Programmazione inizio lezioni 9:00

Unirsi al gruppo telegram, link nel sito

Sul sito web è elencato il materiale didattico: dispense + libro

<https://vigna.di.unimi.it/prog> (o cerca sebastiano vigna su google)

Terminale + editor di testo (co pilot-chatgtp) + installare compilatore go

1 scritto 1 orale. Nell’orale non bisogna mancare di astrazione (non vanno bene solo gli esempi).

**Lezione 1 - 26/09/23**

**Hardware**

Scheda madre: supporto fisico per impiantare i vari chip, dei circuiti integrati basati sul silicio che sono in grado di svolgere delle azioni di base. La cpu è il chip più importante che fa più cose(+conti).

Tra i circuiti integrati troviamo la memoria, dove vengono scritte le informazioni: Ram (random access memory- posso accendere a ogni elemento di informazione in qualunque posizione nello stesso tempo) (si contrappone alla memoria sequenziale, che posso leggere solo dall’inizio alla fine, sono dei nastri – i backup). La ram è alimentata, spento il computer svanisce. Le memorie alimentate sono più veloci di quelle non alimentate, tipo le chiavette usb.

Alla scheda madre si collegano delle periferiche, dei dispositivi fisici che permettono al computer di interagire con il mondo (mouse, tastiera, schermo).

Le memoria di massa sono delle memorie tipo la ram che però ricordano le informazioni senza essere alimentate (tipo usb o HD). Nastri anni negli 60, poi gli Hard Disk (dove c’è qualcosa di meccanico che gira) e ora le SSD (solid state drive, non hanno nulla di meccanico ma solo componenti integrati).

Le ssd sono più piccole degli HD, sono più veloci perché non hanno nulla di meccanico, sono più resistenti degli HD. In genere però contengono meno memoria degli HD e costano di più.

La GPU è un componente integrato che si occupa di disegnare cose sullo schermo.

Alcune memoria di massa sono saldate sulla scheda madre, altre invece sono esterne.

Le memoria di massa sono più lente di diversi ordini di grandezza rispetto alla ram, e si consumano, sia HD che SSD.Quando si salva, le informazioni vengono trascritte dalla ram alla memoria di massa.

Le schede madri nel tempo sono diventate sempre più piccole perché le componenti, i chip, diventano sempre più integrati: un unico chip ne contiene tanti altri -> processo di integrazione.

* **Software**

Reverse engeneering.

Hardware: capisce istruzioni molto elementari ->

Firmware: istruzioni in linguaggio macchina che consente di usare l’hardware con costrutti un po’ più complessi (serve per es quando si accende il computer, accende le varie componenti -> Rom (read-only memory)) ->

Sistema Operativo: utilizza il firmware per dare un controllo più semplice e complesso della macchina. Offre all’utente(più agli applicativi) una API (Application Program Interface) per eseguire istruzioni in maniera semplice. Gestisce la rete e le memoria di massa. ->

Applicativo -> chiama il sistema operativo tramite API per mandare dei comandi alla macchina. La programmazione risiede qui.

Il linguaggio macchina è sequenze di 0 e di 1 che si mettono nel firmware da mandare nell’hardware da eseguire ->

Assembly -> rende il linguaggio macchina più comprensibile, ma fa comunque solo azioni basiche. Le trasforma poi in una sequenza di 0 e 1 da mandare alla macchina -> linguaggio a basso livello.

Linguaggio di alto livello -> con costrutti intuibili del tipo a = b + c. Il programma va poi tradotto in linguaggio macchina con vari metodi:

* Il compilatore (il traduttore) è un programma che poi traduce il linguaggio ad alto livello in linguaggio macchina. Linguaggio complilato: c++
* L’interprete prende lettera per lettera il programma e lo traduce al volo. Può eseguirlo in qualunque circostanza. Linguaggio interpretato: python -> E’ più lento dei linguaggi compilati.
* Tra il L.M. e l’alto livello troviamo un byte code, un nuovo linguaggio che è più simile ad assembly, ma molto più semplice dell’alto livello. La traduzione avviene dall’alto livello al byte code (90% del lavoro, traduce dall’alto livello a un processore virtuale), e dal byte code al linguaggio macchina (traduce dal processore virtuale ad un processore reale). Il processore virtuale non è uguale al processore reale, ma è molto simile. Per questo motivo è più veloce di un interprete. Java, C#. E’ più veloce del linguaggio interpretato ma comunque un po’ più lento del compilato. E’ una via dei mezzi tra i due. Il byte code è uguale dappertutto: posso eseguirlo da diverse macchine. Nel caso di java il byte code è la Java Virtual Machine.

**Lezione 2 - 27/09/23**

Le memorie sono costituite da una sequenza di bit (unità minima per misurare l’informazione, 0 o 1). Per ragione pratiche la memoria è suddivisa in una sequenza di 8 bit, il byte: è l’unità minima per misurare la memoria. Un byte ha 2^8=256 diverse combinazioni.

