Codici

10 marzo 2020

Indice

1 Introduzione

- **Codice a base B** ◁ tecnica che impiega B simboli per rappresentare dati.
  - Un codice a base B consente di memorizzare e/o inviare informazioni su un supporto che può assumere B stati ben distinguibili.
  - I simboli usati da un codice sono detti **cifre** [digits].[1]
- **Parola** [word] ◁ sequenza di cifre che rappresenta un dato.
- **Risoluzione** di un codice [resolution] ◁ lunghezza delle parole usate dal codice.[2]
- **Overflow** ◁ accade se si tenta di rappresentare un dato usando un numero di cifre insufficiente.
- **Bit** ◁ cifra di un codice a base 2 (binary digit).

[1] In Matematica, invece, una cifra è un simbolo usato per rappresentare i numeri interi in notazione posizionale (le cifre in base 10 sono: 0, 1, 2, 3, 4, 5, 6, 7, 8, 9).

[2] La maggior parte dei codici usati in Elettronica e in Informatica sono a risoluzione fissa, ossia usano parole di una lunghezza prestabilita.

2 Notazione posizionale dei numeri positivi

- **Notazione posizionale a base B** di un numero positivo $\triangleleft$ rappresentazione del numero nella seguente forma, usando cifre scelte in un insieme composto da B cifre:

$$D_{n-1} \cdots D_0 . D_{-1} \cdots D_{-p}$$

  - La sequenza di cifre $D_{n-1} \cdots D_0$ rappresenta la **parte intera**.
  - La sequenza di cifre $D_{-1} \cdots D_{-p}$ rappresenta la **parte frazionaria**.
  - D_{n-1} è detta **cifra più significativa**.
    - Per i bit si usa la sigla **MSB** [most significant bit].
  - D_{-p} è detta **cifra meno significativa**.
    - Per i bit si usa la sigla **LSB** [least significant bit].
  - Il numero è associato attraverso il **metodo delle potenze pesate**:

$$D_{n-1}B^{n-1} + \ldots D_0 + D_{-1}B^{-1} + \ldots D_{-k}B^{-p} \quad (1)$$

L'espressione '576.34'

- nella notazione posizionale a base 10 rappresenta il numero

$$
\begin{aligned}
&5 \times 10^2 + 7 \times 10 + 6 + 3 \times 10^{-1} + 8 \times 10^{-2} \\
&= 5 \times 100 + 7 \times 10 + 6 + 3 \times 0.1 + 8 \times 0.01 \\
&= 500 + 70 + 6 + 0.3 + 0.04 \\
&= 576.34
\end{aligned}
$$

- nella notazione posizionale a base 8 rappresenta il numero

$$
\begin{aligned}
&5 \times 8^2 + 7 \times 8 + 6 + 3 \times 8^{-1} + 4 \times 8^{-2} \\
&= 5 \times 64 + 7 \times 8 + 6 + 3/8 + 4/64 \\
&= 320 + 56 + 6 + 0.375 + 0.0625 \\
&= 382.4375
\end{aligned}
$$

2.1 Metodo delle potenze pesate

- Per effettuare l'operazione (1) è comodo suddividerla in calcoli più semplici, tenendo conto dei passaggi attraverso una tabella, come mostrato in questo esempio:

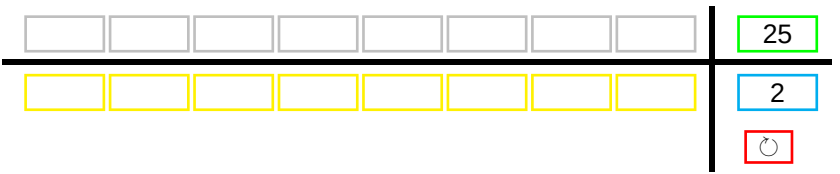
Parola a base 2 :	0	0	0	1	1	0	0	1
Potenze della base:								
Potenze pesate:								
Risultato:					↻			

2.2 Metodo delle divisioni successive

- Consente di determinare la rappresentazione della **parte intera** di numero positivo nella notazione posizionale a B cifre.
- Si inizia dividendo la parte intera del numero per la base:
  - il resto della divisione è la cifra meno significativa;
  - la parte intera del quoziente va ulteriormente divisa per la base:
    - il nuovo resto è la seconda cifra meno significativa;
    - la parte intera del nuovo quoziente è il nuovo numero da dividere.
- Si continua fino a ottenere zero come parte intera del quoziente.
- Per tener conto delle sequenze di numeri che scaturiscono dai calcoli, si può usare una tabella, come mostrato nel seguente esempio.

2.2.1 Esempio

- Nella cella con bordo verde è scritto il numero da codificare.
- Nella cella con bordo azzurro è scritta la base del codice.
- Nelle celle con bordo grigio sono scritte le parti intere dei quozienti ottenuti.
- Nelle celle con bordo giallo sono scritti i resti delle divisioni effettuate — questi resti formano la parola desiderata.



2.3 Metodo delle moltiplicazioni successive

- Consente di determinare la rappresentazione della **parte frazionaria** di numero positivo nella notazione posizionale a B cifre.
- Si inizia moltiplicando la parte frazionaria del numero per la base:
  - la parte intera del prodotto è la cifra più significativa;
  - la parte frazionaria del prodotto va ulteriormente moltiplicata per la base:
    - la parte intera del nuovo prodotto rappresenta la 2ª cifra più significativa;
    - la parte frazionaria del nuovo prodotto è il nuovo numero da moltiplicare.
- Si continua fino a ottenere zero come parte frazionaria del prodotto o la risoluzione prestabilita.[3]
- Per tener conto delle sequenze di numeri che scaturiscono dai calcoli, si può usare una tabella, come mostrato nel seguente esempio.

[3] Se la parte frazionaria dell'ultimo prodotto non è zero, la parola ottenuta rappresenta un'approssimazione troncata della parte frazionaria data.

2.3.1 Esempio

- Nella cella con bordo verde è scritto il numero da codificare.
- Nella cella con bordo azzurro è scritta la base del codice.
- Nelle celle con bordo grigio sono scritte le parti frazionarie dei prodotti ottenuti.
- Nelle celle con bordo giallo sono scritte le parti intere dei prodotti ottenuti — queste parti intere formano la parola desiderata.

3 Codice binario naturale

- E' il codice binario più semplice per rappresentare i **numeri naturali**.

3.1 Definizione

- E' basato sulla notazione posizionale: la parola $b_{n-1} \dots b_0$ rappresenta il numero

$$b_{n-1}2^{n-1} + \dots b_0 \tag{2}$$

3.1.1 Osservazioni

- La versione con parole a n bit permette di rappresentare i numeri naturali compresi tra 0 e $2^n - 1$.
- I numeri pari terminano con 0 e quelli dispari con 1.

3.2 Codifica

- Per determinare la **parola di lunghezza minima** corrispondente a un numero, si può usare il metodo delle divisioni successive (vedi §2.2).
- Per determinare la **parola di n bit** che rappresenta un numero x, basta premettere abbastanza zeri alla parola di lunghezza minima che rappresenta x.

3.2.1 Esempi

- 110 rappresenta 6 a lunghezza minima.
- 00110 rappresenta 6 a 5 bit.
- Non è possibile rappresentare 6 con una parola di 2 bit — si avrebbe l'overflow.

3.3 Decodifica

- Si può usare il metodo delle potenze pesate (vedi esempio §2.1).

3.4 Somma

- Si può usare la procedura valida per i sistemi numerici posizionali:
  1. disponi le due parole su due righe, allineandole a destra;
  2. somma i bit presenti sulla stessa colonna, tenendo conto del riporto;
  3. la striscia delle somme forma la parola rappresentante il risultato.
- Se l'ultimo riporto risulta 1, c'è l'overflow.

