Architetture dei Calcolatori

Codifica dell'Informazione e Aritmetica Binaria

Prof. Francesco Lo Presti

Codifica di Dati e Istruzioni

- Algoritmi
  - Istruzioni che operano su dati
- Per scrivere un programma è necessario rappresentare dati e istruzioni in un formato tale che l'esecutore automatico sia in grado di
  - Memorizzare istruzioni e dati
  - Manipolare istruzioni e dati

Sistemi di Codifica

- Sistema di codifica (o *codifica*, o *codice*)
  - Usa un insieme di simboli di base (alfabeto)
  - I simboli dell' alfabeto possono essere combinati ottenendo differenti configurazioni (o *codici*, o *stati*), distinguibili l' una dall' altra
  - Associa ogni configurazione ad una particolare entità di informazione (la configurazione diventa un modo per rappresentarla)

Sistema di Codifica: Numeri Interi (Decimali)

- Alfabeto
  - Cifre "0", "1", "2", …, "9"
  - separatore decimale (",")
  - separatore delle migliaia (".")
  - Segni positivo ("+") e negativo ("-")
- Regole di composizione (sintassi)
  - Definiscono le combinazioni ben formate
  - 12.318,43
    - ✓ 123.1.8.43 ?
- Codice (semantica)
  - Associano ad ogni configurazione un' entità di informazione
  - 12.318,43 = $1\times10^4+2\times10^3+3\times10^2+1\times10^1+1\times10^0+4\times10^{-1}+3\times10^{-2}$
- Lo stesso alfabeto può essere usato per codici diversi
  - 123,456 = $1\times10^2+2\times10^1+3\times10^0+4\times10^{-1}+5\times10^{-2}+6\times10^{-3}$ [IT]
  - 123,456 = $1\times10^5+2\times10^4+3\times10^3+4\times10^2+5\times10^1+6\times10^0$ [UK]

Codifica Binaria

- Alfabeto binario: due simboli
- Utilizzata nei sistemi informatici
  - Si utilizza una grandezza fisica (luminosità, tensione elettrica, la corrente elettrica), per rappresentare informazione
- BIT (BInary digiT)
  - unità elementare di informazione rappresentabile con dispositivi elettronici
    - ✓ Due soli stati
  - con 1 bit si possono rappresentare 2 stati
  - 0/1, on/off, si/no

Codifica Binaria

- Quanti oggetti/entità si possono codificare con k bit?
  - 1 bit ⇨ 2 stati {0,1} ⇨ 2 oggetti (e.g., Vero, Falso)
  - 2 bit ⇨ 4 stati {00,01,10,11} ⇨ 4 oggetti
  - 3 bit ⇨ 8 stati {000,001,010,…,111} ⇨ 8 oggetti
  - …
  - K bit ⇨2^k stati ⇨ 2^k oggetti
- Quanti bit servono per codificare N oggetti?
  - $N \leq 2^k \Rightarrow k \geq \log_2 N \Rightarrow k = \lceil \log_2 N \rceil$ (intero superiore)
- Attenzione: Ipotesi implicita che i codici abbiano tutti la stessa lunghezza

Esempio di Codifica Binaria

- Problema: assegnare un codice binario univoco a tutti i giorni della settimana
- Giorni della settimana: N=7, ⇨ $k \geq \lceil \log_2 N \rceil \Rightarrow k=3$
  - Con 3 bit, 8 configurazioni
- Possibile soluzione
  - Lunedì 000
  - Martedì 001
  - Mercoledì 010
  - Giovedì 011
  - Venerdì 100
  - Sabato 101
  - Domenica 110

Codifica binaria dei caratteri

- Quanti sono gli oggetti da rappresentare?
  - 26 lettere maiuscole e 26 lettere minuscole
  - 10 cifre
  - Circa 30 simboli di interpunzione (, ; …)
  - Circa 30 caratteri di controllo (EOF, CR, …)
- In totale circa 120 oggetti complessivi
  - $k = \lceil \log_2 120 \rceil = 7$
- *Codice ASCII* (American Standard Code for Information Interchange) utilizza 7 bit
  - Può rappresentare 2^7=128 caratteri detti caratteri ASCII standard
- *Codice ASCII esteso* utilizza 8 bit
  - Può rappresentare 2^8=256 caratteri detti caratteri ASCII estesi
  - Comprende i caratteri ASCII standard e alcuni caratteri semigrafici (cornici, lettere nazionali, simboli matematici, …)

