0. INTRODUZIONE

Gestire la complessità, livelli di astrazione di un calcolatore
Un calcolatore può essere visto da diversi livelli di astrazione.
L'astrazione è una tecnica fondamentale per controllare la complessità di un sistema e consiste nel nascondere tutti i dettagli non importanti di un determinato sistema e tenere conto solo degli aspetti fondamentali.

Per un calcolatore abbiamo diversi livelli di astrazione:

- al livello più basso c'è la **fisica e il moto degli elettroni**.
- più in alto vi sono i **componenti elettronici fondamentali** dei quali è composto il calcolatore ossia i transistor.
- il livello di astrazione successivo riguarda i **circuiti analogici** del sistema. I **circuiti analogici** sono dei circuiti che hanno degli ingressi e delle uscite i cui livelli di tensione variano in modo **continuo**. **I circuiti digitali**, a livello più alto, sono dei circuiti come le porte logiche che limitano i valori di tensione a livelli **discreti**, che sono utilizzati per indicare 0 e 1. Partendo dai circuiti digitali possiamo costruire le **porte logiche** e da esse possiamo arrivare a **blocchi logici** complessi come i sommatori o le memorie.
- la **microarchitettura** unisce il livello di astrazione della logica e quello dell'architettura. combina elementi logici e circuiti digitali per eseguire le istruzioni definite dall'architettura.
- l'**architettura** è un livello che descrive un calcolatore dal punto di vista del programmatore. è definita attraverso una serie di istruzioni, registri che il programmatore può utilizzare per "programmare" i circuiti.

Disciplina
La disciplina è la tecnica di restringere intenzionalmente le scelte di progetto in modo da lavorare in modo più produttivo ad un livello più alto di astrazione. L'utilizzo di parti intercambiabili è un esempio di applicazione della disciplina.

Differenza tra circuiti digitali e analogici
I circuiti digitali utilizzano valori di tensione discreti, mentre **i circuiti analogici** usano valori continui. Quindi i circuiti digitali sono un sottoinsieme dei circuiti analogici con capacità minori. Progettare circuiti digitali però è più semplice e si possono facilmente combinare componenti digitali in un sistema complesso, il quale ha prestazioni migliori rispetto a quelli costruiti con componenti analogici.

Le tre -Y (gerarchia, modularità, regolarità)
Per gestire la complessità di un sistema vengono utilizzati tre "principi" fondamentali: la gerarchia, la modularità e la regolarità

- **gerarchia**: significa che un sistema complesso può essere suddiviso in moduli e ognuno dei moduli può essere suddiviso a sua volta ulteriormente in altri moduli finchè non si arriva a blocchi costitutivi facili da comprendere.
- **modularità**: significa che tutti i moduli definiti abbiano funzionalità e interfacce ben definite così da connettersi tra di loro in maniera semplice.
- **regolarità**: significa cercare uniformità tra i moduli. Infatti i moduli più comuni vengono riutilizzati più volte riducendo il numero di moduli diversi che devono essere progettati. Implica il fatto che in un sistema è meglio avere parti intercambiabili.

Astrazione digitale
La maggior parte delle variabili fisiche è continua. I sistemi digitali invece rappresentano informazioni con variabili dal valore discreto, cioè variabili con un numero finito di valori possibili. I calcolatori utilizzano

una codifica, rappresentazione binaria cioè a due valori dal momento che è più facile distinguere tra due sole tensioni che tra dieci. La tensione maggiore indica 1 e la minore indica uno 0. Una variabile binaria trasporta log2 2 = 1 bit di informazione. Boole ha sviluppato una logica che opera su variabili binarie, nota come logica Booleana, nella quale sono definite le variabili booleane. Ogni variabile booleana può assumere solo uno fra i due valori 'VERO' e 'FALSO' . I calcolatori utilizzano una tensione positiva alta (VDD=1,5 V oggi) per rappresentare 1 e quindi VERO. Mentre una tensione positiva bassa zero volt per rappresentare 0 e quindi FALSO.

