I sistemi di elaborazione

Architettura del calcolatore

L'architettura è ancora quella classica sviluppata da **Von Neumann** nel 1947.

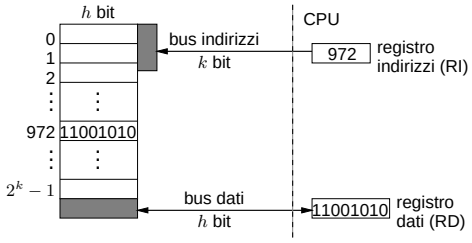
L'architettura di von Neumann riflette le funzionalità richieste da un elaboratore:

- memorizzare i dati e i programmi $\Longrightarrow$ **memoria principale**
- dati devono essere elaborati $\Longrightarrow$ **unità di elaborazione (CPU)**
- comunicazione con l'esterno $\Longrightarrow$ **unità di ingresso/uscita (periferiche)**
- le componenti del sistema di devono scambiarsi informazioni $\Longrightarrow$ **bus di sistema**

Tra le periferiche evidenziamo la **memoria secondaria**.

Memoria centrale (o RAM)

- è una sequenza di **celle di memoria** (dette **parole**), tutte della stessa dimensione
- ogni cella è costituita da una **sequenza di bit** (binary digit)
- il numero di bit di una cella di memoria (dimensione) dipende dall'elaboratore, ed è un multiplo di 8: 8, 16, 32, 64, . . .
- ogni cella di memoria è identificata in modo univoco dal suo **indirizzo**
- il numero di bit necessari per l'indirizzo dipende dal numero di celle di memoria
  k bit $\Longrightarrow$ 2^k celle



Operazione di lettura:

1. CPU scrive l'indirizzo della cella di memoria da cui leggere nel RI
2. esegue l'operazione ("apre i circuiti")
3. il valore della cella indirizzata viene trasferito nel RD

Operazione di scrittura: al contrario

Caratteristiche principali della memoria centrale

- è una memoria ad **accesso casuale**, ossia il tempo di accesso ad una cella di memoria è indipendente dalla posizione della cella $\Longrightarrow$ viene chiamata **RAM** (random access memory)
- può essere sia letta che **scritta**
  - scrittura distruttiva
  - lettura non distruttiva
- **alta velocità** di accesso (attualmente ca. 10ns, per lettura e scrittura)
- è **volatile** (si perde il contenuto quando si spegne il calcolatore)

Dimensione della memoria: misurata in byte

Kilobyte	$=$	2^{10}	$\sim$ 10^3	byte
Megabyte	$=$	2^{20}	$\sim$ 10^6	byte
Gigabyte	$=$	2^{30}	$\sim$ 10^9	byte
Terabyte	$=$	2^{40}	$\sim$ 10^{12}	byte

Tipi di memoria:

RAM random access memory (programmi e dati)

ROM read only memory (firmware)

PROM programmable ROM (possono essere scritte una sola volta)

EPROM erasable PROM (possono anche essere cancellate con luce UV)

EEPROM electrically EPROM (possono essere cancellate da appositi dispositivi)

Memoria secondaria

Caratteristiche:

- non volatile
- capacità maggiore della memoria centrale (decine di GB)
- tempo di accesso lento rispetto alla memoria centrale (ca. 10ms)
- accesso sequenziale e non casuale
- tipi di memoria secondaria: dischi rigidi, floppy, CDROM, CDRW, DVD, nastri, ...

