

## TECNOLOGIA DIGITALE

CPU, memoria centrale e dispositivi sono realizzati con **tecnologia elettronica digitale**

Dati ed operazioni vengono codificati a partire da due valori distinti di grandezze elettriche:

- tensione alta ( $V_H$ , ad es. 5V)
- tensione bassa ( $V_L$ , ad es. 0V)

**In generale presenza od assenza di un fenomeno.**

A tali valori vengono convenzionalmente **associate le due cifre binarie 0 e 1:**

- logica positiva:  $1 \leftrightarrow V_H$ ,  $0 \leftrightarrow V_L$
- logica negativa:  $0 \leftrightarrow V_H$ ,  $1 \leftrightarrow V_L$

1

## TECNOLOGIA DIGITALE (segue)

Dati ed operazioni vengono codificati tramite **sequenze di bit 8 bit = 1 byte**

**01000110101 ....**

CPU è in grado di operare soltanto in aritmetica binaria, effettuando operazioni *elementari*:

- somma e differenza
- scorrimento (shift)
- ...

Lavorando direttamente sull'hardware, **l'utente è forzato a esprimere i propri comandi al livello della macchina, tramite sequenze di bit**

2

## RAPPRESENTAZIONE DELL'INFORMAZIONE

- Internamente a un elaboratore, ogni informazione è **rappresentata** tramite **sequenze di bit (cifre binarie)**
- Una sequenza di bit **non dice "che cosa" essa rappresenta**

Ad esempio, 01000001 può rappresentare:

- l'intero 65, il carattere 'A', il boolean 'vero', ...
- ... il valore di un segnale musicale,
- ... il colore di un puntino sullo schermo...

3

## Rapida Nota sulla Rappresentazione dei Caratteri

Ad esempio, un tipo fondamentale di dato da rappresentare è costituito dai **singoli caratteri**

Idea base: associare **a ciascun carattere un numero intero (codice)** in modo convenzionale

➡ **Codice standard ASCII** (1968)

ASCII definisce univocamente i primi 128 caratteri (7 bit – vedi tabella nel lucido seguente)

I caratteri con codice superiore a 127 possono variare secondo la particolare codifica adottata (dipendenza da linguaggio naturale: ISO 8859-1 per alfabeto latino1, ...)

Visto che i caratteri hanno un codice intero, essi possono essere considerati un insieme ordinato (ad esempio: 'g' > 'O' perché 103 > 79)

4

Tabella ASCII standard

Byte	Cod	Char	Byte	Cod	Char	Byte	Cod	Char	Byte	Cod	Char
00000000	0	Null	00100000	32	Spc	01000000	64	@	01100000	96	~
00000001	1	Start of heading	00100001	33	!	01000001	65	A	01100001	97	a
00000010	2	Start of text	00100010	34	"	01000010	66	B	01100010	98	b
00000011	3	End of text	00100011	35	#	01000011	67	C	01100011	99	c
00000100	4	End of transmit	00100100	36	\$	01000100	68	D	01100100	100	d
00000101	5	Enquiry	00100101	37	%	01000101	69	E	01100101	101	e
00000110	6	Acknowledge	00100110	38	&	01000110	70	F	01100110	102	f
00000111	7	Audible bell	00100111	39	'	01000111	71	G	01100111	103	g
00001000	8	Backspace	00101000	40	(	01001000	72	H	01101000	104	h
00001001	9	Horizontal tab	00101001	41	)	01001001	73	I	01101001	105	i
00001010	10	Line feed	00101010	42	*	01001010	74	J	01101010	106	j
00001011	11	Vertical tab	00101011	43	+	01001011	75	K	01101011	107	k
00001100	12	Form Feed	00101100	44	,	01001100	76	L	01101100	108	l
00001101	13	Carriage return	00101101	45	-	01001101	77	M	01101101	109	m
00001110	14	Shift out	00101110	46	.	01001110	78	N	01101110	110	n
00001111	15	Shift in	00101111	47	/	01001111	79	O	01101111	111	o
00010000	16	Data link escape	00110000	48	0	01010000	80	P	01110000	112	p
00010001	17	Device control 1	00110001	49	1	01010001	81	Q	01110001	113	q
00010010	18	Device control 2	00110010	50	2	01010010	82	R	01110010	114	r
00010011	19	Device control 3	00110011	51	3	01010011	83	S	01110011	115	s
00010100	20	Device control 4	00110100	52	4	01010100	84	T	01110100	116	t
00010101	21	Neg. acknowledge	00110101	53	5	01010101	85	U	01110101	117	u
00010110	22	Synchronous idle	00110110	54	6	01010110	86	V	01110110	118	v
00010111	23	End trans. block	00110111	55	7	01010111	87	W	01110111	119	w
00011000	24	Cancel	00111000	56	8	01011000	88	X	01111000	120	x
00011001	25	End of medium	00111001	57	9	01011001	89	Y	01111001	121	y
00011010	26	Substitution	00111010	58	:	01011010	90	Z	01111010	122	z
00011011	27	Escape	00111011	59	;	01011011	91	[	01111011	123	{
00011100	28	File separator	00111100	60	<	01011100	92	\	01111100	124	
00011101	29	Group separator	00111101	61	=	01011101	93	]	01111101	125	}
00011110	30	Record Separator	00111110	62	>	01011110	94	^	01111110	126	~
00011111	31	Unit separator	00111111	63	?	01011111	95	_	01111111	127	Del