I componenti digitali operano in codice binario. La rappresentazione in binario non è human friendly perchè genera codici molto lunghi. Usiamo quindi il sistema esadecimale poichè 16 è una potenza della base 2. Ciò facilita la codifica e la decodifica di un numero. E' possibile codificare ogni cifra esadecimale con 4 bit, che nel sistema binario corrispondono ad una cifra hex. Nel calcolatore le grandezze numeriche sono elaborate mediante sequenze di lunghezza fissa dette "parole" o word. Sono gruppi di bit la cui grandezza dipende dall'architettura del processore. ARM = 32 bit

Un computer le grandezze numeriche sono elaborate mediante sequenze di simboli di lunghezza fissa dette parole ▪ Poichè una cifra esadecimale codifica 4 bit: ▪ 1 byte (8 bit) → 2 cifre esadecimali ▪ 4 byte (32 bit) → 8 cifre esadecimali ▪ 8 byte (64 bit) → 16 cifre esadecimali

1. RAPPRESENTAZIONE DELL'INFORMAZIONE

informazione: informazione è un qualunque insieme di segnali che condizionano l'evoluzione di un sistema. Problema: occorre definire cosa sia un segnale. restringere il contesto in cui si intende una certa nozione per darne una definizione più precisa

alfabeto: per alfabeto A intenderemo un insieme finito e distinguibile di segni che chiameremo a seconda del contesto cifre, lettere, caratteri, simboli etc.

parola/stringa di un alfabeto A: sequenza finita di simboli dell'alfabeto A

A*: indica tutte le possibili parole generabili a partire dall'alfabeto A

linguaggio L di un alfabeto A: è un qualsiasi sottoinsieme di A*

codifica: è l'azione di associare gli elementi di un insieme D alle parole di un linguaggio L è una **funzione totale f: D-->L**

Se f non è suriettiva allora diremo che è ridondante, la funzione non associa ad ogni elemento di D una parola del linguaggio L
Se f non è iniettiva allora diremo che ambigua, la funzione associa un elemento dell'insieme D a più parole del linguaggio L

Sia ND la cardinalità di D e NL la cardinalità di L

- Se ND >NL qualsiasi codifica f: D → L è ambigua
- Se ND < NL qualsiasi codifica f: D -> L è ridondante

Quindi se f è non ambigua e non ridondate allora f è iniettiva e suriettiva, quindi è una funzione biunivoca

decodifica di D: una funzione g: L→ D

Proprietà delle codifiche
ambigua/non ambigua
ridondante/non ridondante

- Economicità: numero di simboli utilizzati per unità di informazione
- Semplicità nell'operazione di codifica e decodifica
- Semplicità nell'eseguire operazioni sull'informazione codificata

Rappresentazione in base generica B

Rappresentazione in base generica B

- Ad ogni naturale b>1 corrisponde una codifica in base b.
- L'alfabeto A_b consiste in b simboli distinti che corrispondono ai numeri 0,1,…, b-1.
- Analogamente al sistema decimale, un numerale di cifre di A_b rappresenta il numero $s_{m-1} \cdots s_0$

$$\sum_{i=0}^{m-1} s_i \cdot b^i$$

Lunghezza di n rispetto ad una base
Chiamiamo lunghezza di n rispetto a b il numero di cifre che occorrono per rappresentare n in base b
La lunghezza di un numerale decresce al crescere della base di codifica

Valore massimo rappresentabile

$$v_{max} = 10^m - 1$$

base 10

$$b^m - 1$$

qualsiasi base diversa da 10

Cifre necessarie per rappresentare n

Consideriamo il problema inverso: abbiamo un valore n e ci chiediamo quante *cifre m occorrono per rappresentarlo.*

Chiaramente il massimo numero rappresentabile con m cifre dovrà essere maggiore o uguale a n

$$b^m - 1 \geq n$$
$$b^m \geq n + 1$$
$$m \geq \log_b(n+1)$$

In particolare cerchiamo il più piccolo m tale che $m \geq \log_b(n+1)$

$$m = \lceil \log_b(n+1) \rceil$$

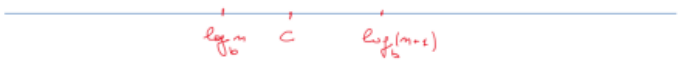
$$\lceil \log_b(n+1) \rceil = \lfloor \log_b n \rfloor + 1$$

- Esercizio: dimostrare che per ogni n e b $\lceil \log_b(n+1) \rceil = \lfloor \log_b n \rfloor + 1$
- Prima osservazione:

Se $x < y$ allora $\lfloor x \rfloor + 1 \leq \lceil y \rceil$

Quindi $\lfloor \log_b n \rfloor + 1 \leq \lceil \log_b(n+1) \rceil$

- Supponiamo per assurdo che $\lfloor \log_b n \rfloor + 1 < \lceil \log_b(n+1) \rceil$
  → chiamiamo c
- Essendo c un intero allora c deve essere compreso fra



Ma allora avremmo:

$$\log_b n < c < \log_b(n+1)$$

$$\Downarrow$$

$$b^{\log_b n} < b^c < b^{\log_b(n+1)}$$

$$n < b^c < n+1$$

b^x

$b \in \mathbb{N}$, $c \in \mathbb{N}$ $\Rightarrow$ $b^c \in \mathbb{N}$

Quindi esisterebbe un numero naturale fra n e n+1, assurdo!

