

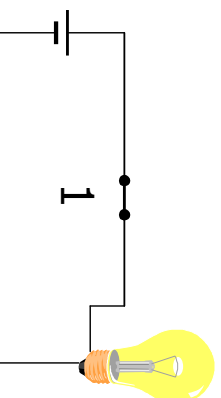
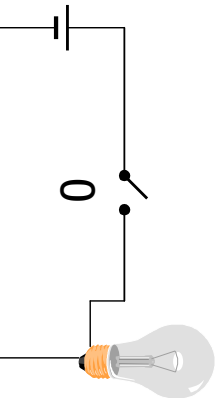


L'algebra Booleana

1

Algebra Booleana

- Contempla due costanti **0** e **1** (**falso** e **vero**)
- Corrispondono a due stati che si escludono a vicenda
- Possono descrivere lo stato di apertura o chiusura di un generico contatto o di un circuito a più contatti



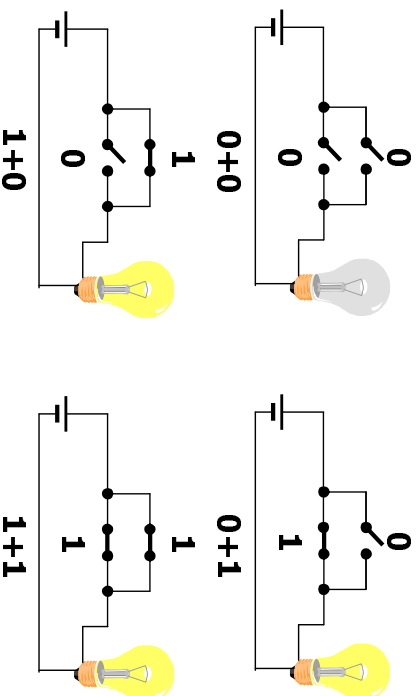
- Si definiscono delle operazioni fra i valori booleani: **AND**, **OR**, **NOT** sono gli operatori fondamentali

2

L'operazione di OR

- Si definisce l'operazione di **somma logica** (OR): il valore della somma logica è il simbolo 1 se il valore di almeno uno degli operandi è il simbolo 1

$$\begin{aligned}0+0 &= 0 \\0+1 &= 1 \\1+0 &= 1 \\1+1 &= 1\end{aligned}$$

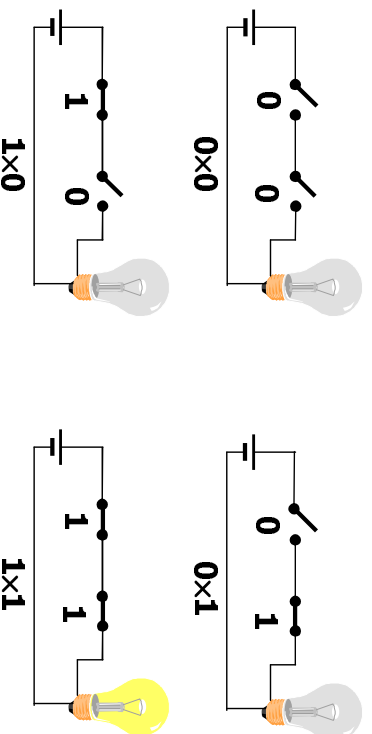


3

L'operazione di AND

- Si definisce l'operazione di **prodotto logico** (AND): il valore del prodotto logico è il simbolo 1 se il valore di tutti gli operandi è il simbolo 1

$$\begin{aligned}0\times0 &= 0 \\0\times1 &= 0 \\1\times0 &= 0 \\1\times1 &= 1\end{aligned}$$



4

La negazione NOT

- Si definisce l'operatore di **negazione** (NOT): l'operatore inverte il valore della costante su cui opera

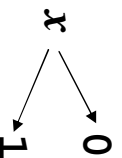
$$\begin{array}{l} \overline{0} = 1 \\ \overline{1} = 0 \end{array}$$

dalla definizione...

$$\begin{array}{l} \overline{\overline{0}} = 0 \\ \overline{\overline{1}} = 1 \end{array}$$

Variabili binarie

- Una variabile binaria indipendente può assumere uno dei due valori **0** e **1**



- Date n variabili binarie indipendenti, la loro somma logica (OR) è

$$x_1 + x_2 + \dots + x_n = \begin{cases} 1 & \text{se almeno una } x_i \text{ vale } 1 \\ 0 & \text{se } x_1 = x_2 = \dots = x_n = 0 \end{cases}$$

AND e NOT con variabili binarie

- Date n variabili binarie indipendenti, il loro prodotto logico (AND) è

$$x_1 \times x_2 \times \dots \times x_n = \begin{cases} 0 & \text{se almeno una } x_i \text{ vale } 0 \\ 1 & \text{se } x_1 = x_2 = \dots = x_n = 1 \end{cases}$$

- La negazione di una variabile x è

$$\begin{aligned} \bar{x} &= 0 & \text{se } x &= 1 \\ \bar{x} &= 1 & \text{se } x &= 0 \end{aligned}$$

Alcune identità

- Si verificano le uguaglianze

$$\begin{array}{ll} x + 1 = 1 & x \times 1 = x \\ x + 0 = x & x \times 0 = 0 \\ x + x = x & x \times x = x \end{array}$$

- Ad esempio...

$$\begin{array}{ccc} & x \times 1 = x & \\ \nearrow x = 0 & & \searrow x = 1 \\ 0 \times 1 = 0 & & 1 \times 1 = 1 \end{array}$$

OK!

Altre proprietà

- Per gli operatori AND e OR valgono le seguenti proprietà:

commutativa $x_1 + x_2 = x_2 + x_1$

$$x_1 \times x_2 = x_2 \times x_1$$

associativa $x_1 + x_2 + x_3 = x_1 + (x_2 + x_3)$

$$x_1 \times x_2 \times x_3 = x_1 \times (x_2 \times x_3)$$

distributiva del prodotto rispetto alla somma $x_1 \times x_2 + x_1 \times x_3 = x_1 \times (x_2 + x_3)$

- Per l'operatore NOT si provano le seguenti identità:

$$x + \bar{x} = 1$$

$$x \times \bar{x} = 0$$

$$\bar{\bar{x}} = x$$

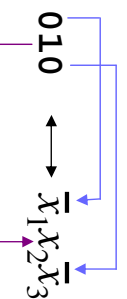
Configurazioni delle variabili

- Date n variabili binarie indipendenti x_1, x_2, \dots, x_n queste possono assumere 2^n configurazioni distinte

Ad esempio per $n=3$ si hanno 8 configurazioni

$$x_1 x_2 x_3 \begin{matrix} \text{000} & \text{001} & \text{010} & \text{011} \\ \text{100} & \text{101} & \text{110} & \text{111} \end{matrix}$$

- Una configurazione specifica è individuata univocamente da un AND (a valore 1) di tutte le variabili, dove quelle corrispondenti ai valori 0 compaiono negate



Funzioni logiche

- Una variabile y è una funzione delle n variabili indipendenti x_1, x_2, \dots, x_n se esiste un criterio che fa corrispondere in modo univoco ad ognuna delle 2^n configurazioni delle x_i un valore di y

$$y = F(x_1, x_2, \dots, x_n)$$

- Una rappresentazione esplicita di una funzione è la **tabella di verità**, in cui si elencano tutte le possibili combinazioni di x_1, x_2, \dots, x_n con associato il valore di y

$y = x_1 + x_2$ →

x_1	x_2	y
0	0	0
0	1	1
1	0	1
1	1	1

11

Una tabella di verità

- Date tre variabili booleane (A, B, C), si scriva la funzione F che vale 1 quando solo due di esse hanno valore 1