Bus di sistema

Suddiviso in tre parti:

- bus indirizzi: k bit, unidirezionale
- bus dati: h bit, bidirezionale
- bus comandi: trasferisce i comandi da master a slave

$\Longrightarrow$ parallelismo (attualmente si arriva a 128 bit)

CPU (Central Processing Unit)

- coordina le attività di tutte le componenti del calcolatore
- interpreta le istruzioni che compongono il programma e le esegue
- 3 componenti principali:

  unità logico-aritmetica (ALU): effettua i calcoli aritmetici e logici

  unità di controllo: coordinamento di tutte le operazioni

  registri: celle di memoria ad accesso molto veloce all'interno della CPU
  - registro istruzione corrente (IR): contiene l'istruzione in corso di esecuzione
  - contatore di programma (PC): contiene l'indirizzo della prossima istruzione da eseguire
  - accumulatori: utilizzati dalla ALU per gli operandi ed il risultato
  - registro dei flag: memorizza alcune informazioni sul risultato dell'ultima operazione (carry, zero, segno, overflow, . . .)
  - registro interruzioni: utilizzato per la comunicazione con le periferiche

Tutte le attività interne alla CPU sono regolate da un orologio (**clock**) che genera impulsi regolari ad una certa frequenza (ad es. 800 MHz, 1 GHz, 2 GHz, . . .).

Ciclo dell'unità di controllo

Il **programma** è memorizzato in celle di memoria consecutive, sulle quali l'unità di controllo lavora eseguendo il ciclo di

prelievo — decodifica — esecuzione

Ogni istruzione è costituita da:

01001001	00110011
codice operativo	operandi

1. fase di **prelievo** (fetch)
   l'unità di controllo acquisisce l'istruzione indirizzata da PC e aggiorna PC in modo che indirizzi la prossima istruzione
2. fase di **decodifica**
   in base al codice operativo viene decodificato il tipo di istruzione per determinare quali sono i passi da eseguire per la sua esecuzione
3. fase di **esecuzione**
   vengono attivate le componenti che realizzano l'azione specificata

Esecuzione dei diversi tipi di istruzione

- ingresso dati: comanda all'unità di ingresso di trasferire i dati in memoria
- uscita dati: comanda all'unità di uscita di trasferire i dati dalla memoria
- operazione aritmetica o logica: comanda il trasferimento dei dati nell'ALU e l'esecuzione dell'operazione
- salto: aggiorna PC in modo opportuno

Le istruzioni dettano quindi il flusso del programma. Vengono eseguite in sequenza, a meno che non vi sia un'istruzione di salto:

do all'infinito
  preleva dalla memoria l'istruzione indirizzata da PC e caricala in IR
  decodifica l'istruzione
  esegui l'istruzione
  if l'istruzione non è un salto
  then incrementa il valore di PC

Dal codice sorgente al codice macchina

I concetti di algoritmo e di programma permettono di astrarre dalla reale struttura del calcolatore, che comprende sequenze di 0 e 1, ovvero un **linguaggio macchina**.

Livelli di astrazione ai quali possiamo vedere i programmi:

Linguaggio macchina (o codice binario): livello più basso di astrazione
- programma è una sequenza di 0 e 1 (suddivisi in parole) che codificano le istruzioni
- dipende dal calcolatore

Linguaggio assemblativo: livello intermedio di astrazione
- dipende dal calcolatore e le sue istruzioni sono in corrispondenza 1-1 con le istruzioni in linguaggio macchina
- istruzioni espresse in forma simbolica $\Longrightarrow$ comprensibile da un umano

Linguaggi ad alto livello: *Esempi:* C, Pascal, C++, Java, Fortran, Lisp, ...
- si basano su costrutti non elementari, comprensibili da un umano
- istruzioni sono più complesse di quelle eseguibili da un calcolatore (corrispondono a molte istruzioni in linguaggio macchina)
- in larga misura indipendenti dallo specifico elaboratore

Quindi, per arrivare dalla formulazione di un problema all'esecuzione del codice che lo risolve, bisogna passare attraverso **diversi stadi**:

problema
⇓ codifica del problema — *progettista*
algoritmo
⇓ codifica dell'algoritmo — *progettista*
codice sorgente (linguaggio ad alto livello)
⇓ compilazione — *compilatore*
codice oggetto (simile al codice macchina, ma contiene riferimenti simbolici)
⇓ collegamento tra diverse parti in codice oggetto — *collegatore (linker)*
codice macchina (eseguibile)
⇓ caricamento — *caricatore (loader)*
codice in memoria eseguito