Esempio con codice a 6 bit:

Riporti:								
1° addendo:	0	1	1	0	0	1		
2° addendo:	0	0	0	1	0	1		
Risultato:								↻

4 Codice binario in modulo e segno

- [Modulus–sign binary code] E' usato per rappresentare i **numeri interi**.

4.1 Definizione

- Il MSB indica il segno:

  '0' sta per '+',

  '1' sta per '−'.

- Il modulo si determina applicando il metodo delle potenze pesate ai restanti bit.

4.1.1 Osservazioni

∴ Le versioni a lunghezza fissa hanno due rappresentazioni dello zero.

- Le parole 0000 e 1000 codificano entrambe lo zero nella versione a 4 bit.

∴ Utilizzando parole di n bit, è possibile rappresentare $2^n - 1$ numeri:

- i positivi arrivano a $2^{n-1} - 1$;
- i negativi arrivano a $-(2^{n-1} - 1)$.

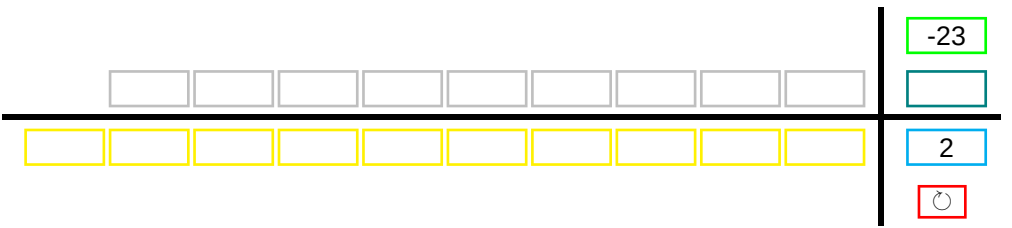
- I numeri pari terminano con 0 e quelli dispari con 1.

4.2 Codifica

- Per ottenere la rappresentazione a n bit di un intero x:

  1. determina la parola di lunghezza minima che rappresenta $|x|$ secondo il codice binario naturale;[4]
  2. premetti alla parola ottenuta abbastanza zeri da portarla alla lunghezza n;
  3. se $x < 0$, cambia a 1 il MSB.

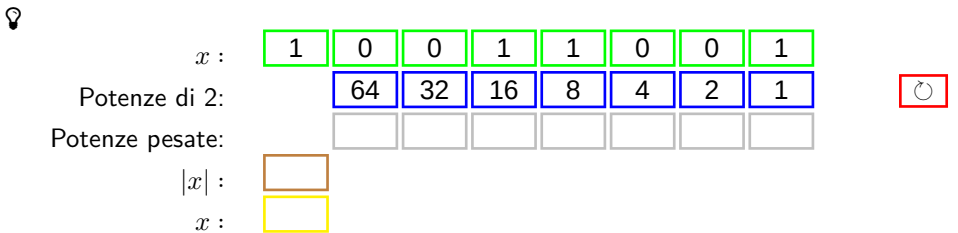
Esempio con codice a 10 bit:



[4] Se la parola ha lunghezza maggiore di $n-1$, non è possibile codificare x a n bit con questo codice.

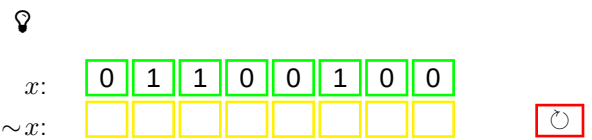
4.3 Decodifica

- Procedura:
  1. scarta il MSB e determina il valore assoluto;
  2. determina il segno valutando il MSB.



5 Codice in complemento a 1

- [One's complement code] E' usato per rappresentare i **numeri interi**.
- **Complemento a 1** di una parola binaria x $\triangleleft$ parola binaria che si ottiene cambiando tutti i bit di x.
  - Il complemento a 1 di x si indica con $\sim x$.



5.1 Definizione

- Il MSB indica il segno:

  '0' sta per '+',

  '1' sta per '−'.

- Se MSB = 0, il valore si determina con il metodo delle potenze pesate.
- L'opposto di x è rappresentato dal suo complemento a 1.

Esempi:

- 0101 rappresenta il numero 5,
- 1010 rappresenta il numero -5.

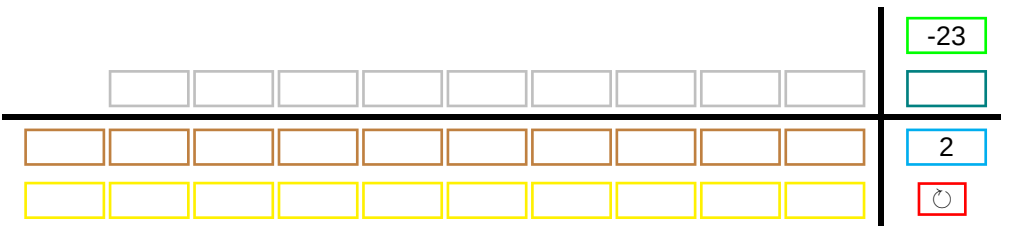
5.1.1 Osservazioni

- Le versioni a lunghezza fissa hanno due rappresentazioni dello zero.
  - le parole 0000 e 1111 codificano entrambe lo zero nella versione a 4 bit.
- Utilizzando parole di n bit, è possibile rappresentare $2^n - 1$ numeri:
  - i positivi arrivano a $2^{n-1} - 1$;
  - i negativi arrivano a $-(2^{n-1} - 1)$.
- Per distinguere pari e dispari bisogna considerare sia LSB sia MSB:
  - i pari positivi terminano con 0, i pari negativi con 1, etc.

5.2 Codifica

- Per ottenere la rappresentazione a n bit di un intero x:
  1. determina la parola di lunghezza minima che rappresenta $|x|$ secondo il codice binario naturale;[5]
  2. premetti alla parola abbastanza zeri da portarla alla lunghezza n;
  3. se $x < 0$, determina l'opposto della parola ottenuta nel passo precedente (applicando il complemento a 1).

Esempio con codice a 10 bit:

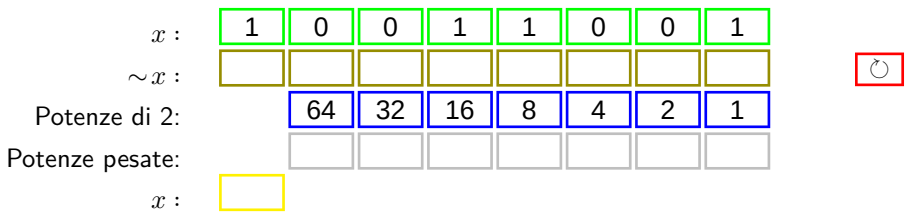


[5] Se la parola ha lunghezza maggiore di $n-1$, non è possibile codificare x a n bit con questo codice.

5.3 Decodifica

- Procedura:

  1. se MSB $= 0$, lascia la parola iniziale inalterata — altrimenti determina il suo opposto;
  2. applica il metodo delle potenze pesate alla parola ottenuta nel passo precedente;
  3. se MSB $= 0$, il numero ottenuto nel passo precedente è quello cercato — altrimenti il numero cercato è il suo opposto.

