• BASI E CONVERSIONE

• BASE BINARIA

• BASE DECIMALE

• PROPRIETÀ DEI NUMERI ~~E DELLE~~

2^{10} 1K

$2^{20} = 1M$

$2^{30} = 1G$

• per rappresentare i numeri si utilizzo:

1) Un alfabeto, cioè un insieme di simboli

2) Una regola di composizione per scrivere Numeri esatti, cioè una codifica.

a codifica più usata è la codifica posizionale

Cosa vuol dire il numero

$$(abcd)_{10} = a \cdot 10^3 + b \cdot 10^2 + c \cdot 10 + d$$

~~dove~~ dove $a, b, c, d \in \{0, 1, 2, \ldots, 9\}$

- per i numeri binari: l'alfabeto è costituito dalle cifre 0 e 1.

cosa vuol dire $1001 = 1 \cdot 2^3 + 0 \cdot 2^2 + 0 \cdot 2 + 1 = 9$

- come effettuare la conversione di base?

2 Algoritmi:

- algoritmo nella base di arrivo

$$(12)_2 = 1 \cdot 10 + 2 = 1 \cdot (1010) + 10$$

occorre costruire la codifica binaria di

0
1
2
⋮
3
10

$= 1010 + 10 = \boxed{1100}$

↑ Risultato.

$$\begin{array}{r} 1010 + \\ 10 \\ \hline 1100 \end{array}$$

• algoritmo nella base di partenza

da

~~$A = a\cdot10^3 + b\cdot10^2 + c\cdot10 + d$~~

se $A = a\cdot 2^3 + b\cdot 2^2 + c\cdot 2 + d$ $\quad 0 \le d \le 1$

~~$\frac{A}{10} = (a\cdot10^3 + b\cdot10^2 + c) + d$~~

$A = (a\cdot 2^2 + b\cdot 2 + c)\cdot 2 + d$

$\frac{A}{2} = (a\cdot 2^2 + b\cdot 2 + c) + \frac{d}{2}$

rimane il resto della divisione intera

~~con $d<10$~~

~~$A = (a\cdot10^2 + b\cdot10 + c)\cdot10 + d$~~

~~$\frac{A}{10} = (a\cdot10^2 + b\cdot10 + c + \frac{d}{10}$~~ ~~← resto~~ della divisione intera.

esempio: $\frac{12}{2} = 6$ con resto 0

$\frac{6}{2} = 3$ con resto 0 $\quad$ ← rappresentazione binaria.

$\frac{3}{2} = 1$ con resto 1

$\frac{1}{2} = 0$ con resto 1

- Base esadecimale: 0
  |
  9 9
  A 10
  B 11
  C 12
  D 13
  E 14
  F 15

- quanto vale FF?

- conversione da esa a bin: a gruppi di 4 cifre.

- potenze del 2

$2^{10} = 1k$

$2^{20} = 1M$

$2^{30} = 1G$

$2^{40} = 1$ Tera.

Codifica dei caratteri

5

- Codifica ASCII: su 7+1 bit
- Codifica Unicode: su 16 bit = 65536 caratteri

i primi 128 caratteri unicode coincidono con l'ASCII.

- Codifica delle istruzioni.

• RAPPRESENTAZIONE DEI NUMERI INTERI

• Modulo e SEGNO

$\pm$ 7 bit
segno

campo di rappresentazione:

$-2^{N-1}+1,\ 2^{N-1}-1$

• complemento a 2 su N bit

$$X = \begin{cases} |x| & \text{se } x \geq 0 \Rightarrow \text{ossia il numero con notazione posizionale} \\ 2^N - |x| & \text{se } x < 0 \Rightarrow \text{è il complemento a 2} \end{cases}$$

$$x = \begin{cases} +X & \text{se } X < 2^{N-1} \\ -(2^N - X) & \text{se } X \geq 2^{N-1} \end{cases}$$

(ossia ~~l'ultimo~~ Il bit più significativo è 0)