Esempio: dati due interi positivi X ed Y, eseguire il loro prodotto usando solo le operazioni di somma e sottrazione

Algoritmo: moltiplicare X per Y, significa sommare X a se stesso per Y volte

leggi X ed Y
inizializza la somma a 0
for Y volte
do incrementa la somma del valore di X
stampa la somma

Raffinamento della parte ripetitiva

leggi X ed Y
inizializza la somma a 0
inizializza il contatore a 0
while contatore $< Y$
do incrementa la somma del valore di X
    incrementa il contatore di 1
stampa la somma

Codifica dell'algoritmo in C

```
#include <stdio.h>
int main (void)
{
  int x, y,
  int i = 0,
  int sum = 0;

  printf("Introduci due interi da moltiplicare\n");
  scanf("%d%d", &x, &y);
  while (i < y) {
    sum = sum + x;
    i = i + 1;
  }
  printf("La somma di %d e %d e' pari a %d\n", x, y, sum);
  return 0;
}
```


Compilazione in linguaggio macchina

Vediamo per comodità il codice in linguaggio assembler (corrisponde 1-1 al codice in linguaggio macchina, ma si legge meglio)

Un **esempio** di **linguaggio assembler**:

Operazione	**Codice assembler**	**Significato**
Caricamento di un dato	`LAOD R1 X` `LAOD R2 X`	Carica nel registro $R1$ (o $R2$) il dato memorizzato nella cella di memoria identificata dal nome simbolico X
Somma Sottrazione	`SUM R1 R2` `SUB R1 R2`	Somma (sottrae) il contenuto di $R2$ al contenuto di $R1$ e memorizza il risultato in $R1$
Memorizzazione	`STORE R1 X` `STORE R2 X`	Memorizza il contenuto di $R1$ ($R2$) nella cella di nome simbolico X
Lettura	`READ X`	Legge un dato e lo memorizza nella cella di nome simbolico X
Scrittura	`WRITE X`	Scrive il valore contenuto nella cella di nome simbolico X
Salto incondizionato	`JUMP A`	La prossima istruzione da eseguire è quella con etichetta A
Salto condizionato	`JUMPZ A`	Se il contenuto di $R1$ è uguale a 0, la prossima istruzione da eseguire è quella con etichetta A
Termine esecuzione	`STOP`	Ferma l'esecuzione del programma

- $R1$, $R2$ indicano i **registri** accumulatori della CPU
- X è un **variabile**, ovvero un nome per una locazione di memoria (a cui è associato un indirizzo)
- A indica l'**indirizzo** di una istruzione del programma (ovvero della cella di memoria che la contiene)

Programma per il prodotto in linguaggio assembler

	Etic.	Istr. assembler	Istruzione C	Significato
0		`READ X`	`scanf`	Leggi valore e mettilo nella cella identificata da X
1		`READ Y`	`scanf`	Leggi valore e mettilo nella cella identificata da Y
2		`LOAD R1 ZERO`	`i = 0`	Inizializzazione di I; metti 0 in $R1$
3		`STORE R1 I`		Metti il valore di $R1$ in I
4		`LOAD R1 ZERO`	`sum = 0`	Inizializzazione di SUM; metti 0 in $R1$
5		`STORE R1 SUM`		Metti il valore di $R1$ in SUM
6	`INIZ`	`LOAD R1 I`	`if (i == y)`	Esecuzione del test; metti in $R1$ il valore di I
7		`LOAD R2 Y`	`jump to FINE`	Metti in $R2$ il valore di Y
8		`SUB R1 R2`		Sottrai $R2$ (ossia Y) da $R1$
9		`JUMPZ FINE`		Se $R1 = 0$ (quindi $I = Y$) salta a $FINE$
10		`LOAD R1 SUM`	`sum = sum + x`	Somma parziale; metti in $R1$ il valore di SUM
11		`LOAD R2 X`		Metti in $R2$ il valore di X
12		`SUM R1 R2`		Metti in $R1$ la somma tra $R1$ ed $R2$
13		`STORE R1 SUM`		Metti il valore di $R1$ in SUM
14		`LOAD R1 I`	`i = i + 1`	Incremento contatore; metti in $R1$ il valore di I
15		`LOAD R2 UNO`		Metti 1 in $R2$
16		`SUM R1 R2`		Metti in $R1$ la somma tra $R1$ ed $R2$
17		`STORE R1 I`		Metti il valore di $R1$ in I
18		`JUMP INIZ`		Salta a $INIZ$
19	`FINE`	`WRITE SUM`	`printf`	Scrive il contenuto di SUM
20		`STOP`		Fine dell'esecuzione