Codifica binaria dei caratteri (2)

- *Codice UNICODE* a 16 bit (http://www.unicode.org)
  - 65.536 = 2^{16} *code points*
  - Semplifica la scrittura del software
  - 336 code points: alfabeti latini
  - 112 accenti e simboli diacritici
  - Greco, cirillico, ebraico, 21.000 ideogrammi cinesi, 11.000 sillabe coreane …
- Un consorzio assegna quello che resta, ma durerà poco
- Codifica dei caratteri e linguaggio di programmazione
  - Java usa UNICODE (16 bit per char)
  - C/C++ usano ASCII esteso (8 bit per char)

Codice ASCII

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 dei Numeri: Numeri e Numerali

- Numero: Entita' Astratta
- Numerale: Sequenza di caratteri che rappresenta un numero in un dato sistema di numerazione
  - Lo stesso numero e' rappresentato da numerali diversi in sistemi diversi
    - ✓ 156 sistema decimale
    - ✓ CLVI in numeri romani
- Un numerale rappresenta un numero solo se si specifica un sistema di numerazione
  - Lo stesso numerale rappresenta diversi numeri in diverse notazioni
  - Es: 100100
    - ✓ Centomilacento in notazione decimale
    - ✓ Trantasei in notazione binaria

Sistemi di Numerazione

- Posizionali: Il valore di un simbolo dipende dalla posizione che esso occupa all'interno della configurazione.

$$a_{n-1}a_{n-2}\ldots a_0\ _B$$

  - Differiscono per la scelta della base B
  - Base indica il numero di simboli usati (0,…,B-1)
  - Decimale (B=10), Binario (B=2), Ottale (B=8), Esadecimale (B=16)
- Non posizionali: Il valore di un simbolo non dipende dalla posizione che esso occupa all'interno della configurazione
  - Numeri Romani

Sistemi di Numerazione Posizionale

- Sistema Decimale: Sistema di Numerazione Posizionale in base B=10

  $a_{n-1}a_{n-2}...a_0\ _{10}$

  rappresenta

  $a_{n-1}*10^{n-1}+...+a_1*10^1+a_0*10^0$
  - Ciascuna cifra rappresenta il coefficiente di una potenza della base
- Sistema Binario: Sistema di Numerazione Posizionale in base b=2

  $a_{n-1}a_{n-2}...a_0\ _2$

  rappresenta

  $a_{n-1}*2^{n-1}+...+a_1*2^1+a_0*2^0$

Conversione Binario⇨Decimale

- $a_{n-1}a_{n-2}...a_0\ _2 \Rightarrow a_{n-1}*2^{n-1}+...+a_1*2^1+a_0*2^0$
- $10110_2 \Rightarrow 1*2^4+0*2^3+1*2^2+1*2^1+0*2^0=2^4+2^2+2^1=16+4+2=22$
  - Si sommano le potenze di due a cui corrisponde un 1 nel numerale

Conversione Decimale⇨Binario

- Il numero binario si ottiene scrivendo la serie dei resti delle divisioni per 2, iniziando dall'ultimo resto ottenuto

573:2	⇨	286	resto 1 -> a_0
286:2	⇨	143	resto 0 -> a_1
143:2	⇨	71	resto 1 -> a_2
71:2	⇨	35	resto 1 -> a_3
35:2	⇨	17	resto 1 -> a_4
17:2	⇨	8	resto 1 -> a_5
8:2	⇨	4	resto 0 -> a_6
4:2	⇨	2	resto 0 -> a_7
2:2	⇨	1	resto 0 -> a_8
1:2	⇨	0	resto 1 -> a_9

$1000111101_2=573_{10}$

Conversione Decimale⇨Binario

18:2=9 resto 0
9:2=4 resto 1
4:2=2 resto 0
2:2=1 resto 0
1:2=0 resto 1

$10010_2=18_{10}$

137:2=68 resto 1
68:2=34 resto 0
34:2=17 resto 0
17:2=8 resto 1
8:2=4 resto 0
4:2=2 resto 0
2:2=1 resto 0
1:2=0 resto 1