(ossia il bit ~~più~~ più significativo è a 1)

:

:

- intervallo di rappresentazione

$$\left(-2^{N-1}, 2^{N-1}-1\right)$$

- Come si ottiene il numero?
  - Se x è negativo, la rappresentazione si ottiene con il complemento di $|X|$ e sommando 1
  - Da X, $|x| = \bar{X}+1$

- esempi: $N=4$;

+1 in complemento a 2 su 4 bit

- ~~[illegible]~~ è 0001

-6

preso 6 che vale 0110

$\bar{6} = 1001$; sommo uno

$|-6|_{\text{complemento a 2}} = 1010$

- Se ho 1010 su 4 bit, che numero rappresenti

è $-|x| = -|(0101)+1| = -(0110) = -6$

• In alternativa posso procedere direttamente tramite definizione.

se $x = -6$

$$X = 2^N - |x|$$

ma $|x| = 0110$

quindi $X = 2^4 - 0110$ ossia

$$\begin{array}{r} 10000\ - \\ 00110 \\ \hline \cancel{0}1010 \end{array}$$

bit

ecco il numero -6 in complemento a 2.

• Esercizi:

calcolare la rappresentazione in complemento a 2 di 4 (viene 0100) e di -4 (viene 1100)

• -1 (1001). e $+1$ (0001)

- perché si usa la rappresentazione in complemento a 2?

  - perché è più semplice fare i conti, ossia realizzare i circuiti che implementano tale operazione.

- Esempio: Sommiamo 2 numeri chiocciati in modulo e segno ed in complemento a 2
  - prendiamo −2 e +1 e sommiamoli su 4 bit

modulo e segno	complemento a due
1 0 1 0 (−2)	1 1 1 0 (−2)
0 0 0 1 (+1)	0 0 0 1 (+1)
1 0 1 1	1 1 1 1
⇓	⇓
errato: vale −3	corretto

- Come dovrei procedere in modulo e segno?

  occorre:

  - confrontare i 2 operandi
  - Se sono di segno diverso, occorre confrontare i valori assoluti e sottrarre il minore dal maggiore
  - Se i segni sono uguali, sommare i valori assoluti.
  - premettere al risultato il segno corretto.

dunque è:

```
 010   2
 001  -1
----------
 001   => aggiungere il segno (negativo)
```

=> dunque 1001 è di difficile implementazione.

- Nella somma di numeri in ~~modulo e segno~~ complemento a 2, quando concordi, occorre trascurare il CARRY -

- RAPPRESENTAZIONE DEI NUMERI IN VIRGOLA FISSA E MOBILE.

- Numeri virgola fissa: con il solito significato.

Numeri in virgola mobile: $N = \pm M \times B^{\pm E}$ ← esponente

mantissa

- Si adotta per rappresentare numeri o molto grandi o molto piccoli.

- la mantissa viene rappresentata in modulo e segno.

- Normalizzazione: la mantissa è del tipo 1._ _ così evito di rappresentare l'1

- polarizzazione: ossia per rappresentare anche negativi, sommo un fattore in modo che venga sempre positivo.

- l'IEEE ha codificato uno standard per tale rappresentazione.

