Introduzione all'Architettura del Calcolatore

Luciano Bononi

bononi@cs.unibo.it

http://www.cs.unibo.it/~bononi/

Figure credits: some of the figures have been taken and modified from existing figures found on the web

Testi consigliati:

William Stallings, *Computer Organization and Architecture, 5/6-th edition*
Prentice Hall, 2003, ISBN: 0-13-035119-9

Andrew Tanenbaum, Structured Computer Organization, Fourth Edition
Prentice Hall (1999) ISBN: 0-13-095990-1

...altri testi potranno essere citati in seguito.

Contenuti di questo modulo

- **Gli argomenti affrontati includono :**
  - Architettura del calcolatore e tecnologia
  - Aritmetica del calcolatore e rappresentazione dati
  - Insiemi di istruzioni del calcolatore
  - es. di progettazione di un processore
    - in emulazione (simulazione) circuitale
    - a partire da un set minimale di istruzioni
  - es. di realizzazione di un semplice programma
    - in esecuzione sull'architettura del processore+memoria

Cos'è l'architettura di un calcolatore?

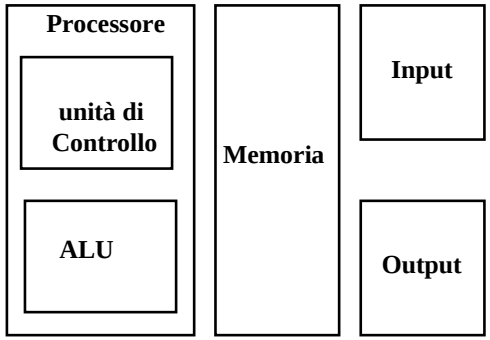
- **L'architettura di un calcolatore è la progettazione di un calcolatore a livello delle interfacce HW e SW**
- **Architettura del Calcolatore =**
  insieme di istruzioni + organizzazione della macchina

Architettura del calcolatore

insieme di istruzioni	organizzazione macchina
Interfaccia del PC	Componenti HW
Interfaccia SW e HW	interfaccia logica

Visione strutturale di alto livello

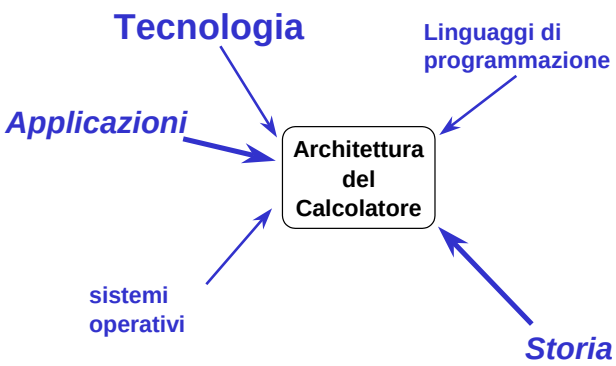
- Queste sono le principali unità che costituiscono il calcolatore



Componenti del calcolatore

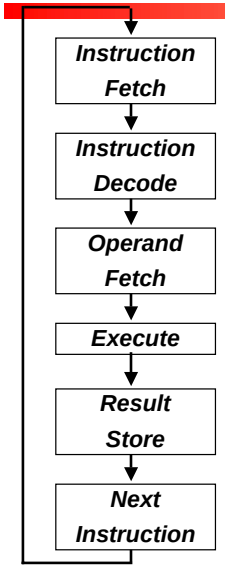
- **Queste sono le principali funzioni svolte dalle componenti del calcolatore**
  - ALU – esegue operazioni aritmetico-logiche
    - es. Somme, moltiplicazioni, shift
  - memoria – mantiene dati e istruzioni (programmi)
    - es. memoria cache, memoria principale, dischi
  - input – acquisisce dati verso il computer
    - es. tastiera, mouse, scheda di rete
  - output – restituisce dati dal computer
    - es. schermo, scheda audio, scheda di rete
  - controllo – gestisce attività delle componenti
    - es. controllore Bus, Gestore unità di memoria (MMU)

Evoluzione dell'architettura del calcolatore



Insieme di istruzioni: interfaccia critica

Ciclo di esecuzione istruzioni



Estrae codice istruzione dalla memoria

determina la dimensione e attiva il controllo

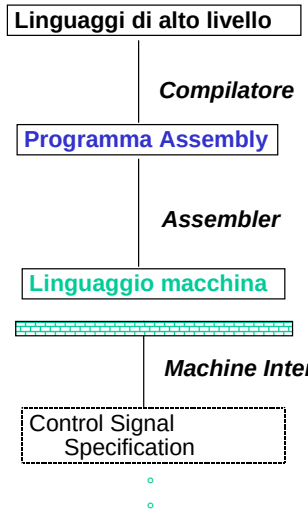
trova e carica i dati (operandi) dell'istruzione

calcola risultati e/o nuovo stato

memorizza risultati in area di memorizzazione

Determina istruzione successiva da eseguire

Livelli di rappresentazione delle istruzioni



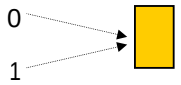

```
temp = v[k];
v[k] = v[k+1];
v[k+1] = temp;
```

```
lw $15, 0($2)
lw $16, 4($2)
sw $16, 0($2)
sw $15, 4($2)
```

```
0000 1001 1100 0110 1010 1111 0101 1000
1010 1111 0101 1000 0000 1001 1100 0110
1100 0110 1010 1111 0101 1000 0000 1001
0101 1000 0000 1001 1100 0110 1010 1111
```