Quindi:

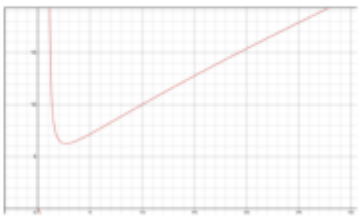
$$\lceil \log_b(n+1) \rceil = \lfloor \log_b n \rfloor + 1$$

Codifica ottimale

- Non bisogna tener presente solo la lunghezza di codifica ma anche il fatto che un calcolatore per operare in una certa base b deve poter rappresentare tutte le cifre di quella base, *quindi gli servono b differenti stati.*
- Una nozione di costo di una codifica che tiene conto anche di questo fattore è data dal prodotto

$$b \cdot m_b \simeq b \cdot \log_b n$$

- Si può mostrare che il valore di b che minimizza tale costo è il numero di Nepero e ≅ 2.7, quindi le codifiche che si avvicinano di più a tale valore ottimale sono quelle in base 2 e 3



Il sistema binario è il sistema di codifica meno economico poiché occorrono tante cifre binarie per rappresentare un numero. Ma bisogna dire che la codifica binaria però è quella ottimale perchè un calcolatore per operare in una certa base deve poter rappresentare tutte le n cifre di quella base, ossia deve poter avere n stati differenti. E' stato dimostrato che il valore ottimale per la codifica in un calcolatore è il numero di nepero e=2.7. Le codifiche che si avvicinano più a tale numero sono quelle in base 2 e base 3. Per i calcolatori si è scelta la base due perchè si codificano solo due stati. Questo è il motivo per cui i calcolatori operano in base binaria.

Numeri con parole di lunghezza fissa

I registri dei moderni calcolatori sono tipicamente parole di 32 o 64 bit Con una parola di lunghezza fissa sono rappresentabili un numero finito di naturali. Nel caso di parole a 64 bit sono rappresentabili i numeri da 0 a 2^64-1 Chiaramente, poiché ogni numero deve essere rappresentato dallo stesso numero di cifre, occorre ricorrere necessariamente a zeri non significativi

Operazioni come l'addizione o la moltiplicazione possono produrre numeri troppo grandi per essere rappresentati con una lunghezza fissata. In questo caso parleremo di trabocco o overflow

Overflow nella somma binaria

Overflow: al termine della somma ho un riporto di 1

```
  11 1
  1101
+ 0101
 10010
```

