DIPARTIMENTO
DI INGEGNERIA
DELL'INFORMAZIONE

Rappresentazione delle informazioni nei calcolatori

(Appendice D del libro di testo)

Parte 1

Obiettivi

- Imparare come vari tipi di dato vengono rappresentati all'interno del computer
  - Numeri naturali (Interi senza segno)
    - Rappresentazione posizionale
  - Numeri Interi
    - Rappresentazione Modulo/Segno
    - Rappresentazione in complemento a 2
  - Numeri Reali
    - Rappresentazione in virgola fissa e in virgola mobile
  - Caratteri
    - ASCII e Unicode

Perche' e' importante studiare la rappresentazione dei dati

- Noi usiamo la notazione decimale per i numeri e dei simboli per le lettere (diversi anche tra linguaggi naturali!)
  - Il computer capisce 0-1
- I valori numerici sono infiniti
  - I bit che il computer dedica alla rappresentazione dei numeri **non** sono infiniti
    - Ci sono dei limiti alla rappresentazione dei valori numerici
      - Massimo/minimo valore rappresentabile
      - Precisione della rappresentazione
- I caratteri sono dei simboli: necessaria tabella di conversione

DIPARTIMENTO
DI INGEGNERIA
DELL'INFORMAZIONE

Notazione posizionale

- I numeri che siamo abituati a utilizzare sono espressi
  - ***In base decimale*** perché usiamo ***dieci cifre diverse*** (da 0 a 9)
  - Con notazione ***posizionale*** perché ***cifre uguali in posizioni diverse hanno valore diverso***
    - Il **peso** di una cifra è uguale alla base (10 in questo caso) elevata alla potenza della posizione della cifra
      - la posizione si incrementa da destra a sinistra a partire da 0

$$434 = 4 \cdot 10^2 + 3 \cdot 10^1 + 4 \cdot 10^0$$

Notazione posizionale

- In generale, con n cifre abbiamo

$$a_{n-1}a_{n-2}\dots a_0 = a_{n-1}10^{n-1} + a_{n-2}10^{n-2} + \dots + a_0 10^0, a_k \in \{0,1,\dots,9\}$$
$$k = 0,1,\dots,n-1$$

  - Ancora più in generale, se la base è *b*

$$a_{n-1}a_{n-2}\dots a_0 = \sum_{k=0}^{n-1} a_k b^k$$

- L'eventuale parte frazionaria, a destra del simbolo separatore, si valuta con potenze ***negative***

$$4.34 = 4 \cdot 10^0 + 3 \cdot 10^{-1} + 4 \cdot 10^{-2}$$

DIPARTIMENTO
DI INGEGNERIA
DELL'INFORMAZIONE

Notazione binaria

- I computer usano invece ***numeri binari***, cioè numeri rappresentati con notazione posizionale ***in base binaria***
  - la base binaria usa solo ***due cifre diverse***, 0 e 1
    - in base X si usano le cifre da 0 a X-1
  - la conversione da base binaria a decimale è semplice

$(1101)_2 =$

DIPARTIMENTO DI INGEGNERIA DELL'INFORMAZIONE

Notazione binaria

- I computer usano invece ***numeri binari***, cioè numeri rappresentati con notazione posizionale ***in base binaria***
  - la base binaria usa solo ***due cifre diverse***, 0 e 1
    - in base X si usano le cifre da 0 a X-1
  - la conversione da base binaria a decimale è semplice

$$(1101)_2 = (1 \cdot 2^3 + 1 \cdot 2^2 + 0 \cdot 2^1 + 1 \cdot 2^0)_{10} = (13)_{10}$$

$$(1.101)_2 =$$

Notazione binaria

- I computer usano invece ***numeri binari***, cioè numeri rappresentati con notazione posizionale ***in base binaria***
  - la base binaria usa solo **due *cifre diverse***, 0 e 1
    - in base X si usano le cifre da 0 a X-1
  - la conversione da base binaria a decimale è semplice

$$(1101)_2 = (1 \cdot 2^3 + 1 \cdot 2^2 + 0 \cdot 2^1 + 1 \cdot 2^0)_{10} = (13)_{10}$$