La rappresentazione dei dati sul calcolatore

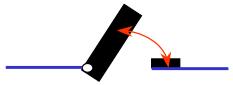
- **il calcolatore dispone di soli due simboli per la memorizzazione, rappresentazione e trasmissione di dati**
  - zero e uno (0,1) = valori assumibili da un binary digit (bit)
  - ogni valore deve essere rappresentato usando questi due simboli (notazione binaria)
  - esistono infinite interpretazioni (codifiche) che possono associare sequenze generiche di simboli 0 e 1 a valori numerici
  - noi vedremo alcune delle più significative



bit = minima unità di memoria

Il bit: valore e logica positiva (lp) e negativa (ln)

levetta:
alta = 1 (lp) 0(ln)
bassa = 0 (lp) 1(ln)



interruttore:
aperto = 0 (lp) 1 (ln)
chiuso = 1 (lp) 0 (ln)

lampadina:
spenta = 0 (lp) 1 (ln)
accesa = 1 (lp) 0 (ln)

tensione elettrica:
bassa = 0 (lp) 1 (ln)
alta = 1 (lp) 0 (ln)

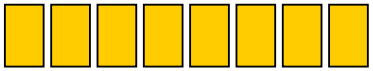
corrente elettrica 50 Hz:
assente = 0 (lp) 1 (ln)
presente = 1 (lp) 0 (ln)

La rappresentazione dei dati sul calcolatore

- **Il calcolatore deve poter memorizzare e elaborare dati e programmi**
  - simboli, valori numerici naturali e interi (positivi e negativi)
    - possono essere rappresentati con un numero finito di bit se appartenenti a un range opportunamente limitato
    - anche i caratteri possono essere mappati su valori interi
  - valori numerici reali, razionali e irrazionali
    - possono essere rappresentati attraverso opportune codifiche binarie, se appartenenti a range opportunamente limitati e tollerando errori di approssimazione
  - programmi espressi attraverso
    - istruzioni (codici) e operandi (dati)
    - es. (Somma 18 3) equivale a (+ 18 3)
      - un simbolo (cioè un valore) può rappresentare un'istruzione del programma memorizzato ed eseguito

La rappresentazione dei dati sul calcolatore

- **i bit possono essere considerati in sequenza (in memoria)**
  - sequenza di 8 bit = 1 Byte



0 0 0 0 0 0 0 0
0 0 0 0 0 0 0 1
0 0 0 0 0 0 1 0
0 0 0 0 0 0 1 1

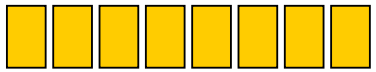
1 1 1 1 1 1 1 0
1 1 1 1 1 1 1 1

2^n possibili sequenze diverse da n bit

n = numero di bit considerati

La rappresentazione dei dati sul calcolatore

- **Dato un byte come possiamo associarvi i simboli o valori?**
  - sequenza di 8 bit = 1 Byte (primo esempio poco utile)

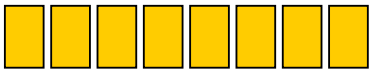


Solo valori da 1 a 8 ? non sfrutto tutte le possibili combinazioni!!!

0 0 0 0 0 0 0 1 = valore 1
0 0 0 0 0 0 1 0 = valore 2
0 0 0 0 0 1 0 0 = valore 3
0 0 0 0 1 0 0 0 = valore 4
0 0 0 1 0 0 0 0 = valore 5
0 0 1 0 0 0 0 0 = valore 6
0 1 0 0 0 0 0 0 = valore 7
1 0 0 0 0 0 0 0 = valore 8

La rappresentazione dei dati sul calcolatore

- **Dato un byte come possiamo associarvi i simboli o valori?**
  - sequenza di 8 bit = 1 Byte

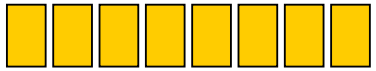


Idea: sfrutto tutte le possibili combinazioni diverse di 8 bit 256 valori [0..255]

0 0 0 0 0 0 0 0 = valore 0
0 0 0 0 0 0 0 1 = valore 1
0 0 0 0 0 0 1 0 = valore 2
0 0 0 0 0 0 1 1 = valore 3
0 0 0 0 0 1 0 0 = valore 4
⋮ ⋮ ⋮ ⋮ ⋮ ⋮ ⋮ ⋮
1 1 1 1 1 1 1 0 = valore 254
1 1 1 1 1 1 1 1 = valore 255

La rappresentazione dei dati sul calcolatore

- **Dato un byte come possiamo associarvi i simboli o valori?**
  - sequenza di 8 bit = 1 Byte

