Corso di Laurea Ingegneria Informatica

Laboratorio di Informatica

Capitolo 2

La rappresentazione dell'informazione

Ottobre 2006

Nota bene

- Alcune parti del presente materiale didattico sono derivate da:
  - trasparenze per il corso di Laboratorio di Informatica A.A. 2005-2006 prodotte da Alfonso Miola
  - materiale didattico prodotto da Carla Limongelli
- L'utilizzo di questo materiale è stato consentito dagli autori

L’informazione all’interno del calcolatore

- La rappresentazione dell’informazione all’interno di un calcolatore è condizionata da due vincoli
  1. il calcolatore gestisce **solo sequenze di bit**
     - con i dispositivi elettronici è facile rappresentare le due cifre 0 e 1
  2. il calcolatore ha una **memoria limitata**
     - ne segue che la quantità di memoria dedicata alla rappresentazione di una singola informazione è anch’essa limitata
- Questi due vincoli sono largamente indipendenti tra loro
  - anche se il calcolatore avesse una memoria infinita dovremmo comunque studiare il modo di rappresentare i dati in binario
  - anche se il calcolatore potesse memorizzare direttamente informazioni generiche (per esempio numeri decimali) avremmo comunque il problema della limitatezza della sua memoria

Contenuti

- La rappresentazione binaria dell'informazione
  - la codifica tabellare
    - la rappresentazione di caratteri
    - la rappresentazione di valori booleani (vero e falso)
  - la conversione di base
    - il sistema di numerazione decimale
    - il sistema di numerazione binario
- Le rappresentazione dell'informazione con memoria limitata
  - i numeri naturali

    0, 1, 2, 3, …
  - i numeri interi relativi

    …, -3, -2, -1, 0, +1, +2, +3, …
  - i numeri razionali

    0.0018, -2345.06, 12.0, …

Codifica dei dati – il bit

- In un calcolatore i dati e le istruzioni sono codificati in forma binaria, ovvero come sequenze finite di cifre 0 e 1
  - il **bit** è il più piccolo dato in un calcolatore
    - un bit può avere valore 0 oppure 1
    - la parola "**bit**" è una forma contratta per **bi**nary digi**t** (cifra binaria)
  - ciascun **bit** è memorizzato da una cella elementare di memoria, fisicamente realizzata come **dispositivo elettronico bistabile** (in cui sono chiaramente distinguibili due stati)
    - questi due stati vengono fatti corrispondere allo 0 e all'1
- Un bit è quindi un dato che può essere usato soltanto per rappresentare una informazione binaria
- Per rappresentare altre tipologie di informazioni sono necessarie sequenze di bit

Codifica dei dati – il byte

- Il byte è la più piccola sequenza di bit
  - un **byte** è una sequenza di 8 bit
- Le possibili combinazioni degli 8 bit in un byte sono $2^8 = 256$
  - un byte può essere utilizzato per rappresentare un valore tra 256 diversi possibili valori
    - ad esempio, un piccolo numero intero, un carattere in un alfabeto che contiene non più di 256 caratteri, ...
  - in generale, sono possibili diverse scelte sull'insieme dei valori possibili
    - ad esempio, con un byte si può rappresentare un numero naturale compreso nell'intervallo da 0 a 255, oppure un numero intero relativo nell'intervallo da –128 a 127
- Per rappresentare altre tipologie di informazioni sono necessarie sequenze di bit più grandi

Strategie per la codifica dei dati

- Nel codificare una informazione in un dato binario possiamo usare due diverse strategie
  - codifica tabellare
    - applicabile quando l'insieme dei valori che può assumere l'informazione è finito
    - si fa corrispondere ad ogni valore dell'informazione una diversa combinazione di 0 e di 1
  - conversione di base
    - applicabile quando l'informazione è un numero in base 10
    - si premette una fase di traduzione in cui il numero viene convertito in base 2

Codifica tabellare

- Quanti bit dobbiamo usare per codificare *m* valori diversi?
- Abbiamo bisogno di almeno *m* combinazioni diverse
  - i matematici non le chiamano “combinazioni”, ma “disposizioni con ripetizione”
- Con un bit abbiamo due sole combinazioni
  - 0 e 1
  - $2 = 2^1$
- Con due bit abbiamo quattro combinazioni diverse
  - 00, 01, 10 e 11
  - $4 = 2^2$
- Con tre bit abbiamo otto combinazioni diverse
  - 000, 001, 010, 011, 100, 101, 110 e 111
  - $8 = 2^3$
- In generale con *n* bit avremo 2^n combinazioni diverse

2^0	2^1	2^2	2^3	2^4	2^5	2^6	2^7	2^8	2^9	2^{10}
(1)	(2)	(4)	(8)	(16)	(32)	(64)	(128)	(256)	(512)	(1024)

Rappresentazione di caratteri

- Sono disponibili diverse codifiche tabellari
- Le principali sono le seguenti
  - ASCII
    - sta per *American Standard Code for Information Interchange*
    - nella prima versione utilizzava 7 bit per codificare un carattere
    - nella versione estesa utilizza un byte per rappresentare un carattere
  - UNICODE
    - utilizza due byte per rappresentare un carattere
    - coincide con la codifica ASCII per i caratteri “di base” dell’alfabeto inglese
    - rappresenta 49.194 caratteri, negli alfabeti delle lingue principali (italiano, inglese, cirillico, arabo, ebraico, cinese, giapponese, ecc. ecc.)
    - è molto diffuso (formati HTML e XML, linguaggio Java, varie applicazioni, ecc.)

Codifica ASCII e UNICODE

Byte	Cod.	Char
00000000	0	Null
00000001	1	Start of heading
00000010	2	Start of text
00000011	3	End of text
00000100	4	End of transmit
00000101	5	Enquiry
00000110	6	Acknowledge
00000111	7	Audible bell
00001000	8	Backspace
00001001	9	Horizontal tab
00001010	10	Line feed
00001011	11	Vertical tab
00001100	12	Form Feed
00001101	13	Carriage return
00001110	14	Shift out
00001111	15	Shift in
00010000	16	Data link escape
00010001	17	Device control 1
00010010	18	Device control 2
00010011	19	Device control 3
00010100	20	Device control 4
00010101	21	Neg. acknowledge
00010110	22	Synchronous idle
00010111	23	End trans. block
00011000	24	Cancel
00011001	25	End of medium
00011010	26	Substitution
00011011	27	Escape
00011100	28	File separator
00011101	29	Group separator
00011110	30	Record Separator
00011111	31	Unit separator

Byte	Cod.	Char
00100000	32	Spc
00100001	33	!
00100010	34	"
00100011	35	#
00100100	36	$
00100101	37	%
00100110	38	&
00100111	39	'
00101000	40	(
00101001	41	)
00101010	42	*
00101011	43	+
00101100	44	,
00101101	45	-
00101110	46	.
00101111	47	/
00110000	48	0
00110001	49	1
00110010	50	2
00110011	51	3
00110100	52	4
00110101	53	5
00110110	54	6
00110111	55	7
00111000	56	8
00111001	57	9
00111010	58	:
00111011	59	;
00111100	60	<
00111101	61	=
00111110	62	>
00111111	63	?