A	B	C	F
0	0	0	0
0	0	1	0
0	1	0	0
0	1	1	1
1	0	0	0
1	0	1	1
1	1	0	1
1	1	1	0

Si può scrivere la funzione come somma logica delle configurazioni corrispondenti agli 1

$$F(A, B, C) = \bar{A}BC + A\bar{B}C + A\bar{B}\bar{C}$$

Forma canonica: somma di prodotti (OR di AND)

- tutte le funzioni logiche si possono scrivere in questa forma

12

Un esempio: lo XOR

- La funzione XOR verifica la disuguaglianza di due variabili

x_1	x_2	XOR
0	0	0
0	1	1
1	0	1
1	1	0

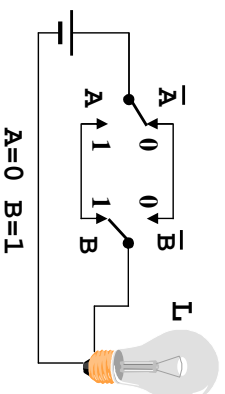
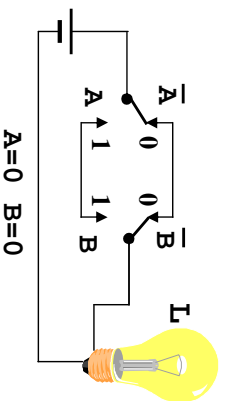
- L'espressione come somma di prodotti è quindi...

$$XOR = \bar{x}_1x_2 + x_1\bar{x}_2$$

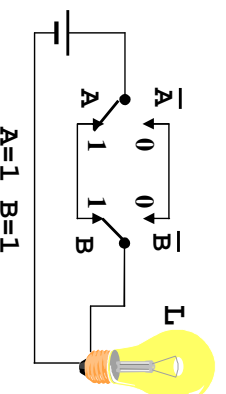
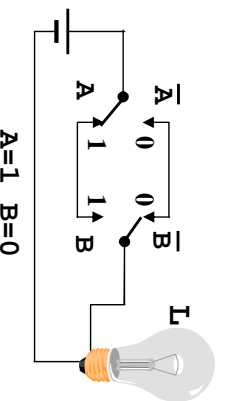
13

Un circuito con due interruttori

- I due interruttori corrispondono a due variabili (A, B) a valori booleani – le variabili assumono i due valori 0 e 1 che corrispondono alle due posizioni dell'interruttore



A	B	L
0	0	1
0	1	0
1	0	0
1	1	1

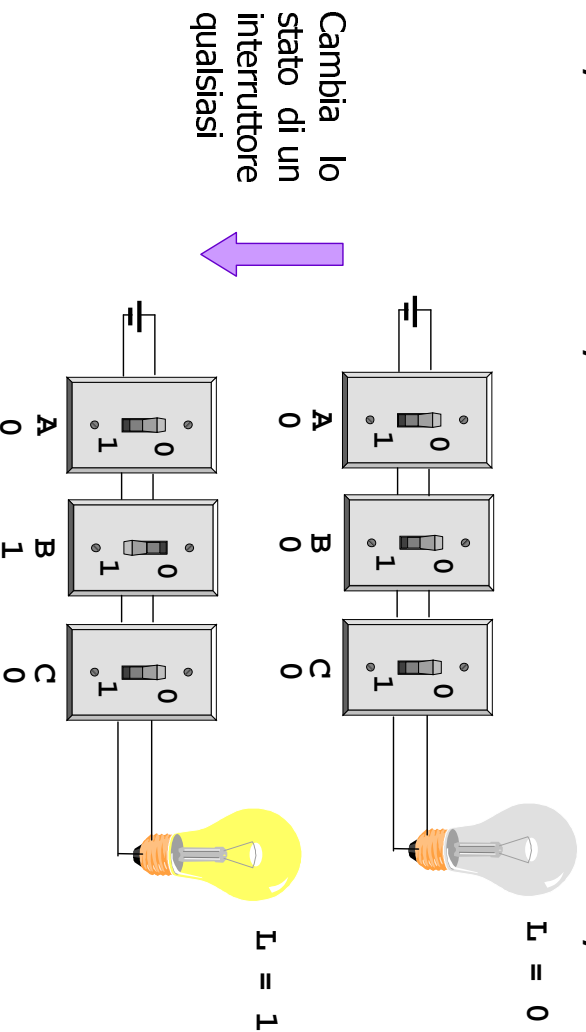


$$L = \bar{A}\bar{B} + A \times B$$

14

Un esercizio

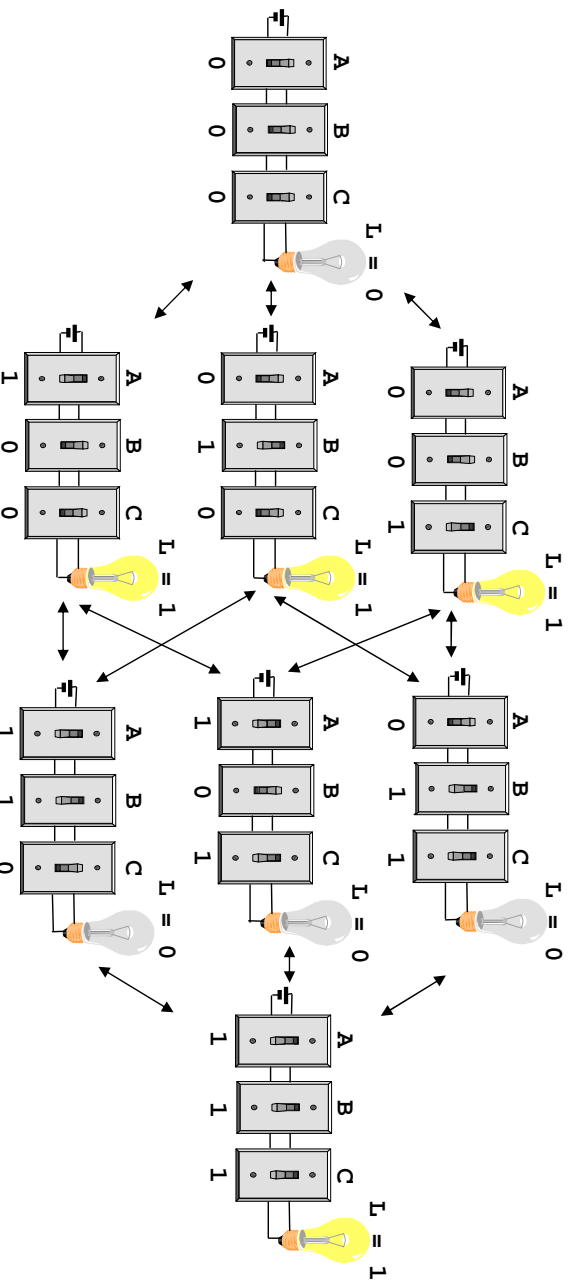
- Progettare un circuito per accendere e spegnere una lampada da uno qualsiasi di tre interruttori indipendenti



15

Analisi delle combinazioni

- Si considera cosa accade a partire dalla configurazione di partenza, cambiando lo stato di un interruttore per volta



16

Scrittura della funzione logica

- Dalle otto combinazioni si ottiene la tabella di verità della funzione logica

A	B	C	L
0	0	0	0
0	0	1	1
0	1	0	1
0	1	1	0
1	0	0	1
1	0	1	0
1	1	0	0
1	1	1	1

- Si può scrivere la funzione L come **somma logica di prodotti logici**

$$L = \bar{A}\bar{B}\bar{C}C + \bar{A}XB\bar{C} + A\bar{X}\bar{B}\bar{C} + AX\bar{B}XC$$