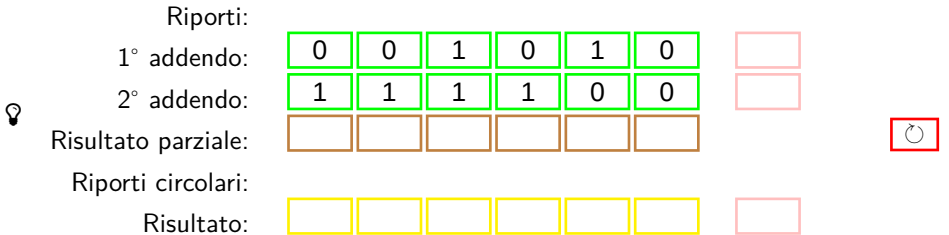


5.4 Somma

- Si può usare il procedimento chiamato **somma circolare**:

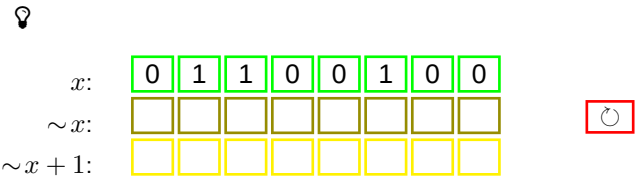
  l'eventuale ultimo riporto va sommato al risultato ottenuto con la somma posizionale.

- Se i due addendi hanno uguale MSB, un risultato con MSB discorde segnala l'overflow.



6 Codice in complemento a 2

- [Two's complement code] E' il codice binario più usato per trattare i **numeri interi**.
- **Complemento a 2** di una parola x a n bit $\triangleleft$ parola a n bit ottenuta sommando 1 a $\sim x$,[6] ignorando l'ultimo riporto.[7]



- Metodo alternativo per determinare il complemento a 2:
  - Cambia tutti i bit che precedono l'1 meno significativo.

01100100 $\rightarrow$ 10011100.

[6] Sommando 1 si ottiene il numero successivo più grande.
[7] L'ultimo riporto risulta 1 solo se x è una sequenza di zeri.

6.1 Definizione

- Il MSB indica il segno:

  '0' sta per '$+$',

  '1' sta per '$-$'.

- Se MSB $= 0$, il valore si determina con il metodo delle potenze pesate.
- L'opposto di x è rappresentato dal suo complemento a 2.

6.1.1 Osservazioni

- Utilizzando parole di n bit, è possibile rappresentare 2^n numeri:
  - i positivi arrivano a $2^{n-1} - 1$;
  - i negativi arrivano a -2^{n-1}.
- Lo zero non ha opposto: il complemento a 2 di una sequenza di zeri è la stessa sequenza.

☺ Il numero -2^{n-1} in un codice a n bit non ha opposto: applicando la procedura di determinazione dell'opposto si ottiene lo stesso -2^{n-1}.

☺ E' facile distinguere pari e dispari: i pari terminano con 0 e i dispari con 1.

- Nella seguente tabella è presentata la corrispondenza tra i numeri esprimibili con il codice in complemento a 2 a tre bit e le corrispondenti rappresentazioni:

3	011
2	010
1	001
0	000
-1	111
-2	110
-3	101
-4	100

6.2 Codifica

- Per ottenere la rappresentazione a n bit di un intero x:
  1. determina la parola di lunghezza minima che rappresenta $|x|$ secondo il codice binario naturale;[8]
  2. premetti alla parola ottenuta abbastanza zeri da portarla alla lunghezza n;
  3. se $x < 0$, determina l'opposto della parola ottenuta nel passo precedente (applicando il complemento a 2).

Esempio con codice a 10 bit:

[8] Se la parola ha lunghezza maggiore di $n-1$, non è possibile codificare x a n bit con questo codice.

6.3 Codifica alternativa

- Per ottenere la rappresentazione a n bit di un intero x: [9]

  1. determina $x' = \begin{cases} 2^n - |x| & \text{se } x < 0 \\ x & \text{altrimenti} \end{cases}$
  2. applica a x' il metodo delle divisioni successive;
  3. premetti alla parola ottenuta abbastanza zeri da portarla alla lunghezza n.[10]

Esempio con codice a 10 bit:

[9] Il fatto che i numeri negativi x siano codificati come $2^n - |x|$ (in codice binario naturale) ha dato il nome al codice: infatti, $B^n - p$ è detto complemento alla base B di modulo B^n del numero naturale p se n è un numero naturale tale che $B^n \geq p$.

[10] Nel caso di numeri negativi la parola è già alla lunghezza n al termine del secondo passo.

Dimostrazione
Per il caso $x \geq 0$, la validità della procedura è palese.

Per il caso $x < 0$, bisogna dimostrare che

$$2^n - |x| = \sum_{i=0}^{n-1} b_i 2^i$$

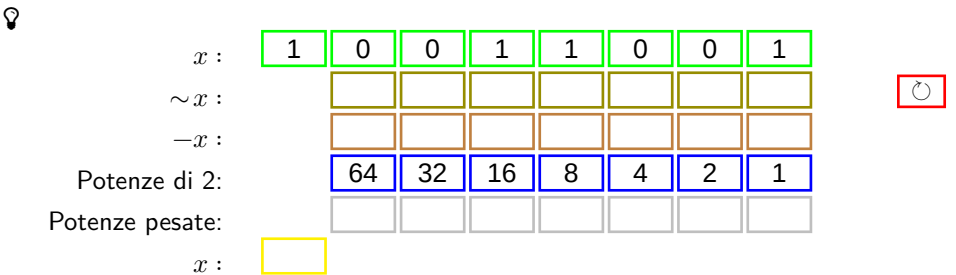
con $b_{n-1} \dots b_0$ rappresentazione di x:

$$\begin{aligned}
2^n - |x| &= 2^n - (-x) \\
&= 2^n - \left(\sum_{i=0}^{n-1} \bar{b}_i 2^i + 1 \right) \\
&= 2^n - \sum_{i=0}^{n-1} (1 - b_i) 2^i - 1 \\
&= 2^n - \sum_{i=0}^{n-1} 2^i + \sum_{i=0}^{n-1} b_i 2^i - 1 \\
\text{(esplicitando la somma parziale della serie geometrica)} &= 2^n - \frac{2^n - 1}{2 - 1} + \sum_{i=0}^{n-1} b_i 2^i - 1 \\
&= \sum_{i=0}^{n-1} b_i 2^i
\end{aligned}$$

6.4 Decodifica

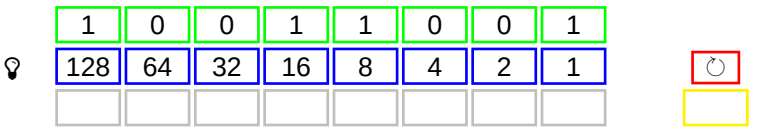
- Procedura:

  1. se MSB $= 0$, lascia la parola iniziale inalterata — altrimenti determina il suo opposto;
  2. applica il metodo delle potenze pesate alla parola ottenuta nel passo precedente;
  3. se MSB $= 0$, il numero ottenuto nel passo precedente è quello cercato — altrimenti il numero cercato è il suo opposto.



6.5 Decodifica alternativa

- Per ottenere il valore di una parola a n bit:
  1. se MSB $= 0$, basta applicare la procedura delle potenze pesate;
  2. se MSB $= 1$:
     (a) sottrai a 2^{n-1} le potenze di 2 corrispondenti ai bit di valore 1;
     (b) cambia segno al valore determinato nel passo precedente.