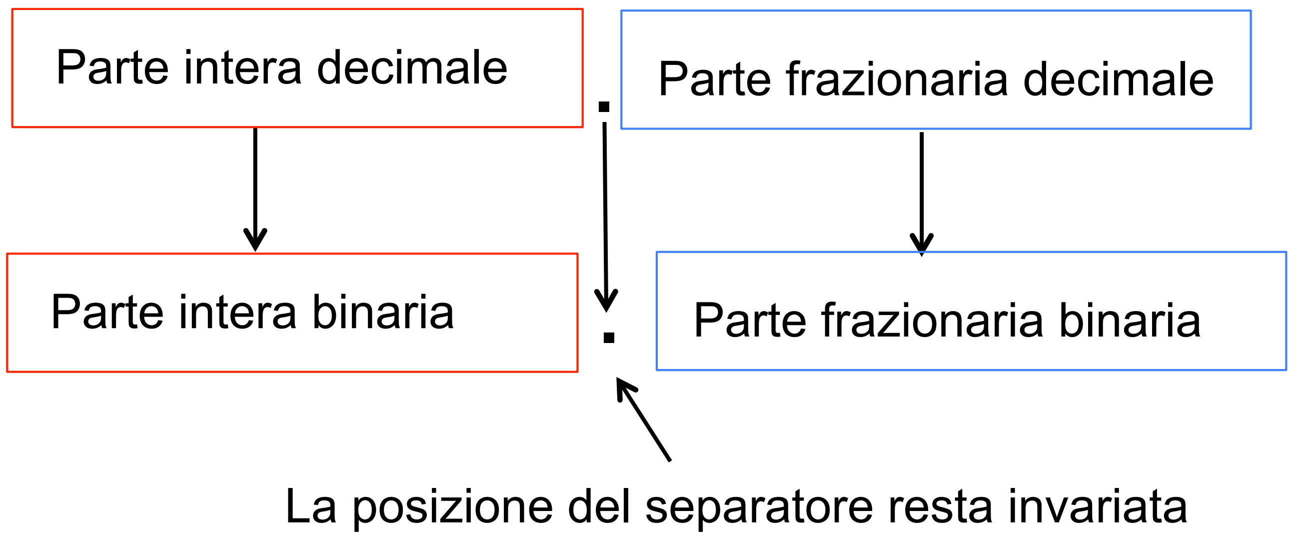
$$(1.101)_2 = (1 \cdot 2^0 + 1 \cdot 2^{-1} + 0 \cdot 2^{-2} + 1 \cdot 2^{-3})_{10} = (1.625)_{10}$$

- La rappresentazione binaria è **più facile** da manipolare per i computer **per motivi tecnologici**
  - perché è meno complicato costruire circuiti logici (digitali) che distinguono tra "acceso" e "spento", piuttosto che fra ***dieci livelli diversi*** di tensione elettrica o di un'altra grandezza fisica (intensità di corrente, luminosità, ecc.)

Notazione binaria

- La ***conversione*** di un numero ***da base decimale a base binaria*** è, invece, un po' più complessa

- La parte intera del numero va elaborata indipendentemente dalla eventuale parte frazionaria



Convertire la parte intera: es. 100_{10}

- Per convertire ***la sola parte intera***:
  - Si divide il numero per 2
  - Si elimina l'eventuale resto
  - Si continua a dividere per 2 il quoziente ottenuto fino a quando non si ottiene quoziente uguale a 0

- Il numero binario si ottiene scrivendo ***la sequenza dei resti*** delle divisioni, ***iniziando dall'ultimo*** resto ottenuto

- **Attenzione**: non fermarsi quando si ottiene **quoziente 1**, ma proseguire fino a **0**