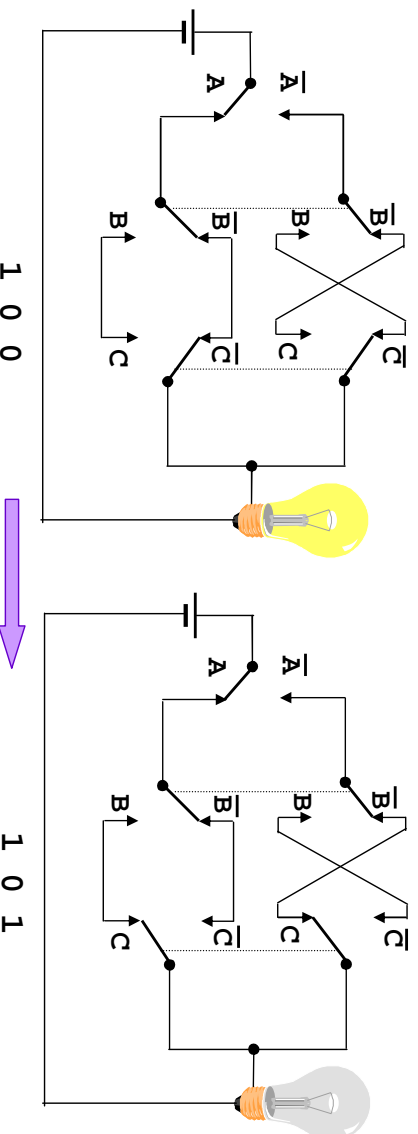
17

Come collegare gli interruttori

- Si può manipolare l'espressione di L usando la proprietà distributiva dell'AND rispetto all'OR

$$L = \bar{A}XB\bar{C}C + \bar{A}XB\bar{C} + \bar{A}XB\bar{C}C + \bar{A}XB\bar{C}C$$

$$L = \bar{A} \times (B\bar{C}C + B\bar{C}\bar{C}) + \bar{A} \times (B\bar{C}C + B\bar{C}\bar{C})$$



18



Sistemi di numerazione

19

Sistemi di numerazione

- Sistemi di numerazione **posizionali**:

La **base** del sistema di numerazione
Le **cifre** del sistema di numerazione

Il numero è scritto specificando le cifre in ordine ed il suo valore dipende dalla **posizione relativa** delle cifre

Esempio: Il sistema decimale (Base 10)

Cifre : 0 1 2 3 4 5 6 7 8 9

$$5641 = 5 \cdot 10^3 + 6 \cdot 10^2 + 4 \cdot 10^1 + 1 \cdot 10^0$$

Position: 3 2 1 0

20

Sistemi in base B

- La base definisce il numero di cifre diverse nel sistema di numerazione
- La cifra di minor valore è sempre lo 0; le altre sono, nell'ordine, 1, 2, ..., B-1; se B > 10 occorre introdurre B-10 simboli in aggiunta alle cifre decimali

Un numero **intero** N si rappresenta con la scrittura $(c_n c_{n-1} \dots c_2 c_1 c_0)_B$

$$N = c_n B^n + c_{n-1} B^{n-1} + \dots + c_2 B^2 + c_1 B^1 + c_0 B^0$$

c_n è la **cifra più significativa**, c_0 la **meno significativa**

Un numero **frazionario** N' si rappresenta come $(0, c_1 c_2 \dots c_n)_B$

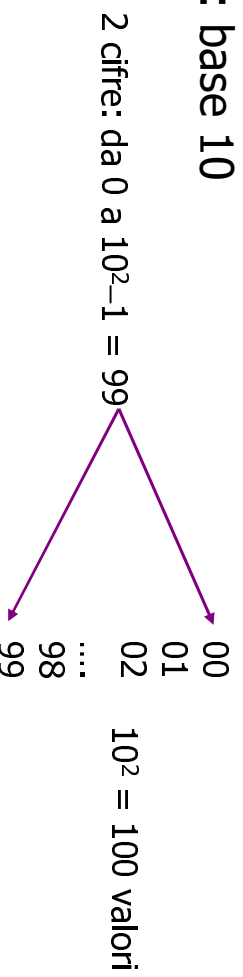
$$N' = c_1 B^{-1} + c_2 B^{-2} + \dots + c_n B^{-n}$$

21

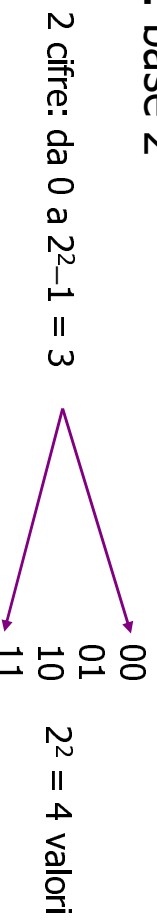
Numeri interi senza segno

- Con n cifre in base B si rappresentano tutti i numeri interi positivi da 0 a $B^n - 1$ (B^n numeri distinti)

Esempio: base 10



Esempio: base 2



22

Il sistema binario (B=2)

- La base 2 è la più piccola per un sistema di numerazione

Cifre: 0 1 – **bit** (binary digit)

Esempi:

Forma
polinomia

$$(101101)_2 = 1 \times 2^5 + 0 \times 2^4 + 1 \times 2^3 + 1 \times 2^2 + 0 \times 2^1 + 1 \times 2^0 = 32 + 0 + 8 + 4 + 0 + 1 = (45)_{10}$$

$$(0,0101)_2 = 0 \cdot 2^{-1} + 1 \cdot 2^{-2} + 0 \cdot 2^{-3} + 1 \cdot 2^{-4} = 0 + 0,25 + 0 + 0,0625 = (0,3125)_{10}$$

$$(11,101)_2 = 1 \cdot 2^1 + 1 \cdot 2^0 + 1 \cdot 2^{-1} + 0 \cdot 2^{-2} + 1 \cdot 2^{-3} = 2 + 1 + 0,5 + 0 + 0,125 = (3,625)_{10}$$

23

Dal bit al byte

- Un **byte** è un insieme di 8 bit (un numero binario a 8 cifre)

$b_7 b_6 b_5 b_4 b_3 b_2 b_1 b_0$

- Con un byte si rappresentano i numeri interi fra 0 e $2^8 - 1 = 255$

00000000
00000001
00000010
00000011
.....
11111110
11111111

$2^8 = 256$ valori distinti

- È l'elemento base con cui si rappresentano i dati nei calcolatori
- Si utilizzano sempre dimensioni multiple (di potenze del 2) del byte: 2 byte (16 bit), 4 byte (32 bit), 8 byte (64 bit)...

24

Dal byte al kilobyte

- Potenze del 2

$$\begin{aligned}2^4 &= 16 \\2^8 &= 256 \\2^{16} &= 65536\end{aligned}$$

$$\begin{aligned}2^{10} &= 1024 && (K=\text{Kilo}) \\2^{20} &= 1048576 && (M=\text{Mega}) \\2^{30} &= 1073741824 && (G=\text{Giga})\end{aligned}$$

- Cosa sono KB (kilobyte), MB (Megabyte), GB (Gigabyte)?

$$\begin{aligned}1 \text{ KB} &= 2^{10} \text{ byte} = 1024 \text{ byte} \\1 \text{ MB} &= 2^{20} \text{ byte} = 1048576 \text{ byte} \\1 \text{ GB} &= 2^{30} \text{ byte} = 1073741824 \text{ byte} \\1 \text{ TB} &= 2^{40} \text{ byte} = 1099511627776 \text{ byte (Terabyte)}\end{aligned}$$

25

Da decimale a binario – 1

- Si divide ripetutamente il numero intero decimale per 2 fino ad ottenere un quoziente nullo; le cifre del numero binario sono i resti delle divisioni; la cifra più significativa è l'ultimo resto