B byte - b bit

K 10^3=1000 Ki 2^10=1024

M 10^6 Mi 2^10

G 10^9 Gi 2^20

Una pagina di un libro occupa 2 KB, 500 pagine occupano 1 MB

I computer a 64 bit signica che può maneggiare 64 bit contemporaneamente in una sola volta.

**Macchina di Von Neumann**

La Cpu è collegata al bus. Al bus sono attaccate la RAM, le periferiche tramite delle interfacce che consentono una comunicazione tra periferiche e bus.

La CPU contiene l’ALU, la parte che fa i conti, contiene dei registri (se il pc è a 64 bit, questi registri possono contenere dati di 64 bit), ed infine l’InstructionRegister e il ProgramCounter, ossia che istruzione sta eseguendo e dove si trova nella memoria per eseguire il programma.

**Fetch-Decode-Execute**

La memoria è una sequenza di byte numerati da 0. La cpu guarda dov’è il PC, che contiene un indirizzo di memoria. La cpu guarda la posizione memorizzata dal PC e si legge i byte, per poi metterli nell’IR e decodificarla. Dopo aver decodificato esegue l’istruzione e quando finisce avanza il PC all’istruzione successiva.

**Esempio CPU MIPS 32 (Assembly -> rappresentazione testuale del linguaggio macchina)**

1) 1000: lw $2, 2000($0)

2) 1004: add $3, $0, $0

3) 1008: add $4, $3, $0

4) 1012: mult $4, $4, $4

5) 1016: slt $5,$4,$2

6) 1020: beq $5, $0, +8

7) 1024: add $3, $3, +1

8) 1028: j 1008

9) 1032: sw $3, 3000($0)

La CPU ha come registri disponibili $0 (che vale sempre 0), $2, $3, ecc. Inizia con il PC.

1. Parte alla posizione 1000 in memoria che ha un’istruzione: lw $2, 2000($0). 2000 è la posizione di memoria dove stanno i dati perchè $0 è un registro che in questo caso vale 0: al valore assegnato, 2000, va aggiunto quello del registro tra le parentesi, $0 (è utile per prendere sequenze di dati che sono vicini: se tra le parentesi ci fosse $1 che vale 2, la CPU prenderebbe i dati in posizione 2002, 2004, ecc). La CPU prende i dati in posizione 2000 (max 32 bit) e li mette nel registro $2.
2. Poi va all’istruzione dopo in posizione 1004 (questa CPU funziona a blocchi di 4 byte, quindi ogni istruzione occupa 4 byte. Non è una cosa che dipende dai bit del processore, ex cpu a 64bit, ma una cosa a se stante che dipende dalla sua architettura): add $3, $0, $0. Prende i due registri a destra, li somma e li mette in quelli a sinistra. Visto che $0 + $0 = 0, $3 diventa 0.
3. somma $3 con $0 e lo mette in $4 (in fin dei conti, visto che $0=0 copia $3 in $4)
4. rimpiazzia il contenuto di $4 con il suo quadrato
5. prima tutte le istruzioni erano aritmetiche, ora è di tipo logico. Se $4 è minore di $2, mette 1 in $5, se invece $4 >= $2 mette uno 0 in $5 (un booleano in pratica). Altera il PC in modo relativo.
6. Beq guarda se i primi due registri sono uguali. Se $5 e $0 sono diversi, non fa nulla, ma se sono uguali aggiunge al PC +8 (salta quindi due operazioni, va in posizione 1032 saltando 1024 e 1028). Questa operazione altera il flusso di esecuzione.
7. Incrementa $3 di uno
8. j, che sta per jump, altera il PC in modo assoluto. Il PC riprende dalla posizione 1008
9. Prende il contenuto di $3 e lo mette nella posizione 3000

Il programma alla fine scrive nella posizione 3000 il contenuto di $3. Se per esempio in posizione 2000 troviamo il valore 17, il programma scriverà in posizione 3000 5.

Questo programma calcola la radice quadrata per eccesso. Se l’input è n l’output è [radq(n)]

**Lezione 3 - 29/09/2023**

**La rappresentazione dell’informazione nel computer**

Si fa per discretizzazione e quantizzazione

**Immagini**

Con la discretizzazione si suddivide l’immagine, lo spazio, in una griglia e si fa corrispondere a ogni cella(pixel) il colore medio dell’immagine presente in essa. Più è fine la griglia della discretizzazione (=la griglia dei pixel), più sarà simile alla realtà. Ogni anno si aumenta la discretizzazione.

quantizzazione: ogni pixel corrisponde a un colore. Bisogna trovare un modo per esprimere i colori, qui interviene l’RGB -> sarebbero numeri da 0 a 1, ma essendo questi infiniti lo discretizziamo in un byte, da 0 a 255 (colori a 8bit). Ogni pixel quindi consumerà 3 byte (uno per R, uno per G e uno per B): per calcolare la grandezza di un’immagine si farà **numero di pixel della base X numero di pixel dell’altezza X 3 byte** (questo è il minimo, se l’immagine non è compressa).