Byte	Cod.	Char
01000000	64	@
01000001	65	A
01000010	66	B
01000011	67	C
01000100	68	D
01000101	69	E
01000110	70	F
01000111	71	G
01001000	72	H
01001001	73	I
01001010	74	J
01001011	75	K
01001100	76	L
01001101	77	M
01001110	78	N
01001111	79	O
01010000	80	P
01010001	81	Q
01010010	82	R
01010011	83	S
01010100	84	T
01010101	85	U
01010110	86	V
01010111	87	W
01011000	88	X
01011001	89	Y
01011010	90	Z
01011011	91	[
01011100	92	\
01011101	93	]
01011110	94	^
01011111	95	_

Byte	Cod.	Char
01100000	96	`
01100001	97	a
01100010	98	b
01100011	99	c
01100100	100	d
01100101	101	e
01100110	102	f
01100111	103	g
01101000	104	h
01101001	105	i
01101010	106	j
01101011	107	k
01101100	108	l
01101101	109	m
01101110	110	n
01101111	111	o
01110000	112	p
01110001	113	q
01110010	114	r
01110011	115	s
01110100	116	t
01110101	117	u
01110110	118	v
01110111	119	w
01111000	120	x
01111001	121	y
01111010	122	z
01111011	123	{
01111100	124	|
01111101	125	}
01111110	126	~
01111111	127	Del

Codifica ASCII e UNICODE – estensione

Byte	Cod.	Char
10000000	128	Ç
10000001	129	ü
10000010	130	é
10000011	131	â
10000100	132	ä
10000101	133	à
10000110	134	å
10000111	135	ç
10001000	136	ê
10001001	137	ë
10001010	138	è
10001011	139	ï
10001100	140	î
10001101	141	ì
10001110	142	Ä
10001111	143	Å
10010000	144	É
10010001	145	æ
10010010	146	Æ
10010011	147	ô
10010100	148	ö
10010101	149	ò
10010110	150	û
10010111	151	ù
10011000	152	ÿ
10011001	153	Ö
10011010	154	Ü
10011011	155	ø
10011100	156	£
10011101	157	Ø
10011110	158	×
10011111	159	ƒ

Byte	Cod.	Char
10100000	160	á
10100001	161	í
10100010	162	ó
10100011	163	ú
10100100	164	ñ
10100101	165	Ñ
10100110	166	ª
10100111	167	º
10101000	168	¿
10101001	169	®
10101010	170	¬
10101011	171	½
10101100	172	¼
10101101	173	¡
10101110	174	«
10101111	175	»
10110000	176	▒
10110001	177	▒
10110010	178	▒
10110011	179	┤
10110100	180	┤
10110101	181	Á
10110110	182	Â
10110111	183	À
10111000	184	©
10111001	185	╣
10111010	186	║
10111011	187	╗
10111100	188	╝
10111101	189	¢
10111110	190	¥
10111111	191	┐

Byte	Cod.	Char
11000000	192	└
11000001	193	┴
11000010	194	┬
11000011	195	├
11000100	196	─
11000101	197	┼
11000110	198	ã
11000111	199	Ã
11001000	200	╚
11001001	201	╔
11001010	202	╩
11001011	203	╦
11001100	204	╠
11001101	205	═
11001110	206	╬
11001111	207	¤
11010000	208	ð
11010001	209	Ð
11010010	210	Ê
11010011	211	Ë
11010100	212	È
11010101	213	ı
11010110	214	Í
11010111	215	Î
11011000	216	Ï
11011001	217	┘
11011010	218	┌
11011011	219	█
11011100	220	▄
11011101	221	¦
11011110	222	Ì
11011111	223	▀

Byte	Cod.	Char
11100000	224	Ó
11100001	225	ß
11100010	226	Ô
11100011	227	Ò
11100100	228	õ
11100101	229	Õ
11100110	230	µ
11100111	231	þ
11101000	232	Þ
11101001	233	Ú
11101010	234	Û
11101011	235	Ù
11101100	236	ý
11101101	237	Ý
11101110	238	¯
11101111	239	´
11110000	240	
11110001	241	±
11110010	242	‗
11110011	243	¾
11110100	244	¶
11110101	245	§
11110110	246	÷
11110111	247	¸
11111000	248	°
11111001	249	¨
11111010	250	·
11111011	251	¹
11111100	252	³
11111101	253	²
11111110	254	■
11111111	255	

Rappresentazione di valori booleani

- Si usa una codifica tabellare
- Per rappresentare un valore booleano sarebbe sufficiente un solo bit
- Tipicamente, però, i sistemi informatici usano uno o più byte, in cui il valore zero rappresenta *false* e tutti gli altri valori rappresentano *true*
  - per esempio, su alcune piattaforme un booleano è rappresentato con 4 byte

Conversione di base

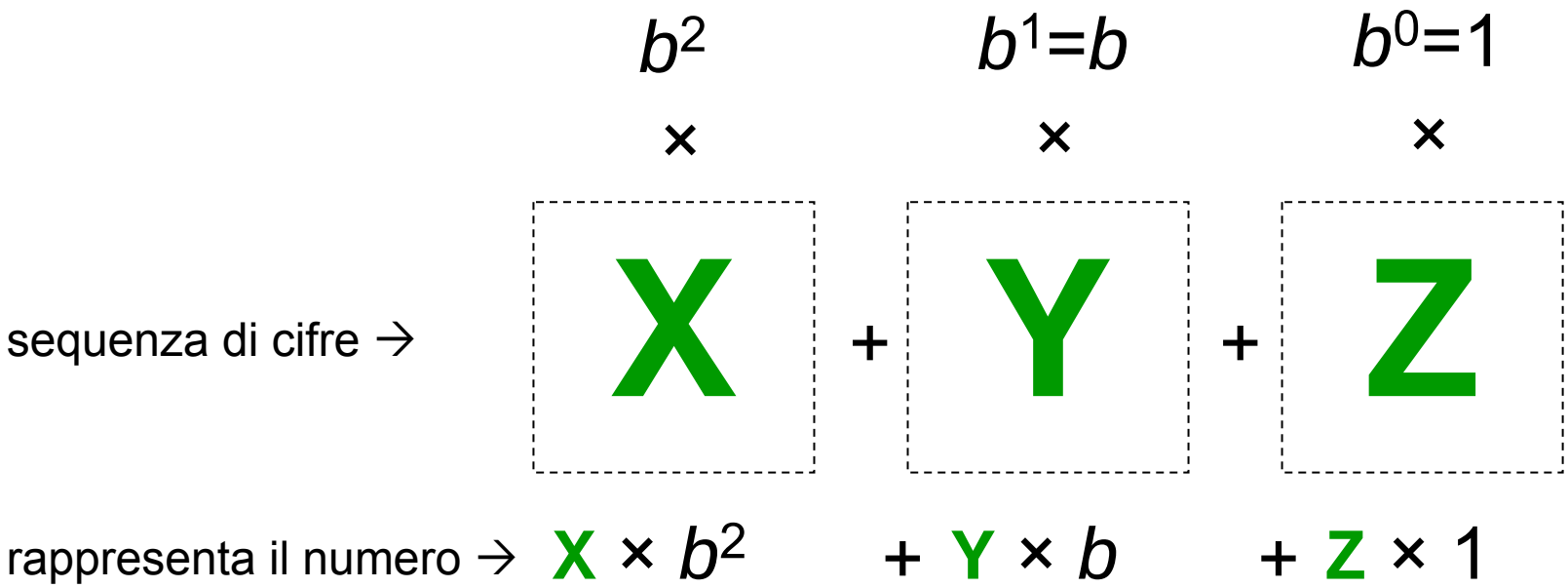
- E' nostra consuetudine rappresentare i numeri naturali tramite il **sistema di numerazione posizionale**
  - i simboli che rappresentano il numero hanno un'interpretazione dipendente dalla loro posizione
- In particolare è largamente diffuso il sistema di numerazione in **base dieci** (detto anche **decimale**)
  - la numerazione in base dieci fa uso di dieci simboli
    0, 1, 2, 3, 4, 5, 6, 7, 8 e 9
  - un numero è denotato da una sequenza di cifre
  - il valore di una cifra dipende dalla cifra stessa e dalla sua posizione nella sequenza
  - per esempio la sequenza di cifre **563** corrisponde al numero
    $5 \times 100 + 6 \times 10 + 3 = 563$