## INFORMAZIONI NUMERICHE

Originariamente la **rappresentazione binaria** è stata utilizzata per la **codifica dei numeri e dei caratteri**

Oggi si digitalizzano comunemente anche suoni, immagini, video e altre informazioni (informazioni multimediali)

La rappresentazione delle **informazioni numeriche** è ovviamente di particolare rilevanza

A titolo di esempio, nel corso ci concentreremo sulla rappresentazione dei **numeri naturali (interi senza segno)**

$$\text{Dominio: } N = \{0, 1, 2, 3, \dots\}$$

## NUMERI NATURALI (interi senza segno)

Dominio:  $N = \{0, 1, 2, 3, \dots\}$

Rappresentabili con diverse notazioni

◆ **non posizionali**

- ad esempio la notazione romana: I, II, III, IV, V, .... IX, X, XI...
- Risulta difficile l'utilizzo di regole generali per il calcolo

◆ **posizionale**

- 1, 2, .. 10, 11, ... 200, ...
- Risulta semplice l'individuazione di regole generali per il calcolo

## NOTAZIONE POSIZIONALE

- Concetto di **base di rappresentazione B**
- Rappresentazione del numero come **sequenza di simboli (cifre)** appartenenti a un **alfabeto di B simboli distinti**
- ogni simbolo rappresenta un valore compreso fra 0 e B-1

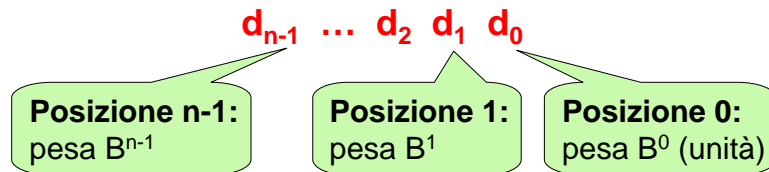
Esempio di rappresentazione su N cifre:

$$d_{n-1} \dots d_2 d_1 d_0$$

## NOTAZIONE POSIZIONALE

Il **valore di un numero** espresso in questa notazione è ricavabile

- ◆ a partire dal valore rappresentato da ogni simbolo
- ◆ **pesandolo in base alla posizione** che occupa nella sequenza



9

## NOTAZIONE POSIZIONALE

In formula:

$$v = \sum_{k=0}^{n-1} d_k B^k$$

dove

- ◆  $B = \text{base}$
- ◆ ogni cifra  $d_k$  rappresenta un valore fra 0 e  $B-1$

Esempio (base  $B=4$ ):

1 2 1 3  
 $d_3 d_2 d_1 d_0$

Valore =  $1 * B^3 + 2 * B^2 + 1 * B^1 + 3 * B^0 = \text{centotre}$

10

## NOTAZIONE POSIZIONALE

Quindi, **una sequenza di cifre non è interpretabile** se non si precisa **la base** in cui è espressa

Esempi:

Stringa	Base	Alfabeto	Calcolo valore	Valore
"12"	quattro	{0,1,2,3}	$4 * 1 + 2$	sei
"12"	otto	{0,1,...,7}	$8 * 1 + 2$	dieci
"12"	dieci	{0,1,...,9}	$10 * 1 + 2$	dodici
"12"	sedici	{0,...,9, A, F}	$16 * 1 + 2$	diciotto

11

## NOTAZIONE POSIZIONALE

Inversamente, ogni numero può essere espresso, **in modo univoco, come sequenza di cifre in una qualunque base**

Esempi:

Numero	Base	Alfabeto	Rappresentazione
venti	due	{0,1}	"10100"
venti	otto	{0,1,...,7}	"24"
venti	dieci	{0,1,...,9}	"20"
venti	sedici	{0,...,9, A, F}	"14"

**Non bisogna confondere un numero con una sua RAPPRESENTAZIONE!**

12

## NUMERI E LORO RAPPRESENTAZIONE

- **Internamente**, un elaboratore adotta per i **numeri interi (non negativi)** una **rappresentazione binaria (base B=2)**
- **Esternamente**, le costanti numeriche che scriviamo nei programmi e i valori che stampiamo a video/leggiamo da tastiera sono invece **sequenze di caratteri ASCII**

Il passaggio dall'una all'altra forma richiede dunque un **processo di conversione**

13

## Esempio: RAPPRESENTAZIONE INTERNA/ESTERNA

- Numero: *centoventicinque*
- Rappresentazione interna binaria (16 bit):

00000000 01111101

- Rappresentazione esterna in base 10:

occorre produrre la **sequenza di caratteri ASCII '1', '2', '5'**

00110001 00110010 00110101

vedi tabella ASCII

## Esempio: RAPPRESENTAZIONE INTERNA/ESTERNA

- Rappresentazione esterna in base 10:

È data la **sequenza di caratteri ASCII**  
'3', '1', '2', '5', '4'

vedi tabella ASCII

00110011 00110001 00110010 00110101 00110100

- Rappresentazione interna binaria (16 bit):

01111010 00010110

- Numero:

*trentunomiladuecentocinquantaquattro*

15

## CONVERSIONE STRINGA/NUMERO

Si applica la definizione:

$$v = \sum_{k=0}^{n-1} d_k B^k$$

=  $d_0 + B * ( d_1 + B * ( d_2 + B * ( d_3 + ... )))$

le cifre  $d_k$  sono note, il valore  $v$  va calcolato

Ciò richiede la valutazione di un polinomio

→ **Metodo di Horner**

16

## CONVERSIONE NUMERO/STRINGA

- Problema: **dato un numero, determinare la sua rappresentazione in una base data**
- Soluzione (**notazione posizionale**): **manipolare la formula** per dedurre un algoritmo

$$v = \sum_{k=0}^{n-1} d_k B^k$$

v è noto,  
le cifre  $d_k$  vanno calcolate

$$= d_0 + B * ( d_1 + B * ( d_2 + B * ( d_3 + ... )))$$

17

## CONVERSIONE NUMERO/STRINGA

- Per trovare le cifre bisogna **calcolarle una per una**, ossia bisogna trovare un modo per **isolarne una dalle altre**

$$v = d_0 + B * (...)$$

Osservazione:

**$d_0$  è la sola cifra non moltiplicata per  $B$**

Conseguenza:

**$d_0$  è ricavabile come  $v$  modulo  $B$**

18

## CONVERSIONE NUMERO/STRINGA

### Algoritmo delle divisioni successive

- si divide  $v$  per  $B$ 
  - **il resto** costituisce la cifra meno significativa ( $d_0$ )
  - **il quoziente** serve a iterare il procedimento
- **se tale quoziente è zero, l'algoritmo termina;**
- **se non lo è, lo si assume come nuovo valore  $v'$ , e si itera il procedimento con il valore  $v'$**

19

## CONVERSIONE NUMERO/STRINGA

Esempi:

Numero	Base	Calcolo valore	Stringa
quattordici	4	14 / 4 = 3 con resto 2 3 / 4 = 0 con resto 3	→ "32"
undici	2	11 / 2 = 5 con resto 1 5 / 2 = 2 con resto 1 2 / 2 = 1 con resto 0 1 / 2 = 0 con resto 1	→ "1011"
sessantatre	10	63 / 10 = 6 con resto 3 6 / 10 = 0 con resto 6	→ "63"
sessantatre	16	63 / 16 = 3 con resto 15 3 / 16 = 0 con resto 3	→ "3F"

20

## NUMERI NATURALI: valori rappresentabili

- Con  $N$  bit, si possono fare  $2^N$  combinazioni
- Si rappresentano così i numeri da 0 a  $2^N-1$

### Esempi

□ con 8 bit, [ 0 .... 255 ]

In C: unsigned char = byte

□ con 16 bit, [ 0 .... 65.535 ]

In C: unsigned short int (su alcuni compilatori)

In C: unsigned int (su alcuni compilatori)

□ con 32 bit, [ 0 .... 4.294.967.295 ]

In C: unsigned int (su alcuni compilatori)

In C: unsigned long int (su molti compilatori)

21

## OPERAZIONI ED ERRORI

La rappresentazione binaria rende possibile fare **addizioni e sottrazioni con le usuali regole algebriche**

Esempio:

```
5 +      0101
3 =      0011
---
8        1000
```

22

## ERRORI NELLE OPERAZIONI

Esempio (supponendo di avere solo 7 bit per la rappresentazione)

```
60 +      01111100
75 =      11000111
-----
135       10011111
```

Errore!  
Massimo numero  
rappresentabile:  
 $2^7-1$  cioè 127

- Questo errore si chiama **overflow**
- Può capitare **sommando due numeri dello stesso segno** il cui risultato non sia rappresentabile utilizzando **il numero massimo di bit designati**

23

## ESERCIZIO RAPPRESENTAZIONE

Un elaboratore rappresenta numeri interi su **8 bit** dei quali **7** sono dedicati alla rappresentazione del modulo del numero e **uno** al suo **segno**. Indicare come viene svolta la seguente operazione aritmetica:

$59 - 27$

in codifica binaria

24

## ESERCIZIO RAPPRESENTAZIONE

---

### Soluzione

59 →<sub>0</sub> 0111011

-27 →<sub>1</sub> 0011011

Tra i (moduli dei) due numeri si esegue una sottrazione:

```
  0111011
-  0011011
-----
  0100000
```

che vale 32 in base 10

25

## Differenze tra numeri binari

---

Che cosa avremmo dovuto fare se avessimo avuto  
27-59 ???

**Avremmo dovuto invertire i due numeri,**  
calcolare il risultato, e poi ricordarci di **mettere a 1**  
**il bit rappresentante il segno**

Per ovviare a tale problema, si usa la **notazione in**  
**“complemento a 2”**, che permette di eseguire tutte le  
differenze tramite semplici somme

26