Il complemento a 2

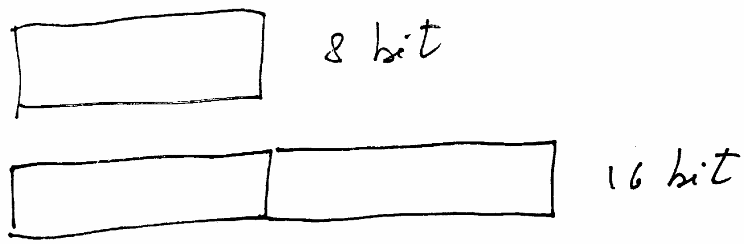
- Metodi alternativi per calcolare la rappresentazione di –X a partire da quella di X
  - Effettuare il complemento di ogni bit di X e aggiungere poi 1
    - rappresentazione di $+6_{dieci} = 0110_{C2}$ (NB ci vogliono 4 bit!!)
    - complemento di tutti i bit $\Rightarrow 1001_{C2}$ (corrisponderebbe a -7_{dieci})
    - aggiungere 1 $\Rightarrow 1010_{C2}$ (che corrisponde a -6_{dieci})
  - Partendo da destra e andando verso sinistra, lasciare invariati tutti i bit fino al primo 1 compreso, complementare tutti gli altri bit.
    - rappresentazione di $+6_{dieci} = 0110_{C2}$ (NB ci vogliono 4 bit!!)
    - gli ultimi due bit (_._.**1.0**) rimangono invariati
    - gli altri due bit vengono complementati (**1.0.1.0**$_{C2}$)

Complemento a 2 – Alcune osservazioni

- i numeri positivi iniziano con **0**, quelli negativi con **1**
- data la rappresentazione di un numero su **k** bit, la rappresentazione dello stesso numero su **k+1** bit si ottiene aggiungendo (a sinistra) un bit uguale al primo (**estensione del "segno"**)
  - Rappresentazione di –6 su 4 bit = 1010
  - Rappresentazione di –6 su 5 bit = 11010
  - Rappresentazione di –6 su 8 bit = 11111010
- la sottrazione si effettua come somma algebrica
  - 4 – 6 = +4 + (–6) = 0100 + 1010 = 1110 = –2
  - 9 – 6 = +9 + (–6) = 01001 + 11010 = [1]00011 = +3

Esercizi:

1) • Si dispone delle seguenti locazioni:



• Scegliere in quale locazione, e con quale formato, è possibile rappresentare (e rappresentarli)

- ± 5
- ± 254 | -255
- ± 125.000

2)

- Si dispone di 8 bit
- Si vogliono svolgere le seguenti somme:

  2+5; 12-27; -125-100

- Discutere l'implementabilità dell'operazione su 8 bit e calcolare il risultato dell'operazione

- Si vogliono sommare 125 e 100 su 8 bit in complemento a 2.
  - Discuterne la rappresentabilità.

A6

Sistemi di numerazione

A6.1 Numeri binari

La notazione decimale rappresenta i numeri come potenze di 10, ad esempio

$$1728_{\text{decimale}} = 1 \times 10^3 + 7 \times 10^2 + 2 \times 10^1 + 8 \times 10^0$$

Non c'è nessun motivo particolare per la scelta del numero 10, a parte il fatto che molti sistemi di numerazione storici sono stati messi a punto da persone che contavano con le dita. Altri sistemi numerici, con base 12, 20 o 60, sono stati usati da varie culture nel corso della storia dell'umanità. I computer, invece, usano un sistema numerico con base 2 perché è molto più facile costruire componenti elettronici che funzionino con due valori, che possono essere rappresentati da una corrente che scorre oppure no, piuttosto che rappresentare 10 valori diversi di un segnale elettrico. Un numero scritto in base 2 viene anche detto numero *binario*. Ad esempio

$$1101_{\text{binario}} = 1 \times 2^3 + 1 \times 2^2 + 0 \times 2^1 = 8 + 4 + 1 = 13$$

Per le cifre che seguono il punto "decimale", si usano le potenze negative di 2.

$$1.101_{\text{binario}} = 1 \times 2^0 + 1 \times 2^{-1} + 0 \times 2^{-2} + 1 \times 2^{-3} = 1 + 1/2 + 1/8 = 1 + 0.5 + 0.125 = 1.625$$

In generale, per convertire un numero binario nel suo equivalente decimale, valutate semplicemente le potenze di 2 che corrispondono alle cifre di valore 1 e sommatele.

La Tabella 1 mostra le prime potenze di 2.

