**Operazioni aritmetiche tra numeri binari**

Il calcolatore ha al suo interno dei dispositivi adatti ad effettuare le operazioni elementari sui numeri binari: moltiplicazione, somma, divisione e sottrazione. Dato che il sistema di numerazione binario è posizionale, è evidente che le leggi che regolano tali operazioni sono identiche a quelle dell'aritmetica dei numeri decimali, tenendo conto della base del sistema di rappresentazione e del numero di bit che si vuole utilizzare. Supponiamo d'avere due numeri binari $A = A_{n-1}\ldots A_{i+1}\, A_i\, A_{i-1}\ldots A_1A_0$ e $B = B_{n-1}\ldots B_{i+1}\, B_i\, B_{i-1}\ldots B_1B_0$ che utilizzano $n$ bit (se uno dei due ha $m < n$ bit basta aggiungere $n - m$ zeri davanti).

*Addizione*

Consideriamo la generica cifra di posto $i$: $A_i$ e $B_i$. Le possibili combinazioni che possono assumere le cifre suddette sono:

| $A_i$ | $B_i$ | Somma | Riporto | |
|---|---|---|---|---|
| 0 | 0 | 0 | 0 | 0 + 0 = 0, riporto 0 |
| 0 | 1 | 1 | 0 | 0 + 1 = 1, riporto 0 |
| 1 | 0 | 1 | 0 | 1 + 0 = 1, riporto 0 |
| 1 | 1 | 0 | 1 | 1 + 1 = 0, riporto 1 |

Si noti che $1_{(10)} + 1_{(10)} = 2_{(10)}$, ma 2 è proprio la base di rappresentazione binaria e quindi otteniamo $10_{(2)}$ ossia 0 col riporto di 1. Facciamo lo stesso in decimale: 5 + 5 vale 0 con il riporto di 1.

Esempi:

| | Binario | Decimale |
|---|---|---|
| Riporti | 010 | 0 |
| $A$ | 1010 + | 10 + |
| $B$ | 0011 = | 3 = |
| Somma | 1101 | 13 |

| | Binario | Decimale |
|---|---|---|
| Riporti | 111 | 1 |
| $A$ | 0111 + | 7 + |
| $B$ | 0011 = | 3 = |
| Somma | 1010 | 10 |

Nel secondo esempio, relativamente al bit di posto 1, si è sommato 1 + 1 + 1 ottenendo 1 e riportando 1, ossia $11_{(2)}$ che equivale per l'appunto a $3_{(10)}$. Si noti che $A$, $B$ e la loro somma utilizzano 4 bit. La somma potrebbe anche superare i 4 bit, ad esempio 1110 + 0100 = 10010 ($14_{(10)} + 4_{(10)} = 18_{(10)}$). In tal caso se nel calcolatore avessimo stabilito di utilizzare 4 bit il risultato visualizzato sarebbe stato 0010 ossia $2_{(10)}$, giacché il bit più significativo verrebbe ignorato. Quindi viene generato un errore facilmente individuabile (una volta stabilito il numero di bit) noto con il nome di *tracimazione* (*overflow*).

*Sottrazione*

Consideriamo la generica cifra di posto $i$: $A_i$ e $B_i$. Le possibili combinazioni che possono assumere le cifre suddette sono:

| $A_i$ | $B_i$ | Sottrazione | Prestito | |
|---|---|---|---|---|
| 0 | 0 | 0 | 0 | 0 – 0 = 0, prestito 0 |
| 0 | 1 | 1 | 1 | 0 – 1 = 1, prestito 1 |
| 1 | 0 | 1 | 0 | 1 – 0 = 1, prestito 0 |
| 1 | 1 | 0 | 0 | 1 – 1 = 0, prestito 0 |

Poiché sappiamo scrivere solo numeri binari interi positivi (al momento) la sottrazione ha senso solo nel caso in cui $A \geq B$.
Esempi:

| | Binario | Decimale |
|---|---|---|
| Prestito | 1 1 1 | 1 |
| *A* | 1 0 1 0 – | 10 – |
| *B* | 0 0 1 1 = | 3 = |
| Sottrazione | 0 1 1 1 | 7 |

| | Binario | Decimale |
|---|---|---|
| Prestito | 1 1 | |
| *A* | 1 1 0 0 – | 12 – |
| *B* | 0 0 0 1 = | 1 = |
| Sottrazione | 1 0 1 1 | 11 |