Convertire la parte intera: es. 100_{10}

- Per convertire ***la sola parte intera***:
  - Si divide il numero per 2
  - Si elimina l'eventuale resto
  - Si continua a dividere per 2 il quoziente ottenuto fino a quando non si ottiene quoziente uguale a 0

100	/	2	=	50	resto	0
50	/	2	=	25	resto	0
25	/	2	=	12	resto	1
12	/	2	=	6	resto	0
6	/	2	=	3	resto	0
3	/	2	=	1	resto	1
1	/	2	=	0	resto	1

- Il numero binario si ottiene scrivendo ***la sequenza dei resti*** delle divisioni, ***iniziando dall'ultimo*** resto ottenuto

$$(100)_{10} = (1100100)_2$$

- **Attenzione:** non fermarsi quando si ottiene **quoziente 1**, ma proseguire fino a **0**

DIPARTIMENTO
DI INGEGNERIA
DELL'INFORMAZIONE

wooclap

Convertire la parte frazionaria, es 0.35d

- Per convertire ***la sola parte frazionaria***
  - si moltiplica il numero per 2
  - si sottrae 1 dal prodotto se questo è maggiore di 1
  - continuo fino a che il risultato è uguale a 0 oppure è un risultato già ottenuto
- Il numero binario si ottiene scrivendo la sequenza delle parti intere dei prodotti ottenuti, iniziando dal primo
- Se si ottiene un risultato già ottenuto in precedenza, il numero sarà periodico, anche se non lo era in base decimale

Convertire 0.35d

- Per convertire ***la sola parte frazionaria***, si moltiplica il numero per 2, sottraendo 1 dal prodotto se questo è maggiore di 1 e continuando a moltiplicare per 2 il risultato così ottenuto fino a quando non si ottiene un risultato uguale a 0 oppure un risultato già ottenuto in precedenza

- Il numero binario si ottiene scrivendo la sequenza delle parti intere dei prodotti ottenuti, iniziando dal primo

- Se si ottiene un risultato già ottenuto in precedenza, il numero sarà periodico, anche se non lo era in base decimale

$$
\begin{array}{l}
0.35 \cdot 2 = 0.7 \\
0.7 \cdot 2 = 1.4 \\
0.4 \cdot 2 = \boxed{0}.8 \\
0.8 \cdot 2 = \boxed{1}.6 \\
0.6 \cdot 2 = \boxed{1}.2 \\
0.2 \cdot 2 = \boxed{0}.4
\end{array}
$$

$$(0.35)_{10} = (0.01\overline{0110})_2$$

Provate

wooclap

Rappresentazione in virgola fissa

- La rappresentazione completa si ottiene componendo la parte intera e quella frazionaria
- **Rappresentazione in virgola fissa:** il separatore si trova sempre nello stesso punto rispetto alla sequenza di bit

$$(100)_{10} = (1100100)_2$$

$$(0.35)_{10} = (0.01\overline{0110})_2$$

$$(100.35)_{10} = (1100100.01\overline{0110})_2$$

Quanti numeri?

- Abbiamo visto che per i numeri interi non negativi si usa la ***rappresentazione binaria posizionale***

  $(101100)_2 = (44)_{10}$

- Se si usa una rappresentazione a **n** bit, si possono rappresentare i 2^n numeri naturali che sono compresi nell'intervallo

  $[0 , 2^n - 1] \cap Z$

n è la dimensione (in bit) della cella di memoria che contiene il numero

Esempio

- Con 8 cifre binarie (cioè 8 bit) si possono rappresentare 2^8 configurazioni, pari a 256 numeri diversi
  - $0_{10} = 0000\ 0000_2$
  - $1_{10} = 0000\ 0001_2$
  - $2_{10} = 0000\ 0010_2$
  - $3_{10} = 0000\ 0011_2$
  - $4_{10} = 0000\ 0100_2$
  - ...
  - $254_{10} = 1111\ 1110_2$
  - $255_{10} = 1111\ 1111_2 = 2^7+2^6+2^5+2^4+2^3+2^1+2^0$