Osservazioni sul codice assembler

- ad una istruzione C corrispondono in genere più istruzioni assembler (e quindi linguaggio macchina)

  Esempio: `sum = sum + x`

  $\Longrightarrow$
  1. carica il valore di X in un registro
  2. carica il valore di SUM in un altro registro
  3. effettua la somma tra i due registri
  4. memorizza il risultato nella locazione di memoria di SUM

- `JUMP` e `JUMPZ` interrompono la sequenzialità delle istruzioni

In realtà il compilatore (ed il linker) genera **linguaggio macchina**

- ogni istruzione è codificata come una sequenza di bit
- ogni istruzione occupa una (o più celle di memoria)
- istruzione costituita da 2 parti:
  - codice operativo
  - operandi

  (nel nostro caso abbiamo solo istruzioni a un solo operando)

Un esempio di linguaggio macchina

Istruzione assembler	Codice operativo
`LOAD R1` ind	0000
`LOAD R2` ind	0001
`STORE R1` ind	0010
`STORE R2` ind	0011
`SUM R1 R2`	0100
`SUB R1 R2`	0101
`JUMP` ind	0110
`JUMPZ` ind	0111
`READ` ind	1000
`WRITE` ind	1001
`STOP`	1011

	Indirizzo	Codice binario: Codice operativo	Codice binario: Indirizzo operando	Istr. assembler
0	00000	1000	10101	READ X
1	00001	1000	10110	READ Y
2	00010	0000	10111	LOAD R1 ZERO
3	00011	0010	11001	STORE R1 I
4	00100	0000	10111	LOAD R1 ZERO
5	00101	0010	11000	STORE R1 SUM
6	00110	0000	11001	LOAD R1 I
7	00111	0001	10110	LOAD R2 Y
8	01000	0101	-----	SUB R1 R2
9	01001	0111	10011	JUMPZ FINE
10	01010	0000	11000	LOAD R1 SUM
11	01011	0001	10101	LOAD R2 X
12	01100	0100	-----	SUM R1 R2
13	01101	0010	11000	STORE R1 SUM
14	01110	0000	11001	LOAD R1 I
15	01111	0001	11010	LOAD R2 UNO
16	10000	0100	-----	SUM R1 R2
17	10001	0010	11001	STORE R1 I
18	10010	0110	00110	JUMP INIZ
19	10011	1001	11000	WRITE SUM
20	10100	1011	-----	STOP
21	10101			X
22	10110			Y
23	10111	0000	00000	ZERO
24	11000			SUM
25	11001			I
26	11010	0000	00001	UNO

Rappresentazione binaria dell'informazione

Per informazione intendiamo tutto quello che viene manipolato da un calcolatore:

- numeri (naturali, interi, reali, . . .)
- caratteri
- immagini (statiche, dinamiche)
- suoni
- programmi

In un calcolatore tutte le informazioni sono rappresentate in **forma binaria**, come sequenze di **0** e **1**.

Per **motivi tecnologici**: distinguere tra due valori di una grandezza fisica è più semplice che non ad esempio tra dieci valori.