Tabella 1 Potenze di due

2^0	1
2^1	2
2^2	4
2^3	8
2^4	16
2^5	32
2^6	64
2^7	128
2^8	256
2^9	512
2^{10}	1024
2^{11}	2048
2^{12}	4096
2^{13}	9192
2^{14}	16384
2^{15}	32768
2^{16}	65536

Per convertire in binario un numero intero decimale, dividetelo ripetutamente per 2, tenendo traccia dei resti e fermandovi quando il dividendo diventa 0. Scrivete quindi i resti come numero binario, iniziando dall'*ultimo*. Ad esempio

$$
\begin{aligned}
100 \div 2 &= 50 \text{ resto } \mathbf{0}\\
50 \div 2 &= 25 \text{ resto } \mathbf{0}\\
25 \div 2 &= 12 \text{ resto } \mathbf{1}\\
12 \div 2 &= 6 \text{ resto } \mathbf{0}\\
6 \div 2 &= 3 \text{ resto } \mathbf{0}\\
3 \div 2 &= 1 \text{ resto } \mathbf{1}\\
1 \div 2 &= 0 \text{ resto } \mathbf{1}
\end{aligned}
$$

Quindi, $100_{\text{decimale}} = 1100100_{\text{binario}}$.

Per convertire, invece, un numero frazionario minore di 1 nel formato binario, moltiplicatelo ripetutamente per 2. Se il risultato è maggiore di 1, sottraete 1; fermatevi quando il numero da moltiplicare diventa 0. Scrivete quindi le cifre che precedono il punto decimale come cifre binarie della parte frazionaria, iniziando dalla *prima*. Ad esempio

$$
\begin{aligned}
0.35 \cdot 2 &= \mathbf{0}.7\\
0.7 \cdot 2 &= \mathbf{1}.4\\
0.4 \cdot 2 &= \mathbf{0}.8\\
0.8 \cdot 2 &= \mathbf{1}.6\\
0.6 \cdot 2 &= \mathbf{1}.2\\
0.2 \cdot 2 &= \mathbf{0}.4
\end{aligned}
$$

A questo punto lo schema si ripete, quindi la rappresentazione binaria di 0.35 è 0.01 0110 0110 0110 ...

Per convertire in binario un numero in virgola mobile, convertite la parte intera e la parte frazionaria separatamente.

A6.2 Numeri interi in complemento a due

Per rappresentare numeri interi negativi esistono due comuni notazioni, chiamate "modulo e segno" e "complemento a due". La notazione con "modulo e segno" è semplice: si usa il bit più a sinistra per il segno (0 = positivo, 1 = negativo). Ad esempio, usando numeri di 8 bit:

$$-13 = 10001101_{\text{modulo e segno}}$$

Tuttavia, costruire circuiti per sommare numeri diventa un po' più complicato quando occorre considerare il segno. La rappresentazione in complemento a due risolve questo problema. Per comporre il complemento a due di un numero:

- Cambiate il valore di tutti i bit.
- Quindi, aggiungete 1.

Ad esempio, per calcolare –13 come valore di 8 bit, dapprima cambiate il valore di tutti i bit di 00001101, ottenendo 11110010, quindi aggiungete 1:

$$-13 = 11110011_{\text{complemento a due}}$$

Ora, non serve nessun circuito specifico per sommare due numeri: seguite semplicemente le normali regole dell'addizione, con il riporto nella posizione successiva nel caso in cui la somma delle cifre e del riporto precedente sia 2 o 3. Ad esempio:

```
         1 1111 111
+13       ¦0000¦1101
-13       ¦1111¦0011
         ___________
         1¦0000¦0000
```

Sono importanti, però, soltanto gli ultimi 8 bit, per cui +13 e –13 hanno somma 0, come dovrebbero.

In particolare, la rappresentazione in complemento a due di –1 è 1111...1111, cioè tutti i bit valgono 1.

Il bit più a sinistra di un numero in complemento a due vale 0 se il numero è positivo e 1 se è negativo.

La rappresentazione in complemento a due con un dato numero di bit può rappresentare un numero negativo in più rispetto al numero di valori positivi rappresentabili; ad esempio, i numeri in complemento a due con 8 bit variano da –128 a +127.