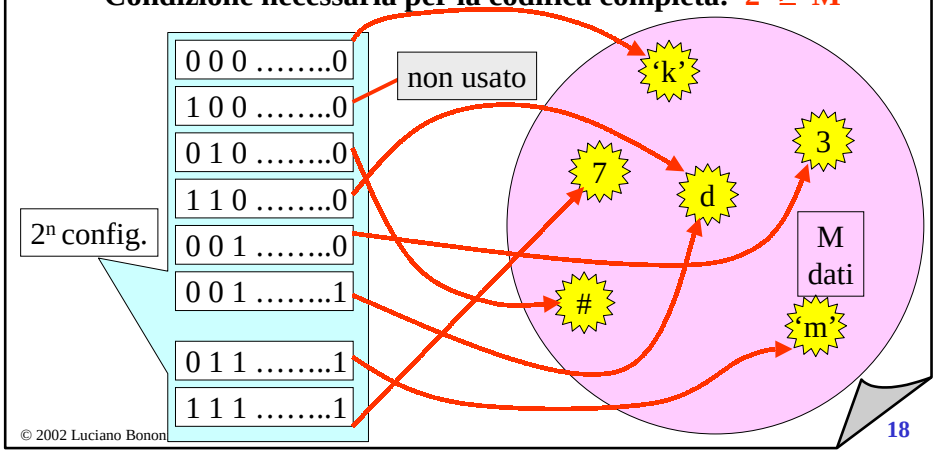


Idea: sfrutto tutte le possibili combinazioni diverse di 8 bit
256 simboli

0	0	0	0	0	0	0	0	= simbolo A
0	0	0	0	0	0	0	1	= simbolo B
0	0	0	0	0	0	1	0	= simbolo C
0	0	0	0	0	0	1	1	= simbolo E
0	0	0	0	0	1	0	0	= simbolo F
⋮	⋮	⋮	⋮	⋮	⋮	⋮	⋮	
1	1	1	1	1	1	1	0	= simbolo %
1	1	1	1	1	1	1	1	= simbolo $

Codici binari: convenzioni per rappresentare info.

Funzioni dall'insieme delle 2^n configurazioni di n bit ad un insieme di M informazioni o dati
(*valori, simboli, istruzioni,* ecc.**).**
Condizione necessaria per la codifica completa: $2^n \geq M$



Codici binari: quanti sono?

La scelta di un codice è condivisa da sorgente e destinazione ed ha due gradi di libertà:

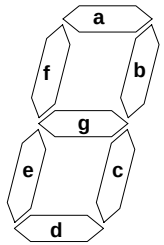
- **il numero di bit n (qualsiasi, a patto che sia $2^n \geq M$)**

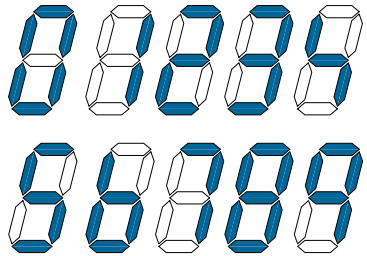
- **l'associazione tra configurazioni e informazioni; a parità di n e di M le associazioni possibili sono**

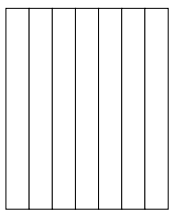
$$C = 2^n! / (2^n - M)!$$

n = 1, M = 2	C = 2 (logica pos. e neg.)
n = 2, M = 4	C = 24
n = 3, M = 8	C = 64.320
n = 4, M = 10	C = 29.000.000.000

Es. codici speciali: codice 7 segmenti







A=?
b=?
C=?
.....

Sistemi di numerazione

Posizionali

• il valore di un simbolo dipende dalla posizione che esso occupa all'interno della configurazione, seguendo una legge nota. I vari sistemi di numerazione posizionale differiscono per la scelta della base B. La base B indica il numero di simboli usati.

• **decimale B=10**, **binario B=2**, **ottale B=8**, **esadecimale B=16**

Non posizionali

Il valore di un simbolo non dipende dalla posizione che esso occupa all'interno della configurazione. (es: Numeri Romani)

Interpretazione (funzione valore)

• Nei sistemi posizionali, i simboli di una configurazione possono essere interpretati come i coefficienti del seguente polinomio [1] (detto funzione Valore)

$$V = \sum_{i=-m}^{n-1} d_i \cdot B^i$$

B = base
d_i = i-esima cifra $\in$ [0..B-1]
n = numero di cifre parte intera
m = numero di cifre parte frazionaria

La virgola e' posta tra le cifre di posizione 0 e –1.

Interpretazione (funzione valore)

Esempio: sistema decimale

Il numero 245.6 decimale può essere rappresentato come segue:

B = 10	base
n=3	numero cifre parte intera
m=1	numero cifre parte frazionaria
d = {2,4,5,6}	insieme delle cifre

cifra	2	4	5	6
posizione	2	1	0	-1
peso	10^2	10^1	10^0	10^{-1}

$$V = \sum_{i=-m}^{n-1} d_i \cdot B^i \quad = 2 \bullet 10^2 + 4 \bullet 10^1 + 5 \bullet 10^0 + 6 \bullet 10^{-1} = 245.6$$

Esempio di interpretazione valore binario naturale

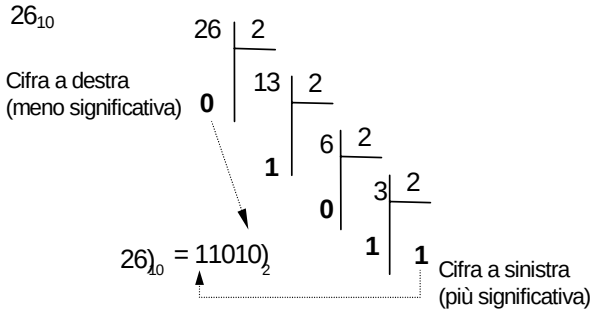
Esempio: Quale è il valore decimale corrispondente al numero binario 1101.010_2 ?