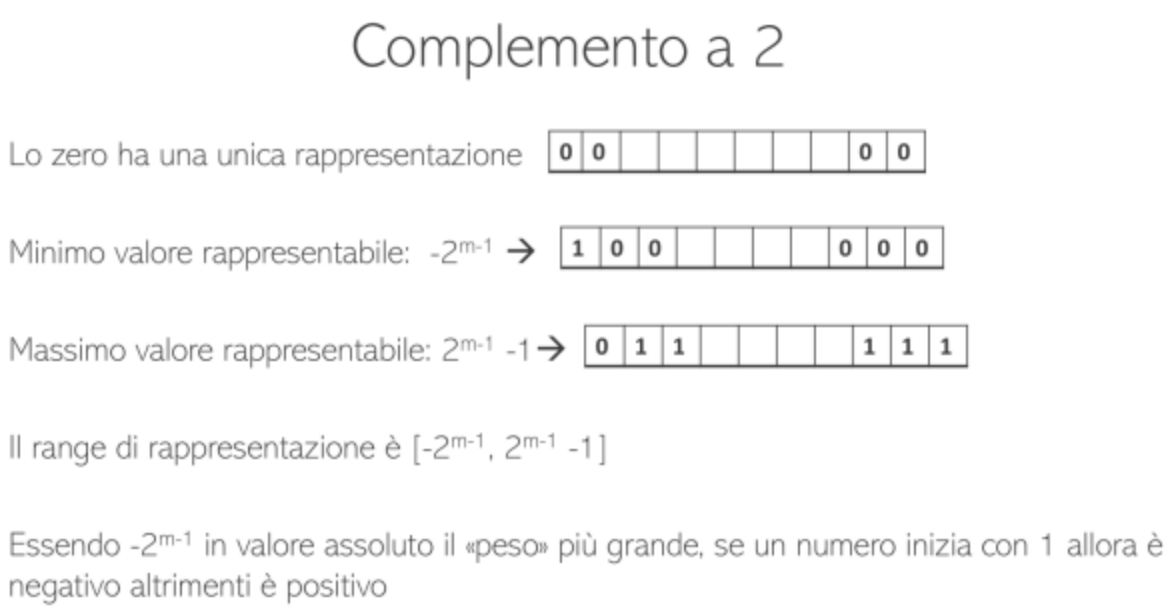
Rappresentazione di interi

Nella rappresentazione di numeri interi mediante parole di lunghezza fissata, dobbiamo codificare non solo il valore assoluto del numero intero ma anche il suo segno. Esistono diversi sistemi che possono essere usati per rappresentare i numeri binari negativi. Le principali rappresentazioni di numeri interi sono:

- **Rappresentazione con modulo e segno**
- **complemento a due**

Rappresentazione con segno
In tale rappresentazione la cifra più significativa di una parola rappresenta il segno. Per convenzione 0 rappresenta il segno più, 1 rappresenta il segno meno. Si utilizza il bit più significativo per esprimere il segno e rimanenti m-1 bit per il modulo. Lo zero ha due possibili rappresentazioni: 0000 e 1000
Intervallo di variabilità: {-2^n-1 + 1 --- 2^n-1 - 1}
Svantaggio: nelle operazioni di addizione o sottrazione occorre controllare il segno e i valori assoluti dei due operandi per determinare il segno del risultato.

Complemento a 2
In questa rappresentazione il bit più significativo vale **-2^n-1**
Lo zero ha un 'unica rappresentazione
Calcolo del complemento a 2: consiste nell'invertire tutti i bit all'interno del numero e poi aggiungere 1



Il numero più negativo -2^N-1 è anomalo perchè il suo complemento a 2 è ancora il numero stesso. Questo numero non ha una controparte positiva

Overflow nel complemento a 2
Sommare un numero negativo e uno positivo non genera overflow
l'overflow non avviene come per gli unsigned quando ho riporto finale di 1
Nella somma a N bit si può verificare un overflow se il risultato è maggiore di vmax 2^N-1 -1 o minore di vmin -2^N-1
In particolare, l'overflow si verifica quando i due numeri che vengono sommati hanno lo stesso bit di segno e il risultato ha un bit di segno opposto
entrambi gli operandi sono negativi (primo bit=1) o entrambi positivi (primo bit=0) è il risultato ha segno opposto

Estensione del segno: Per estendere un numero in una rappresentazione con più bit basta riprodurre a sinistra il bit più significativo (quello di segno)

Binary Coded Decimal
I codici Binary Coded Decimal hanno lo scopo di fornire una naturale rappresentazione binaria del sistema numerico decimale.
Essendo 10 i simboli da codificare ('0',…,'9') avremo bisogno di 4 bit.
Poiché le combinazioni da 10 a 15 non si usano, la codifica BCD è ridondante
I numeri vengono rappresentati con le singole cifre a 4 bit, cifra per cifra. 23=0010 0011
i codici 1010, 1011, 1100, 1101, 1110 ,1111 non sono utilizzati (codifica ridondante)
Naturalmente, la codifica BCD viene usata solo in funzione di interfaccia per rendere comprensibile ad operatori umani i risultati di una elaborazione numerica binaria.
La codifica BCD del numero è, infatti, una operazione propedeutica alla sua visualizzazione su un display numerico decimale (es. display a sette segmenti)

Rappresentazione di Numeri con virgola

Virgola fissa
Nella rappresentazione in virgola fissa si suddividono gli n bit del numero in due sottoparole

- i primi h bit sono dedicati alla codifica della parte intera del numero
- i rimanenti k bit sono dedicati alla codifica della parte frazionaria del numero

Rappresentazione in virgola mobile
In generale, fissare a priori la lunghezza della parte intera h e di quella frazionaria k costituisce una scelta rigida. Alcune applicazioni scientifiche operano con valori ancora più piccoli, altre invece con valori molto grandi

Per ovviare a queste difficoltà è stata introdotta una rappresentazione detta in virgola mobile I numeri in virgola mobile sono equivalenti alla notazione scientifica. Superano la limitazione di avere un numero costante di bit interi e frazionari, permettendo quindi la rappresentazione di numeri molto piccoli ma anche molto grandi.

