicamente sono:

- Lettura (*fetch*): acquisizione dalla memoria di una delle istruzioni del programma;
- Decodifica (*decode*): riconoscimento dell'istruzione fra quelle che compongono l'insieme delle istruzioni e identificazione delle operazioni che devono essere svolte per completarne l'esecuzione;
- Esecuzione (*execute*): effettuazione delle operazioni corrispondenti all'istruzione;

Più tecnicamente avviene:
- Il contenuto del registro PC viene trasmesso alla memoria lungo il bus indirizzi per prelevare l'istruzione corrente (sul bus di controllo è quindi indicato: leggere).
- La memoria scrive l'istruzione richiesta sul bus dati che viene poi trasferita nel registro IR; contemporaneamente si aggiorna il PC in modo che contenga l'indirizzo dell'istruzione successiva;
- Il contenuto del registro IR viene trasmesso all'ALU che, dopo aver eseguito i calcoli, riporta l'esito sul PSW.
Questo tipo di architettura prevede che qualsiasi tipo di operazione si effettui su dati già caricati sui registri, per questo è chiamata *load/store* (caricamento, archiviazione dei risultati). Le istruzioni che è in grado di gestire sono:
- **Aritmetico-logiche**: utilizzando gli operandi presenti nei registri.
- **Di trasferimento dati**: tra i registri e la memoria (oppure le periferiche di I/O).
- **Di salto** (da un'istruzione all'altra): può essere condizionato o non condizionato (ossia avviene o no in una determinata condizione).

Prestazioni

Il tempo che la CPU impiega per svolgere un processo si può esprimere come il numero di cicli di clock necessari (un clock segna il ciclo lettura-decodifica-esecuzione) moltiplicato per il periodo di clock oppure diviso per la frequenza di clock (la frequenza è l'inverso del periodo). Il numero medio di cicli di clock per eseguire un'istruzione si ricava invece dividendo il numero di cicli di clock per il numero delle istruzioni.
Per incrementare le funzioni di un processore si sono sviluppate due filosofia di progettazione: la **CISC** vuole realizzare a livello hardware funzioni sempre più complesse consentendo di ridurre il numero di istruzioni richieste per il completamento del processo. Tuttavia i compilatori, in grado di tradurre le funzioni in linguaggio macchina, non erano in grado di sfruttare al meglio un insieme di istruzioni complesse. Da ciò prende spunto la **RISC** basata sull'idea di rendere molto semplici le istruzioni riconosciute dalla CPU. Col tempo la prima prenderà alcuni principi della seconda e nascerà la CRISC.
Un primo indice di prestazioni si ottiene misurando il numero di istruzioni in linguaggio macchina che la CPU può eseguire in un secondo: *Mega Instructions Per Second* (**MIPS**) che corrisponde al numero di istruzioni da eseguire per completare un processo diviso il tempo di CPU per 10^{-6}.
Un indice alternativo si basa sul numero di operazioni (e non istruzioni) eseguite in un secondo: *FLoating point Operation Per Second* (**FLOPS**) che corrisponde al numero delle operazioni (riguardanti numeri reali) diviso il tempo di CPU per 10^{-6}.

CPU pipeline

Per sfruttare in maniera più efficiente le diverse unità della CPU è necessario adottare una modalità operativa che riduca i tempi morti facendo sì che ognuna di esse lavori costantemente: il lavoro viene suddiviso in parti ed ognuna di esse viene eseguita da una parte della CPU che corrisponde ad uno stadio, finito il primo si passa al successivo in sequenza mentre i precedenti vengono subito riempiti nuovamente. In questo modo si ha un flusso continuo come un liquidi in una tubatura, pipeline per l'appunto. L'aspetto negativo è che poiché gli stadi della pipeline sono collegati in successione e devono operare in modo sincrono, la durata di un ciclo di clock è dunque determinata dal tempo richiesto dallo stadio più lento (come la velocità di un convoglio di navi limitata dal mezzo più lento). Tale difetto si annulla all'aumentare del numero di istruzioni considerate e se si considerano istruzioni di lunghezza sempre maggiore.
Le pipeline che superano i venti stadi si considerano superpipeline e se ogni stadio è in grado di elaborare due o più istruzioni contemporaneamente la CPU viene definita superscalare.