Sistema di numerazione in base dieci

	$10^2=100$		$10^1=10$		$10^0=1$
	×		×		×
la sequenza di cifre →	**5**	+	**6**	+	**3**
rappresenta il numero →	5 × 100		+ 6 × 10		+ 3 × 1
	500		+ 60		+ 3
	563				

Sistema di numerazione in base qualsiasi

- Il sistema di numerazione posizionale può essere generalizzato a qualsiasi base *b*, disponendo di *b* simboli diversi



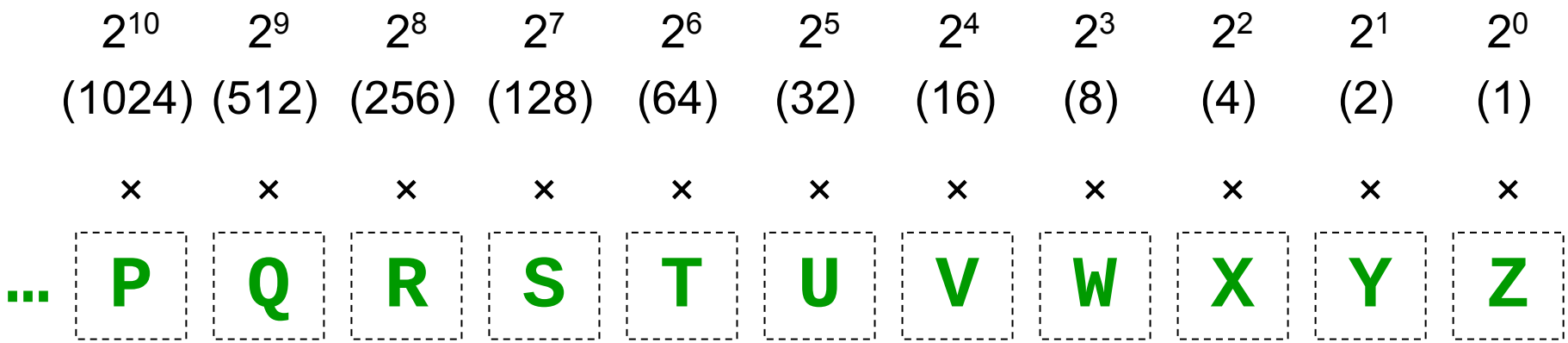
le cifre **X**, **Y** e **Z** sono prese dall'insieme dei *b* simboli a disposizione

Basi di uso frequente

- Sistema di numerazione in base dieci (o decimale)
  - utilizza i simboli 0,1,2,3,4,5,6,7,8 e 9
  - adottato per gli ovvi rapporti con la rappresentazione tramite flessione delle dita (indigitazione)
- Sistema di numerazione in base due (o binario)
  - utilizza i simboli 0 e 1
  - adottato dai calcolatori elettronici per la facilità di rappresentare mediante grandezze elettromagnetiche due stati, corrispondenti ai due simboli
- Sistema di numerazione in base sedici (o esadecimale)
  - utilizza i simboli 0,1,2,3,4,5,6,7,8,9,A,B,C,D,E ed F
  - adottato per la sua compattezza e per la facilità di conversione con la rappresentazione binaria
- Sistema di numerazione in base uno (o unario)
  - utilizza solo il simbolo I
  - non è veramente un sistema posizionale ($1^0 = 1^1 = 1^2 = 1^3 = \ldots = 1$)
  - adottato talvolta perché il numero può essere incrementato facilmente con l'aggiunta di una cifra (esempio: III+I = IIII)

Sistema di numerazione in base due

- Il sistema di numerazione in base due (detto anche *binario*) fa uso dei due soli simboli **0** e **1**



Operazioni con i numeri binari

- Esempi di somma di interi

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

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

Operazioni con i numeri binari

- Esempio di sottrazione di interi
  - lo sappiamo fare solo se il primo numero (minuendo) è maggiore del secondo numero (sottraendo)

```
  0  1  0  0  0  1  1  1      prestiti
  1  0  1  0  1  1  1  0  -
  0  1  0  0  0  1  1  1  =
-------------------------
  0  1  1  0  0  1  1  1
```

Operazioni con i numeri binari

- Esempi di moltiplicazione di interi

```
      1 0 1 1 0 ×
        1 0 1 =
-----------------
      1 0 1 1 0
    0 0 0 0 0
  1 0 1 1 0
-----------------
1 1 0 1 1 1 0
```

```
        1 0 0 0 1 ×
          1 1 0 0 =
-------------------
        0 0 0 0 0
      0 0 0 0 0
    1 0 0 0 1
  1 0 0 0 1
-------------------
1 1 0 0 1 1 0 0
```

Operazioni con i numeri binari

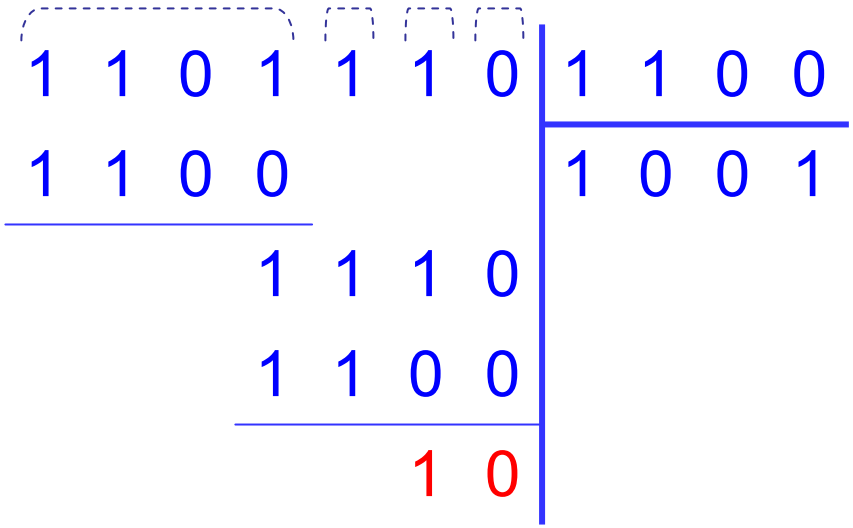
- Moltiplicare per 2^k equivale a traslare a sinistra di k posizioni

```
  1 0 1 1 0 ×
      1 0 =
-------------
  0 0 0 0 0
1 0 1 1 0
-------------
1 0 1 1 0 0
```