Esempio: convertire in binario $(43)_{10}$

$$\begin{array}{r}43 : 2 = 21 + 1 \\21 : 2 = 10 + 1 \\10 : 2 = 5 + 0 \\5 : 2 = 2 + 1 \\2 : 2 = 1 + 0 \\1 : 2 = 0 + 1\end{array}$$

resti

bit più significativo

$$(43)_{10} = (101011)_2$$

26

Da decimale a binario – 2

- Si moltiplica ripetutamente il numero frazionario decimale per 2, fino ad ottenere una parte decimale nulla o, dato che la condizione potrebbe non verificarsi mai, per un numero prefissato di volte; le cifre del numero binario sono le parti intere dei prodotti successivi; la cifra più significativa è il risultato della prima moltiplicazione

Esempio: convertire in binario $(0,21875)_{10}$ e $(0,45)_{10}$

$$\begin{array}{rcl} 0,21875 \times 2 & = & 0,4375 \\ 0,4375 \times 2 & = & 0,875 \\ 0,875 \times 2 & = & 1,75 \\ 0,75 \times 2 & = & 1,5 \\ 0,5 \times 2 & = & 1,0 \end{array}$$



$$(0,21875)_{10} = (0,00111)_2$$

$$\begin{array}{rcl} 0,45 \times 2 & = & 0,9 \\ 0,90 \times 2 & = & 1,8 \\ 0,80 \times 2 & = & 1,6 \\ 0,60 \times 2 & = & 1,2 \\ 0,20 \times 2 & = & 0,4 \text{ etc.} \end{array}$$



$$(0,45)_{10} \approx (0,01110)_2$$

27

Da binario a decimale

- Oltre all'espansione esplicita in potenze del 2 – **forma polinomia...**
 $(101011)_2 = 1 \times 2^5 + 0 \times 2^4 + 1 \times 2^3 + 0 \times 2^2 + 1 \times 2^1 + 1 \times 2^0 = (43)_{10}$
- ...si può operare nel modo seguente: si raddoppia il bit più significativo e si aggiunge al secondo bit; si raddoppia la somma e si aggiunge al terzo bit... si continua fino al bit meno significativo

Esempio: convertire in decimale $(101011)_2$

bit più significativo

$$\begin{array}{rcl} 1 \times 2 & = & 2 + 0 \\ 2 \times 2 & = & 4 + 1 \\ 5 \times 2 & = & 10 + 0 \\ 10 \times 2 & = & 20 + 1 \\ 21 \times 2 & = & 42 + 1 = 43 \end{array}$$

$$(101011)_2 = (43)_{10}$$

28

Sistema esadecimale

- La base 16 è molto usata in campo informatico

Cifre: 0 1 2 3 4 5 6 7 8 9 A B C D E F

La corrispondenza in decimale delle cifre oltre il 9 è

$$\begin{array}{ll} A = (10)_{10} & D = (13)_{10} \\ B = (11)_{10} & E = (14)_{10} \\ C = (12)_{10} & F = (15)_{10} \end{array}$$

Esempio:

$$(3A2F)_{16} = 3 \times 16^3 + 10 \times 16^2 + 2 \times 16^1 + 15 \times 16^0 = 3 \times 4096 + 10 \times 256 + 2 \times 16 + 15 = (14895)_{10}$$

29

Da binario a esadecimale

- Una cifra esadecimale corrisponde a 4 bit

0 corrisponde a 4 bit a 0	0000	0	1000	8
	0001	1	1001	9
	0010	2	1010	A
	0011	3	1011	B
	0100	4	1100	C
	0101	5	1101	D
	0110	6	1110	E
	0111	7	1111	F
			F corrisponde a 4 bit a 1	

- Si possono rappresentare numeri binari lunghi con poche cifre (1/4)
- La conversione da binario ad esadecimale è immediata, raggruppando le cifre binarie in gruppi di 4 (da destra) e sostituendole con le cifre esadecimali secondo la tabella precedente

30

Dai bit all'hex

- Un numero binario di $4n$ bit corrisponde a un numero esadecimale di n cifre

Esempio: 32 bit corrispondono a 8 cifre esadecimali

1101	1001	0001	1011	0100	0011	0111	1111
D	9	1	B	4	3	7	F

$(D91B437F)_{16}$

Esempio: 16 bit corrispondono a 4 cifre esadecimali

0000	0000	1111	1111
0	0	F	F

$(00FF)_{16}$

31

Da esadecimale a binario

- La conversione da esadecimale a binario si ottiene espandendo ciascuna cifra con i 4 bit corrispondenti

Esempio: convertire in binario il numero esadecimale $0x0c8f$

Notazione usata in molti linguaggi di programmazione per rappresentare numeri esadecimali

0	c	8	f
0000	1100	1000	1111

Il numero binario ha $4 \times 4 = 16$ bit

32

Esempi – 1

- In qualsiasi base, l'essere il sistema di numerazione *posizionale* impone che combinazioni diverse di cifre uguali rappresentino numeri diversi; ad esempio:

$$\begin{aligned} &\rightarrow (319)_{10} \neq (193)_{10} \\ &\rightarrow (152)_6 \neq (512)_6, \text{ infatti...} \end{aligned}$$

$$\begin{aligned} (152)_6 &= 1 \times 6^2 + 5 \times 6^1 + 2 \times 6^0 = 36 + 30 + 2 = (68)_{10} \\ (512)_6 &= 5 \times 6^2 + 1 \times 6^1 + 2 \times 6^0 = 180 + 6 + 2 = (188)_{10} \end{aligned}$$

- Numeri che hanno identica rappresentazione, in basi diverse, hanno valori diversi:

$$\begin{aligned} &\rightarrow (234)_{10} \neq (234)_8, \text{ infatti...} \\ (234)_8 &= 2 \times 8^2 + 3 \times 8^1 + 4 \times 8^0 = 2 \times 64 + 3 \times 8 + 4 = 128 + 24 + 4 = (156)_{10} \end{aligned}$$

Osservazione: più piccola è la base, minore è il valore del numero rappresentato dalla stessa sequenza di cifre

33

Esempi – 2

- La notazione posizionale si applica anche per il calcolo del valore dei numeri frazionari, infatti...

$$\begin{aligned} &\rightarrow (0,872)_{10} = 8 \times 10^{-1} + 7 \times 10^{-2} + 2 \times 10^{-3} \\ &= 8/10 + 7/100 + 2/1000 = 0,8 + 0,07 + 0,002 \end{aligned}$$

- Quante cifre occorreranno per rappresentare il numero decimale 36 in base 2? Sappiamo che con n cifre si rappresentano i numeri da 0 a $2^n - 1$, quindi...

$$\begin{aligned} &\rightarrow \text{per } n=1 \text{ (con una cifra) si rappresentano } 0 \dots 2^1 - 1 \Rightarrow 0,1 \\ &\rightarrow \text{per } n=2: 0 \dots 2^2 - 1 \Rightarrow 0 \dots 3 \\ &\rightarrow \text{per } n=3: 0 \dots 2^3 - 1 \Rightarrow 0 \dots 7 \\ &\rightarrow \text{per } n=4: 0 \dots 2^4 - 1 \Rightarrow 0 \dots 15 \\ &\rightarrow \text{per } n=5: 0 \dots 2^5 - 1 \Rightarrow 0 \dots 31 \\ &\rightarrow \text{per } n=6: 0 \dots 2^6 - 1 \Rightarrow 0 \dots 63 \end{aligned}$$

Effettivamente possiamo verificare che $(100100)_2 = (36)_{10}$, infatti...
 $100100 = 1 \times 2^5 + 0 \times 2^4 + 0 \times 2^3 + 1 \times 2^2 + 0 \times 2^1 + 0 \times 2^0$
 $= 32 + 4 = 36$
con 6 cifre necessarie per la codifica del numero in base 2