Numeri interi relativi

- Come possiamo rappresentare **i numeri negativi**?
  - la rappresentazione più "naturale" o intuitiva è quella ***con modulo e segno***
  - si rappresenta il segno positivo o negativo del numero con il primo bit della sequenza (quello più a sinistra, ovvero il **più significativo**)
    - 0 rappresenta + mentre 1 rappresenta –
  - si rappresenta il modulo o valore assoluto del numero (che ovviamente è un numero non negativo)
    - Utilizzando i restanti bit a disposizione
    - Con la notazione binaria posizionale vista per i numeri non negativi

Numeri interi relativi

- Esempio: rappresentazione usando 6 bit
  - 1 bit di segno (0=positivo, 1=negativo)
  - 5 bit bit destinati al valore assoluto del numero
    - il loro spazio viene riempito di zeri a sinistra, dopo il bit del segno, se serve

$$(101100)_{2MS} = (-12)_{10}$$

$$(001100)_{2MS} = (+12)_{10}$$

DIPARTIMENTO
DI INGEGNERIA
DELL'INFORMAZIONE

Numeri interi relativi

- Se si usa una rappresentazione a **n** bit modulo/segno
  - si possono rappresentare i $2^n - 1$ numeri interi nell'intervallo

$$[-(2^{n-1} - 1), 2^{n-1} - 1] \cap Z$$

- 1 bit è infatti riservato al segno
- I restanti n-1 bit sono utilizzati per rappresentare
  - 2^{n-1} numeri non negativi (quando il primo bit è a 0)
  - 2^{n-1} numeri non positivi (quando il primo bit è a 1)
  - 0 ha due rappresentazioni

Rappresentazione in modulo/segno

- Esempio con 8 bit: $[-2^{8-1}-1, 2^{8-1}-1] = [-127, 127]$
  - $+127_{10} = 0\ 111\ 1111_2$
  - $+126_{10} = 0\ 111\ 1110_2$
  - … = …
  - $+001_{10} = 0\ 000\ 0001_2$
  - $+000_{10} = 0\ 000\ 0000_2$
  - $-000_{10} = 1\ 000\ 0000_2$
  - $-001_{10} = 1\ 000\ 0001_2$
  - … = …
  - $-126_{10} = 1\ 111\ 1110_2$
  - $-127_{10} = 1\ 111\ 1111_2$

Bit di segno +
(bit piu' significativo)

Bit di segno -
(bit piu' significativo)

Rappresentazione in modulo/segno

- In pratica non si usa
  - (piccolo) Problema: c'è una doppia rappresentazione per lo zero (+0 e –0), per cui **si "spreca" una configurazione**
  - **Problema (più grave): l'algoritmo per l'addizione di numeri così rappresentati è complesso**

- Idea sull'algoritmo?

Rappresentazione in modulo/segno

- Addizione **S = A + B** eseguita con numeri relativi rappresentati in modulo e segno
- Se segno(A) = segno(B)

  segno(S) = segno(A), $|S| = (|A| + |B|)$

  altrimenti

  se $|A| \geq |B|$

  segno(S) = segno(A), $|S| = (|A| - |B|)$

  altrimenti

  segno(S) = segno(B), $|S| = (|B| - |A|)$

Complemento a due

- Una rappresentazione più efficiente è quella denominata ***complemento a due***, così definita
  - dato un numero intero relativo

    $a \in [-2^{n-1}, 2^{n-1} - 1] \cap Z$

    la sua rappresentazione in complemento a due con n bit è

$$C2_n(a) = \begin{cases} \text{rappresentazione binaria di a con n bit} & \text{se } a \geq 0 \\ \text{rappresentazione binaria di } (a+2^{n}) \text{ con n bit} & \text{se } a < 0 \end{cases}$$

La rappresentazione in complemento a due di un valore **a** dipende (anche) dal numero **n** di bit a disposizione