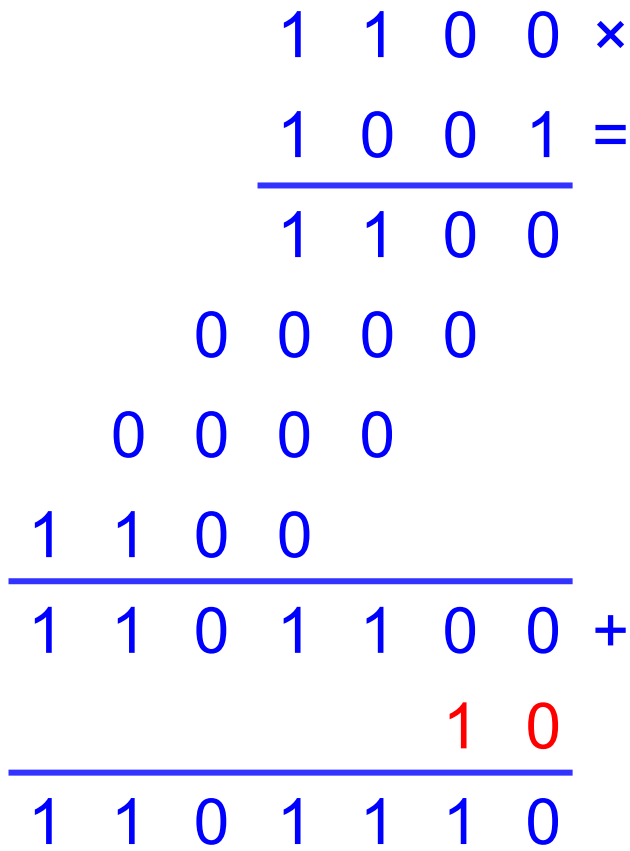
```
    1 0 1 1 0 ×
        1 0 0 =
---------------
    0 0 0 0 0
  0 0 0 0 0
1 0 1 1 0
---------------
1 0 1 1 0 0 0
```

Operazioni con i numeri binari

- Esempi di divisione tra interi

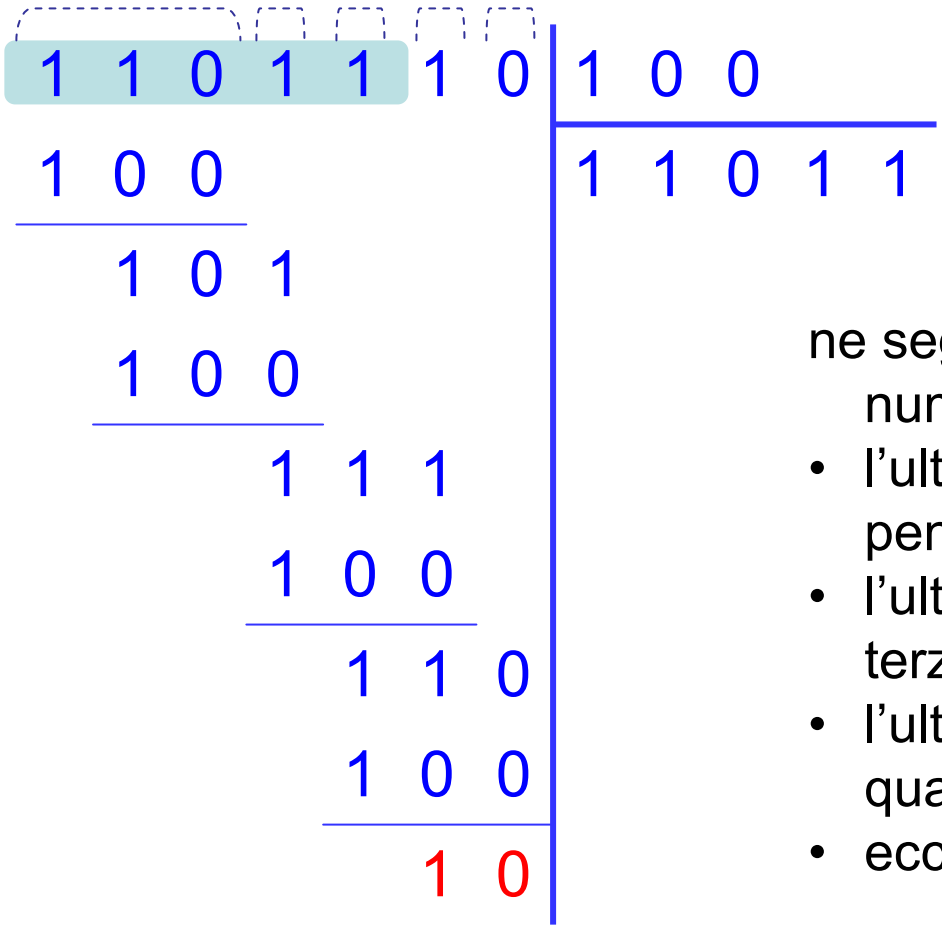


riprova:



Operazioni con i numeri binari

- Dividere per 2^k equivale a traslare a destra di k posizioni



ne segue che, se X è un numero binario:

- l'ultima cifra di X/2 è la penultima cifra di X
- l'ultima cifra di X/4 è la terzultima cifra di X
- l'ultima cifra di X/8 è la quartultima cifra di X
- ecc

Esercizi sulle operazioni in binario

- Calcolare i risultati delle seguenti operazioni
  - 0100101 + 0101 =
  - 100001010 + 11111111 =
  - 101010010 – 1111000 =
  - 100001 – 11 =
  - 10010001 × 111 =
  - 10101101 × 1010 =
  - 100010101 × 100000 =
  - 1010010 : 1000 = (con resto)
  - 1110010 : 1101 = (con resto)
  - 100101 : 110 = (con resto)

Conversione di un decimale in un binario

- Conversione del numero $X_{(10)}$ (decimale) in $X_{(2)}$ (binario)
  - supponiamo che $X_{(2)}$ sia composto dalle cifre .. $x_4\ x_3\ x_2\ x_1\ x_0$
  - è immediato determinare se $X_{(10)}$ è pari o dispari
    - se $X_{(10)}$ è dispari anche $X_{(2)}$ è dispari
      - ne segue che $x_0 = 1$
    - se $X_{(10)}$ è pari anche $X_{(2)}$ è pari
      - ne segue che $x_0 = 0$
  - immaginiamo di dividere $X_{(10)}$ e $X_{(2)}$ per due, scartando il resto (che è proprio x_0)
    - se $X_{(10)}/2$ è dispari anche $X_{(2)}/2$ è dispari
      - l'ultima cifra di $X_{(2)}/2$ è 1 $\Rightarrow$ la penultima cifra di $X_{(2)}$ è 1 $\Rightarrow$ $x_1 = 1$
    - se $X_{(10)}/2$ è pari anche $X_{(2)}/2$ è pari
      - l'ultima cifra di $X_{(2)}/2$ è 0 $\Rightarrow$ la penultima cifra di $X_{(2)}$ è 0 $\Rightarrow$ $x_1 = 0$
  - iteriamo il procedimento finché non otteniamo tutte le cifre di $X_{(2)}$