34

Esempi – 3

Quesito: Per quale base B risulterà vera l'uguaglianza

$$17 + 41 + 22 = 102 ?$$

Se i numeri sono rappresentati in base B, sappiamo che:

$$(17)_B = 1 \times B^1 + 7 \times B^0 = B + 7$$

$$(41)_B = 4 \times B^1 + 1 \times B^0 = 4B + 1$$

$$(22)_B = 2 \times B^1 + 2 \times B^0 = 2B + 2$$

$$(102)_B = 1 \times B^2 + 0 \times B^1 + 2 \times B^0 = B^2 + 2$$

$$\text{da cui...} \quad B + 7 + 4B + 1 + 2B + 2 = 7B + 10 = B^2 + 2$$

\Rightarrow Si ottiene un'equazione di 2° grado: $B^2 - 7B - 8 = 0$

$$\text{Risolvendo: } \Delta = 49 + 32 = 81$$

$$B = (7 \pm \sqrt{\Delta})/2 = \begin{cases} (7+9)/2 = 8 & \leftarrow \text{È la soluzione!} \\ (7-9)/2 = -1 & \leftarrow \text{Non può essere una base} \end{cases}$$

35

La rappresentazione dei dati
e l'aritmetica degli elaboratori

36



Numeri interi positivi

- I numeri interi positivi sono rappresentati all'interno dell'elaboratore utilizzando un multiplo del byte (generalmente 2/4 byte)
- Se l'intero si rappresenta con un numero di cifre minore, vengono aggiunti zeri nelle cifre più significative

Esempio: 12 viene rappresentato in un byte come...

00001100

37



Numeri con segno

- Per rappresentare numeri con segno, occorre utilizzare un bit per definire il segno del numero
- Si possono usare 3 tecniche di codifica

- Modulo e segno
- Complemento a 2

38

Modulo e segno

- Il bit più significativo rappresenta il segno: 0 per i numeri positivi, 1 per quelli negativi
- Esiste uno zero positivo (00...0) e uno zero negativo (10...0)
- Se si utilizzano n bit si rappresentano tutti i numeri compresi fra $-(2^{n-1}-1)$ e $+2^{n-1}-1$

Esempio: con 4 bit si rappresentano i numeri fra -7 ($-(2^3-1)$) e +7 (2^3-1)

0000	+0	1000	-0
0001	+1	1001	-1
0010	+2	1010	-2
0011	+3	1011	-3
0100	+4	1100	-4
0101	+5	1101	-5
0110	+6	1110	-6
0111	+7	1111	-7

positivi

negativi

Complemento a 2

- Il complemento a 2 di un numero binario $(N)_2$ a n cifre è il numero $2^n - (N)_2 = 10 \dots 0 - (N)_2$
- Tale numero si ottiene...
 - Effettuando il complemento di ogni cifra del numero di partenza (complemento a 1): si trasforma ogni 0 in 1 e ogni 1 in 0
 - Aggiungendo 1 al numero ottenuto
- Oppure: a partire da destra, lasciando invariate tutte le cifre fino al primo 1 compreso, quindi invertendo il valore delle rimanenti

$$\begin{array}{r}
 01010111 \\
 10101000 \\
 \hline
 10101001 \\
 + 1 \\
 \hline
 10101001
 \end{array}
 \quad
 \begin{array}{l}
 \text{complemento cifre} \\
 + 1
 \end{array}
 \quad
 \begin{array}{l}
 10000000 \leftarrow 2^8 \\
 + 01111111 \leftarrow 2^8 - 1 \\
 - 01010111 \leftarrow N \\
 \hline
 10101000 \leftarrow 2^8 - 1 - N
 \end{array}
 \quad
 \begin{array}{l}
 10101001 \leftarrow 2^8 - 1 - N + 1
 \end{array}$$

Interi in complemento a 2

- I **numeri positivi** sono rappresentati come in modulo e segno
- I **numeri negativi** hanno un 1 nella posizione più significativa e sono rappresentati in complemento a 2
- Lo zero è rappresentato come numero positivo (con una sequenza di n zeri)
- Il campo dei numeri rappresentabili è da -2^{n-1} a $+2^{n-1}-1$

Esempio: numeri a 4 cifre

0000	+0	1000	-8	
0001	+1	1001	-7	
0010	+2	1010	-6	
0011	+3	1011	-5	
0100	+4	1100	-4	
0101	+5	1101	-3	
0110	+6	1110	-2	
0111	+7	1111	-1	

Nota: 0111 +7
1000 -8

41

Interi a 16 bit

- Numeri interi rappresentati su 16 bit in complemento a 2:

Il più grande numero intero positivo è $2^{15}-1=(32767)_{10}$

0111 1111 1111 1111
0x 7 F F F

Il più piccolo numero intero negativo è $-2^{15}=(-32768)_{10}$

1000 0000 0000 0000 → 0111 1111 1111 1111 +
0x 8 0 0 0 1
1000 0000 0000 0000

Il numero intero -1 è rappresentato come

1111 1111 1111 1111 → 0000 0000 0000 0000 +
0x F F F F 1
0000 0000 0000 0001

42

Addizione binaria

- Le regole per l'addizione di due bit sono

$$\begin{aligned} 0 + 0 &= 0 \\ 0 + 1 &= 1 \\ 1 + 0 &= 1 \\ 1 + 1 &= 0 \text{ con riporto di } 1 \end{aligned}$$

- L'ultima regola è... $(1)_2 + (1)_2 = (10)_2 \dots (1+1=2)_{10} \text{ !!}$

Esempio

1	11	1	riporti	$\begin{array}{r} 91+ \\ 90 \\ \hline 181 \end{array}$
0101	1011	1+	➡	
0101	1010			
1011	10101			

43

Sottrazione binaria – 1

- Le regole per la sottrazione di due bit sono

$$\begin{aligned} 0 - 0 &= 0 \\ 1 - 0 &= 1 \\ 1 - 1 &= 0 \\ 10 - 1 &= 1 \text{ con prestito di } 1 \text{ dalla cifra precedente a sinistra} \end{aligned}$$

Esempio

0 ¹⁰	1	1	0	0	1	-	25 -
	1	1	0	0	1	-	5
	1	0	1				20
	1	0	1	0	0		

➡

- La sottrazione può divenire complicata: quando si ha una richiesta sulla cifra precedente a sinistra, che è uno 0, l'operazione si propaga a sinistra fino alla prima cifra ad 1 del sottraendo

44

Sottrazione binaria – 2

- Utilizzando la rappresentazione in complemento a 2, addizione e sottrazione sono trattate come un'unica operazione

$$N_1 - N_2 = N_1 + (2^n - N_2) - 2^n \quad \text{si trascura il bit } n + 1$$

complemento a 2 di N_2 : $(-N_2)$

- 1 Si calcola il complemento a 2 di N_2
- 2 Si somma N_1 con il complemento a 2 di N_2
- 3 Si trascura il bit più significativo del risultato

Esempio: $(010001)_2 - (000101)_2 = (17)_{10} - (5)_{10}$

$$\begin{array}{r} 010001 + \\ \underline{111011} \\ \boxed{1}001100 \rightarrow (12)_{10} \end{array}$$

45

Overflow

L'**overflow** si ha quando il risultato di un'operazione non è rappresentabile correttamente con n bit

Esempio: 5 bit $\rightarrow [-16, +15]$

$$\begin{array}{r} 14 + \quad 01110 + \\ 10 \quad \underline{01010} \quad -8 + \quad 11000 + \\ 24 \quad \underline{11000} \rightarrow -8 \quad -10 \quad \underline{10110} \\ \quad \quad -18 \quad \underline{101110} \rightarrow +14 \end{array}$$

- Per evitare l'overflow occorre aumentare il numero di bit utilizzati per rappresentare gli operandi
- C'è overflow se c'è riporto al di fuori del bit di segno e non sul bit di segno, o se c'è riporto sul bit di segno, ma non al di fuori

Punteggio nei vecchi videogame...