Simultaneous multithreading

Un processore con questa tecnologia si presenta come una coppia di processori di cui uno è quello principale e l'altro è costituito dalla seconda unità di controllo che interviene sui cicli liberi, ossia su quelle unità operative della CPU che, non ricevendo più informazioni, sono libere.

Processori multicore

Questa tecnologia consiste nell'utilizzare più processori indipendenti contemporaneamente, in questo modo si ottiene un incremento delle prestazioni anche abbassando la frequenza di funzionamento del processore che permette di ridurre il consumo energetico e il surriscaldamento.

MEMORIA

Memoria centrale: destinata a contenere i programmi in esecuzione e i relativi dati; agisce principalmente come supporto alla CPU, cui deve appunto fornire dati e istruzioni a elevata velocità per non penalizzare in maniera inaccettabile le prestazione complessive del sistema (tecnologie elettroniche).
Memoria di massa: destinata a contenere grandi moli di dati che non vengono utilizzati frequentemente, ma che devono essere memorizzati in modo permanente per essere accessibili anche dopo lo spegnimento. La caratteristica non è quindi la velocità ma la stabilità (tecnologie magnetiche ed ottiche).

Caratteristiche

Le unità minime indirizzabili sono chiamate **celle**, costituite da una successione di bit chiamata **parola**, ogni cella ha poi un **indirizzo**. Si definisce perciò *lettura* di una cella l'operazione che rende disponibile sul bus dati la parola presente nella cella selezionata dal bus indirizzi; *scrittura* è invece l'operazione con cui alla parola della cella si sostituisce la parola presente sul bus dati. Per valutare le prestazioni di un'unità di memoria si usano i seguenti parametri:

- **Tempo di accesso**: l'intervallo tra il momento in cui viene presentata dalla CPU la richiesta di accesso alla memoria e l'istante in cui essa termina il proprio compito rendendo disponibile il dato. Può essere:
  - **Sequenziale**: le celle sono posizionate in successione, così che l'accesso a un dato comporta la lettura di tutti quelli che lo precedono.
  - **Casuale**: l'accesso ad una cella non richiede la lettura delle precedenti (Random Access Memory).
  - **Misto**: poiché l'indirizzo non è sufficiente ad identificare l'esatta collocazione della cella, con diversi accessi si giunge in prossimità del dato per poi effettuare una ricerca sequenziale ed arrivare ad esso.
  - **Associativo**: la parola viene selezionata in base ad una parte del suo contenuto invece che a partire dal suo indirizzo (tipico delle memorie cache).
- **Velocità di trasferimento**: quantità di dati trasferiti nell'unità di tempo da o verso la memoria.

Organizzazione gerarchica

- **Registri**;
- **Cache** I livello; II livello; III livello;
- **Memoria centrale** (**RAM**, volatile, e **ROM**, Read Only Memory non volatile, utilizzata per contenere le informazioni di inizializzazione del calcolatore che vengono usate ad ogni accensione o altre funzioni che devono essere permanentemente utilizzabili come il sistema di diagnosi del corretto funzionamento delle varie unità);
- **Memoria di massa** (disco fisso e dischi esterni).

I livelli più alti sono caratterizzati da una maggiore **velocità** di trasferimento e accesso (per non penalizzare la CPU) e offrono uno spazio ridotto per motivi economici mentre quelli più bassi consentono un ampio **spazio** a basso **prezzo** con il limite di una bassa velocità di accesso e trasferimento. Per poter sfruttare i vantaggi dei due estremi della gerarchie si applica un trasferimento di specifici blocchi di memoria tra i livelli tentando di portare in quelli più alti i dati (per poter incrementare le prestazioni) in base a criteri di **località spaziale** (quando un programma fa riferimento ad un elemento è molto probabile che quello stesso programma faccia riferimento entro breve tempo ad altri elementi il cui indirizzo è vicino a quello dell'elemento riferito) e di **località temporale** (quando un programma fa riferimento ad un elemento è molto probabile che quello stesso programma faccia riferimento entro breve tempo allo stesso elemento, ciò vuol dire che i blocchi da cancellare sono sempre i più vecchi).