$cifra_2$	1	1	0	1	.	0	1	0...
peso	2^3	2^2	2^1	2^0	.	2^{-1}	2^{-2}	2^{-3}
valore	1•8	1•4	0•2	1•1	.	0•1/2	1•1/4	0•1/8

$1101.010_2 = 1 \bullet 2^3 + 1 \bullet 2^2 + 0 \bullet 2^1 + 1 \bullet 2^0 + 0 \bullet 2^0 + 1 \bullet 2^0 = 13.25_{10}$

Metodo della divisione: Base 10 -> Base B

Es. base 2:
Per valori interi positivi:
Per ottenere il valore in base B, di un numero intero codificato nel sistema decimale, si procede utilizzando un metodo iterativo di successive divisioni per la base: al termine del procedimento i resti delle divisioni, dall'ultimo al primo, rappresentano il valore iniziale in base B.

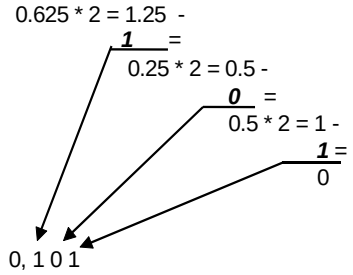


Metodo della divisione: Base 10 -> binario naturale

Es. base 2
Per valori con cifre decimali: si separa la parte intera da quella frazionaria, La parte intera si calcola come nel caso precedente La parte frazionaria si ottiene come segue:

1. Si moltiplica la parte frazionaria per 2
2. Se il numero ottenuto è maggiore di 1, si sottrae 1 e si considera come prima cifra dopo la virgola un '1'.
3. Se invece il numero è nella forma 0,….. => la cifra da inserire è uno '0'.
4. Si ripete dal passo 1 fino a che il numero di partenza non è zero.

Esempio: $0.625_{10} = 0.101_2$



Somma di valori in binario naturale

Assumiamo di avere solo n=3 bit a disposizione per rappresentare i valori e il risultato, quindi posso rappresentare B^n = 8 valori diversi interi positivi (da 0 a 7).

$5_{10}+1_{10}=6_{10}$

```
  1
1 0 1 +
0 0 1 =
-------
1 1 0
```

$2_{10}+3_{10}=5_{10}$

```
  1
0 1 0 +
0 1 1 =
-------
1 0 1
```

$5_{10}+3_{10}=8_{10}$

```
1 1 1
  1 0 1 +
  0 1 1 =
---------
1 0 0 0
```

Overflow!!

Per rappresentare il risultato della somma di $5_{10} + 3_{10}$ sono necessari 4 bit !

Altre basi: ottale e esadecimale

Quando per la rappresentazione di un numero si utilizzano molte cifre binarie può convenire usare altri sistemi di numerazione.

I sistemi ottale ed esadecimale sono utilizzati principalmente per rappresentare in modo più compatto i numeri binari.

L'algoritmo della divisione per passare da base 10 a ottale o esadecimale vale anche in questo caso. Provare.

I simboli del sistema **Ottale** sono 8 (da 0 a 7):
{ 0, 1, 2, 3, 4, 5, 6, 7, 10, 11, 12, 13, 14, 15, 16, 17, 20, ...}

I simboli del sistema **Esadecimale** sono 16 (da 0 a F):
{ 0, 1, 2, 3, 4, 5, 6, 7, 8, 9, A, B, C, D, E, F, 10, 11, 12, 13, 14, ...}

Cambiamenti di base (metodo veloce)

Esiste un metodo veloce quando base di partenza BP e di arrivo BA sono esprimibili come BA=BP^k.

BP:Binario -> BA:Ottale , K=3, (8=2^3)

Per passare dalla codifica Binaria a quella Ottale, si raggruppano le cifre binarie a gruppi di 3 (**a partire da destra, allineandosi alla virgola**) e le si sostituiscono con una cifra del sistema ottale.

Esempio : $111001010_2 = 712_8$

Ottale -> Binario

Per passare dalla codifica Ottale a quella Binaria, si sostituisce ad ogni cifra ottale la corrispondente codifica binaria (composta da 3 cifre).

Esempio : $302_8 = 011000010_2$

Cambiamenti di base (metodo veloce)

BP: Binario -> BA:esadecimale , K=4, (16=2^4)

Per passare dal codice Binario a quello Esadecimale, si raggruppano le cifre a gruppi di 4 (a partire da destra) e le si sostituiscono con una cifra del sistema esadecimale.

Esempio : $100100011111_2 = 91F_{16}$

Esadecimale -> Binario

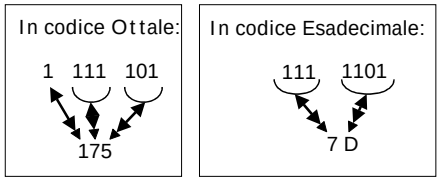
Per passare dal codice Esadecimale a quello Binario, si sostituisce ad ogni cifra esadecimale la corrispondente configurazione binaria (composta da 4 cifre).