$$\begin{array}{r} 0111 \ 1111 \ 1111 \ 1111 \ + \ 1 = 1000 \ 0000 \ 0000 \ 0000 \\ 32767 \quad \quad \quad + \ 1 = \quad \quad \quad -32768 \end{array}$$

46

Moltiplicazione binaria

- Le regole per la moltiplicazione di due bit sono

$$\begin{aligned} 0 \times 0 &= 0 \\ 0 \times 1 &= 0 \\ 1 \times 0 &= 0 \\ 1 \times 1 &= 1 \end{aligned}$$

Esempio

$$\begin{array}{r} 1100111 \times 101 \\ \hline 1100111 \\ 0000000 \\ 1100111 \\ \hline 1000000011 \end{array}$$

- Moltiplicare per 2^n corrisponde ad aggiungere n zeri in coda al moltiplicando

$$110011 \times 10000 = 1100110000 \quad \times 16 = 2^4$$

47

Divisione binaria

- La divisione binaria di A per B viene calcolata in modo analogo alla divisione decimale, così da ottenere un quoziente Q ed un resto R, tali che $A = B \times Q + R$
- La divisione binaria si compone di una serie di sottrazioni

$$\begin{array}{r} 110 \hat{1} \hat{1} \hat{1} \hat{0} \\ \underline{101} \\ 111 \\ \underline{101} \\ 100 \end{array} \quad \begin{array}{r} 101 \\ \underline{1010} \end{array}$$

$54 = 5 \times 10 + 4$

- Dividere per 2^n equivale a scorrere il numero a destra di n posizioni; le cifre scartate costituiscono il resto

$$110011 : 1000 = 11 \text{ con resto } 11 \quad \longleftrightarrow \quad 51 : 16 = 3 \text{ con resto } 3$$

48

Esercizi

Esercizio 1

Assumendo che un elaboratore rappresenti i numeri interi con segno su quattro bit in complemento a 2, si calcolino entrambi i membri della seguente identità:

$$(A-C)+B = (A+B)-C,$$

con $A=7$, $B=5$, $C=7$. Si ottiene lo stesso risultato dal calcolo dei due membri? Discutere le motivazioni della risposta.

Esercizio 2

- a) Si determini, se esiste, la base b di un sistema di numerazione tale che

$$(842)_b = (1202)_{10}$$
- b) Si determini, se esiste, la base $b \leq 16$ di un sistema di numerazione tale che

$$(725)_b - (626)_b = (224)_{10}$$

49

Confronto delle rappresentazioni

	Interi positivi	Segno e modulo	Complemento a 1	Complemento a 2
0000	0	+0	+0	+0
0001	1	+1	+1	+1
0010	2	+2	+2	+2
0011	3	+3	+3	+3
0100	4	+4	+4	+4
0101	5	+5	+5	+5
0110	6	+6	+6	+6
0111	7	+7	+7	+7
1000	8	-0	-7	-8
1001	9	-1	-6	-7
1010	10	-2	-5	-6
1011	11	-3	-4	-5
1100	12	-4	-3	-4
1101	13	-5	-2	-3
1110	14	-6	-1	-2
1111	15	-7	-0	-1

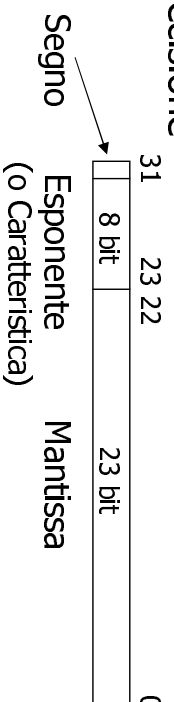
50

Numeri in virgola mobile

- La rappresentazione dei numeri in virgola mobile è in relazione con la **notazione scientifica** (es. $1.2 \times 10^2 = 120$)
- La IEEE ha previsto uno standard per la rappresentazione in virgola mobile

- **singola precisione** (32 bit = 4 byte)
- **doppia precisione** (64 bit = 8 byte)
- **quadrupla precisione** (128 bit = 16 byte)

Singola precisione



Se $E \neq 0$ e $E \neq 255$, il valore è $(-1)^S 1.M \times 2^{E-127}$

Eccesso: vale $2^{t-1}-1$, se t è il numero di cifre riservate alla caratteristica → rappresentazione "interna" dell'esponente sempre positiva

51

Numeri in virgola mobile: casi speciali

se $E=255$ e $M \neq 0$ NaN (not a number)

0 11111111 10101000111000101001100

se $E=255$, $M=0$, $S=1$ -Infinity

1 11111111 000000000000000000000000

se $E=255$, $M=0$, $S=0$ +Infinity

0 11111111 000000000000000000000000

se $E=0$, $M=0$, $S=0$ +0

0 00000000 000000000000000000000000

se $E=0$, $M=0$, $S=1$ -0

1 00000000 00000000000000000000100000

se $E=0$, $M \neq 0$ $(-1)^S 0.M \times 2^{-126}$

0 00000000 00000100000110001100000

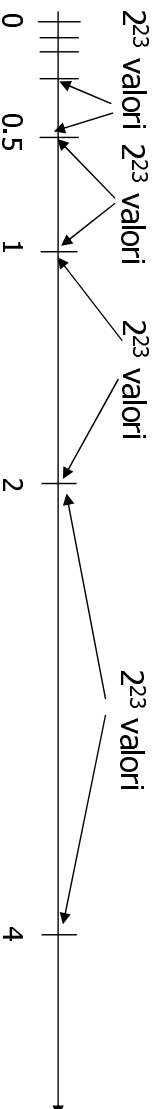
Ad esempio
dividendo un
numero per 0 puo
ottenere un NaN

Si dice valore
denormalizzato

52

Singola precisione

- Il numero più grande rappresentabile è $2^{127} \times (2 - 2^{-23}) \approx 2^{128} \approx 3.402 \times 10^{38}$
 $0\ 11111110\ 111111111111111111111111$
 $2^{254-127}\ 1.111111111111111111111111$
- Il più piccolo numero positivo è $2^{-126} \times 2^{-23} = 2^{-149} \approx 1.4013 \times 10^{-45}$
 $0\ 00000000\ 00000000000000000000001$
 $2^{-126}\ 0.000000000000000000000001$
- In totale si rappresentano 2^{32} numeri distinti, metà positivi, metà negativi
- Circa metà dei numeri sono compresi fra -1 e 1 ($E-127 < 0$)



53

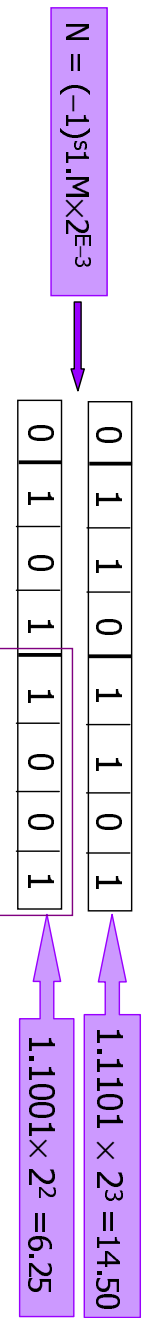
L'aritmetica floating-point: addizione

- Metodo per il calcolo dell'addizione
 - Se le caratteristiche dei numeri sono diverse, si considera il numero con caratteristica minore e...
 - Si trasla la mantissa di un posto a destra
 - Si incrementa la caratteristica di 1Si ripete 1.1 e 1.2 fino a quando le due caratteristiche sono uguali.
 - La mantissa del risultato è ottenuta dalla somma delle due mantisse
 - La caratteristica del risultato è inizialmente quella definita al punto 1. Se l'addizione comporta un riporto oltre la cifra più significativa, si trasla la mantissa del risultato a destra di un posto e si incrementa la caratteristica di 1.