$10001001_2=137_{10}$

Conversione Decimale⇨Binario

- Sistema di Numerazione in base B (B=2 caso particolare)
- $a_{n-1}a_{n-2}...a_0{}_B \Rightarrow a_{n-1}*B^{n-1}+...+a_1*B^1+a_0*B^0=N$
  $\Rightarrow a_{n-1}*B^{n-1}+...+a_1*B+a_0=N$
- Dividendo il numero per la base B, si ottiene:
- $N1=a_{n-1}*B^{n-2}+a_{n-2}*B^{n-3}+...+a_1+a_0/B$
  - Quozionte: $N1=a_{n-1}*B^{n-2}+a_{n-2}*B^{n-3}+...+a_1$
  - Resto: a_0
- Il resto della divisione corrisponde all'ultima cifra a_0 della rappresentazione in base B del numero N
- Applicando lo stesso procedimento al quoziente N1 si ottiene la penultima cifra a_1 della rappresentazione in base B
- Ripetendo la procedura e' possibile ottenere tutte le altre cifre
  - Ci si ferma quando il quoziente e' uguale a 0

Intervalli rappresentati

- Rappresentando gli interi positivi e lo zero in notazione binaria con n cifre (bit) si copre l' intervallo $[0, 2^n-1]$
- Si sfruttano tutte le 2^n disposizioni

Esempio: n=3 [0,7]

0	000
1	001
2	010
3	011
4	100
5	101
6	110
7	111

NB *Anche gli 0 non significativi devono essere rappresentati*

Conversione decimale-binario per i numeri con parte frazionaria

- Procedimento che a partire da una frazione F<1
  - Ottiene la rappresentazione di F in binario (o una sua approssimazione)
- Sequenza ripetuta di moltiplicazioni per 2
  - Si ottiene una parte intera (che può essere 0 o 1) ed una parte decimale F_1
  - La parte intera rappresenta il bit più significativo, mentre F_1 viene moltiplicato per 2
  - Si ottiene una parte intera (che può essere 0 o 1) ed una parte decimale F_2
  - La parte intera rappresenta il secondo bit più significativo, mentre F_2 viene moltiplicato per 2
  - … fino a quando F_i è uguale e 0 oppure abbiamo esaurito il numero di cifre a disposizione per la rappresentazione
    - ✓ Nel secondo caso introduciamo un' approssimazione

Esempio: conversione parte frazionaria

- Conversione di 0.73 con 8 bit

0.73·2 = 1.46	1	cifra più significativa
0.46·2 = 0.92	0	
0.92·2 = 1.84	1	
0.84·2 = 1.68	1	
0.68·2 = 1.36	1	
0.36·2 = 0.72	0	
0.72·2 = 1.44	1	
0.44·2 = 0.88	0	cifra meno significativa

- Quindi $0.73_{10} = 0.10111010_2$
- Con 8 bit rappresentiamo il numero 0.7265625

Ordini di grandezza

- Le potenze di 2
  - 2^0 ... 2^9 = 1, 2, 4, 8, 16, 32, 64, 128, 256, 512..
  - 2^{10} = 1024 ~ 10^3 1K
  - 2^{20} = 2^{10} 2^{10} =1.048.576 ~ 10^6 1M
  - 2^{30} = 2^{10} 2^{10} 2^{10} =1.073.741.824 ~ 10^9 1G
  - 2^{40} = ... =1.099.511.627.776 ~ 10^{12} 1T
  - 2^{50} = ... =1.125.899.906.842.624 ~ 10^{15} 1P
- Esempio: 2^{26} = $2^6 \cdot 2^{20}$ = 64 M
- Il numero di bit di un indirizzo determina le dimensioni della memoria indirizzabile

bit indirizzo	*Memoria*
16 bit	64 K
20 bit	1 M
32 bit	4 G

Il sistema esadecimale

- Per i numerali esadecimali occorrono 16 cifre
  - {0,1, …, 9,A,B,C,D,E,F}
- Il valore delle cifre
  - 0,1, …, 9 hanno il valore consueto
  - A vale 10, B vale 11, …, F vale 15
- Stesso meccanismo della notazione decimale pesata
- Esempio: rappresentazione di BAC_{16}

$$N = 11 \cdot 16^2 + 10 \cdot 16^1 + 12 \cdot 16^0 = 2988$$

- Conversione Decimale ⇨ Esadecimale?