Conversione di un decimale in un binario

- Esempio: $X_{(10)} = 75$; $X_{(2)} = ?$
  - 75 : 2 = 37 con resto 1 ($x_0 = 1$)
  - 37 : 2 = 18 con resto 1 ($x_1 = 1$)
  - 18 : 2 = 9 con resto 0 ($x_2 = 0$)
  - 9 : 2 = 4 con resto 1 ($x_3 = 1$)
  - 4 : 2 = 2 con resto 0 ($x_4 = 0$)
  - 2 : 2 = 1 con resto 0 ($x_5 = 0$)
  - 1 : 2 = 0 con resto 1 ($x_6 = 1$)
- $X_{(2)} = x_6\ x_5\ x_4\ x_3\ x_2\ x_1\ x_0 = 1001011$

Conversione da un binario ad un decimale

- Conversione di un numero $X_{(2)}$ (binario) in $X_{(10)}$ (decimale)
  - supponiamo che $X_{(2)}$ sia composto dalle cifre .. x_4 x_3 x_2 x_1 x_0
  - applichiamo direttamente la formula:
    $X_{(10)} = x_0 \cdot 2^0 + x_1 \cdot 2^1 + x_2 \cdot 2^2 + x_3 \cdot 2^3$ …
- Esempio: $X_{(2)} = 10011$; $X_{(10)} = ?$
  - $X_{(10)} = 1 \cdot 2^0 + 1 \cdot 2^1 + 0 \cdot 2^2 + 0 \cdot 2^3 + 1 \cdot 2^4$
    $= 1 + 2 + 16 = 19$
- Altro esempio: $X_{(2)} = 11010$; $X_{(10)} = ?$
  - $X_{(10)} = 0 \cdot 2^0 + 1 \cdot 2^1 + 0 \cdot 2^2 + 1 \cdot 2^3 + 1 \cdot 2^4$
    $= 2 + 8 + 16 = 26$

Metodi di conversione veloci

- Ordine di grandezza di un numero binario
  - si può ottenere ricordando che $2^{10} = 1024 \approx 1000 = 10^3$
    - per esempio $11010010100010101100 10_{(2)}$
    - può essere così suddiviso 11 0100101000 1010110010
    - il suo ordine di grandezza è dunque $3 \times 1000 \times 1000 = 3\,000\,000_{(10)}$
      - il suo valore effettivo è $3\,449\,522_{(10)}$
- Numeri ottenibili per sottrazione o somma
  - alcuni numeri possono essere ottenuti per sottrazione o somma di numeri notevoli il cui equivalente decimale è noto o facile da calcolare
  - esempio
    - per i numeri decimali $999_{(10)} = 1000_{(10)} - 1_{(10)}$
    - analogamente, per i numeri binari $1111_{(2)} = 10000_{(2)} - 1_{(2)}$
    - dunque $1111_{(2)} = 2^4 - 1 = 16 - 1 = 15_{(10)}$

Esercizio

- Completa la seguente tabella

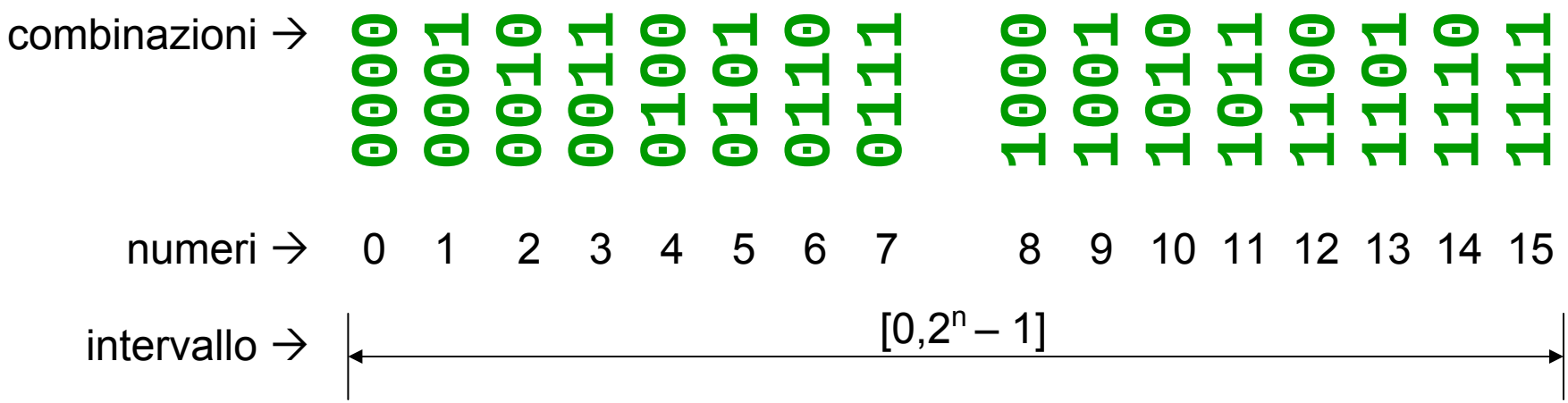
decimale	binario
0	0
57	
143	
201	
	101010
	11101011
	100111001

Rappresentazione con memoria limitata

- Quando la memoria è limitata e l'insieme dei valori da rappresentare è infinito è possibile solo una rappresentazione parziale dell'informazione
  - cioè: non tutti i valori possono essere rappresentati
- Nel seguito supporremo di avere a disposizione n bit con i quali vogliamo rappresentare i seguenti dati
  - un numero naturale (0, 1, 2, …)
  - un numero relativo (…, -3, -2, -1, 0, +1, +2, +3, …)
  - un numero razionale (0.0018, 2345.06, 12.0, …)