Esempio

Esempio con $n = 3$, $2^n = 8$

In pratica, prendo l'intervallo dei **numeri negativi** e ne faccio una **traslazione**, spostandolo subito dopo l'intervallo occupato dai numeri positivi

4 2 1

- $a = 0$, $C2_3(0) = \mathbf{000}$
- $a = +1$, $C2_3(+1) = \mathbf{001}$
- $a = +2$, $C2_3(+2) = \mathbf{010}$
- $a = +3$, $C2_3(+3) = \mathbf{011}$

Osserviamo che la trasformazione è (ovviamene) invertibile, perché le rappresentazioni di numeri diversi sono diverse.

4 2 1

- $a = -4$, $C2_3(-4) = \mathbf{100}$ (è la rappresentazione con 3 bit di $a+2^n = 4$)
- $a = -3$, $C2_3(-3) = \mathbf{101}$ (è la rappresentazione con 3 bit di $a+2^n = 5$)
- $a = -2$, $C2_3(-2) = \mathbf{110}$ (è la rappresentazione con 3 bit di $a+2^n = 6$)
- $a = -1$, $C2_3(-1) = \mathbf{111}$ (è la rappresentazione con 3 bit di $a+2^n = 7$)

Qual e' l'intervallo che posso rappresentare?

$\forall a \in [-2^{n-1}, 2^{n-1} - 1] = [-2^2, 2^2 - 1] = [-4,3]$

Intervallo di rappresentazione in complemento a due

- $\forall a \in [-2^{n-1}, 2^{n-1} - 1] \cap Z$

$$C2_n(a) = \begin{cases} \text{rappresentazione binaria di a con n bit} & \textbf{se } \mathbf{a \geq 0} \\ \text{rappresentazione binaria di } (a+2^{\mathbf{n}}) \text{ con n bit} & \text{se } a < 0 \end{cases}$$

- Intervallo di rappresentazione dei numeri $0 \leq a \leq 2^{n-1}-1$
  - $0 \leq a \leq 2^{n-1} - 1 < 2^{\mathbf{n-1}} = 2^{\mathbf{n}} - 2^{\mathbf{n-1}}$
    - (perché $2^{\mathbf{n-1}} + 2^{\mathbf{n-1}} = 2 \times 2^{\mathbf{n-1}} = 2^{\mathbf{n}}$)

$$\boxed{\mathbf{0 \leq a < 2^{n} - 2^{n-1}}}$$

Intervallo di rappresentazione in complemento a due

- $\forall a \in [-2^{n-1}, 2^{n-1} - 1] \cap Z$

$$C2_n(a) = \begin{cases} \text{rappresentazione binaria di a con n bit} & \text{se } a \geq 0 \\ \text{rappresentazione binaria di } (a+2^n) \text{ con n bit} & \text{se } a < 0 \end{cases}$$

- Intervallo di rappresentazione dei numeri $-2^{n-1} \leq a < 0$
  - Devo determinare l'intervallo di $b = a + 2^n$
  - b è un numero positivo, $0 < -2^{n-1} + 2^n \leq b < 0 + 2^n$

$$\boxed{2^n - 2^{n-1} \leq b < 2^n}$$

Intervallo di rappresentazione in complemento a due

$0 \leq a < 2^n - 2^{n-1}$ $\quad$ $2^n - 2^{n-1} \leq b < 2^n$

- Questi intervalli hanno intersezione vuota
- L'intervallo totale $[0,2^n-1]$ è compatibile con il massimo numero positivo rappresentabile con **n** bit, che è proprio $2^n - 1$
- NB: nell'intervallo di rappresentazione binaria $[0,2^n-1]$ codifico i numeri interi appartenenti all'intervallo numerico $[-2^{n-1}, 2^{n-1} - 1] \cap Z$