Conversioni tra base 16 e base 2

- Conversione esadecimale-binario
  - A ciascuna cifra esadecimale si fa corrispondere il gruppo di 4 bit che ne rappresenta il valore
  - Esempio

F	5	7	A	3	1
1111	0101	0111	1010	0011	0001

- Conversione binario-esadecimale
  - Partendo da destra si fa corrispondere a ciascun gruppo di 4 cifre binarie la cifra esadecimale che ne rappresenta il valore
- Si usano spesso codici esadecimali per rappresentare in modo compatto codici binarie
  - Rappresentazione più compatta: con sole 2 cifre esadecimali si rappresentano valori memorizzati in 8 bit
  - Esempio: rappresentazione dei colori RGB, indirizzi di memoria, indirizzi MAC

Esercizi

- Convertire in base 2
  - 175_{10}
  - 98_{10}
  - 125.32_{10} (parte intera e parte frazionaria)
- Convertire in base 10
  - 10010101101_2
  - 00110010101_2
- Convertire in base 16
  - 110101101011010_2
- Convertire in base 2
  - 175_{16}
- Un' altra base utilizzata è la base 8. Come convertire un numero da base 8 in base 2 (e viceversa) senza passare per la base 10?
- Se $(79)_{10} = (142)_b$ determinare il valore della base b

Interi positivi e negativi

- Finora abbiamo considerato solamente la rappresentazione dei numeri positivi (unsigned)
- Si utilizzano varie rappresentazioni per gli interi relativi
  - Modulo e segno
  - Complemento a 1
  - Complemento a 2
  - Eccesso
- Per rappresentare gli interi relativi, a parità di cifre *si dimezza l'intervallo dei valori assoluti*

Rappresentazione con modulo e segno

- Per rappresentare un numero con segno, si usa:
  - un bit per il segno: 0 per +, 1 per -
  - n-1 bit per il modulo
  - Intervallo di rappresentazione: $[-2^{n-1}+1, +2^{n-1}-1]$
- Esempio
  n = 4 bit intervallo [-7,+7]
  5 = 0101 -5 = 1101
- Osservazioni
  - Intervallo di rappresentazione *simmetrico*
  - Problema: *doppia rappresentazione dello zero*
    - ✓ Ad es. nel caso di 8 bit: 10000000 e 00000000
  - Ulteriore problema: addizione e sottrazione complicata da segno dei numeri, modulo dei numeri

Rappresentazione in complemento a 1

- un bit per il segno: 0 per +, 1 per -
- Per i numeri positivi: la prima cifra (0) indica il segno, le restanti n-1 il modulo
- Es: 5 ⇨ 0101 (4 cifre)
- *Per un numero negativo –A (A>0) si parte dal numerale corrispondente ad A e si complementa il numerale bit a bit*
- Es 5= 0101 ⇨ -5=1010
- I numerali positivi iniziano per 0, i negativi per 1
- Intervallo di rappresentazione con n bit: $[-2^{n-1}+1, +2^{n-1}-1]$
- È una notazione posizionale
  Pesi delle cifre: $(-2^{n-1}+1)$ 2^{n-2} ... 2^1 2^0

Esempio
n = 4 bit intervallo di rappresentazione [-7, +7]
5 = 0101
-5 = 1010 (-7+2)

Rappresentazione in complemento a 2

- un bit per il segno: 0 per +, 1 per -
- Per i numeri positivi: la prima cifra (0) indica il segno, le restanti n-1 il modulo
  - Es: 5 ⇨ 0101 (4 cifre)
- *Per un numero negativo –A (A>0)*
- *Prima metodo:*
  - *si parte dal numerale corrispondente ad A, si complementa il numerale bit a bit ... e si somma 1 al numerale ottenuto*
  - Esempio (4 bit)
    - ✓ $+6_{10} = 0110_{CP2}$
    - ✓ Complemento di tutti i bit: 1001
    - ✓ Aggiungere 1: $1001+1 = 1010_{CP2} = -6_{10}$
- Seconda metodo (pratico):
  - *Partendo da destra si lasciano invariati tutti i bit fino al primo 1 compreso, e poi si complementano i restanti*
  - Stesso esempio
    - ✓ Gli ultimi 2 bit rimangono invariati: __10
    - ✓ Complementare gli altri 2 bit: $1010_{CP2} = -6_{10}$

Rappresentazione in complemento a 2

- I numeri positivi hanno la stessa rappresentazione che hanno in complemento a 1
- I negativi si ottengono sommando 1 alla loro rappresentazione in complemento a 1
- Intervallo di rappresentazione con n bit: $[-2^{n-1}, +2^{n-1}-1]$
- Notazione Posizionale: Pesi delle cifre: -2^{n-1} 2^{n-2} ... 2^1 2^0
- Consideriamo il numero espresso in base 2

