2 - Conversione binaria di numeri negativi, numeri in virgola fissa e mobile, cenni di porte logiche

NUMERI NEGATIVI

Un **numero negativo** non è rappresentabile nel sistema binario puro. Dunque, sono stati adottati diversi metodi di rappresentazione convenzionali per ovviare a tale limite.

- **Rappresentazione modulo e segno**:

  In tale rappresentazione si considera uno spazio di rappresentazione di 1 byte (8 bit): i 7 bit a destra rappresentano il valore assoluto del numero, mentre il **MSB** (quello più a sinistra) vale 1 se il numero è negativo, 0 se è positivo.

  Esempio: $4_{10} = 0000\ 0100_2,\ \ -4_{10} = 1000\ 0100_2.$

  I problemi di un simile modo di rappresentazione risultano principalmente essere lo **“spreco” del MSB** e la conseguente perdita di capacità di rappresentazione, oltre che **l’ambiguità data da due possibili configurazioni dello zero**: 0000 0000 e 1000 0000 rappresentano 0 (= -0).

- **Complemento a 1**:

  L’operazione CA1 consiste nel convertire normalmente il modulo di un numero negativo, per poi effettuarne il **complemento** (ovvero la sostituzione di ogni bit con il bit opposto).

  Esempio: $4_{10} = 0000\ 0100_2,\ \ -4_{10} = 1111\ 1011_2$

  Anche utilizzando questa configurazione si incorre nell’**ambiguità dello zero**: 0000 0000 e 1111 1111 rappresentano 0 (= -0).

- **Complemento a 2**:

  L’operazione CA2 consiste nel convertire il numero negativo con CA1, per poi **sommare 1** al risultato finale. Inoltre, il MSB ha valore 1 per i numeri negativi e 0

per i positivi.

Non persiste il problema legato all'ambiguità dello zero.

Esistono 3 metodi per calcolare il CA2 di un numero X:

- $Ca2(x) = 2^n - x$, dove x è il valore assoluto del numero in base 10 e n è il numero di bit disponibili.

  Esempio: $-4_{10} = 2^8 - x = 252_{10} \rightarrow 1111\ 1100_2$

- Si calcola **CA1** e si aggiunge **1**.
- A partire da destra, si trascrivono tutti gli **zeri** del valore assoluto. il primo **1** incontrato viene trascritto, dopodichè si **complementano** tutti i bit restanti.

  Esempio: $4_{10} = 0000\ 0100_2 \rightarrow -4_{ca_2} = 11111100_2.$

OPERAZIONI ARITMETICHE

- **Modulo e segno**:
  - Somma: il bit di segno è quello della configurazione di valore assoluto maggiore. La somma si effettua bit-a-bit.
  - Sottrazione: se i bit di segno sono diversi, il segno sarà quello del minuendo, mentre, se i bit di segno sono uguali, il segno sarà quello della configurazione di valore assoluto maggiore. La sottrazione si effettua bit-a-bit (tenendo conto che A-B = A+(-B)).
- **Complemento a 2**:
  - Somma algebrica: si sommano bit-a-bit tutti gli addendi (segno compreso), ma il carry oltre l'MSB viene scartato. L'overflow si ha se gli operandi hanno egual segno e se il segno del risultato non è concorde con essi.

SHIFT RIGHT E SHIFT LEFT

L'operazione **shift left** equivale a moltiplicare il numero per la base. Il MSB viene scaricato: se il bit è 1, si ha overflow.

L'operazione **shift right** equivale a dividere il numero per la base. Il MSB mantiene il segno della base, si aggiunge uno zero subito dopo e si scarica il LSB.

ECCESSO 2^{n-1} (128)

Nell'**eccesso** 2^{n-1} il numero x si rappresenta come $x + 2^{n-1}$. Tali numeri si ottengono complementando l'MSB del CA2.

Esempio: $x = 4 \rightarrow 4 + 128 = 132 \rightarrow 1000\ 0100_2$.

NUMERI IN VIRGOLA FISSA

Dato un numero non intero, esso è rappresentato in **virgola fissa** utilizzando 1 bit per il segno, N bit per la parte intera e D bit per la parte decimale. La posizione della virgola è **implicita e uguale** $\forall$ numero.

Per convertire da base 10 a virgola fissa, si converte prima la parte intera, poi la parte decimale viene moltiplicata per 2 in successione, tante volte quanti i bit disponibili: la parte decimale binaria è la parte intera del risultato di operazioni (nell'ordine in cui esse vengono eseguite).

Esempio: $4,125_{10} \rightarrow 4 = 0100;\ 0,125 * 2 = 0,25\ (i0) * 2 = 0,5\ (i0) * 2 = 1\ (i1) \rightarrow 0100.001_2$

Per convertire da virgola fissa in base 10, si utilizza il metodo sopra citato per la conversione di interi, ma si moltiplicano le cifre dopo la virgola per le potenze negative di 2.

I problemi della rappresentazione in virgola fissa sono la **rigidità** della posizione della virgola e l'impatto che essa ha sulla **precisione** della conversione.

NUMERI IN VIRGOLA MOBILE

La rappresentazione in **virgola mobile** utilizza 1 bit di segno, M bit per la mantissa e E bit per l'esponente. La posizione della virgola è **variabile** in quanto la rappresentazione avviene, di fatto, in notazione scientifica (si moltiplica per 2^n), con un'unica cifra a sinistra della virgola. Il numero a sinistra della virgola è sempre 1, mentre la mantissa contiene tutte le cifre successive.

⚠️ ATTENZIONE! Per rappresentare un esponente è necessario sommare 127 ad esso (eccesso 128).

Un numero X è scritto nella forma $X = (-1)^{segno} * M * 2^{esponente}$.

Esempio: $1001.0001 \rightarrow 1.0010001 * 2^3 \rightarrow m : 1.0010001 \;\; e : 127 + 3 = 130 = 1000\,0010.$

Lo standard **IEEE754** utilizza 8 bit per l'esponente e 23 bit per la mantissa (precisione semplice), oppure 11 bit per l'esponente e 52 per la mantissa.

ALTRE INFORMAZIONI

I caratteri possono essere rappresentati mediante:

- **ASCII standard**: 7 bit per carattere.
- **ASCII estesa**: 8 bit per carattere.
- **Unicode**: da 8 a 32 bit per carattere (fino a 4 miliardi di caratteri diversi). Ogni carattere è rappresentato da un code point (8 cifre esadecimali) mediante lo standard **UTF-8**.