Rappresentazione di numeri naturali

- Un numero naturale è un oggetto matematico, che può essere **rappresentato** mediante una **sequenza di simboli** di un alfabeto fissato.
- È importante distinguere tra numero e sua rappresentazione: il **numerale** "234" è la rappresentazione del numero 234.
- Si distinguono **2 tipi di rappresentazione**:

  additiva: ad es. le cifre romane

  posizionale: una cifra contribuisce con un valore diverso al numero a seconda della posizione in cui si trova

Noi consideriamo solo la rappresentazione posizionale.

Un numero è rappresentato da una **sequenza finita di cifre** di un certo **alfabeto**:

$$c_{n-1}c_{n-2}\cdots c_1c_0 \;=\; N_b$$

c_0 ..... viene detta cifra **meno significativa**

c_{n-1} .... viene detta cifra **più significativa**

Il numero b di cifre diverse (dimensione dell'alfabeto) è detto **base** del sistema di numerazione. Ad ogni cifra è associato un valore compreso tra 0 e $b-1$.

Base	**Alfabeto**	**Sistema**
2	`0`, `1`	binario
8	`0`, . . . , `7`	ottale
10	`0`, . . . , `9`	decimale
16	`0`, . . . , `9`, `A`, . . . , `F`	esadecimale

Il significato di una sequenza di cifre (il numero N che essa rappresenta) dipende da b:

$$c_{n-1} \cdot b^{n-1} + c_{n-2} \cdot b^{n-2} + \cdots + c_1 \cdot b^1 + c_0 \cdot b^0 \quad = \quad \sum_{i=1}^{n} c_i \cdot b^i \quad = \quad N$$

Esempio: Il numerale `101` rappresenta numeri diversi a seconda del sistema usato:

Sistema	**Base** b	$\texttt{101})_b$	**Valore** (in decimale)
binario	2	$(\texttt{101})_2$	5
ottale	8	$(\texttt{101})_8$	65
decimale	10	$(\texttt{101})_{10}$	101
esadecimale	16	$(\texttt{101})_{16}$	257

Intervallo di rappresentazione con n cifre in base b: **da** 0 **a** $b^n - 1$

Esempio:

3 cifre in base 10 : da 0 a $999 = 10^3 - 1$

8 cifre in base 2 (1 byte) : da 0 a $255 = 2^8 - 1$

16 cifre in base 2 (2 byte) : da 0 a $65\,535 = 2^{16} - 1$

32 cifre in base 2 (4 byte) : da 0 a $4\,294\,967\,296 = 2^{32} - 1$

2 cifre in base 16 : da 0 a $255 = 16^2 - 1$

8 cifre in base 16 : da 0 a $4\,294\,967\,296 = 16^8 - 1$

Conversioni di base

Conversione da base b a base 10

Usando direttamente

$$c_{n-1} \cdot b^{n-1} + c_{n-2} \cdot b^{n-2} + \cdots + c_1 \cdot b^1 + c_0 \cdot b^0 \quad = \quad \sum_{i=1}^{n} c_i \cdot b^i \quad = \quad N$$

esprimendo le cifre e b in base 10 (e facendo i conti in base 10)

Esercizio: Scrivere l'algoritmo di conversione da base b a base 10.

Conversione da base 10 a base b

$$\begin{aligned} N &= c_0 + c_1 \cdot b^1 + c_2 \cdot b^2 + \cdots + c_{k-1} \cdot b^{k-1} \\ &= c_0 + b \cdot (c_1 + b \cdot (c_2 + \cdots + b \cdot (c_{k-1}) \cdots)) \end{aligned}$$

Vogliamo determinare le cifre $c_0, c_1, \ldots, c_{k-1}$

Consideriamo la divisione di N per b:

$$\begin{aligned} N &= R + b \cdot Q \qquad\qquad (0 \leq R < b) \\ &= c_0 + b \cdot (c_1 + b \cdot (\cdots)) \end{aligned}$$