54

Un esempio di addizione

- Supponiamo che per la rappresentazione floating-point vengano utilizzati otto bit, di cui uno per il segno, tre per la caratteristica e quattro per la mantissa



- La caratteristica del secondo operando è più piccola di una unità, quindi la mantissa deve scorrere di una posizione a destra

$$1.1001 \times 2^2 \Rightarrow 0.1100 \times 2^3$$

se non esistono sufficienti bit, ci può essere un errore di troncamento

- La caratteristica del risultato è 110, la mantissa è

$$\begin{array}{r}
 1.1101 + \\
 0.1100 \\
 \hline
 10.1001
 \end{array}$$

55

Un esempio di addizione

$$E=110, M=10.1001$$

- La caratteristica deve essere aumentata di 1 e la mantissa del risultato tralata a destra di una posizione:

0	1	1	1	0	1	0	0
---	---	---	---	---	---	---	---

- Il risultato codifica il numero $1.0100 \times 2^4 = 20$ ma il risultato corretto è 20.75: errore di troncamento.
- Errori di approssimazione sono inevitabili con i numeri reali in ambito scientifico! Occorre stimare il massimo errore che si può commettere.

56

L'aritmetica floating-point: moltiplicazione

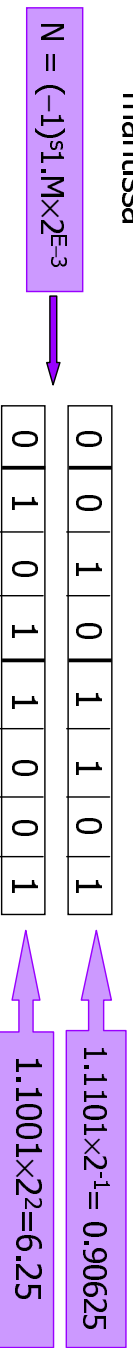
- Metodo per il calcolo della moltiplicazione

1. Si moltiplicano le due mantisse
2. Si addizionano le due caratteristiche
3. Si trasla a sinistra il prodotto delle due mantisse fino ad ottenere un 1 come cifra più significativa; si diminuisce la caratteristica di 1 per ogni traslazione eseguita
4. Si tronca la mantissa al numero di bit utilizzati nella rappresentazione; la mantissa del prodotto è il risultato del troncamento
5. Si sottrae l'eccesso alla somma delle caratteristiche, ottenendo la caratteristica del prodotto

57

Un esempio di moltiplicazione

- Supponiamo che per la rappresentazione floating-point vengano utilizzati otto bit, di cui uno per il segno, tre per la caratteristica e quattro per la mantissa



- Moltiplicando le mantisse e sommando le caratteristiche si ottiene:

$$M=1.1101 \times 1.1001=10.11010101 \quad E=010+101=111$$

- La mantissa del risultato deve essere traslata di un posto a destra, e la somma delle caratteristiche deve essere incrementata di 1;

$$M=10.11010101 \Rightarrow M=1.011010101 \quad E=111 \Rightarrow E=1000$$

58

Un esempio di moltiplicazione

M=1.011010101 E=1000

- infine la mantissa deve essere troncata alle 4 cifre significative e l'eccesso (011) sottratto alla caratteristica:

$$\begin{array}{r} 1\ 0\ 0\ 0 \\ -\ 0\ 1\ 1 \\ \hline 0\ 1\ 0\ 1\ 1\ 0\ 1\ 0\ 1\ 0 \end{array}$$



Codifica il numero 5.5, ma il risultato corretto è 5.6640625: errore di troncamento

59

L'aritmetica degli elaboratori – 1

- L'aritmetica "interna" degli elaboratori differisce notevolmente dall'aritmetica classica
- Sebbene le stesse operazioni possano essere realizzate secondo modalità diverse su elaboratori diversi, si riscontrano alcune caratteristiche comuni:
 - ✦ Rappresentazione binaria dei numeri
 - ✦ Rango finito dei numeri rappresentabili
 - ✦ Precisione finita dei numeri
 - ✦ Operazioni espresse in termini di operazioni più semplici

60

L'aritmetica degli elaboratori – 2

Rango finito dei numeri rappresentabili

- Qualunque sia la codifica utilizzata, esistono sempre il più grande ed il più piccolo numero rappresentabile
- I limiti inferiore e superiore del rango di rappresentazione dipendono sia dal tipo di codifica, sia dal numero di bit utilizzati
- Se il risultato di un'operazione non appartiene al rango dei numeri rappresentabili, si dice che si è verificato un overflow (un **underflow**, più precisamente, se il risultato è più piccolo del più piccolo numero rappresentabile)

61

L'aritmetica degli elaboratori – 3

Precisione finita dei numeri

- La **precisione** della rappresentazione di un numero frazionario è una misura di quanto essa corrisponda al numero che deve essere rappresentato
- Negli elaboratori, i numeri frazionari sono rappresentati in virgola mobile (**floating-point**), utilizzando un numero finito di bit
- È plausibile che un numero reale non ammetta una rappresentazione finita, quindi dovrà essere codificato in maniera approssimata
- Negli elaboratori si rappresentano soltanto numeri razionali (fino ad una data precisione)

62

L'aritmetica degli elaboratori – 4

- **Operazioni espresse in termini di operazioni più semplici**
 - La maggior parte degli elaboratori non possiede circuiti in grado di eseguire direttamente tutte le operazioni:
 - La sottrazione si realizza per mezzo di una complementazione e di un'addizione
 - La moltiplicazione si realizza per mezzo di una successione di addizioni e di shift (traslazioni)
 - La divisione si realizza per mezzo di una successione di shift e sottrazioni
 - Le operazioni più semplici sono eseguite direttamente da appositi circuiti (in hardware); le operazioni più complesse sono realizzate mediante l'esecuzione di successioni di operazioni più semplici, sotto il controllo di programmi appositamente realizzati, e generalmente memorizzati permanentemente (in firmware)

63

Codifica dei caratteri alfabetici

- Oltre ai numeri molte applicazioni informatiche elaborano caratteri (simboli)
 - Gli elaboratori elettronici trattano numeri
 - Si codificano i caratteri e i simboli con dei numeri
 - Per poter scambiare dati (testi) in modo corretto occorre definire uno standard di codifica
- | | | |
|----|-------|----------|
| 3 | ————→ | 00110011 |
| \$ | ————→ | 00100100 |

- Quando si scambiano dati deve essere noto il tipo di codifica utilizzato
- In genere un sistema informatico deve supportare più standard di codifica
- La codifica deve prevedere le lettere dell'alfabeto, le cifre numeriche, i simboli, la punteggiatura, i caratteri speciali per certe lingue (æ, ã, è, ...,)

64

Codifica ASCII

American Standard Code for Information Interchange

- Definisce una tabella di corrispondenza fra ciascun carattere e un codice a **7 bit** (128 caratteri)
- I caratteri in genere sono rappresentati con **1 byte** (8 bit). I caratteri con il bit più significativo a 1 (quelli con codice dal 128 al 255) fanno parte di una estensione della codifica
- La tabella comprende sia **caratteri di controllo** (codici da 0 a 31) che **caratteri stampabili**
- I caratteri alfabetici hanno codici ordinati secondo l'ordine alfabetico

0 48	A 65	a 97
1 49	B 66	b 98
.....
8 56	Y 89	y 121
9 57	Z 90	Z 122
cifre	maiuscole	minuscole

65

Caratteri di controllo ASCII

- I caratteri di controllo (codice da 0 a 31) hanno funzioni speciali
- Si ottengono o con tasti specifici o con una sequenza **Ctrl**

Ctrl	Dec	Hex	Code	Nota
^@	0	0	NULL	carattere nullo
^A	1	1	SOH	partenza blocco
.....
^G	7	7	BEL	beep
^H	8	8	BS	backspace
^I	9	9	HT	tabulazione orizzontale
^J	10	A	LF	Line feed (cambio linea)
^K	11	B	VT	tabulazione verticale
^L	12	C	FF	Form feed (alim. carta)
^M	13	D	CR	carriage return (a capo)
.....
^Z	26	1A	EOF	fine file
^[27	1B	ESC	escape
....
^_	31	1F	US	separatore di unità