Rappresentazione binaria di numeri naturali

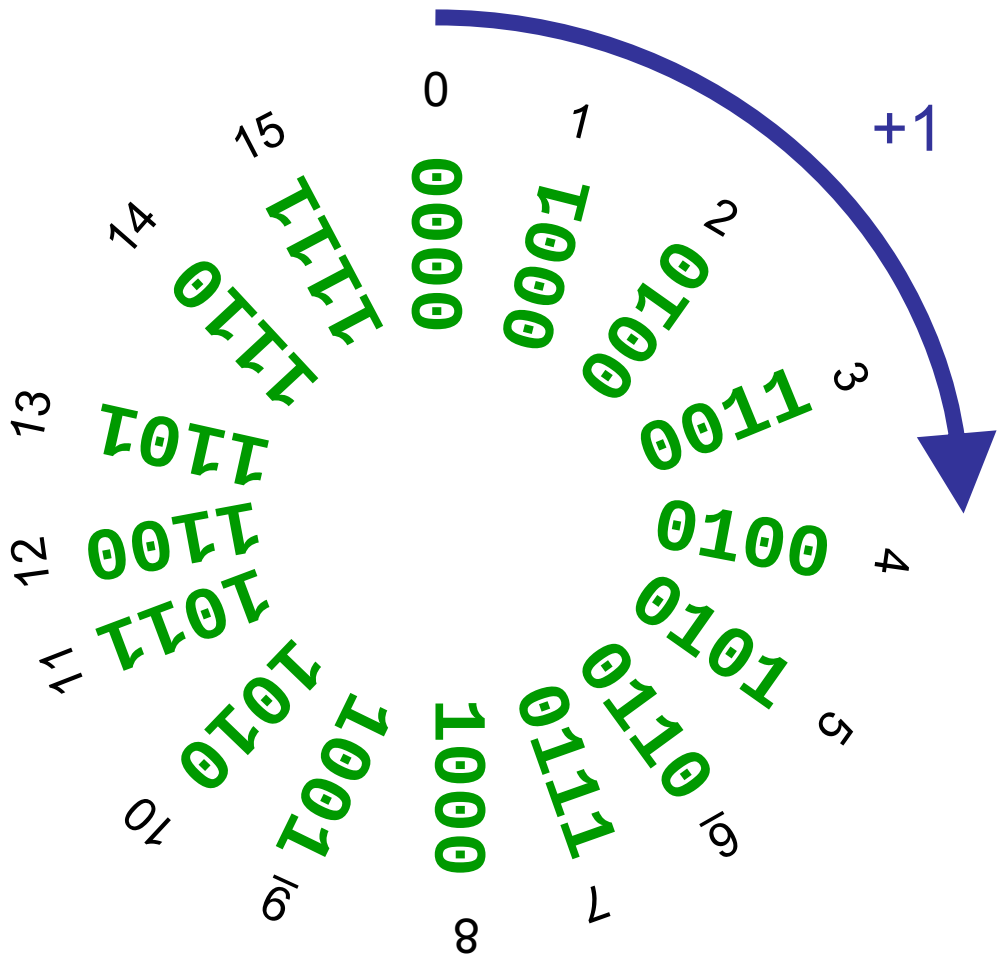
- Un numero naturale (0, 1, 2, …) può essere rappresentato dalla sua notazione in binario
- Quanti numeri possiamo rappresentare con n cifre in base due?
  - abbiamo 2^n combinazioni diverse che possono essere utilizzate per rappresentare i numeri da zero a $2^n - 1$
  - esempio con 4 cifre (n=4)



Note sulla rappresentazione dei naturali

- Avendo un numero limitato di bit per la rappresentazione dei numeri naturali non è possibile rappresentare numeri arbitrariamente grandi
  - il numero più grande che possiamo rappresentare con n bit è 2^n-1
- Che numero otteniamo se sommiamo 1 a 2^n-1?

Circolarità della rappresentazione

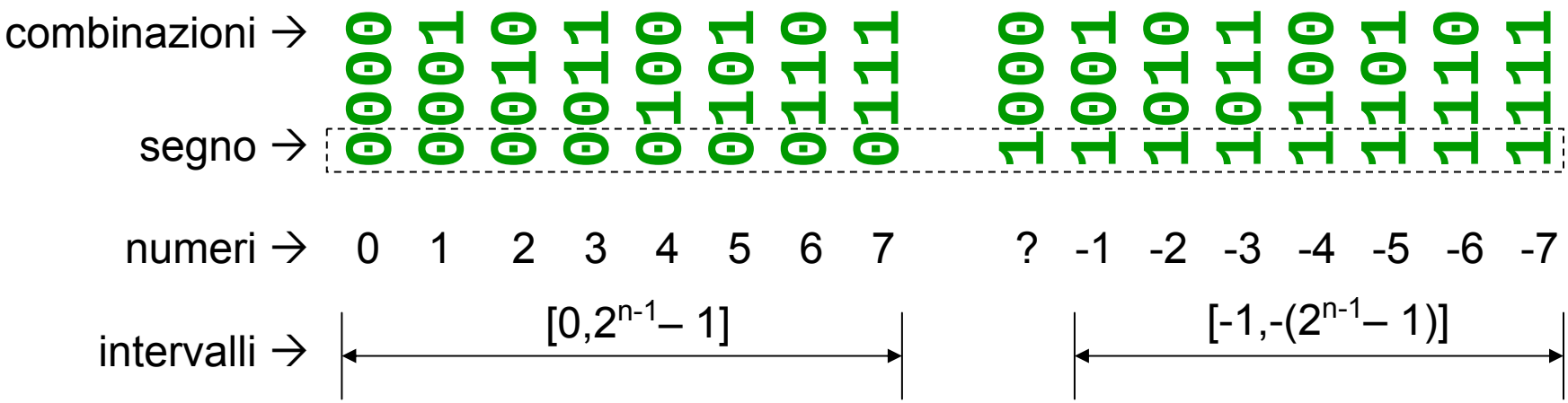


Rappresentazione di interi relativi

- Nella rappresentazione dei numeri relativi occorre destinare alcune combinazioni di bit alla rappresentazione dei numeri negativi
  - Un sistema intuitivo è quello di rappresentare esplicitamente segno e modulo
    - questa rappresentazione è infatti chiamata “modulo-segno”
  - Il sistema più utilizzato è però quello di usare il complemento alla base
    - in questo caso il complemento a due

Rappresentazione modulo-segno

- Il primo bit della sequenza rappresenta il segno
  - 0 = numero positivo, 1 = numero negativo
  - esempio con sequenze di quattro bit (n=4)



- Questa rappresentazione è di facile lettura, ma è difficile eseguire operazioni aritmetiche, perché il segno va trattato in maniera difforme dagli altri bit

Operazione di complementazione

- Data una sequenza di k cifre rappresentante il numero N in base b, si definisce:
  - **complemento alla base di N** il valore

$$C_b = b^k - N$$

  - **complemento diminuito di N** il valore

$$C_d = C_b - 1$$

  - segue, quindi, che

$$C_b = C_d + 1$$

Operazione di complementazione

- Nel sistema binario, C_b prende il nome di **complemento a due,** mentre C_d **complemento a uno**
- Osservazione: nel sistema binario, il complemento a uno di un numero si ottiene semplicemente sostituendo nella sequenza di bit gli 0 con 1 e gli 1 con 0
- N.B. valgono le seguenti proprietà