$\Longrightarrow$ $R = c_0$ ovvero, il resto R della divisione di N per b dà c_0 (cifra meno significativa)
$Q = c_1 + b \cdot (\cdots)$

A partire dal quoziente Q si può iterare il procedimento per ottenere le cifre successive (fino a che Q diventa 0).

algoritmo converte N da base 10 a base b
  $i \leftarrow 0$
  while $N \neq 0$
  do $c_i \leftarrow N$ mod b
    $N \leftarrow N$ div b
    $i \leftarrow i + 1$

N.B. Le cifre vengono determinate dalla meno significativa a quella più significativa.

Esempio: $(25)_{10} = (???)_2$

$N : b$	Q	R	cifra
$25 : 2$	12	1	c_0
$12 : 2$	6	0	c_1
$6 : 2$	3	0	c_2
$3 : 2$	1	1	c_3
$1 : 2$	0	1	c_4

$(25)_{10} = (11001)_2$

N.B. servono 5 bit (con cui possiamo rappresentare i numeri da 0 a 31)

Conversione quando una base è potenza di un'altra

Da base b^k a base b (esempio: da base $8 = 2^3$ a base 2)

$$\begin{aligned} N = \; & c_{n-1} \cdot (b^k)^{n-1} + \\ & c_{n-2} \cdot (b^k)^{n-2} + \\ & \vdots \\ & c_0 \cdot (b^k)^0 \end{aligned}$$

Ogni cifra c_i è compresa tra 0 e $b^k - 1$ $\Longrightarrow$ possiamo rappresentarla in base b con al più k cifre

$$c_{i,k-1} \; c_{i,k-2} \; \cdots \; c_{i,0}$$

che rappresentano il numero

$$c_i = c_{i,k-1} \cdot b^{k-1} + c_{i,k-2} \cdot b^{k-2} + \cdots + c_{i,0} \cdot b^0$$

Sostituendo nell'equazione precedente si ottiene

$$\begin{aligned} N \;=\; & (c_{n-1,k-1}\cdot b^{k-1}+\cdots+c_{n-1,0}\cdot b^0)\cdot (b^k)^{n-1}+\\ & (c_{n-2,k-1}\cdot b^{k-1}+\cdots+c_{n-2,0}\cdot b^0)\cdot (b^k)^{n-2}+\\ & \quad\vdots\\ & (c_{0,k-1}\cdot b^{k-1}+\cdots+c_{0,0}b^0)\cdot (b^k)^0 \end{aligned}$$

Quindi i coefficienti in base b sono

$$c_{n-1,k-1}\; c_{n-1,k-2}\; \cdots\; c_{n-1,0}\; c_{n-2,k-1}\; \cdots\; c_{0,0}$$

$\Longrightarrow$ Si può convertire da base b^k a base b **convertendo ogni cifra separatamente**.

Esempio:

$$\begin{aligned} (542)_8 &= 5\cdot 8^2+4\cdot 8+2\\ &= (101)_2\cdot 2^6+(100)_2\cdot 2^3+(010)_2\\ &= (101\,100\,010)_2 \end{aligned}$$

Da base b a base b^k (esempio: da base 2 a base $16=2^4$):
Si può procedere in modo duale, raggruppando le cifre in gruppi di k a partire da destra.

Esempio: $(1100\,0101)_2=(C9)_{16}$

Rappresentazione di numeri interi

dobbiamo rappresentare anche il **segno** $\Longrightarrow$ si usa uno dei bit (quello più significativo)

Rappresentazione tramite modulo e segno

- il bit più significativo rappresenta il segno
- le altre $n-1$ cifre rappresentano il valore assoluto
- **problemi**:
  - doppia rappresentazione per lo zero ($00\cdots00$ e $10\cdots00$)
  - le operazioni aritmetiche sono complicate (analisi per casi)

$\Longrightarrow$ invece della rappresentazione tramite modulo e segno si usa una rappresentazione in complemento