Esempio

- Rappresentazione in complemento a 2 con 8 bit
  - Con 8 bit si rappresentano numeri in
    - $[-2^{n-1}, 2^{n-1} - 1] ==[-2^7,2^7-1]=[-128,127]$
    - $+127_{10} = 0\ 111\ 1111_2$
    - $+126_{10} = 0\ 111\ 1110_2$
    - … = …
    - $+001_{10} = 0\ 000\ 0001_2$
    - $+000_{10} = 0\ 000\ 0000_2$
    - $-001_{10} = 1\ 111\ 1111_2$
    - … = …
    - $-127_{10} = 1\ 000\ 0001_2$
    - $-128_{10} = 1\ 000\ 0000_2$

NB: Il bit più significativo Indica ancora il bit di segno

NB: c'è un'unica rappresentazione dello zero

Conversione in decimale

- La sequenza di n bit $a_{n-1}\ a_{n-2} \ldots a_1\ a_0$ rappresentata in complemento a 2
  - Descrive valori nell'intervallo $[-2^{n-1}, 2^{n-1}-1]$
  - Corrisponde al valore intero

$$-a_{n-1}2^{n-1} + a_{n-2}2^{n-2} + \ldots + a_1 2^1 + a_0 2^0$$

  - Esempio: 10011 in complemento a 2 con 5 bit
    - $-1 \times 2^4 + 0 \times 2^3 + 0 \times 2^2 + 1 \times 2^1 + 1 \times 2^0 = -16+2+1 = -13$

wooclap

Complemento a due

- Esempio: calcolare la rappresentazione in complemento a due del numero -13 a 8 bit
  - Calcolo $-13+2^8 = -13+256 = 243$
  - Converto in binario: $\mathbf{(243)_{10} = (11110011)_2}$
    - Verificare per casa!
- Alternativa più semplice ("inversione" in complemento a due)
  - Scrivo +13: **00001101**
  - Scambio 0 e 1 (ovvero "complemento a uno"): **11110010**
  - Aggiungo 1 al risultato: **11110011**

Complemento a due

- Esempio: calcolare la rappresentazione in complemento a due del numero -13 a 8 bit
  - Calcolo $-13+2^8 = -13+256= 243$
  - Converto in binario: $(\mathbf{243})_{10}$ =

```
243 / 2 = 121 resto 1
121 / 2 =  60 resto 1
 60 / 2 =  30 resto 0
 30 / 2 =  15 resto 0
 15 / 2 =   7 resto 1
  7 / 2 =   3 resto 1
  3 / 2 =   1 resto 1
  1 / 2 =   0 resto 1
```

  - $(\mathbf{11110011})_2$

Complemento a due

Esempio: calcolare la rappresentazione in complemento a due del numero -13 a 8 bit

- Alternativa più semplice ("inversione" in complemento a due)
  - Scrivo +13: 1101
    - **Poiche' sono richiesti 8 bit i primi 4 li metto a 0: 00001101**
  - Scambio 0 e 1 (ovvero "complemento a uno")
    - **00001101 diventa 11110010**
  - Aggiungo 1 al risultato: **11110011**

```
13 / 2 = 6 resto 1
 6 / 2 = 3 resto 0
 3 / 2 = 1 resto 1
 1 / 2 = 0 resto 1
```

```
11110010 +
       1 =
----------
11110011
```

wooclap

Complemento a 2: proprieta'

- Proprietà (dimostrabili)
  - il segno di un numero rappresentato in complemento a due è ancora il bit più a sinistra della rappresentazione
    - 0 se positivo o nullo, 1 se negativo
  - **la parte restante della rappresentazione NON è il valore assoluto del numero**
    - lo è **soltanto** per i numeri non negativi
  - non ci sono più configurazioni "sprecate"
    - con **n** bit si rappresentano 2^n numeri diversi
  - L'addizione di numeri rappresentati in complemento a due si esegue
    - facendo l'addizione binaria delle rappresentazioni
    - Ignorando un eventuale riporto nella posizione n