- La procedura è giustificata dalla formula:

$$\text{valore}_{TCC}(b_{n-1} \dots b_0) = -\left(b_{n-1}2^{n-1} - \sum_{i=0}^{n-2} b_i 2^i\right) \tag{3}$$

Dimostrazione
Se $b_{n-1} = 0$, allora la parola rappresenta un intero positivo o zero. Quindi, basta dimostrare che

$$-\left(b_{n-1}2^{n-1} - \sum_{i=0}^{n-2} b_i 2^i\right) = \sum_{i=0}^{n-1} b_i 2^i$$

Osserviamo che, essendo $b_{n-1} = 0$, la precedente equazione può essere riscritta senza considerare i due termini in cui figura b_{n-1}:

$$-\left(-\sum_{i=0}^{n-2} b_i 2^i\right) = \sum_{i=0}^{n-2} b_i 2^i$$

Quindi, semplificando a sinistra otteniamo, l'identità:

$$\sum_{i=0}^{n-2} b_i 2^i = \sum_{i=0}^{n-2} b_i 2^i$$

Se $b_{n-1} = 1$, allora la parola rappresenta un intero negativo. Quindi, basta dimostrare che

$$-\left(b_{n-1}2^{n-1} - \sum_{i=0}^{n-2} b_i 2^i\right)$$

△

è l'opposto del valore di $\bar{b}_{n-1} \ldots \bar{b}_0 + 1$, ovvero

$$-\left(b_{n-1}2^{n-1} - \sum_{i=0}^{n-2} b_i 2^i\right) = -\left(\sum_{i=0}^{n-1} \bar{b}_i 2^i + 1\right)$$

ovvero, semplificando il segno meno,

$$b_{n-1}2^{n-1} - \sum_{i=0}^{n-2} b_i 2^i = \sum_{i=0}^{n-1} \bar{b}_i 2^i + 1$$

Tenendo conto che $b_{n-1} = 1$ e, quindi $\bar{b}_{n-1} = 0$, possiamo riscrivere

$$2^{n-1} - \sum_{i=0}^{n-2} b_i 2^i = \sum_{i=0}^{n-2} \bar{b}_i 2^i + 1$$

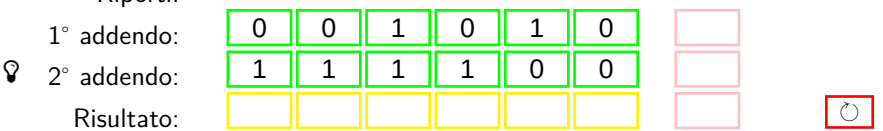
Tenendo conto che $\bar{b} = 1 - b$, possiamo scrivere

$$\begin{aligned}
\sum_{i=0}^{n-2} \bar{b}_i 2^i + 1 &= \sum_{i=0}^{n-2} (1 - b_i) 2^i + 1 \\
&= \sum_{i=0}^{n-2} 2^i - \sum_{i=0}^{n-2} b_i 2^i + 1 \\
&= 2^{n-1} - 1 - \sum_{i=0}^{n-2} b_i 2^i + 1 \\
&= 2^{n-1} - \sum_{i=0}^{n-2} b_i 2^i
\end{aligned}$$

in accordo a quanto volevasi dimostrare.

6.6 Somma

- La somma è effettuata seguendo il procedimento valido per i sistemi numerici posizionali, ma scartando l'eventuale ultimo riporto.
- Se i due addendi hanno uguale MSB, un risultato con MSB discorde segnala l'overflow.



7 Codice binario polarizzato

- [Biased binary code] E' un codice binario usato per rappresentare i **numeri interi**.

7.1 Definizione

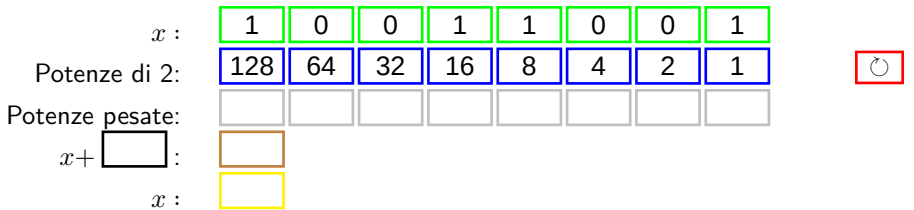
- Un numero x viene codificato a n bit seguendo questa procedura:
  1. somma a x il numero 2^{n-1} (metà di tutte le combinazioni);
     - il numero 2^{n-1} è detto **polarizzazione** [bias] o **eccesso**;[11]
  2. codifica il numero ottenuto (sicuramente non negativo) in codice binario naturale.

[11] Per questo motivo il codice è anche detto **notazione in eccesso di** 2^{n-1}.

7.1.1 Esempi

- Esempio di codifica a 8 bit:

- Esempio di decodifica:



7.1.2 Osservazioni

- I numeri rappresentabili con il codice di lunghezza n sono gli interi compresi nell'intervallo

$$[-2^{n-1}, 2^{n-1} - 1]$$

☺ L'ordinamento delle parole corrisponde all'ordinamento dei numeri:

3	111
2	110
1	101
0	100
-1	011
-2	010
-3	001
-4	000

☹ Il MSB dei numeri negativi è 0.

8 Codici BCD

- **Codici a cifre decimali codificate in binario** [Binary-coded decimal-digit codes] ◁ sono usati per rappresentare numeri naturali mediante la rappresentazione delle loro cifre decimali in accordo a un codice binario per numeri naturali.
  - Ogni cifra decimale è codificata con 4 bit, essendo 4 la quantità minima di bit per esprimere almeno 10 combinazioni.
- Questi codici sono usati per pilotare display compositi (come quelli per le file nei supermercati).

8.1 Codice BCD standard

- **Codice BCD standard** [standard BCD code] ◁ codice BCD basato sul codice binario posizionale.

Codifico in codice BCD standard il numero 947:

9	4	7	← Considero separatamente ciascuna cifra decimale
1001	0100	0111	← Sostituisco a ciascuna cifra la corrispondente parola di 4 bit
1001 0100 0111			← Unisco le parole di 4 bit

Esempio interattivo di codifica:

Numero in formato posizionale decimale: 72

Corrispondente formato BCD standard:

Esempio interattivo di decodifica:

Numero in formato BCD standard: 10010001

Corrispondente formato posizionale decimale:

8.2 Codice BCD Gray

- **Codice BCD Gray** [Gray BCD code] ◁ codice BCD basato sul codice Gray.

8.2.1 Codice Gray

- **Codice Gray** [Gray ...] ◁ codice binario per numeri naturali, basato su questa regola:
  - zero è rappresentato da una sequenza di zeri;
  - la rappresentazione del numero n si ottiene dalla rappresentazione del numero $n-1$ cambiando il bit più a destra che origina una nuova combinazione (rispetto alle combinazioni già scritte).

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

Tabella 1 - Tabella di codifica del codice Gray a 3 bit.

- E' usato per evitare i glitch in dispositivi basati sui contatori.[12]
  - Esempi di dispositivi basati sui contatori sono i sensori di posizione e di velocità angolare.

[12] **I glitch** sono errori temporanei (che possono causare errori stabili) causati dall'arrivo in tempi diversi di dati che dovrebbero arrivare simultaneamente.

8.2.2 Codifica e decodifica in codice BCD Gray

0	0000
1	0001
2	0011
3	0010
4	0110
5	0111
6	0101
7	0100
8	1100
9	1101

Tabella 2 - Tabella di codifica del codice BCD Gray.

Codifico in codice BCD Gray il numero 947:

9	4	7	← Considero separatamente ciascuna cifra decimale
1101	0110	0100	← Sostituisco a ciascuna cifra la corrispondente parola di 4 bit
1101 0110 0100			← Unisco le parole di 4 bit

Esempio interattivo di codifica:

Numero in formato posizionale decimale: 72

Corrispondente formato BCD Gray:

Esempio interattivo di decodifica:

Numero in formato BCD Gray: 11010001

Corrispondente formato posizionale decimale:

9 Codici esadecimali

- [Hexadecimal codes] Sono i codici a base 16.
- Utilizzano come cifre i simboli: **0**, **1**, **2**, **3**, **4**, **5**, **6**, **7**, **8**, **9**, **A**, **B**, **C**, **D**, **E**, **F**.