Rappresentazione in complemento

In quanto segue:

- b indica la base
- n indica il numero complessivo di cifre
- consideriamo solo numeri in valore assoluto $\leq b^n$

Residuo modulo b^n di un intero X:

$$|X|_{b^n}=X-\lfloor X/b^n\rfloor\cdot b^n$$

dove $\lfloor Y\rfloor$ indica la parte intera inferiore di Y

Esempio: $b=10,\quad n=2$

$$\begin{aligned} |25|_{10^2} &= 25-\lfloor 25/10^2\rfloor\cdot 10^2=25-0=25\\ |-25|_{10^2} &= -25-\lfloor -25/10^2\rfloor\cdot 10^2=-25-(-1)\cdot 100=75 \end{aligned}$$

$\Longrightarrow$ per un numero positivo, il residuo è pari al numero stesso
per un numero negativo, il residuo è pari al complemento rispetto a b^n

- ad un numero corrisponde un unico residuo
- ad un residuo
  - corrispondono in generale due numeri, uno positivo ed uno negativo
  - corrisponde però un unico numero nell'intervallo $[-b^n/2,\ b^n/2)$

$\Longrightarrow$ **definizione semplificata di residuo**:

$$|X|_{b^n} = \begin{cases} X, & \text{se } 0 \leq X < b^n/2 \\ b^n - |X|, & \text{se } -b^n/2 \leq X < 0 \end{cases}$$

Il residuo viene utilizzato per la rappresentazione in complemento alla base.

Rappresentazione in complemento alla base (b con n cifre)

- è la rappresentazione di interi relativi nell'intervallo $[-b^n/2,\ b^n/2)$ tramite il residuo modulo b^n
  - se $X \geq 0$: RCB di X è compresa in $[0,\ b^n/2)$
  - se $X < 0$: RCB di X è compresa in $[b^n/2,\ b^n)$
- lo 0 ha una sola rappresentazione
- se $b = 2$ $\Longrightarrow$ **rappresentazione in complemento a 2**
  - positivi: cifra più significativa è 0 (rappresentati nella parte inferiore dell'intervallo)
  - negativi: cifra più significativa è 1 (rappresentati nella parte superiore dell'intervallo)

Esempio: $b = 2, \quad n = 6, \quad X = -9$

$$b^n - |X| = 1000000 - 1001 = 110111 (= 55_{10} = 64 - 9)$$

Osservazione:

$$\begin{array}{rcll} 2^n - |X| & = & 2^n - |X| + 1 - 1 & \\ & = & 2^n - 1 & n \text{ uni} \\ & & -|X| & \text{inverto i bit di } |X| \\ & & +1 & \text{sommo } 1 \end{array}$$

Esempio: $b = 2, \quad n = 6, \quad X = -9$

$$(9)_{10} = (001001)_2$$

inverto i bit: `110110`

sommo 1: `110111`

Metodo ancora più rapido:

1. si rappresenta $|X|$ con n bit
2. a partire da destra si lasciano inalterate tutte le cifre fino al primo 1 compreso
3. si invertono le rimanenti cifre

Operazioni su interi relativi in complemento a 2

Somma di due numeri

- si effettua **bit a bit**
- non è necessario preocuparsi dei segni
- il risultato sarà corretto (in complemento a 2 se negativo)
  (per la giustificazione vedi dispensa)
- può verificarsi **trabocco** (overflow) $\Longrightarrow$ il risultato non è corretto

  Si verifica quando il numero di bit a disposizione non è sufficiente a rappresentare il risultato.

Esempio: $n = 5, \quad \pm 9 \pm 3, \quad \pm 9 \pm 8$

intervallo di rappresentazione: da -2^4 a $2^4 - 1$ (ovvero, da -16 a 15)

riporto	00011		11001		00111		11111	
	$+9$	01001	$+9$	01001	-9	10111	-9	10111
	$+3$	00011	-3	11101	$+3$	00011	-3	11101
	$+12$	01100	$+6$	00110	-6	11010	-12	10100

In questo caso non si ha trabocco.

riporto	01000		10000	
	$+9$	01001	-9	10111
	$+8$	01000	-8	11000
	-15	10001	$+15$	01111
	(e non $+17$)		(e non -17)	

In questo caso si ha trabocco.