$$C_b(C_b(N)) = N$$

$$C_d(C_d(N)) = N$$

Operazione di complementazione

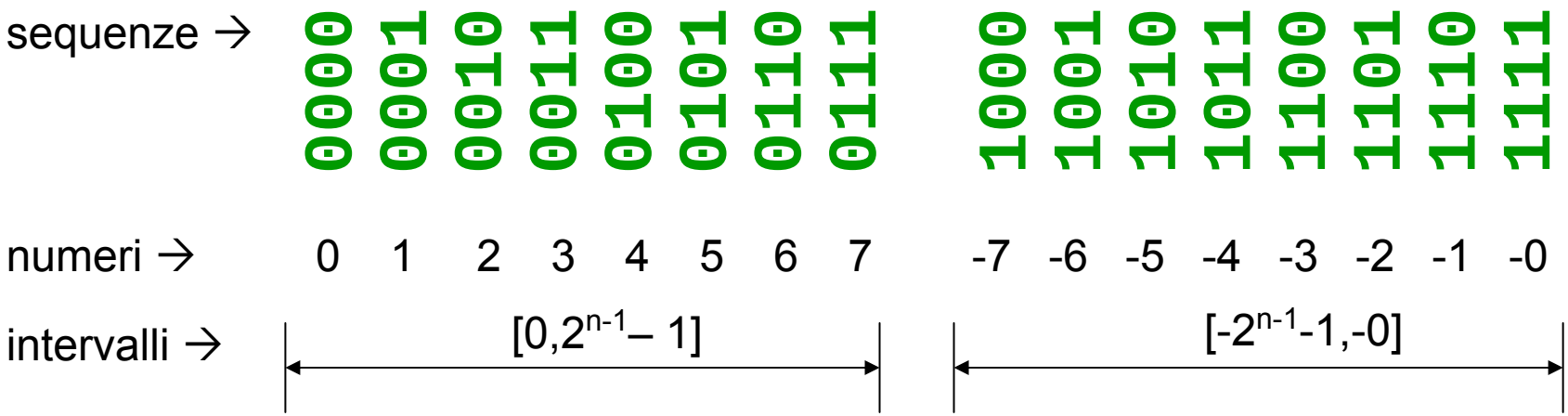
- Esempi:
  - $N = 001_{(2)}$
    - $C_d = 110_{(2)}$
    - $C_b = 110_{(2)} + 1 = 111_{(2)}$
  - $N = 100_{(2)}$
    - $C_d = 011_{(2)}$
    - $C_b = 011_{(2)} + 1 = 100_{(2)}$
  - $N = 000_{(2)}$
    - $C_d = 111_{(2)}$
    - $C_b = 111_{(2)} + 1 = 000_{(2)}$

Complemento a uno

- Data una sequenza che rappresenta un numero assoluto, si ottiene la sequenza che rappresenta il corrispondente numero:
  - **positivo**: usando la rappresentazione binaria tradizionale
  - **negativo**: complementando ad uno l'intera rappresentazione del corrispondente numero positivo

Complemento ad uno

- Gli intervalli rappresentati sono simmetrici ma con doppia rappresentazione dello zero

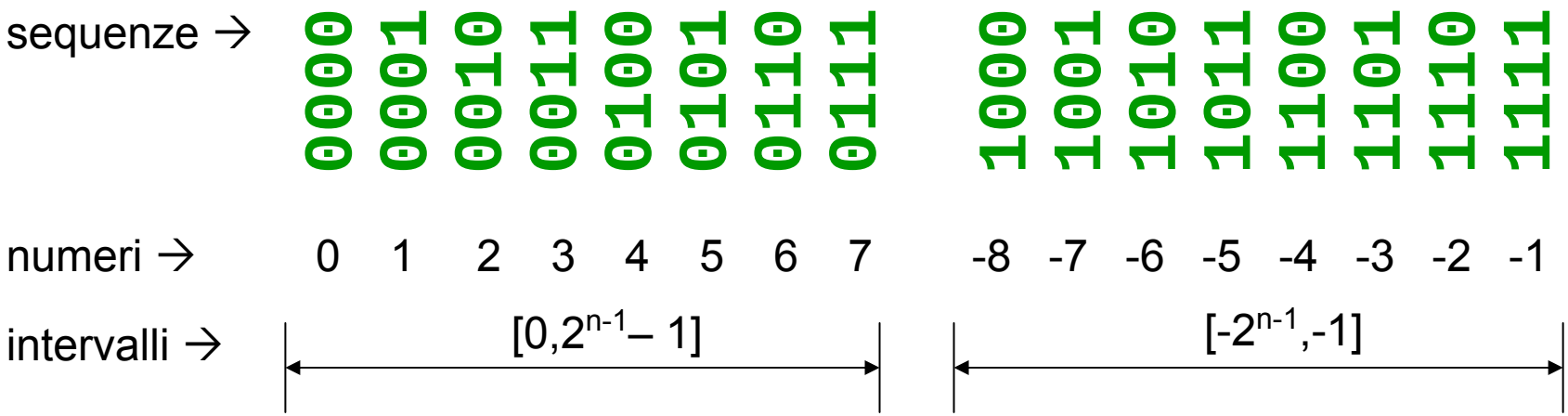


Esempi di complemento ad uno

Assoluto	Positivo	Negativo
$0101_{(2)}$	$0101_{(2)}$	$1010_{(2)}$
$01011_{(2)}$	$01011_{(2)}$	$10100_{(2)}$
$0000_{(2)}$	$0000_{(2)}$	$1111_{(2)}$

Rappresentazione in complemento a due

- Nella rappresentazione in **complemento a due** i numeri negativi si ottengono dai corrispondenti numeri positivi complementando ogni cifra e sommando uno
  - esempio
    - 5 = 0101
    - -5 = 1010 + 1 = 1011



- La prima cifra continua a rappresentare il segno

Rappresentazione in complemento a due

- Metodo più veloce per ottenere la rappresentazione di un numero negativo da quella del corrispondente positivo:
  - partendo da destra si lasciano invariate tutte le cifre fino al primo uno (incluso) e poi si invertono tutti i seguenti
  - esempio
    - N = 1001101010011000 0
    - 1001101010011000 0 ← invariato fino al primo uno (incluso)
    - 0110010101101000 0 ← invertite tutte le cifre seguenti
    - -N = 0110010101101000 0
- L'operazione di cambiamento di segno è eseguibile tramite il complemento a due
  - esempio
    - 5 = 0101
    - -5 = 1011 (ottenuto complementando a due il precedente)
    - 5 = 0101 (ottenuto complementando a due il precedente)