Esempio : $A7F_{16} = 101001111111_2$

Esempi

Esempio 1

Codifica del numero $125_{10} = 1111101_2$



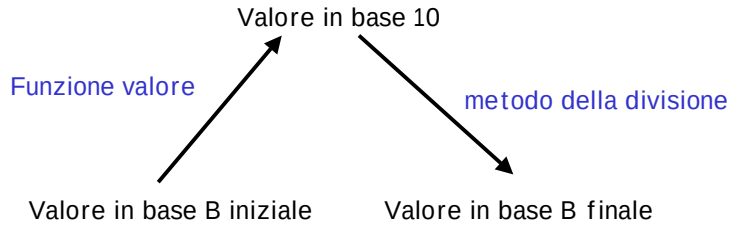
Esempio 2

Decimale	Binario	Ottale	Esadecimale
5	101	5	5
12	1100	14	C
78	1001110	116	4E
149	10010101	225	95

Esempi

Decimale	Binario **dcba**	Ottale	Esadecimale
0	0000	00	0
1	0001	01	1
2	0010	02	2
3	0011	03	3
4	0100	04	4
5	0101	05	5
6	0110	06	6
7	0111	07	7
8	1000	10	8
9	1001	11	9
10	1010	12	A
11	1011	13	B
12	1100	14	C
13	1101	15	D
14	1110	16	E
15	1111	17	F

Riassunto: passaggi di base generici



Codifica dei caratteri ASCII (7 bit -> 128 caratteri)

Il codice ASCII è non ridondante, perchè i simboli che vengono codificati sono in numero pari alle configurazioni ottenibili con 7 cifre binarie.

	000	001	010	011	100	101	110	111
0000	NUL	DLE		0	@	P	°	p
0001	SOH	DC1	!	1	A	Q	a	q
0010	STX	DC2	“	2	B	R	b	r
0011	ETX	DC3	#	3	C	S	c	s
0100	EOT	DC4	$	4	D	T	d	t
0101	ENQ	NAK	%	5	E	U	e	u
0110	ACK	SYN	&	6	F	V	f	v
0111	BEL	ETB	‘	7	G	W	g	w
1000	BS	CAN	(	8	H	X	h	x
1001	HT	EM	)	9	I	Y	i	y
1010	LF	SUB	*	:	J	Z	j	z
1011	VT	ESC	+	;	K	[	k	{
1100	FF	FS	,	<	L	\	l	|
1101	CR	GS	-	=	M	]	m	}
1110	SO	RS	.	>	N	^	n	~
1111	SI	US	/	?	O	_	o	DEL

MSB

LSB

Rappresentazione binaria di numeri interi

Nella rappresentazione binaria di numeri dotati di segno viene tipicamente usato un bit per discriminare tra valori positivi e valori negativi. Dati n bit per la rappresentazione il bit usato per il segno è quello più significativo (More Significant Bit (MSB), in posizione n-1).

Rappresentazione *Modulo e Segno (M-S)*

In questo rappresentazione al valore assoluto del numero viene *prefisso* un bit per indicarne il segno.

Il valore 0 di questo bit codifica il segno *più* e il valore 1 il segno *meno*.

Esempio n=8 (rappresento i valori su n=8 bit):

$+57_{10} = 00111001_2$ $\qquad -57_{10} = 10111001_2$

segno **modulo**

Rappresentazione modulo e segno

La rappresentazione M-S è vantaggiosa per la sua semplicità ma richiede circuiti complessi per l’esecuzione di somme algebriche.

Prima di eseguire una somma algebrica tra due operandi A e B e’ necessario determinare quale dei due e’ maggiore in valore assoluto.

- Se A e’ maggiore di B si esegue la differenza A-B e si assegna al risultato il segno di A.
- Se A e’ minore di B si esegue la differenza B-A e si assegna al risultato il segno di B

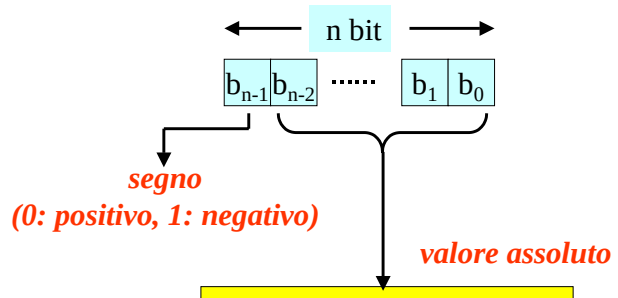
N.B.

Usando n bit (es. 8) per la codifica, il range di valori rappresentabili risulta: **$[-2^{n-1}-1..+2^{n-1}-1]$**.
Un bit (MSB) è usato per il segno.
Es. con 8 bit sono rapresentabili valori nell’intervallo [-127 ..+127].

Vi sono due configurazioni per lo zero (00000000_2 e 10000000_2) e ciò rappresenta un problema quando l’architettura deve verificare la condizione di uguaglianza a zero.