L’immagine è quindi ridotta a una serie di byte. La graphic card trasforma i bit dell’immagine contenuta nella memoria in un segnale visibile sul monitor. Quando si disegna sullo schermo, la CPU alloca ai pixel disegnati nella memoria un valore RGB diverso (se si disegna con il nero 0,0,0) e dopo la scheda grafica, leggendo la memoria, si rende conto che il valore RGB è cambiato e lo rappresenta sul monitor.

Nello streaming non si memorizza ogni fotogramma (sono minimo 24 al secondo), ma essendo questi molto simili tra di loro si memorizza la differenza tra un frame e l’altro.

( La fotocamera trasforma da analogico a digitale, tramite dei sensori trasforma ogni cella del suo sensore in un pixel salvato come una sequenza di byte. Il computer poi fa il contrario ).

**Audio**

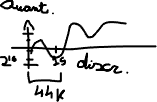
Discretizzazione: si divide il tempo in tanti piccoli punti e si registra il segnale di quei punti sulla griglia. Il segnale elettrico diventa quindi ora una lista di misurazioni.

Teorema di Foye?: se si campiona il segnale con una certa frequenza si riescono a ricostruire le sequenze del segnale fino a metà del campionamento: se il suono è a 44k Hz, si può ricostruire a 22k Hz. In un CD a 44k Hz la differenza di tempo tra questi punti è di 1/44.000 di secondo.

La frequenza corrisponde quindi a quanti punti di suddivisione si hanno ogni secondo.

Ora bisogna quantizzare: lo standard della quantizzazione è 16-bit. Si limita il range del suono dal silenzio a circa 130 decibel (la rottura del timpano), e lo si divide in 2^16 intervalli (numero di valori possibili che si possono salvare in 2 byte). Ora abbiamo una sequenza di byte.

Peso dell’audio: **Hz \* numero di byte (se è 16 bit è 2) \* numero di canali stereo.** La CPU memorizza questi 2 byte 44k volte al secondo, e il DAC (Digital to Analog Converter) si occupa di farci sentire l’audio.



**Testi**

Si fa tramite il codice ASCII, dove ogni carattere occupa un byte con un valore da 0 a 255.

**Compressione Audio-Video**

Qualcuno trasmette una lista di byte. Invece di mandarli grezzi, conviene comprimere indicando quante volte è ripetuto il byte. Funziona perché si assume che il segnale venga ripetuto: nel caso non sia così, il file verà espando e non compresso; non si può comprimere tutto.

Esempio: 0,0,2,2,2,100,100,2,2,2,2,3,… -> 2, 0, 3, 2, 2, 100, 4, 2, 1, 3, … (Compressione Lossless)

Nel caso la compressione non sia abbastanza, allora si effettua una Compressione Lossy, ossia con perdita di informazione. Invece di memorizzare le ripetizioni, si memorizza il numero di ripetizioni diviso due. Comprimendo vengono scalate le sfumature, diventando più visibili.

Esempio: 0,0,2,2,2,100,100,2,2,2,2,3,… -> 1, 0, 1, 2, 1, 100, 2, 2, 0, 3, … -

Se si ricostruisce l’audio, diventa: 0,0,2,2,100,100,2,2,2,2,3. Abbiamo perso un 2.

La compressione si nota con segnali lunghi: se ho 500 zeri, nel lossless dovrò occupare 4 byte: 256,0,244,0; mentre nel lossy diventerà 250,0: si risparmiano 2 byte. In larga scala questo risparmio diventa grande (bisogna considerare che a 44k Hz ci sono 44k segnali ogni secondo).

Questo esempio di lossy è solo per far comprendere il concetto, non è implementato in maniera ovviamente così semplicistica e grezza.

Il formato zip è lossless

**Lezione 4 - 03/10/2023**

**Ciclo di vita del Software**

* Esigenza: si parte da un’esigenza da parte di un cliente.
* Studio di Fattibilità: a partire dall’esigenza si decide che cosa sia possibile fare. Se non è possibile si torna indietro alle esigenze.
* Analisi delle specifiche: stabilito cosa si vuole fare e cosa si può fare si fa un’analisi delle specifiche.
* Progettazione: dopodichè c’è la progettazione del software
* Sviluppo: i programmatori scrivono il software usando il progetto. Se il progetto non è adatto si può tornare in fase di progettazione.
* Testing and Debugging: si verifica il software cercando di capire se funzioni rispettando le specifiche e se non funziona parte il debugging: si torna allo sviluppo, si capisce perché non va e lo si sistema. Il testing dimostra che una cosa non funzioni, ma non che il programma funzioni al 100%, non si può mai essere sicuri.
* Rilascio: se il software passa la fase di testing viene rilasciato
* Utilizzo e mantenimento: gli utenti trovano dei problemi, si ritorna alla fase di testing.

Testing and Debugging è la maggior parte di lavoro dello sviluppo di un software.

Tutte queste fasi sono informali, sono fatte verbalmente, con l’eccezione dello sviluppo e del testing e debugging.