In un codice esadecimale posizionale la parola 5A rappresenta il numero

$$5 \times 16 + 10 \times 16^0 = 80 + 10 = 90$$

- Il codice esadecimale posizionale è usato dai tecnici per **scrivere in forma compatta parole binarie**, essendo molto semplici il passaggio dal codice binario posizionale al codice esadecimale posizionale e il passaggio inverso.

9.1 Conversione da binario a esadecimale

- Per trasformare una parola binaria in una parola esadecimale basta:

  1. premettere alla parola binaria un numero sufficiente di 0 tali da renderla costituita da un numero di bit multiplo di 4;
  2. considerare la parola binaria costituita da sequenze di 4 bit;
  3. sostituire, a ciascuna sequenza di 4 bit, la cifra esadecimale corrispondente al valore della sequenza secondo il codice binario naturale.

Trasformo la parola binaria 101010 in una parola esadecimale:

00101010 ← Aggiungo due 0 in modo da portarne la lunghezza a un multiplo di 4
0010 1010 ← Divido la parola in due sequenze di 4 bit
2 A ← Sostituisco a ciascuna sequenza la cifra esadecimale corrispondente

Esempio interattivo:

Parola binaria: 10010001

Corrispondente parola esadecimale:

9.2 Conversione da esadecimale a binario

- Per trasformare una parola esadecimale in una parola binaria, basta sostituire ciascuna cifra esadecimale con la corrispondente parola di 4 bit secondo il codice binario naturale.

Trasformo la parola esadecimale B47 in una parola binaria:

B	4	7	← Considero separatamente ciascuna cifra
1011	0100	0111	← Sostituisco a ciascuna cifra la corrispondente parola di 4 bit
1011	0100	0111	Unisco le parole di 4 bit

Esempio interattivo:

Parola esadecimale: 7B

Corrispondente parola binaria:

10 Codici a virgola fissa

- [Fixed-point codes] Sono codici per numeri reali che usano:
  - n cifre per la parte intera;
  - p cifre per la parte frazionaria.
- Un codice binario a virgola fissa può:
  - essere basato sulla notazione posizionale e usare il primo bit per il segno;
  - usare il codice binario in complemento-a-2 per la parte intera e il codice binario naturale per la parte frazionaria.

11 Codici a virgola mobile

- [Floating-point codes] Sono codici per numeri reali basati sulla **notazione scientifica**.
- **Notazione scientifica a base B** di un numero positivo $\lhd$ rappresentazione nella forma:

$$S \times B^E$$

  S è detto **significando** [significand] ed è un numero reale espresso in notazione posizionale a base B;

  E è detto **esponente** [exponent] ed è un numero intero espresso in notazione posizionale a base B.

- **Notazione scientifica normalizzata** di un numero $\lhd$ notazione scientifica in cui il significando ha parte intera composta da una sola cifra diversa da zero.

- Esempi:
  - 9.75×10^{-3} è espresso in notazione scientifica normalizzata a base 10: 9.75 è il significando e -3 è l'esponente
  - 97.5×10^{3} non è espresso in notazione scientifica normalizzata, in quanto la parte intera del significando è formata da due cifre (9 e 7);
  - 0.35×10^{3} non è espresso in notazione scientifica normalizzata, in quanto la parte intera del significando è nulla.

- Per passare un numero dalla notazione posizionale alla notazione scientifica normalizzata, basta calcolare l'esponente come:
  - $m - 1$ con m numero di cifre della parte intera, se il valore assoluto del numero è maggiore di 1;
  - $-z$ con z numero di zeri che precedono la prima cifra non-nulla, se il valore assoluto del numero è minore di 1.

  Esempi:

$$975.3 = 9.753 \times 10^{2}$$

$$0.0042 = 4.2 \times 10^{-3}$$

11.1 Standard IEEE 754

- Definisce i seguenti codici basati sulla notazione scientifica normalizzata:[13]
  - *formato a mezza precisione* (16 bit, precisione a 3 cifre decimali),
  - *formato a precisione singola* (32 bit, precisione a 7 cifre decimali),
  - *formato a precisione doppia* (64 bit, precisione a 15 cifre decimali),
  - *formato a precisione quadrupla* (128 bit, precisione a 34 cifre decimali),
  - *formato a precisione ottupla* (256 bit, precisione a 71 cifre decimali).

[13] Questo standard è stato proposto nel 1985 e aggiornato nel 2008 dalla IEEE (*Institute of Electrical and Electronic Engineers*) e, oltre ai codici citati, definisce altri codici per la memorizzazione dei numeri e regole per effettuare le operazioni aritmetiche e gli arrotondamenti.

11.1.1 Formato a precisione singola

- Le parole sono di 32 bit.
- Il bit più significativo codifica il segno: '0' sta per '$+$', '1' sta per '$-$'.
- I seguenti 8 bit codificano l'esponente tramite codice polarizzato con 127 (anziché 128).
  - I numeri normalizzati hanno esponente che parte da -126 (anziché -128).
- I 23 bit meno significativi sono riservati alla parte frazionaria del significando, in accordo alla notazione posizionale.

  La parte intera del significando è sottintesa, in quanto vale sempre 1, poiché è adottata la notazione scientifica normalizzata.

- La sequenza di soli '1' nell'**esponente** può corrispondere a:[14]
  - $+\infty$ o $-\infty$ (se tutti i bit del significando sono zero);
  - NaN (se c'è almeno un '1' tra i bit del significando).[15]

$$0\,\underbrace{11111111}_{\text{exponent}}\,\underbrace{000\ldots0}_{\text{significand}} \quad = +\infty$$

$$1\,\underbrace{11111111}_{\text{exponent}}\,\underbrace{000\ldots0}_{\text{significand}} \quad = -\infty$$

$$0\,\underbrace{11111111}_{\text{exponent}}\,\underbrace{010\ldots0}_{\text{significand}} \quad \mapsto \text{NaN}$$

[14] Questa sequenza non può corrispondere a nessun numero in quanto la polarizzazione è 127 anziché 128.

[15] *Not a number* ovvero *numero indeterminato*: può risultare da una divisione tra zeri, da una divisione tra infiniti, da un prodotto infinito per zero, etc.

- La sequenza di soli '0' nell'esponente è riservata ai cosiddetti numeri **denormalizzati**.
- **Numeri denormalizzati** $\triangleleft$ numeri in cui sono nulli i bit riservati alla parte intera del significando e all'esponente.
  - Questi numeri hanno una minore precisione relativa (avendo un numero di cifre significative inferiore a 24) ma consentono di ridurre i casi di underflow.
  - Tra questi numeri c'è lo **zero** corrispondente alla sequenza di solo zeri anche nella parte frazionaria del significando (c'è uno zero positivo e uno negativo).

Codifico il numero: -13.25

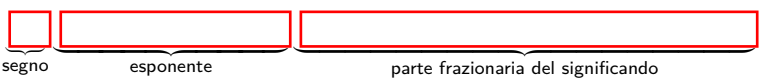
a) Valore assoluto in notazione posizionale binaria (fino a 24 bit):[16]

b) Valore assoluto in notazione scientifica a base 2 normalizzata:[17]

$1.\underbrace{\qquad}_{\text{significando}} \times \underbrace{\langle 2 \rangle}_{\text{base}} \underbrace{\langle \quad \rangle}_{\text{esponente}}$

c) Esponente espresso in codice binario polarizzato con 127:

d) Risultato:



[16] La parte intera si determina con il metodo delle divisioni successive e la parte frazionaria con il metodo delle moltiplicazioni successive.

[17] Per determinare l'esponente puoi far finta che il numero binario sia espresso in base 10.

11.1.2 Formato a precisione doppia

- Differenze rispetto al formato a precisione singola:
  - l'esponente è codificato da 11 bit (varia tra -1022 e 1023);
  - il significando è codificato da 52 bit.