Si ha **trabocco** quando il riporto sul bit di segno è diverso dal riporto dal bit di segno verso l'esterno.

Differenza tra due numeri: sommo al primo il complemento del secondo

L'aritmetica reale

L'insieme dei reali (e dei razionali) è infinito $\Longrightarrow$ non è possibile rapprentarli tutti

Rappresentazione in virgola fissa

Si rappresentano separatamente, usando un numero fissato di cifre

- parte intera e
- parte frazionaria

(si usa una virgola per separare le due parti)

$$N_b = c_{n-1}\ c_{n-2}\ \cdots\ c_1\ c_0\ ,\ c_{-1}\ c_{-2}\ \cdots\ c_{-m}$$

rappresenta il numero

$$N = c_{n-1} \cdot b^{n-1} + \cdots + c_0 \cdot b^0 + c_{-1} \cdot b^{-1} + \cdots + c_{-m} \cdot b^{-m}$$

Rappresentazione in virgola mobile

Rappresentazione in **forma normalizzata in base** b

$$X = m \cdot b^e$$

- e è la **caratteristica** in base b di X: intero relativo
- m è la **mantissa** in base b di X: numero frazionario tale che $1/b \leq |m| < 1$
  Se è rappresentata dalla sequenza di cifre

  $$c_1\ c_2\ c_3\ \cdots$$

  allora rappresenta il valore

  $$c_1 \cdot b^{-1} + c_2 b^{-2} + \cdots$$

Esempio: $X = (5)_{10} = (101)_2$

$$\begin{aligned} m &= |m| = (0.101\cdots0000)_2 \\ e &= (11)_2 \end{aligned}$$

Fissati

- k bit per mantissa
- h bit per caratteristica
- 1 bit per il segno

l'**insieme di reali rappresentabili** è fissato (e limitato)

$$\begin{aligned} 1/2 &\leq |m| \leq \textstyle\sum_{i=1}^{k} 2^{-i} \\ |e| &\leq 2^{h-1} - 1 \end{aligned}$$

Questo fissa anche massimo e minimo (in valore assoluto) numero rappresentabile.

Assunzione realistica: reali rappresentati con 32 bit:

- 24 bit per la mantissa
- 7 bit per la caratteristica (in complemento)
- 1 bit per il segno della mantissa (0 positivo, 1 negativo)

Insieme F dei numeri rappresentabili in virgola mobile

- sottoinsieme finito dei numeri razionali rappresentabili (con n bit)
- simmetrico rispetto allo 0
- gli elementi **non** sono uniformemente distribuiti sull'asse reale
  - densi intorno allo 0
  - radi intorno al massimo rappresentabile
- molti razionali non appartengono ad F (ed es. $1/3$, $1/5$, ...)
- non è chiuso rispetto ad addizioni e moltiplicazioni
- per rappresentare un reale X si sceglie l'elemento di F più vicino ad X
- la funzione che associa ad un reale X l'elemento di F più vicino ad X è detta funzione di arrotondamento

Limitazioni aritmetiche

Dovute al fatto che il numero di bit usati per rappresentare un numero è limitato $\Longrightarrow$

- perdita di precisione
- **arrotondamento**: mantissa non è sufficiente a rappresentare tutte le cifre significative del numero
- **errore di overflow**: caratteristica non è sufficiente (numero troppo grande)
- **errore di underflow**: numero troppo piccolo viene rappresentato come 0

Formati standard proposti dalla IEEE (Institute of Electrical and Electronics Engineers)

- singola precisione: 32 bit
- doppia precisione: 64 bit
- quadrupla precisione: 128 bit