Vediamo meglio il primo esempio:

| Inizio | Passo 1 | Passo 2 |
|---|---|---|
| 1 0 1 0 – | 1 0 **0** **10** – | **0** **10** 0 10 – |
| 0 0 1 1 = | 0 0 1 1 = | 0 0 1 1 = |
| | **1** | 1 |

| Passo 3 | Passo 4 | Passo 5 |
|---|---|---|
| 0 **1** **10** 10 – | 0 1 10 10 – | 0 1 10 10 – |
| 0 0 1 1 = | 0 0 1 1 = | 0 0 1 1 = |
| **1** 1 | **1** 1 1 | **0** 1 1 1 |

(Inizio) Sul bit 0 si ha 0 – 1: non si può effettuare
(Passo 1) Il bit 0 si fa prestare un'unità dal bit 1: il bit 1 diventa 0, sul bit 0 si ha $10_{(2)}$-$1_{(2)}$= $1_{(2)}$
(Passo 2) Sul bit 1 si ha 0 – 1: non si può effettuare. Il bit 1 deve farsi prestare 1 un'unità dal bit 2, ma questo vale 0 e si deve far prestare un'unità dal bit 3 che diventa 0. Il bit 2 diventa $10_{(2)}$
(Passo 3) Il bit 2 presta un'unità al bit 1 diventando 1, sul bit 1 si ha $10_{(2)}$-$1_{(2)}$ = $1_{(2)}$
(Passo 4) Sul bit 2 si ha 1 – 0 = 1
(Passo 5) Il bit 4 vale adesso 0 e quindi 0 – 0 = 0

*Moltiplicazione*
Consideriamo la generica cifra di posto *i*: $A_i$ e $B_i$. Le possibili combinazioni che possono assumere le cifre suddette sono:

| $A_i$ | $B_i$ | Prodotto | |
|---|---|---|---|
| 0 | 0 | 0 | $0 \times 0 = 0$ |
| 0 | 1 | 0 | $0 \times 1 = 0$ |
| 1 | 0 | 0 | $1 \times 0 = 0$ |
| 1 | 1 | 1 | $1 \times 1 = 1$ |

Esempi:

| | Binario | Decimale |
|---|---|---|
| *A* | 0110 × | 6 × |
| *B* | 0011 = | 3 = |
| | 0110 + | 18 |
| | 0110 = | |
| Prodotto | 10010 | |

| | Binario | Decimale |
|---|---|---|
| *A* | 0011 × | 3 × |
| *B* | 0010 = | 2 = |
| | 0000 + | 6 |
| | 0011 = | |
| Prodotto | 00110 | |

Come si vede, in entrambe i casi non si è riportato il prodotto degli zeri sui bit più significativi di *B* essendo ininfluenti ai fini del risultato. Si noti che nel caso del primo esempio (con risultato 18) se avessimo stabilito di utilizzare 4 bit il risultato sarebbe stato $0010_{(2)}$ con un evidente errore dovuto alla tracimazione del bit di posto 4 nella somma.

*Divisione*
Trattando numeri interi la divisione sarà espressa tramite quoziente e resto. Il resto varrà zero solo nel caso in cui il dividendo è un multiplo del divisore. Vale la regola che il resto è sempre minore della base *r* che utilizziamo. Nel caso decimale quindi il resto può essere un numero da 0 a 9, mentre nel binario può essere soltanto 0 oppure 1.

Esempi:

| Binario | | Decimale | | Binario | | Decimale | |
|---|---|---|---|---|---|---|---|
| **101**0 | 11 | **10** | 3 | **10**10 | 10 | **10** | 2 |
| 11 | 11 | 9 | 3 | 10 | 101 | 10 | 5 |
| 0**100** | | 1 | | 0010 | | 0 | |
| 011 | | | | 00 | | | |
| 001 | | | | | | | |

Consideriamo la prima divisione:
1 è minore di 11, aggiungiamo il bit seguente.
10 è minore di 11, aggiungiamo ancora il bit seguente.
101 : 11 = 1 con resto 010, ossia 10.
10 è minore di 11, aggiungiamo l'ultimo bit.
100 : 11 = 1 con resto 1.