Poichè le operazioni vanno eseguite sulla sola parte di valore assoluto è semplice la determinazione dell’**overflow**

Numeri relativi: rappres. Modulo e Segno (M-S)



$$|N| = \left(b_{n-2}.2^{n-2} + \ldots + b_0.2^0\right)$$

$$N \in \left[-(2^{n-1} - 1), +(2^{n-1} - 1)\right]$$

esempio di operazioni (M-S)

$(23+ 16)_{10}$

Stesso segno, si esegue la somma escludendo il bit più significativo. Entrambi gli operandi hanno lo stesso segno (bit 7 = 0), quindi il segno viene mantenuto (bit 7 = 0)

```
                1
23₁₀ +      0010111₂ +
16₁₀ =      0010000₂ =
------    ----------
39₁₀        0100111₂
```

Risultato : $+39_{10}$ 00100111_2

$(22 - 17)_{10}$

Il primo operando ha modulo maggiore del secondo (|22| > |17|) => si esegue la differenza tra 22 e 17.
In questo caso il segno del risultato (5) è positivo e il bit 7 deve essere posto a 0.

```
                1
22₁₀ -      0010110₂  -
17₁₀ =      0010001₂  =
------    ----------
 5₁₀        0000101₂
```

Risultato : $+5_{10}$ 00000101_2

esempio di operazioni (M-S)

$(8 - 16)_{10}$

Il secondo operando ha modulo maggiore (|16| >|8|) del primo. Si esegue la differenza tra 16 e 8.
Il segno e' quello dell'operando di valore assoluto maggiore. 16 è negativo e il bit 7 deve essere posto a 1.

```
                1
16₁₀        0010000₂  -
 8₁₀        0001000₂  =
------    ----------
 8₁₀        0001000₂
```

Risultato -8_{10} 10001000_2

$(-112 - 39)_{10}$

Entrambi gli operandi di segno negativo => si sommano i valori assoluti.

```
                1
112₁₀       1110000₂  +
 39₁₀       0100111₂  =
------    ----------
151₁₀      10010111₂
```

Overflow. Sono necessari 8 bit per rappresentare 151 !
Usando la rappresentazione M-S sono disponibili per il modulo solo 7 bit => (Overflow).

Rappresentazione Complemento a uno (1's C)

Dati n bit per la codifica del modulo e del segno:

la rappresentazione di un numero negativo in complemento a 1 (1's C) si ottiene complementando tutti i bit della rappresentazione in binario naturale del corrispondente positivo, e viceversa

$$V = \sum_{i=0}^{n-2} d_i \cdot 2^i \quad \text{se } d_{n-1}=0 \qquad V = \sum_{i=0}^{n-2} (\neg d_i) \cdot 2^i \quad \text{se } d_{n-1}=1$$

- Con tale rappresentazione possono essere codificati i valori compresi nell'intervallo $[-2^{n-1}+1, 2^{n-1}-1]$.
- **Esistono due rappresentazioni per lo zero: es. per n=4, 0000, 1111**
- I numeri positivi restano inalterati, come nel binario naturale
- I numeri negativi sono calcolabili partendo dal corrispondente valore positivo, invertendo tutti i bit.

Rappresentazione Complemento a due (2's C)

Dati n bit per la codifica del modulo e del segno:

la rappresentazione in complemento a 2 (2's C) di un numero si ottiene sommando (sottraendo nel caso di numeri negativi) a 2^n il numero codificato in valore assoluto ed eliminando l'eventuale bit di riporto in posizione n;

- Con tale rappresentazione possono essere codificati i valori compresi nell'intervallo [-2^{n-1}, ($2^{n-1}-1$)].
- **Esiste solo una rappresentazione dello zero**
- I numeri positivi restano inalterati, come nel binario naturale
- I numeri negativi sono calcolabili partendo dal corrispondente valore positivo, invertendo tutti i bit (complemento a 1, 1's) e sommando 1

Rappresentazione Complemento a due (2's C)

Funzione Valore di un numero codificato in complemento a 2.

In una configurazione binaria di n bit codificata in complemento a 2, il bit più significativo (MSB in posizione n-1) assume un peso negativo pari a -2^{n-1}.

$$V_{10} = -2^{n-1} \cdot d_{n-1} + \sum_{i=0}^{n-2} d_i \cdot 2^i \qquad d_i \in [0,1]$$

I numeri positivi (d_{n-1}=0) codificati in complemento a 2 rimangono del tutto analoghi al binario naturale.

Esempio: n=4

1011_2 in complemento a 2 equivale a:

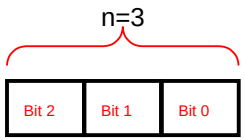
$1011_2 = -1\cdot2^3 + 0\cdot2^2 + 1\cdot2^1 + 1\cdot2^0 = -8 + 0 + 2 + 1 = -5_{10}$

Mentre 0111_2 in complemento a 2 equivale a :

$0111_2 = 0\cdot2^3 + 1\cdot2^2 + 1\cdot2^1 + 1\cdot2^0 = 0 + 4 + 2 + 1 = +7_{10}$

Rappresentazione Complemento a due (2's C)

Siano dati **3 bit** (n=3) per la rappresentazione di numeri con segno



Con tale configurazione di bit potranno essere codificati i numeri da –4 a +3, cioè [-2^2, 2^2-1]

La rappresentazione in complemento a 2 (2's C) si ottiene come segue:

si somma (o si sottrae) a $2^3 = 8_{10} = 1000_2$ la rappresentazione in valore assoluto del numero che si vuole rappresentare in 2's C,

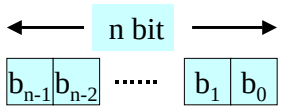
Es +1: Per i numeri positivi si somma a 1000_2 il modulo del numero ($|1_{10}|=001_2$).