Esempi di somme in complemento a due

- 2 + 3 = 5

		0	0	1	0		riporti
2	+	0	0	1	0	+	
3	=	0	0	1	1	=	
5		0	1	0	1		

- (-2) + (-3) = -5

		1	1	0	0		riporti
-2	+	1	1	1	0	+	
-3	=	1	1	0	1	=	
-5		1	0	1	1		

Esempi di operazioni in complemento a due

- 2 + (-3) = -1

		0	0	0	0	riporti
2	+	0	0	1	0	+
-3	=	1	1	0	1	=
-1		1	1	1	1	

- Sottrazione: 3 - 2 = 3 + (-2) = 1

		1	1	1	0	riporti
3	+	0	0	1	1	+
-2	=	1	1	1	0	=
1		0	0	0	1	

Overflow e underflow

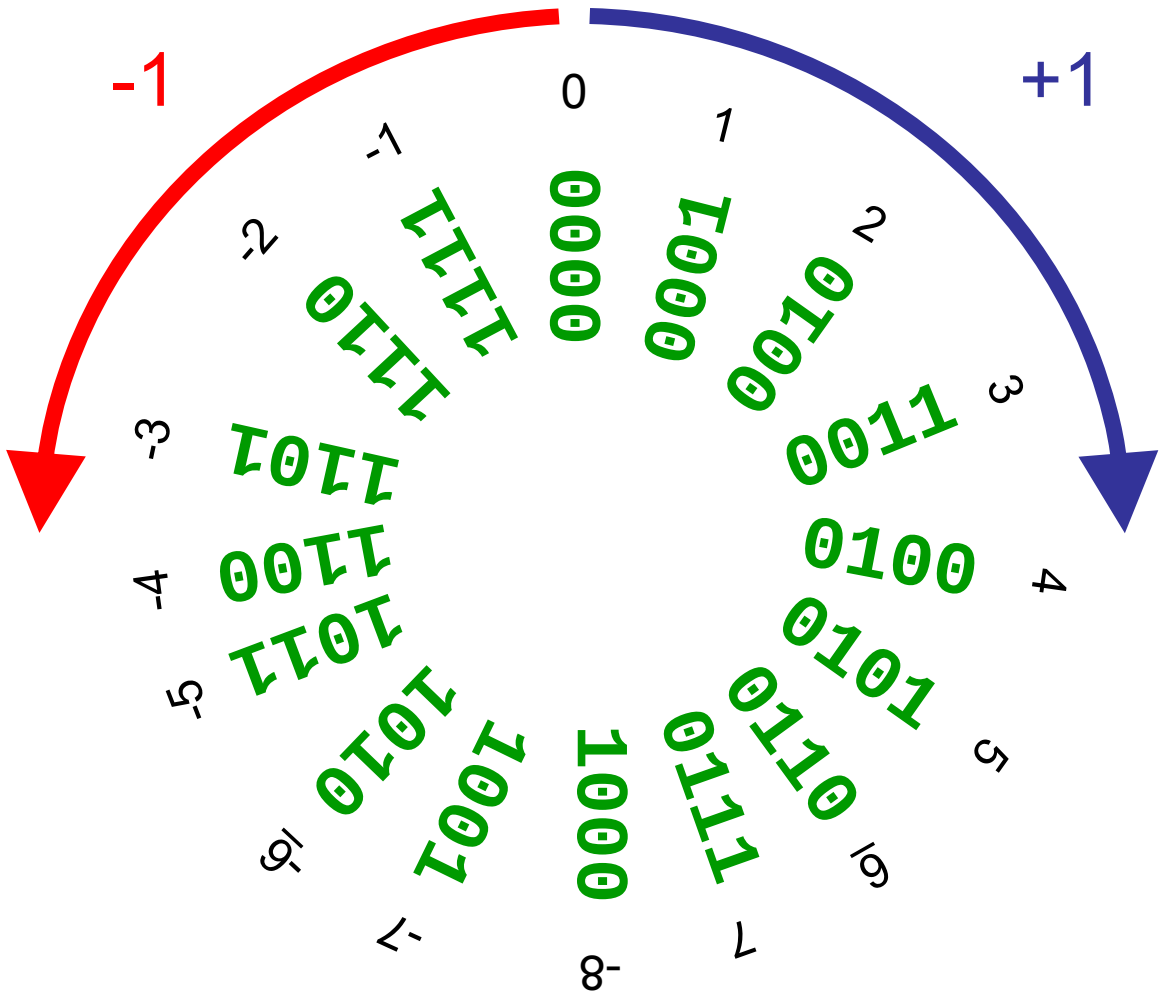
- Overflow: 2 + 6 = -8

		0	1	1	0	riporti
2	+	0	0	1	0	+
6	=	0	1	1	0	=
-8		1	0	0	0	

- Underflow: (-2) + (-7) = 7

		1	0	0	0	riporti
-2	+	1	1	1	0	+
-7	=	1	0	0	1	=
7		0	1	1	1	

Circolarità della rappresentazione



Aritmetica con il complemento a due

- la somma di due numeri relativi di n cifre si esegue come nel caso dei numeri naturali
  - l'eventuale riporto sulla cifra n+1 viene ignorato
  - se il risultato eccede $2^{n-1} - 1$ si ottiene un numero negativo che non corrisponde al risultato desiderato
  - se il risultato è minore di -2^{n-1} si ottiene un numero positivo che non corrisponde al risultato desiderato
- la sottrazione tra due numeri relativi di n cifre si esegue complementando a due il secondo numero ed eseguendo la somma

Domande

- Supponi che la piattaforma "**`Kimbo`**" consenta di rappresentare i numeri interi relativi in complemento a due tramite uno dei seguenti formati
  - formato "**`byte`**": 8 bit
  - formato "**`short`**": 16 bit
  - formato "**`int`**": 32 bit
  - formato "**`long`**": 64 bit
- Quali numeri è possibile rappresentare con ognuno di questi formati?

Risposte

- Con 8 bit è possibile rappresentare in complemento a due i numeri da -2^7 a 2^7-1
  - l'intervallo, dunque, è [-128,+127]
- Con 16 bit è possibile rappresentare in complemento a due i numeri da -2^{15} a $2^{15}-1$
  - $2^{15} = 2^5 \times 2^{10} = 32 \times 1024 \approx 32 \times 1000 = 32.000$
  - l'intervallo è infatti [–32.768,+32.767]
- Con 32 bit è possibile rappresentare in complemento a due i numeri da -2^{31} a $2^{31}-1$
  - $2^{31} = 2 \times 2^{10} \times 2^{10} \times 2^{10} = 2 \times 1024 \times 1024 \times 1024 \approx 2 \times 1000 \times 1000 \times 1000 = 2.000.000.000$
  - l'intervallo è infatti [–2.147.483.648,+2.147.483.647]
- Con 64 bit è possibile rappresentare in complemento a due i numeri da -2^{63} a $2^{63}-1$
  - $2^{63} = 2^3 \times 2^{10} \times 2^{10} \times 2^{10} \times 2^{10} \times 2^{10} \times 2^{10} \approx 2 \times 10^{18}$
  - l'intervallo è
    [–9.223.372.036.854.775.808, +9.223.372.036.854.775.807]

Rappresentazione di numeri razionali

- La difficoltà della rappresentazione dei numeri razionali deriva dal fatto che un intervallo arbitrariamente piccolo ne contiene infiniti
- E' dunque possibile rappresentarne solo un sottoinsieme
- Una delle rappresentazioni più utilizzate è quella in virgola mobile, anche detta floating point

Rappresentazione in virgola mobile

- Un qualunque numero razione può essere riscritto in modo da evidenziare una **mantissa** ed un **esponente**
  - esempi
    - $235.023 = .235023 \times 10^{3}$ (.235023 è la mantissa, 3 è l'esponente)
    - $-0.0000235 = -.235 \times 10^{-4}$ (-.235 è la mantissa, -4 è l'esponente)
- Ciò è vero anche per i numeri in base due
  - esempio
    - $1001001.0101 = .10010010101 \times 2^{-7}$
- La rappresentazione in virgola mobile consiste proprio nel rappresentare mantissa ed esponente del numero razionale espresso in base due
  - un bit viene generalmente usato per il segno di m
  - una porzione dei bit viene dedicata alla rappresentazione della mantissa
  - i rimanenti bit vengono utilizzati per rappresentare l'esponente in complemento a due

Rifermimenti al libro di testo

- Per lo studio di questi argomenti, oltre a queste trasparenze, si fa riferimento al libro di testo, e in particolare al capitolo 1 (Architettura dei calcolatori):
  - 1.3 Tecnologia dei calcolatori
    - 1.3.1 Codifica dei dati
- Lo studente potrà trovare giovamento anche dalla lettura dei seguenti paragrafi del capitolo 11 (Tipi ed Espressioni)
  - 11.3 Tipi primitivi per numeri interi
    - 11.3.3 Rappresentazione dei numeri interi
    - 11.3.4 Altri tipi primitivi numerici interi
  - 11.4 Tipi primitivi per numeri reali