66

Caratteri ASCII stampabili

Dec	Hx	Chr	Dec	Hx	Chr	Dec	Hx	Chr	Dec	Hx	Chr	Dec	Hx	Chr	Dec	Hx	Chr
32	20	SPACE	48	30	0	64	40	@	80	50	P	96	60	`	112	70	p
33	21	!	49	31	1	65	41	A	81	51	Q	97	61	a	113	71	q
34	22	"	50	32	2	66	42	B	82	52	R	98	62	b	114	72	r
35	23	#	51	33	3	67	43	C	83	53	S	99	63	c	115	73	s
36	24	\$	52	34	4	68	44	D	84	54	T	100	64	d	116	74	t
37	25	%	53	35	5	69	45	E	85	55	U	101	65	e	117	75	u
38	26	&	54	36	6	70	46	F	86	56	V	102	66	f	118	76	v
39	27	'	55	37	7	71	47	G	87	57	W	103	67	g	119	77	w
40	28	(56	38	8	72	48	H	88	58	X	104	68	h	120	78	x
41	29)	57	39	9	73	49	I	89	59	Y	105	69	i	121	79	y
42	2A	*	58	3A	:	74	4A	J	90	5A	Z	106	6A	j	122	7A	z
43	2B	+	59	3B	;	75	4B	K	91	5B	[107	6B	k	123	7B	{
44	2C	,	60	3C	<	76	4C	L	92	5C	\	108	6C	l	124	7C	
45	2D	-	61	3D	=	77	4D	M	93	5D]	109	6D	m	125	7D	}
46	2E	.	62	3E	>	78	4E	N	94	5E	^	110	6E	n	126	7E	~
47	2F	/	63	3F	?	79	4F	O	95	5F	_	111	6F	o	127	7F	DEL

Nota: il valore numerico di una cifra può essere calcolato come differenza del suo codice ASCII rispetto al codice ASCII della cifra 0 (es. '5' - '0' = 53-48 = 5)

Tabella ASCII estesa

- I codici oltre il 127 non sono compresi nello standard originario

128	C	144	È	160	é	176	☼	192	Ł	208	Ł	224	œ	240	=
129	Ù	145	⌘	161	í	177	☼	193	Ł	209	Ł	225	ß	241	±
130	É	146	Æ	162	ó	178	☼	194	Ł	210	Ł	226	Ŧ	242	≥
131	ê	147	ô	163	ú	179		195	Ł	211	Ł	227	Ŧ	243	≤
132	ë	148	õ	164	ü	180	†	196	-	212	Ł	228	Ŧ	244	ƒ
133	ä	149	ö	165	ñ	181	†	197	†	213	Ł	229	c	245	j
134	å	150	ù	166	ı	182	†	198	†	214	Ł	230	Ł	246	÷
135	ƒ	151	ù	167	°	183	Ŧ	199	Ł	215	Ł	231	Ł	247	≈
136	ê	152	—	168	¿	184	Ŧ	200	Ł	216	Ł	232	ø	248	°
137	ë	153	Ö	169	—	185	Ł	201	Ł	217	Ł	233	©	249	·
138	ê	154	Ü	170	—	186	Ł	202	Ł	218	Ł	234	©	250	·
139	ı	155	£	171	¼	187	Ŧ	203	Ł	219	Ł	235	ø	251	√
140	ı	156	£	172	½	188	Ł	204	Ł	220	Ł	236	∞	252	—
141	ı	157	£	173	¾	189	Ł	205	Ł	221	Ł	237	ø	253	—
142	ı	158	—	174	ı	190	Ł	206	Ł	222	Ł	238	ø	254	■
143	ı	159	ı	175	ı	191	Ł	207	Ł	223	Ł	239	ø	255	■

Codifica UNICODE

- E' lo standard emergente per la codifica dei caratteri nei testi. E' basato sulle caratteristiche del codice ASCII ma va oltre la limitazione di poter rappresentare in modo coerente solo l'alfabeto latino
- Fornisce un **unico codice per ogni carattere di ogni lingua scritta** indipendentemente dalla piattaforma, dal linguaggio o dal programma
- Lo standard iniziale prevedeva di codificare i caratteri con 16 bit, per un totale di oltre 65000 caratteri rappresentabili
- La versione 3.0 dello standard fornisce i codici per **49194** caratteri dagli alfabeti usati nel mondo, dagli insiemi di ideogrammi, dalle collezioni di simboli
- L'ultima versione dello standard definisce 3 tipi diversi di codifica che permettono agli stessi dati di essere trasmessi in byte (8 bit - **UTF-8**), word (16 bit - **UTF-16**) o double word (32 bit - **UTF-32**). Tutte le codifiche hanno al più bisogno di 32 bit per carattere

69

Codifica UTF-8

- E' utilizzata nel file HTML

```
<META content="text/html; charset=utf-8" http-equiv=Content-Type>
```

- I caratteri sono codificati con un numero variabile di byte
- La codifica UTF-8 è compatibile con la codifica ASCII, ovvero i caratteri contenuti nella tabella ASCII standard hanno esattamente la stessa codifica UTF-8. Questo facilita la compatibilità fra i programmi

T → U+0054
 ↑
 hex

- Lo standard definisce il significato del simbolo (es. Lettera T maiuscola dell'alfabeto latino) ma non come questo viene rappresentato sullo schermo o sulla stampa (**glifo** - **font**, **dimesione**, **orientamento**).

70



Codifica delle immagini – 1

- Le immagini vengono anch'esse codificate come una sequenza di bit: il processo di "traduzione" da un'immagine ad una sequenza binaria prende il nome di *digitalizzazione*
- L'immagine è suddivisa in punti o **pixel** (per *picture element*), e ciascun punto viene codificato con un numero, che corrisponde ad un colore o ad un particolare tono di grigio
- Si utilizzano un numero di colori o di sfumature che sia una potenza del 2, in modo da codificare l'informazione legata a ciascun pixel con un opportuno numero di bit

71



Codifica delle immagini – 2

- Le immagini vengono memorizzate come lunghe sequenze di bit: per interpretarle è necessario conoscere...
 - ...le dimensioni dell'immagine (base ed altezza in numero di pixel), detta anche **risoluzione**
 - ...il numero di colori (o toni di grigio) disponibili per ogni pixel
- Se un'immagine viene codificata ad una data risoluzione, potrà comunque essere presentata su un dispositivo a più bassa risoluzione, a patto di "ignorare" alcuni dei bit che descrivono i pixel

72



Codifica delle immagini – 3

- Come è avvenuto per i caratteri, anche per le immagini sono stati definiti standard di codifica, che assicurano la compatibilità fra sistemi diversi, per quanto concerne la trasmissione e la visualizzazione delle immagini
 - **TIFF** – *Tagged Image File Format*
 - **JPEG**
 - **PNG** – *Portable Network Graphics*
- Per ridurre lo spazio necessario per memorizzare le immagini si utilizzano tecniche di **compressione** (utili anche per la trasmissione su Internet)

73



Codifica delle immagini – 4

- Le tecniche di compressione si dividono in...
 - **Tecniche lossless**: non provocano perdita di informazione, sono adatte a codificare immagini in cui sono presenti ampie aree monocromatiche
⇒ si codificano in maniera compatta insieme di pixel aventi le stesse caratteristiche
 - **Tecniche lossy**: provocano perdita di informazione, facendo decadere la qualità dell'immagine
- Normalmente ai formati JPEG e PNG, molto diffusi per lo scambio di immagini su Internet, si applicano metodi di compressione lossy

74