Somma in complemento a due

Somma con le regole della somma in colonna

0+0=0   0+1=1+0=1   1+1=0 con riporto di 1

```
128 64 32 16  8 4 2 1
0000 0101 +        +5+
0000 0010          +2
---------          --
0000 0111          +7
```

```
  1111   1
  0000 0101 +      +5+
  1111  1110       -2
  ----------       --
1|0000 0011        +3
```

```
1111   111
0111  1111 +       +127+
0000 0001             +1
----------         -----
1000 0000          -128!!!!
```

```
  1000 0000 +      -128+
  1111  1111          -1
  ----------       -----
1|0111 1111        +127!!!
```

Errori di overflow

Regola per l'errore di overflow

- Si ottiene un errore di ***overflow*** quando sommando due numeri con lo stesso segno si ottiene un risultato di segno opposto
  - Di fatto si ha superamento dei limiti di rappresentazione con i bit a disposizione
- Si osservano le due posizioni più significative
  - Se c'è riporto in una sola delle due allora c'è OVERFLOW

```
 1111  111
0111 1111 +
0000 0001
---------
1000 0000
```

Esercizi in complemento a 2

- **Esercizio**: qual è il massimo intero positivo rappresentabile con 4 bit?

- **Esercizio:** Eseguire la somma ***6+2*** a 4 bit

- **Esercizio**: eseguire la somma ***(-3) + (-7)*** a 4 bit (qual è il minimo intero negativo rappresentabile con 4 bit?)

- 4 bit: $[-2^3,+2^3-1]=[-8,+7]$

8 4 2 1

- Rappresento 6: 0110
- Rappresento 2: 0010
- Sommo: 1000

- $-a_{n-1}2^{n-1}+a_{n-2}2^{n-2}+\ldots+a_1 2^1+a_0 2^0 = -1 \times 2^3 = -8$ **!!!!! Sbagliato**

8 4 2 1

- $-7+2^4 = +9 \quad => 1001$
- $-3+2^4=+13 \quad => 1101$
- Sommo 1|0110

- Alternativa (complemento a 1, piu' 1)
  - 7 = 0111 => 1000 => 1001
  - 3 = 0011 => 1100 => 1101
  - Sommo 1|0110

- $-a_{n-1}2^{n-1}+a_{n-2}2^{n-2}+\ldots+a_1 2^1+a_0 2^0 = 1 \times 2^2 + 1 \times 2^1 = 6$ **!!!!! Sbagliato**

L'importanza del formato

- Data una sequenza di bit che rappresentano un numero in binario, per convertirlo in decimale dobbiamo conoscerne il formato
- Complemento a 2 a 8 bit
  - $1111\ 1111_2 = \mathbf{-2^7}+2^6+2^5+2^4+2^3+2^2+2^1+2^0=-1_{10}$
- Complemento a 2 a 16 bit
  - $0000\ 0000\ 1111\ 1111_2 = 2^7+2^6+2^5+2^4+2^3+2^2+2^1+2^0=+255_{10}$
- Modulo e segno a 8 bit
  - $1111\ 1111_2 = -(2^6+2^5+2^4+2^3+2^2+2^1+2^0)=-127_{10}$

DIPARTIMENTO
DI INGEGNERIA
DELL'INFORMAZIONE

Riassunto

- Rappresentazione binaria posizionale
  - n bit: 2^n valori: da 0 a 2^{n-1}
- Numeri con parte frazionaria
  - Notazione in virgola fissa: converto separatamente la parte intera e la parte frazionaria e le ricombino
- Numeri relativi
  - Modulo e segno: bit più significativo (primo a sx) per il segno, gli altri n-1 bit il valore assoluto; valori da $-2^{n-1}-1$ a $2^{n-1}-1$
  - Complemento a 2 con n bit:
    - numeri ≥ 0 solita rappresentazione, numeri < 0 rappresento $a+2^n$
    - Utilizzo tutte le 2^n combinazioni in modo univoco
  - Da -2^{n-1} a $2^{n-1}-1$