12 Codici per caratteri

- **Carattere** [character] ◁ elemento base di un testo.
- I codici per caratteri sono spesso indicati con i termini inglesi **encoding** e **charset**.

12.1 American Standard Code for Information Interchange

- **Ascii** ◁ codice binario su cui sono basati i più importanti codici per caratteri.[18]
- E' stato proposto dalla Ansi nel 1963 e la sua ultima revisione è del 1986.[19]
- E' un codice a 8 bit:
  - il bit più significativo funge da bit di parità (vedi §14.4);
  - i restanti bit servono a rappresentare i caratteri: ciascuna parola di 7 bit codifica in codice binario naturale il numero corrispondente al carattere.

[18] La Iana ha aggiornato il nome a Us-Ascii.
[19] L'American National Standard Institute è un'organizzazione non-profit degli USA.

- E' stato pensato per rappresentare testi scritti in lingua inglese-americana:
  - ☺ contiene le lettere dell'alfabeto inglese in maiuscolo e in minuscolo;
  - ☺ contiene le cifre decimali;
  - ☺ contiene i simboli della punteggiatura inglese;
  - ☹ contiene solo alcuni operatori matematici;
  - ☹ contiene il simbolo del dollaro, ma non quelli di altre valute (neppure la sterlina inglese).
- E' obsoleto, in quanto l'insieme dei caratteri rappresentabili è troppo piccolo.[20]

[20] E' ancora supportato da sistemi di scrittura che prevedono la rielaborazione dei caratteri immessi dall'utente (ad esempio, TEX e derivati).

- I primi 32 caratteri sono caratteri di controllo.[21]

Code point	Hexadecimal CP	Name	Abbreviation	C escape-sequence
0	0	Null	NUL	\0
1	1	Start of heading	SOH	
		⋮		
7	7	Bell	BEL	\a
8	8	Backspace	BS	\b
9	9	Horizontal tab	HT	\t
10	A	Line feed	LF	\n
11	B	Vertical tab	VT	\v
12	C	Form feed	FF	\f
13	D	Carriage return	CR	\r
		⋮		
27	1B	Escape	ESC	\e
		⋮		
31	1F	Unit separator	US	

Tabella 3 - Alcuni dei primi 32 caratteri Ascii

[21] In Ascii l'indice esadecimale differisce dalla rappresentazione esadecimale, in quanto non tiene conto del bit di parità.

- I successivi caratteri sono riportati nella tabella 4.

	0	1	2	3	4	5	6	7	8	9	A	B	C	D	E	F
0x2	SP 32	! 33	" 34	# 35	$ 36	% 37	& 38	' 39	(40	) 41	* 42	+ 43	, 44	- 45	. 46	/ 47
0x3	0 48	1 49	2 50	3 51	4 52	5 53	6 54	7 55	8 56	9 57	: 58	; 59	< 60	= 61	> 62	? 63
0x4	@ 64	A 65	B 66	C 67	D 68	E 69	F 70	G 71	H 72	I 73	J 74	K 75	L 76	M 77	N 78	O 79
0x5	P 80	Q 81	R 82	S 83	T 84	U 85	V 86	W 87	X 88	Y 89	Z 90	[91	\ 92	] 93	^ 94	_ 95
0x6	` 96	a 97	b 98	c 99	d 100	e 101	f 102	g 103	h 104	i 105	j 106	k 107	l 108	m 109	n 110	o 111
0x7	p 112	q 113	r 114	s 115	t 116	u 117	v 118	w 119	x 120	y 121	z 122	{ 123	| 124	} 125	~ 126	DEL 127

Tabella 4 - Caratteri Ascii seguenti ai primi 32. SP è il carattere spazio. DEL è il carattere di controllo Delete.

12.2 Codici Ascii regionali

- Sono codici a 8 bit rappresentanti 256 caratteri:
  - i primi 128 caratteri coincidono con quelli dell'Ascii;
  - i restanti caratteri dipendono da:
    - la regione geografica a cui il codice è dedicato,
    - la funzione che il codice deve svolgere.
- Le più usate famiglie di codici regionali sono:
  - Iso/IEC 8859-x ← creata nel 1987 e aggiornata fino al 2003;[22]
  - Windows 125x ← creata per i sistemi operativi Windows.[23]

[22] La sigla Iana è iso-8859-x
[23] La sigla Iana è windows-125x

12.2.1 Codici Ascii per l'Europa occidentale

- **Iso/IEC 8859-1** (anche detto **Latin-1**) ◁ estende il codice Ascii con le vocali accentate e altri simboli usati in Italia, Francia, Germania, Regno Unito, Spagna.
  - E' stato approvato come standard Iso nel 1987 e revisionato nel 1998.
  - Non usa i code-point compresi tra 0x80 e 0x9F.
- **Windows 1252** (anche detto **Windows occidentale**) ◁ estende l'Iso/IEC 8859-1 aggiungendo caratteri con code-point compresi tra 0x80 e 0x9F.
  - ☺ Ha il simbolo dell'euro.
  - E' stato il codice per caratteri più usato negli anni '90 e nel 1° decennio del 2000.
  - I browser usano questo codice al posto del Latin-1, essendone una estensione.

	0	1	2	3	4	5	6	7	8	9	A	B	C	D	E	F
0X8	€		‚	ƒ	„	…	†	‡	ˆ	‰	Š	‹	Œ		Ž	
0X9		‘	’	“	”	•	–	—	˜	™	š	›	œ		ž	Ÿ
0XA		¡	¢	£	¤	¥	¦	§	¨	©	ª	«	¬		®	¯
0XB	°	±	²	³	´	µ	¶	·	¸	¹	º	»	¼	½	¾	¿
0XC	À	Á	Â	Ã	Ä	Å	Æ	Ç	È	É	Ê	Ë	Ì	Í	Î	Ï
0XD	Ð	Ñ	Ò	Ó	Ô	Õ	Ö	×	Ø	Ù	Ú	Û	Ü	Ý	Þ	ß
0XE	à	á	â	ã	ä	å	æ	ç	è	é	ê	ë	ì	í	î	ï
0XF	ð	ñ	ò	ó	ô	õ	ö	÷	ø	ù	ú	û	ü	ý	þ	ÿ

Figura 1 - Seconda parte della code-page di Windows 1252. Le due righe gialle sono vuote nel Latin-1. In 0xA0 c'è lo *spazio unificatore* [non-breaking space]. In 0xAD c'è il *soft hyphen* (un simbolo invisibile che indica dove poter spezzare una parola per andare a capo).

- **Iso/IEC 8859-15** (anche detto **Latin-9**) ◁ è stato approvato nel 1999 e differisce dal Latin-1 per 8 simboli, tra cui quello dell'euro (che ha sostituito il simbolo di valuta generica ¤).

	0	1	2	3	4	5	6	7	8	9	A	B	C	D	E	F
0X8																
0X9																
0XA		¡	¢	£	€	¥	Š	§	š	©	ª	«	¬		®	¯
0XB	°	±	²	³	Ž	µ	¶	·	ž	¹	º	»	Œ	œ	Ÿ	¿
0XC	À	Á	Â	Ã	Ä	Å	Æ	Ç	È	É	Ê	Ë	Ì	Í	Î	Ï
0XD	Ð	Ñ	Ò	Ó	Ô	Õ	Ö	×	Ø	Ù	Ú	Û	Ü	Ý	Þ	ß
0XE	à	á	â	ã	ä	å	æ	ç	è	é	ê	ë	ì	í	î	ï
0XF	ð	ñ	ò	ó	ô	õ	ö	÷	ø	ù	ú	û	ü	ý	þ	ÿ

Figura 2 - 2ª parte della code-page di Iso/IEC 8859-15. Le celle gialle sono vuote.