**Operazione di scorrimento (SHIFT)**
Si vuole introdurre una nuova operazione detta di *scorrimento* o meglio di *shift* che non trova un diretto riscontro sulle operazioni che facciamo sul sistema decimale. Per tale operazione è fondamentale stabilire il numero di bit che si utilizzano. L'operazione di scorrimento può essere effettuata sia a *sinistra* sia a *destra*.

*Shift a sinistra*
Se ho scorrimento a sinistra il bit di posto *i* va ad occupare la posizione *i* + 1, il bit più significativo viene eliminato e il bit meno significativo assume valore zero.
Supponiamo di avere il numero 001011 di 6 bit e di effettuare shift a sinistra.

| **Bit** | **5** | **4** | **3** | **2** | **1** | **0** | **Decimale** |
|---|---|---|---|---|---|---|---|
| **Numero iniziale** | 0 | 0 | 1 | 0 | 1 | 1 | $11_{(10)}$ |
| **1° shift a sinistra** | 0 | 1 | 0 | 1 | 1 | **0** | $22_{(10)}$ |
| **2° shift a sinistra** | 1 | 0 | 1 | 1 | **0** | **0** | $44_{(10)}$ |
| **3° shift a sinistra** | 0 | 1 | 1 | **0** | **0** | **0** | $24_{(10)}$ |
| **4° shift a sinistra** | 1 | 1 | **0** | **0** | **0** | **0** | $48_{(10)}$ |
| **5° shift a sinistra** | 1 | **0** | **0** | **0** | **0** | **0** | $32_{(10)}$ |
| **6° shift a sinistra** | **0** | **0** | **0** | **0** | **0** | **0** | $0_{(10)}$ |

Come si vede, finché non esce il primo bit di valore 1 (ossia diverso da zero), è come se moltiplicassimo per la base (essendo la base 2 in pratica raddoppiamo il numero). Ciò avviene in

modo simile in base 10: se stabiliamo di avere 4 cifre, il numero 0047 dopo uno shift varrà 0470, dopo due shift 4700.
In ogni caso dopo aver operato un numero di shift pari al numero di bit stabiliti il numero varrà zero e continuerà ad esserlo operando ulteriori scorrimenti a sinistra.

*Shift a destra*
Se ho scorrimento a destra il bit di posto *i* va ad occupare la posizione *i* – 1, il bit meno significativo viene eliminato e il bit più significativo assume valore zero.
Supponiamo di avere il numero 100100 di 6 bit e di effettuare shift a destra.

| **Bit** | **5** | **4** | **3** | **2** | **1** | **0** | **Decimale** |
|---|---|---|---|---|---|---|---|
| **Numero iniziale** | 1 | 0 | 0 | 1 | 0 | 0 | $36_{(10)}$ |
| **1° shift a destra** | **0** | 1 | 0 | 0 | 1 | 0 | $18_{(10)}$ |
| **2° shift a destra** | **0** | **0** | 1 | 0 | 0 | 1 | $9_{(10)}$ |
| **3° shift a destra** | **0** | **0** | **0** | 1 | 0 | 0 | $4_{(10)}$ |
| **4° shift a destra** | **0** | **0** | **0** | **0** | 1 | 0 | $2_{(10)}$ |
| **5° shift a destra** | **0** | **0** | **0** | **0** | **0** | 1 | $1_{(10)}$ |
| **6° shift a destra** | **0** | **0** | **0** | **0** | **0** | **0** | $0_{(10)}$ |

Come si vede, finché non esce il primo bit di valore 1 (ossia diverso da zero), è come se dividessimo per la base, (essendo la base 2 in pratica dimezziamo il numero). Ciò avviene in modo simile in base 10: se stabiliamo di avere 4 cifre, il numero 1280 dopo uno shift varrà 0128.
In ogni caso dopo aver operato un numero di shift pari al numero di bit stabiliti il numero varrà zero e continuerà ad esserlo operando ulteriori scorrimenti a destra.