Questo fenomeno è fonte di errori di programmazione. Ad esempio, considerate il codice seguente:

```
byte b = ...;
if (b < 0) b = -b;
```

Questo codice non garantisce che, al termine, `b` sia non negativo. Se `b` vale inizialmente 128, il calcolo del suo opposto fornisce nuovamente il valore –128. (Provate: prendete 10000000, cambiate tutti i bit, e sommate 1).

A6.3 Numeri in virgola mobile

Lo standard IEEE-754 (IEEE, Institute for Electrical and Electronics Engineering) definisce le rappresentazioni per i numeri in virgola mobile. La Figura 1 mostra come i valori in singola precisione (float) e in doppia precisione (double) siano composti da:

- un bit di segno
- un esponente
- una mantissa

I numeri in virgola mobile usano la notazione scientifica, nella quale un numero viene rappresentato come

$$0.\,b_1 b_2 b_3 \ldots \times 2^e$$

In questa rappresentazione, e è l'esponente, mentre le cifre $b_1 b_2 b_3 \ldots$ compongono la mantissa. La rappresentazione *normalizzata* è quella avente $b_1 \neq 0$. Per esempio

$$\mathtt{100}_{\text{decimale}} = \mathtt{1100100}_{\text{binario}} = \mathtt{0.1100100}_{\text{binario}} \times 2^7$$

Poiché nel sistema di numerazione binario il primo bit di una rappresentazione normalizzata deve necessariamente essere 1, in realtà non viene memorizzato nella mantissa, per cui dovete sempre aggiungerlo per ottenere il valore vero. Ad esempio, la mantissa 0.1100100 viene memorizzata come 100100.

La parte della rappresentazione IEEE riservata all'esponente non usa né la rappresentazione in complemento a due, né quella in modulo e segno, ma viene aggiunto all'esponente vero una quantità fissa, detta *bias*. Tale quantità è 127 per i numeri in singola precisione e 1023 per quelli in doppia precisione. Ad esempio, l'esponente e = 7 verrebbe memorizzato come 134 in un numero in singola precisione.

Quindi

$$100_{\text{decimale}} = 0|100000110|10010000000000000000000_{\text{IEEE singola precisione}}$$

Figura 1 Rappresentazione IEEE per numeri in virgola mobile

1 bit	8 bit	23 bit
Segno	Esponente con bias $e + 127$	Mantissa (senza 1 iniziale)

Singola Precisione

1 bit	11 bit	52 bit
Segno	Esponente con bias $e + 1023$	Mantissa (senza 1 iniziale)

Doppia Precisione

Ci sono, poi, alcuni valori speciali. Fra questi:

- *Zero*: esponente con bias = 0, mantissa = 0.
- *Infinito*: esponente con bias = 11...1, mantissa = 0.
- *NaN* (*not a number*, non è un numero valido): esponente con bias = 11...1, mantissa ≠ 10...0.

A6.4 Numeri esadecimali

Poiché i numeri binari sono di difficile lettura per le persone, spesso i programmatori usano il sistema di numerazione esadecimale, con base 16. Le cifre vengono indicate con 0, 1, ..., 9, A, B, C, D, E, F (osservate la Tabella 2).

Tabella 2 Cifre esadecimali

Esadecimale	Decimale	Binario
0	0	0000
1	1	0001
2	2	0010
3	3	0011
4	4	0100
5	5	0101
6	6	0110
7	7	0111
8	8	1000
9	9	1001
A	10	1010
B	11	1011
C	12	1100
D	13	1101
E	14	1110
F	15	1111

Quattro cifre binarie corrispondono ad una cifra esadecimale: ciò rende semplici le conversioni fra valori binari e valori esadecimali. Ad esempio

$$11|1011|0001_{\text{binario}} = 3B1_{\text{esadecimale}}$$

In Java, i numeri esadecimali sono usati come valori per i caratteri Unicode, ad esempio `\u03B1` (la lettera greca alfa minuscola). I numeri interi esadecimali vengono indicati con il prefisso `0x`, come, ad esempio, `0x3B1`.