La differenza fra virgola fissa e virgola mobile è che se analizziamo la distribuzione dei valori rappresentabili con n bit lungo una retta:

- **in virgola fissa** abbiamo un passo piccolo ma costante e abbiamo un range fissato di rappresentabilità, il che implica la perdita di precisione per numeri molto piccoli o grandi
- **in virgola mobile** il passo si infittisce vicino allo 0 per rappresentare anche numeri molto piccoli, per poi dilatarsi man mano. La precisione vicino allo 0 è maggiore di quella in v. fissa, ma man mano che che ci si allontana dallo 0 abbiamo una precisione peggiore di quella in v. fissa perchè il passo si allunga sempre di più

Se abbiamo 32 bit potremmo rappresentare 2^32 combinazioni di numeri sia in v.fissa che in v.mobile , la differenza sta nella precisione del passo.

I numeri in virgola mobile hanno
-un segno 0 positivo, 1 negativo
-una mantissa m -una base b
-un esponente e

Un numero reale è quindi rappresentato da una tripla (s,m,e)
In binario questo vuol dire che la mantissa è sempre del tipo **1,xyz**

Chiaramente nella rappresentazione della mantissa non sprecherò memoria per rappresentare il primo bit "1" e lo considero sottointeso

Standard IEEE 754
Una rappresentazione largamente adottata è quella dell'Institute of Electrical and Electronical Engineering (IEEE)
Singola precisione: 32 bit totali, 1 per il segno, 23 per la mantissa e 8 per l'esponente
Doppia precisione: 64 bit totali, 1 per il segno, 52 per la mantissa e 11 per l'esponente

Per la codifica dell'esponente si utilizza la "rappresentazione polarizzata" (e-P) e=e'+P Costante di polarizzazione:

$$P = 2^{k-1} - 1$$

- IEEE 754 precisione singola: 8 bit per l'esponente
  - $P = 2^7 - 1 = 127$
- IEEE 754 precisione doppia: 11 bit per l'esponente
  - $P = 2^{10} - 1 = 1023$

Avendo 8 bit a disposizione per l'esponente, P = 2^8-1 -1 = 2^7 -1 = 127
Per la mantissa si adotta la rappresentazione scientifica 1,xyz…
e' = si ottiene spostando la virgola vicino al numero più significativo (sempre 1)

Casi speciali IEEE
Lo standard IEEE 754 di rappresentazione dei numeri in virgola mobile prevede codici speciali per rappresentare numeri come lo 0, l'infinito, e i valori impossibili. Per esempio risulta problematica la rappresentazione dello 0 in virgola mobile a causa dell'uno più significativo implicito. Vengono usati per questi casi particolari esponenti di tutti 0 o tutti 1.
Numero 0
Si hanno 2 rappresentazioni, a seconda che il segno sia 1 (negativo) o 0 (positivo)
s = X
e = 00000000 (0)
m = 00000000000000000000000 (0)

+Infinito
s = 1
e = 11111111 (255)
m = 00000000000000000000000 (0)

Numeri molto vicini allo 0
s = X
e = 00000000 (0)
m = diversa da 0

NaN not a number
s = X
esponente = 11111111 (255)

m = diversa da 0

Codifica caratteri alfa-numerici
Si parla di caratteri "alfanumerici" per sottolineare che in un testo sono presenti:
▪ caratteri alfabetici (a,b,c,d,…)
▪ caratteri numerici (0,…,9) ▪ segni di punteggiatura (!,?,…)
▪ simboli particolari vario tipo (£, &, @, ...)

▪ Un testo è una sequenza di caratteri. I codici associano un numero intero ad ogni carattere

1968 ASCII.
Codice a 7 bit: 95 caratteri stampabili e 33 di controllo.

1980 Extended ASCII.
Varie estensioni a 8 bit, con simboli grafici e lettere accentate.

1991 Unicode.
Codice a 21 bit (1 milione di simboli). Attualmente (v. 9.0) definiti circa 128.000 caratteri! Viene ulteriormente codificato in UTF-8.

1992 UTF-8.
Codifica di Unicode a lunghezza variabile (da 1 a 4 byte). Retro-compatibile con ASCII. UTF-8 è la codifica consigliata per XML e HTML.