  $x_{n-1}x_{n-2}...x_0$
  - Il bit più significativo x_{n-1} assume peso negativo -2^{n-1}
- Quindi, il valore di un numero N espresso in complemento a 2 è

$$N = -x_{n-1}2^{n-1} + \sum_{i=0}^{n-2} x_i 2^i$$

Intervallo più esteso ma asimmetrico

Una sola rappresentazione dello zero

Conversione decimale-binario (CP1 e CP2)

- Abbiamo già visto come fare se il numero X è *positivo*
- Se il numero X è *negativo*:
  - Determinare il numero minimo di bit da usare (n_{min})
  - Convertire il valore assoluto di X in notazione a n_{min} bit
  - Complementare (a 1 o 2) il numerale così ottenuto
- Esempio: convertire $(-347)_{10}$ in CP2

  $2^8 = 256 < 347 < 512 = 2^9$

  intervallo con n bit: $[-2^{n-1}, +2^{n-1}-1]$

  pertanto $n_{min}=10$

  +347 in notazione a 10 bit:

-512	256	128	64	32	16	8	4	2	1
0	1	0	1	0	1	1	0	1	1

  complementando a 2:

-512	256	128	64	32	16	8	4	2	1
1	0	1	0	1	0	0	1	0	1

Esercizi

- Convertire in complemento a 2 con il numero di bit indicato
  - 15_{10} con 6 bit
  - -12_{10} con 6 bit
  - -5_{10} con 7 bit
  - -5_{10} con 6 bit
  - -5_{10} con 5 bit
  - -1_{10} con 32 bit
- Come si può convertire la rappresentazione di un numero in base 2 su *n* bit (in complemento a 2) ad una rappresentazione in base 2 su *m>n* bit?
  - Non è necessario passare per la rappresentazione in base 10

Soluzione

- Per convertire la rappresentazione di un numero in base 2 su *n* bit (in complemento a 2) ad una rappresentazione in base 2 su *n+1* bit
  - Si aggiunge a sinistra un bit uguale al bit più significativo (operazione di estensione del segno – *sign extension*)
- Esempio
  - Rappresentazione di –6 su 4 bit: 1010_{CP2}
  - Rappresentazione di –6 su 5 bit: 11010_{CP2}
  - Rappresentazione di –6 su 8 bit: 11111010_{CP2}
  - Rappresentazione di –6 su 16 bit: 1111111111111010_{CP2}

Rappresentazione in Eccesso

- Con n bit si possono rappresentare (fino a) 2^n valori
  - Unsigned: $[0,2^n-1]$
  - Modulo e segno: $[-2^{n-1}+1,+2^{n-1}-1]$
  - CP1: $[-2^{n-1}-1,+2^{n-1}-1]$
  - CP2: $[-2^{n-1},+2^{n-1}-1]$

- Rappresentazione in Eccesso k ($k \geq 0$)
  - Rappresentare con n bit l'intervallo $[-k,2^n-k-1]$
    - ✓ k deve essere tale che $k \leq 2^n$

Rappresentazione in Eccesso

- Il numero $X \in [-k,2^n-k-1]$ e' rappresentato dalla codifica binaria unsigned di X+k
  - $X \in [-k,2^n-k-1] \Rightarrow X+k \in [0,2^n-1]$
- Esempio

  n=4 bit: eccesso 8, intervallo [-8,+7]

  -3 -3+8=5 : 0101

  +4 +4+8=12 : 1100

- Proprieta'
  - Codifica -k = 000…000
  - Codifica 2^n-k-1=111…111
- Caso particolare $k=2^{n-1}$
  - Intervallo rappresentato e' $[-2^{n-1},+2^{n-1}-1]$ come in CP2

Rappresentazioni a confronto

Decimale	M&S	CP1	CP2	Ecc 8
+ 7	0111	0111	0111	1111
+ 6	0110	0110	0110	1110
+ 5	0101	0101	0101	1101
+ 4	0100	0100	0100	1100
+ 3	0011	0011	0011	1011
+ 2	0010	0010	0010	1010
+ 1	0001	0001	0001	1001
+ 0	0000	0000	0000	1000
- 0	1000	1111	-----	-----
- 1	1001	1110	1111	0111
- 2	1010	1101	1110	0110
- 3	1011	1100	1101	0101
- 4	1100	1011	1100	0100
- 5	1101	1010	1011	0011
- 6	1110	1001	1010	0010
- 7	1111	1000	1001	0001
- 8	-----	-----	1000	0000