Appendice B

L'insieme dei caratteri ASCII

	0	1	2	3	4	5	6	7	8	9
0	nul	soh	stx	etx	eot	enq	ack	bel	bs	ht
1	nl	vt	ff	cr	so	si	dle	dc1	dc2	dc3
2	dc4	nak	syn	etb	can	em	sub	esc	fs	gs
3	rs	us	sp	!	"	#	$	%	&	'
4	(	)	*	+	,	-	.	/	0	1
5	2	3	4	5	6	7	8	9	:;	
6	<	=	>	?	@	A	B	C	D	E
7	F	G	H	I	J	K	L	M	N	O
8	P	Q	R	S	T	U	V	W	X	Y
9	Z	[	\	]	^	_	'	a	b	c
10	d	e	f	g	h	i	j	k	lm	
11	n	o	p	q	r	s	t	u	v	w
12	x	y	z	{	|	}	~	del		

I numeri a sinistra della tabella rappresentano le cifre più significative del codice del carattere, espresso in notazione decimale (0-127), mentre quelli in cima alla tabella rappresentano le cifre meno significative del codice del carattere. Per esempio, il codice del carattere 'F' è 70, mentre quello di '&' è 38.

Nota: L'insieme di caratteri ASCII è un sottogruppo dell'insieme di caratteri Unicode utilizzati da Java per rappresentare i caratteri delle principali lingue del mondo. Per ulteriori informazioni circa l'insieme di caratteri Unicode, è possibile visitare il sito `http://unicode.org`.

rappresentato con una cifra nella stessa base: più precisamente, il segno positivo viene rappresentato come 0, mentre il segno negativo viene rappresentato come 1 (nel caso di una generica base b il segno positivo viene rappresentato come 0, mentre il segno negativo viene rappresentato come $b - 1$). Come esempio si consideri il numero 6 in notazione decimale. Per la rappresentazione binaria in modulo e segno, occorrono 4 bit, con il bit più significativo che rappresenta il segno: $+6 = 0110$ e $-6 = 1110$.

Questo tipo di rappresentazione è *ridondante,* dato che permette due diverse rappresentazioni per il valore zero: *00....0* e *10...0.* Fissato il numero n di bit (compreso il bit di segno), il campo dei numeri rappresentabili è *simmetrico*, dato che risulta costituito da tutti gli N tali che: $-2^{n-1} - 1 \leq N \leq 2^{n-1} - 1$ (in generale, per una base B, risulta $-B^{n-1} - 1 \leq N \leq B^{n-1} - 1$) e la codifica del numero $-N$ si ottiene immediatamente da quella di $+N$ semplicemente cambiando il bit di segno.

Tabella 2.6 Rappresentazioni binarie su 4 bit di numeri interi con segno.

	Valore rappresentato		
Valore binario $b_3b_2b_1b_0$	**Notazione modulo e segno**	**Notazione complemento a 2**	**Notazione complemento a 1**
0000	+ 0	0	0
0001	+ 1	+ 1	+ 1
0010	+ 2	+ 2	+ 2
0011	+ 3	+ 3	+ 3
0100	+ 4	+ 4	+ 4
0101	+ 5	+ 5	+ 5
0110	+ 6	+ 6	+ 6
0111	+ 7	+ 7	+ 7
1000	− 0	− 8	− 7
1001	− 1	− 7	− 6
1010	− 2	− 6	− 5
1011	− 3	− 5	− 4
1100	− 4	− 4	− 3
1101	− 5	− 3	− 2
1110	− 6	− 2	− 1
1111	− 7	− 1	− 0

La rappresentazione in modulo e segno sembra quella più naturale perché, nei nostri calcoli manuali, siamo abituati a trattare con valori decimali espressi in modulo e