12.3 Unicode

- Il suo intento è la rappresentazione di tutti i caratteri dell'Universo.
- Attualmente è il codice per caratteri più usato.
- E' stato pubblicato nel 1991 dallo Unicode Consortium.[24]
- E' uno standard Iso/IEC dal 1993.
- Viene periodicamente aggiornato con l'aggiunta di nuovi caratteri.
- L'abbinamento (carattere, parola binaria) è effettuato in due passi:
  - ciascun carattere è abbinato a un numero naturale, detto **code-point**;
    - questi code-point sono di norma espressi in esadecimale con il prefisso U+;

    il code-point di "A" è indicato con "U+41" (essendo 65 in decimale);
  - ciascun code-point è abbinato a una parola binaria mediante un codice detto **formato di trasformazione** [transformation format].[25]

[24] Unicode Consortium è un'organizzazione non-profit a cui sono associate anche le principali software-house (Adobe, Apple, Facebook, Google, IBM, Microsoft, Oracle)

[25] Negli Ascii regionali il formato di trasformazione è il codice binario naturale.

- E' una estensione dell'Iso/IEC 8859-1.
  - Tuttavia solo per i primi 128 caratteri, possono coincidere anche le parole binarie (dipende dal formato di trasformazione).
  - Nelle righe vuote dell'Iso/IEC 8859-1 sono assegnati diversi caratteri di controllo.
- I caratteri sono raggruppati in blocchi, in base al campo d'uso e alla frequenza d'uso.
  - Il blocco "Cyrillic" contiene i simboli base dell'alfabeto cirillico, mentre i blocchi "Cyrillic Supplement", "Cyrillic Extended-A", "Cyrillic Extended-B" contengono i simboli dell'alfabeto cirillico usati meno comunemente.
  - Il blocco "Arrows" contiene i simboli di freccia più usati, mentre i blocchi "Supplemental Arrows-A" e "Supplemental Arrows-B" contengono simboli di freccia meno comuni.
  - L'applicazione **Gnome Character Map** permette di visualizzare i caratteri suddivisi nei blocchi di Unicode.
- Attualmente Unicode supporta quasi 2^{21} caratteri (quelli assegnati sono molto meno).

12.3.1 UTF-8

- E' il formato di trasformazione di Unicode nettamente più usato.
- Abbinamento tra code-point e parola binaria:
  - Se per il code-point bastano 7 bit, allora la parola binaria corrisponde a quella ottenuta con il codice binario naturale a 8 bit;
  - Altrimenti, occorre usare più byte:
    - il 1° byte inizia con tanti 1 quanti sono i byte da usare, seguiti da uno 0;
    - i restanti byte iniziano con 10;
    - nelle posizioni rimanenti sono copiati i bit del code-point.

Bit sufficienti	Code-point	Parola UTF-8
7	$b_6...b_0$	$0\ b_6...b_0$
11	$b_{10}...b_0$	$110\ b_{10}...b_6 \quad 10\ b_5...b_0$
16	$b_{15}...b_0$	$1110\ b_{15}..b_{12} \quad 10\ b_{11}...b_6 \quad 10\ b_5...b_0$
21	$b_{20}...b_0$	$11110\ b_{20}...b_{18}\ 10\ b_{17}...b_{12}\ 10\ b_{11}...b_6\ 10\ b_5...b_0$

Tabella 5 - Regola dell'UTF-8


Carattere:
W
Code point:
U+
= 0b
Parola UTF-8:

12.3.2 UTF-16

- E' un codice a lunghezza multipla di 16 bit:
  - per code-point fino a 16 bit è equivalente al codice binario naturale;
  - un code-point superiore a 16 bit è diviso in (due) numeri, detti **surrogati** [surrogate], appartenenti a intervalli appositi in cui Unicode non assegna caratteri, che vengono tradotti in codice binario naturale a 16 bit.

```
Per ottenere i due surrogati, si procede così:
 x = code_point - 2^16 // ottengo un numero esprimibile con 20 bit
 surrogato1 = parte_intera(x / 2^10) + 0xD800 // sommo 0xD800 ai 10 bit più significativi di x
 surrogato2 = resto(x / 2^10) + 0xDC00 // sommo 0xDC00 ai 10 bit meno significativi di x

Viceversa, per ottenere il code-point, si procede così:
 code_point = (surrogato1 - 0xD800)*2^10 + (surrogato2 - 0xDC00) + 2^16
```

- E' usato internamente da Windows, Java, Javascript.

☹ E' sconsigliato per le pagine HTML, i documenti di testo semplice, gli script, i codici sorgente, in quanto i file così codificati occupano una memoria quasi doppia rispetto ai file codificati in UTF-8.

12.3.3 Altri formati

- **UCS-2** è il codice binario naturale a 16 bit: può essere usato, dunque, solo per i caratteri con code-point inferiori a 2^{16}.
- **UTF-32** è il codice binario naturale a 32 bit.
- **GB18030** è a lunghezza multipla del byte ed è stato registrato dalla Repubblica Popolare Cinese.

13 Codici per colori

14 Codici ridondanti

14.1 Distanza di Hamming

- **Distanza di Hamming tra due parole binarie** $\lhd$ numero di bit per cui le due parole differiscono.

  Le due parole x e y

  $$\begin{array}{lccccc} x: & 0 & 1 & 1 & 0 & 1 \\ y: & 1 & 1 & 0 & 1 & 1 \\ & \uparrow & & \uparrow & \uparrow & \end{array}$$

  hanno distanza di Hamming uguale a 3, in quanto differiscono in tre posizioni.

- La distanza di Hamming tra due parole si può determinare sommando i bit del risultato dello Xor tra le due parole.

  $$\begin{array}{lccccc} x: & 0 & 1 & 1 & 0 & 1 \\ y: & 1 & 1 & 0 & 1 & 1 \\ x \oplus y: & 1 & 0 & 1 & 1 & 0 \overset{+}{\rightarrow} 3 \end{array}$$

  $\triangle$

- **Distanza di Hamming di un codice** ◁ la più piccola distanza di Hamming tra tutte le parole del codice.
  - I codici che usano tutte le combinazioni di bit hanno distanza di Hamming uguale a 1.

  Il codice Ascii ha distanza di Hamming uguale a 2.

  I codici Ascii regionali (Iso e Windows) hanno 1 come distanza di Hamming.
- **Codice minimale** ◁ codice con distanza di Hamming uguale a 1.
- **Codice ridondante** ◁ codice con distanza di Hamming maggiore di 1.

14.2 Rilevazione degli errori

- **Codice rilevatore di errori** [error-detecting code] $\lhd$ codice che permette di rilevare parole sbagliate.

- In un codice con distanza di Hamming d è possibile rilevare le parole sbagliate che presentano fino a $w = d - 1$ errori.

  Infatti, se in una parola del codice vengono cambiati meno di d bit, questa parola è trasformata in una parola che non appartiene al codice.

$\Leftrightarrow$ Un codice che permette di rilevare le parole sbagliate che presentano fino a w errori ha distanza di Hamming

$$d = w + 1$$

14.3 Correzione degli errori

- **Codice correttore** [correcting code] $\lhd$ permette di correggere parole sbagliate.
- In un codice con distanza di Hamming d è possibile correggere le parole sbagliate che presentano fino a $w = (d-1)/2$ errori.

  Infatti, se in una parola del codice vengono cambiati un numero di bit inferiore alla metà della distanza di Hamming, la parola è lasciata più simile alla parola originaria rispetto a tutte le altre del codice.