Addizioni binarie

- Le addizioni fra numerali si effettuano cifra a cifra (come in decimale) portando il riporto alla cifra successiva

  0 + 0 = 0

  0 + 1 = 1

  1 + 0 = 1

  1 + 1 = 0 con il riporto di 1

- Esempio

3 + 2 = 5

```
  0011 +
  0010 =
------
  0101
```

Se il numero di cifre non permette di rappresentare il risultato si ha un trabocco (*overflow*) nella propagazione del riporto

Overflow

- Se si considerano due numeri interi senza segno rappresentati con n bit, si verifica la condizione di overflow ogni volta che il risultato supera 2^n-1
- Esempio
  - Sommiamo 5 e 11 su 4 bit (max rappr. = $2^4-1=15$)

    $(5)_{10} = (0101)_2$

    $(11)_{10} = (1011)_2$

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

Overflow

Addizioni in complemento a 2

- L' addizione per interi in complemento a due è identica a quella per interi senza segno
  - Si somma dal bit meno significativo portando il riporto ad ogni passo
  - Si trascura l' eventuale riporto in uscita dal bit più significativo
- L' unica differenza è nel modo in cui si controlla l' overflow
- Esempio
  - Sommiamo –3 e 4 su 4 bit
  - -3 = 1101
  - +4 = 0100

```
 1101 +
 0100 =
-------
10001
```

L'ultimo bit di riporto viene ignorato: il risultato è corretto

La regola di overflow

- Si verifica un overflow nell' addizione di interi in complemento a 2 se e solo se
  - Il risultato della somma di due interi positivi è un intero negativo
  - Il risultato della somma di due interi negativi è un intero positivo
- Esempio
  - Sommiamo 5 e 4 su 4 bit (max rappr. = $2^3-1=7$)
  - +5 = 0101
  - +4 = 0100

```
 0101 +
 0100 =
-------
 1001
```

Overflow: la somma di due interi positivi non può dare un intero negativo

Aritmetica in CP2: operazione di negazione

- Sia A rappresentato in complemento a due

  $A = a_{n-1}a_{n-2}...a_1a_0$
- La rappresentazione di -A si ottiene nel modo seguente:
  - Si complementa ogni bit a_i ottenendo il bit $\underline{a_i}$ = 1 se a_i=0
  - Si considera la stringa ottenuta come un intero senza segno e si somma 1 — 0 se a_i=1
- Esempio: A=18 su 8 bit
  - Rappresentazione di A in CP2: 00010010
  - Complemento (bit a bit): 11101101
  - Sommiamo 1: 11101101 +

```
 00000001 =
----------
 11101110
```

  - 11101110 = $-2^7 + 2^6 + 2^5 + 2^3 + 2^2 + 2^1 = -18$

Sottrazioni in complemento a 2

- Per sottrarre ad un minuendo (M) un sottraendo (S) basta sommare M e –S
- Conosciamo la maniera semplice di calcolare la negazione di un numero in complemento a due
  - Riusciamo a realizzare la sottrazione come se fosse una somma
  - Risparmiando così sulla circuteria della ALU
- Esercizio
  - Calcolare 2 – 5 su 4 bit
  - Calcolare (–3) – 1 su 4 bit

Overflow per la sottrazione

- Può verificarsi l' overflow con la sottrazione?
  - Si!
- Poiché la sottrazione è in effetti implementata con una negazione ed una addizione
- Vale la regola di overflow della somma
  - Si verifica se e soltanto se:
    - ✓ Il risultato della somma di due interi positivi è un intero negativo
    - ✓ Il risultato della somma di due interi negativi è un intero positivo

Condizioni di overflow

Operazione	A	B	Risultato
A+B	≥0	≥0	<0
A+B	<0	<0	≥0
A-B	≥0	<0	<0
A-B	<0	≥0	≥0

Esercizio

- Calcolare il risultato delle seguenti operazioni binarie tra numeri interi con segno rappresentati in complemento a 2 su 8 bit
- Scrivere l' equivalente rappresentazione dei numeri in decimale (verificando la correttezza del risultato ottenuto)
  - 00001101 + 00111101
  - 00001100 + 10110110
  - 00010100 - 01101111
  - 11110100 + 11101000