Quindi $+1_{10} = 1000_2 + 001_2 = x001_2$ in complemento a 2

Es -1: Per i numeri negativi si sottrae da 1000_2 il modulo del numero $|1_{10}|=001_2$.

Quindi $-1_{10} = 1000_2 - 001_2 = x111_2$ in complemento a 2

Numeri relativi: rappres. 2'sC



Esempi (*n=4*)

+1 = 0 0 0 1	+7 = 0 1 1 1
-1 = 1 1 1 0 +	- 7 = 1 0 0 0 +
1	1
= 1 1 1 1	= 1 0 0 1

N.B. - anche nella rappresentazione in complemento a 2 il bit più significativo indica il segno (0:positivo, 1:negativo).

$$N = -2^{n-1} \cdot b_{n-1} + b_{n-2} \cdot 2^{n-2} + \ldots + b_0 \cdot 2^0$$

$$N \in \left[-2^{n-1}, +(2^{n-1} - 1)\right]$$

Rappresentazione Complemento a due (2's C)

N.B.
Come ci si attende, applicando due volte la regola del complemento a 2 si ottiene il numero originale.

Esempio

n = 4 , voglio rappresentare –5 a partire da 5 = 0101_2

-5_{10} in complemento a 2 risulta 1011_2

Applicando nuovamente il complemento a 2 si ottiene il valore assoluto del numero

1's +1
-5 -> 1011_2 -> 0100_2 -> 0101_2 = + 5_{10}

Vantaggi del Complemento a due

- Vi è una sola rappresentazione per lo zero (00…000), quindi i confronti di uguaglianza a zero sono veloci
- non vi e' differenza nell'eseguire somme o sottrazioni, quindi posso usare lo stesso circuito per le somme e sottrazioni, a condizione di assumere i valori espressi in 2'sC. Infatti: (A – B) = A + (-B)
-Inoltre, per la sottrazione non e' necessario individuare il maggiore, in valore assoluto, tra i due operandi come nel caso della rappresentazione S-VA.

Esercizio di verifica
Utilizzando una rappresentazione 2'sC a 4 bit, e facendo le somme, calcolare

```
 3   +1 =  ?          [Soluzione  0100]
 3   -1 =  ?          [Soluzione  0010]
-1   -2 =  ?          [Soluzione  1101]
 3   -7 =  ?          [Soluzione  1100]
```


Gestione dell'overflow

M-S, 1'sC

Nel caso della rappresentazione con modulo e segno e 1'sC, la presenza di eventuali situazioni overflow puo' essere rilevata analizzando il bit di *carry-out* relativo al bit più significativo del modulo (da notare che in 1'sC occorre anche complementare i bit dei valori negativi prima della somma o sottrazione).

2's C

Nel caso di somme algebriche con numeri rappresentati in complemento a 2 la rilevazione della condizione di overflow si ottiene controllando se il bit di *carry-in* e il bit di *carry_out* relativi al bit più significativo (il bit n-1) della codifica sono diversi. Questa operazione puo' essere eseguita utilizzando l'operatore logico or-esclusivo.

Esempi: Con **N=3** possono essere rappresentati i numeri tra [+3, -4] in 2's C

3+3 = 6 (overflow)	2+1 = 3	-3-3 = -6 (overflow)	-3-1 = -4	-3+2 = -1
0 1 1	0 0 0	1 0 1	1 1 1	0 0 0
0 1 1+	0 1 0+	1 0 1+	1 0 1+	1 0 1+
0 1 1 =	0 0 1 =	1 0 1 =	1 1 1 =	0 1 0 =
0 1 1 0	0 0 1 1	1 0 1 0	1 1 0 0	0 1 1 1

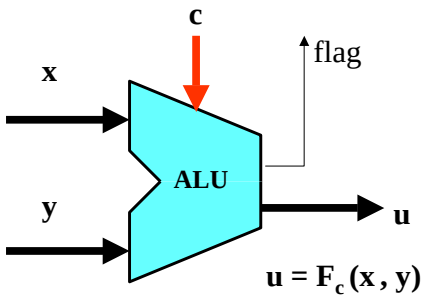
Gestione dell'overflow

2's C

Nel caso di somme algebriche con numeri rappresentati in complemento a 2 la rilevazione della condizione di overflow si ottiene solo se i due valori da sommare sono concordi (stesso segno) e il risultato è discorde (segno diverso), oppure se i due valori da sottrarre sono discordi e il risultato è discorde dal primo operando (infatti basta notare che dati A-B discordi/concordi ciò equivale a A+B concordi/discordi).