$\Leftrightarrow$ Un codice che permette di correggere le parole sbagliate che presentano fino a w errori ha distanza di Hamming

$$d = 2w + 1$$

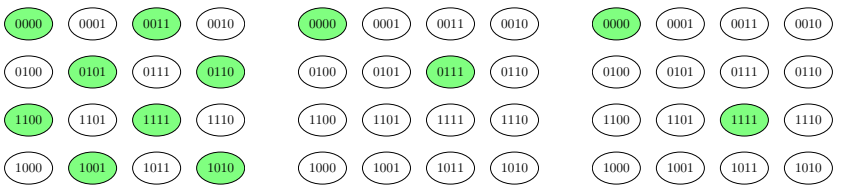


Figura 3 - Rappresentazione di tre codici aventi rispettive distanze di Hamming 2, 3, 4. I pallini verdi rappresentano le parole lecite.

14.4 Codici con 1 bit di parità

- Sono codici costruiti a partire da codici minimali in cui alla parola originaria è aggiunto (solitamente come bit più significativo) un bit il cui valore è scelto in modo tale che la somma dei bit della parola risulti pari.
  - Il bit aggiunto è chiamato **bit di parità** [parity bit].
  - Per vedere se la somma dei bit di una parola è pari, basta vedere se la somma logica esclusiva dei suoi bit è zero:

$$b_{n-1} \oplus \dots b_0 = 0$$

- Il bit di parità è determinato effettuando la somma logica esclusiva di tutti gli altri bit:

$$b_{n-1} = b_{n-2} \oplus \dots b_0$$

- I codici con 1 bit di parità hanno distanza di Hamming uguale a 2.
- I codici con 1 bit di parità permettono di rilevare la presenza di un numero dispari di errori in una parola.

Codice C di un programma che controlla se la parola passata dall'utente è sbagliata:

```c
#include <stdio.h>

int main () {
   const int N = 4;
   char w[N+1];
   printf("Enter the %d-bit word: ", N);
   scanf("%s", w);
   int checksum = 0;
   for (int i = 0; i < N; i++) {
      checksum += w[i];
   }
   if (checksum % 2) {
      printf("The word is wrong!\n");
   } else {
      printf("The word is right!\n");
   }
}
```

14.5 Codice di Hamming

- E' un codice con distanza di Hamming uguale a 3.[26]
- Dunque:
  - permette di rilevare parole in cui sono presenti 1 o 2 bit sbagliati;
  - permette di correggere parole che presentano 1 bit sbagliato.

[26] Questo codice è stato pubblicato dal matematico Richard Wesley Hamming (Chicago, 1915 – Monterey (California), 1998) nel 1950 a conclusione di uno studio per automatizzare la correzione degli errori prodotti dal computer Bell Model V realizzato dall'azienda Bell Labs.

14.5.1 Teorema di Hamming

- Per un codice di Hamming, avente m bit di informazione per parola, il numero dei bit di parità p va stabilito in accordo alla seguente relazione:

$$2^p - p - 1 \geq m \tag{4}$$

p	$\max(m)$
2	1
3	4
4	11
5	26
6	57
7	120
8	247
9	502
10	1013
11	

Dimostrazione
Chiamiamo n la lunghezza delle parole del codice.

Visto che il codice usa m bit di informazione per parola, le parole lecite sono 2^m.

Visto che il codice permette la correzione di 1 bit errato per parola, ogni parola richiede (n+1) configurazioni dedicate: una lecita e n illecite ottenute da quella lecita modificando un solo bit.

Quindi, il codice richiede che siano disponibili $(n+1)2^m$ configurazioni.

D'altra parte, visto che ciascuna parola è formata da n bit, le configurazioni disponibili sono 2^n.

Pertanto, deve valere la relazione

$$(n+1)\ 2^m \leq 2^n$$

ovvero, ricordando che $n = m + p$,

$$(m+p+1)\ 2^m \leq 2^m\ 2^p$$

da cui, semplificando e isolando m da un lato, segue la relazione della tesi.

14.5.2 Costruzione di un codice di Hamming

1. Le posizioni all'interno della parola sono numerate a partire da 1.
   - In questo testo uso l'ordine crescente verso destra.
2. I bit di parità sono inseriti tra i bit di informazione in modo da occupare le posizioni corrispondenti alle potenze di 2.
   - Nella parola $h_1h_2h_3h_4h_5h_6$ i bit di parità sono h_1, h_2, h_4.
3. Il bit di informazione avente posizione i è controllato dai bit di parità aventi posizioni $i_1, \dots i_k$ tali che
   $$i_1 + \dots i_k = i$$
   - Il bit h_3 è controllato dai bit h_1 e h_2 (essendo $3 = 2 + 1$).
   - Il bit h_{13} è controllato dai bit h_1, h_4, h_8 (essendo $13 = 8 + 4 + 1$).[27]
4. Il valore di un bit di parità è scelto uguale alla somma esclusiva dei bit che controlla.
   - Se h_1 controlla h_3 e h_5, allora il suo valore è scelto così:
   $$h_1 = h_3 \oplus h_5$$

[27] Nella ricerca della scomposizione di un numero in potenze di 2 conviene piazzare prima le potenze più grandi.

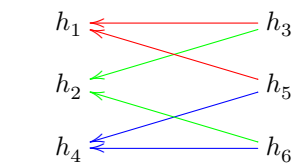
14.5.3 Esempio

Costruiamo il dizionario di un codice di Hamming con tre bit di informazione ($m = 3$).

Per il teorema di Hamming occorrono almeno tre bit di parità. Quindi scegliamo $p = 3$.

I bit di parità sono h_1, h_2, h_4, mentre i bit di informazione sono h_3, h_5, h_6.

In figura mostriamo graficamente come sono controllati i bit di informazione.



Dalla precedente figura ricaviamo le formule per i valori dei bit di parità:

$$h_1 = h_3 \oplus h_5$$
$$h_2 = h_3 \oplus h_6$$
$$h_4 = h_5 \oplus h_6$$

Le parole del codice sono riportate in tabella.

h_1	h_2	h_3	h_4	h_5	h_6
0	0	0	0	0	0
0	1	0	1	0	1
1	0	0	1	1	0
1	1	0	0	1	1
1	1	1	0	0	0
1	0	1	1	0	1
0	1	1	1	1	0
0	0	1	0	1	1

14.5.4 Individuazione del bit sbagliato

- Se è presente un solo bit sbagliato nella parola, il suo indice è dato dalla somma degli indici dei bit di parità che sembrano sbagliati.

Per la seguente parola

1	0	0	1	0	0
h_1	h_2	h_3	h_4	h_5	h_6

risulta:

- $h_1 \oplus h_3 \oplus h_5 = 1 \oplus 0 \oplus 0 = 1$ $\leftarrow$ h_1 sembra sbagliato
- $h_2 \oplus h_3 \oplus h_6 = 0 \oplus 0 \oplus 0 = 0$ $\leftarrow$ OK
- $h_4 \oplus h_5 \oplus h_6 = 1 \oplus 0 \oplus 0 = 1$ $\leftarrow$ h_4 sembra sbagliato

Pertanto, se si è verificato un solo errore, il bit sbagliato è h_5 $(1 + 4 = 5)$.

Codice C di un programma che corregge la parola passata dall'utente per il codice di Hamming a 6 bit:

```c
#include <stdio.h>

int main () {
   const int N = 6;
   char w[N+1];
   printf("Enter the %d-bit word: ", N);
   scanf("%s", w);
   int k = (w[0] + w[2] + w[4]) % 2;
   k += ((w[1] + w[2] + w[5]) % 2)*2;
   k += ((w[3] + w[4] + w[5]) % 2)*4;
   if (k > N) {
      printf("The word has two wrong bits!\n");
   } else if (k > 0) {
      w[k-1] = w[k-1] == '0' ? '1' : '0';
      printf("The fixed word is %s\n", w);
   } else {
      printf("The word is right!\n");
   }
}
```