Somma A e B concordi	A>0	B>0	A+B > 0 OK	A+B<0 Overflow
	A<0	B<0	A+B > 0 Overflow	A+B < 0 OK
Sottrazione A e B discordi	A>0	B<0	A-B > 0 OK	A-B < 0 Overflow
	A<0	B>0	A-B > 0 Overflow	A-B < 0 OK

ALU: Aritmetical Logical Unit



ALU - Rete combinatoria in grado di eseguire diverse operazioni di tipo aritmetico o logico. L'operazione di volta in volta eseguita dipende dal valore attribuito ai bit di programmazione (codice operazione C)

Rappresentazione binaria polarizzata

- Dati n bit, esiste una Base di Polarizzazione (BP) arbitraria
  - es. n=8, Base di polarizzazione = 01111111 = valore 127
- La BP una volta fissata, serve a rappresentare il valore Zero
- Le rimanenti stringhe di n bit, ordinate per valori crescenti, rappresentano i valori interi dell'intorno dello zero.
- La base di polarizzazione può essere spostata per consentire la rappresentazione di intervalli asimmetrici di interi positivi e negativi.
  N.B. rappresentazione ambigua finchè non viene fissata la BP!!!
- Per passare da valore a rappresentazione polarizzata
  - esprimo il valore in binario 2'sC e gli sommo la BP
- Per passare da rappresentazione polarizzata a valore
  - sottraggo la BP ottenendo il valore in binario naturale

Rappresentazione binaria polarizzata (pol)

- Esempio: "Valore" = val(stringa BinarioNaturale)-BP;
  stringaPolarizzata = "Valore" + BP

v. binario naturale	v. polarizzata (BP = 127)	v. polarizzata (BP = 128)
11111111	(-BP) + (2^n)-1 = 128	(-BP) + (2^n)-1 = 127
11111110	(-BP) + (2^n)-2 = 127	(-BP) + (2^n)-2 = 126
⋮	⋮	⋮
10000001	(-BP) + BP+2 = +2	(-BP) + BP+1 = +1
BP= 10000000	(-BP) + BP+1 = +1	(-BP) + BP = 0
BP= 01111111	(-BP) + BP = 0	(-BP) + BP-1= -1
01111110	(–BP) + BP-1 = -1	(–BP) + BP-2 = -2
⋮	⋮	⋮
00000011	(–BP) + 3 = - 124	(–BP) + 3 = - 125
00000010	(–BP) + 2 = -125	(–BP) + 2 = -126
00000001	(–BP) + 1 = -126	(–BP) + 1 = -127
00000000	(–BP) + 0 = -127	(–BP) + 0 = -128

Rappresentazione Standard IEEE 754

- Serve a rappresentare valori in virgola mobile (floating point)
  - •singola precisione (32 bit)
  - •doppia precisione (64 bit)
- Serve idealmente a rappresentare valori "reali", ovviamente con criteri di approssimazione, su un intervallo di rappresentazione molto ampio
- •Permette un abile compromesso tra densità dei valori e ampiezza del range di rappresentazione.
- Sfrutta la notazione scientifica
  - •Valore = ddddd.ddddd x B^k,
    es. 12345.678 x 10^(-3) = 12.345678 x 10^0 = 12.345678
    es. 110010.1001 x 2^(+3) = 110010100.1 x 2^0 = 110010100.1

Mantissa Esponente

Rappresentazione Standard IEEE 754

- La notazione scientifica si dice normalizzata se nella mantissa esiste una sola cifra a sinistra della virgola, diversa da zero
- in binario ciò equivale ad avere un solo UNO a sinistra della virgola
- Per normalizzare una mantissa (senza modificare il valore) posso spostare la virgola a destra dell'UNO più significativo e
  - aumentare di uno l'esponente per ogni spostamento della virgola a sinistra
    (equivale a moltiplicare per la Base incrementando l'esponente, e a dividere per la base spostando la virgola a sinistra)
  - decrementare di uno l'esponente per ogni spostamento della virgola a destra
    (equivale a dividere per la Base decrementando l'esponente, e a moltiplicare per la base spostando la virgola a destra)

es. 110010.1001 x 2^(3) = 1.100101001 x 2^8 = 110010100.1

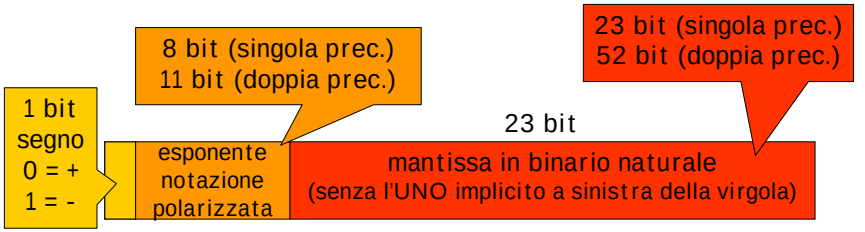
es. 0.000101001 x 2^(-2) = 1.01001000 x 2^(-6) = 0.00000101001

Rappresentazione Standard IEEE 754

Dato un valore in forma scientifica normalizzata

es. 110010.1001 x 2^(3) = 1.100101001 x 2^8 = 110010100.1

lo posso portare in notazione IEEE 754 attraverso questa forma:



- Di solito la Base di Polarizzazione per l'esponente è 01111111 = 127
  - i possibili esponenti potrebbero variare da –127 a +128
- es. 0 10000111 1001010010...0 è il valore +1.100101001x2^8 e quindi equivale a 110010100.1 in binario naturale = 404.5 in base 10