



## Le operazioni di Bit Shift

[http://www.vbsimple.net/news/news\\_07.htm](http://www.vbsimple.net/news/news_07.htm)

Com'è noto i personal computer memorizzano e manipolano i dati in [bit](#). Un gruppetto di 8 bit costituisce un [byte](#). Durante l'elaborazione di certi dati può essere necessario effettuare delle modifiche ai singoli bit di un numero.

L'operazione di Shift di bit (**BitShift**) consiste nello spostare tutti i bit costituenti un valore di un certo numero di posizioni specificato nell'operazione di Shift, che può avvenire in entrambi i sensi. Quando lo spostamento avviene verso sinistra, è detto *ShiftLeft* ed è identificato dall'operatore << in linguaggio C oppure **SHL** in linguaggio Assembly; quando, viceversa, lo spostamento avviene verso destra l'operazione è detta *ShiftRight* ed è identificata dall'operatore >> in linguaggio C oppure **SHR** in linguaggio Assembly.

Prima di vedere con calma di cosa si tratti si raccomanda l'utilizzo dell'utility **BitViewer**, liberamente scaricabile dall'[angolo VBUP2](#).

L'operazione in se stessa effettua una moltiplicazione (*ShiftLeft*) oppure una divisione (*ShiftRight*) di multipli di 2; questo perché il valore assegnato ad un bit dipende dalla sua posizione. Vedi anche le [informazioni aggiuntive sui sistemi di numerazione](#).

Supponiamo che il valore del nostro numero sia -987654321, un numero negativo. Utilizzando BitViewer scopriamo che la sua rappresentazione binaria è:

```
1100 0101 0010 0001 1001 0111 0100 1111
```

La cifra più a sinistra è l'ultimo (32°) bit mentre quella all'estrema destra è il primo bit. La numerazione avviene pertanto dal bit in posizione 0 al bit in posizione 31.

L'operazione di shift a sinistra consiste nello spostare verso sinistra i vari bit del numero di posizioni specificato. Nel nostro esempio il risultato di uno shift a sinistra di 1 bit è:

```
1100 0101 0010 0001 1001 0111 0100 1111 << 1
1000 1010 0100 0011 0010 1110 1001 1110
```

Tutti i bit del numero sono stati spostati verso sinistra di una posizione. Il bit all'estrema destra viene impostato a 0. Visual Basic nativamente non provvede alcuna soluzione per effettuare shift di bit a sinistra; la soluzione più semplice sembrerebbe quella di effettuare una moltiplicazione del numero per 2 se il numero di bit da spostare è 1, per 4 se il numero di bit da spostare è 2, etc...

Torniamo al nostro numero -987654321, una moltiplicazione per due (corrispondente ad uno shift a sinistra di 1 bit) restituisce il valore -1975308642, rappresentabile in binario con:

```

1100 0101 0010 0001 1001 0111 0100 1111 * 2
1000 1010 0100 0011 0010 1110 1001 1110 (-1975308642)

```

Utilizzando la moltiplicazione per 2 in questo semplice caso il risultato ottenuto è quello corretto e senza errori. Ma se il numero di bit da spostare è ad esempio 2 o 3 viene generato un errore di *overflow* durante la computazione della moltiplicazione.

È pertanto necessario trovare un'altra soluzione, differente dalla semplice moltiplicazione per 2. Una soluzione, proposta da [Dave Foley](#), consiste nell'azzerare quei bit che genererebbero l'errore di overflow **prima** di effettuare la moltiplicazione e riattivare l'eventuale bit di segno solo in seguito. Vediamo subito l'esempio pratico:

```

-987654321 << 5 =
-987564321 * 32 (2^5) = Errore di Overflow

```

Tramite il codice di Foley invece:

```

1100 0101 0010 0001 1001 0111 0100 1111 AND
0000 0011 1111 1111 1111 1111 1111 1111 (2^(31-5)-1) =
0000 0001 0010 0001 1001 0111 0100 1111 (18978639)

```

Sarà adesso necessario rintracciare l'eventuale bit di segno da reimpostare dopo che sarà effettuata la moltiplicazione. L'isolamento del bit di segno è fatto come segue:

```

1100 0101 0010 0001 1001 0111 0100 1111 AND
0000 0100 0000 0000 0000 0000 0000 0000 (2^(31-5)) =
0000 0100 0000 0000 0000 0000 0000 0000

```

Il bit acceso in tale posizione indica il nuovo bit di segno dopo l'operazione di Shift. Se il risultato di tale operazione è diverso da 0, sarà salvato il bit di segno in una variabile ed esso verrà aggiunto solo in seguito all'operazione di Shift, effettuata mediante la normale moltiplicazione del valore calcolato in precedenza.

```

0000 0001 0010 0001 1001 0111 0100 1111 * 32 (2^5)
0010 0100 0011 0010 1110 1001 1110 0000 OR
1000 0000 0000 0000 0000 0000 0000 0000 (Segno)
1010 0100 0011 0010 1110 1001 1110 0000 (-1540167200)

```

L'operazione ha avuto effetto! Il numero originale -987564321 è stato spostato a sinistra di 5 bit ottenendo il numero -1540167200.

L'operazione di shift a destra consiste nello spostare verso destra i vari bit del numero di posizioni specificato. Nel nostro esempio il risultato di uno shift a destra di 1 bit è:

```

1100 0101 0010 0001 1001 0111 0100 1111 >> 1
0110 0010 1001 0000 1100 1011 1010 0111

```

Tutti i bit del numero sono stati spostati verso destra di una posizione. Il bit all'estrema sinistra viene impostato a 0. Visual Basic nativamente non provvede alcuna soluzione per effettuare shift di bit a sinistra o a destra. La soluzione più semplice in caso di un ShiftRight consiste nell'effettuare una divisione intera del numero per 2 se il numero di bit

da spostare è 1, per 4 se il numero di bit da spostare è 2, etc...

Torniamo al nostro numero -987654321, una divisione intera per due (corrispondente ad uno shift a destra di 1 bit) restituisce il valore -493827160 con il resto di 1, rappresentabile in binario con:

```
1100 0101 0010 0001 1001 0111 0100 1111 \ 2
1110 0010 1001 0000 1100 1011 1010 1000 (-493827160)
```

Notiamo subito che l'ultimo gruppetto di cifre a destra (ma potevano essere anche più di quattro cifre binarie) è differente dal risultato corretto visto in precedenza. In particolare esso è incrementato di un'unità ( $0111 + 1 = 1000$ ). Per correggere questo comportamento della divisione intera è necessario impostare subito i bit che verranno scartati (quelli all'estrema destra) a 0. L'operazione potrà essere eseguita mediante un calcolo tramite l'operatore **AND**. Il numero risultante sarà pertanto:

```
1100 0101 0010 0001 1001 0111 0100 1111 AND
1111 1111 1111 1111 1111 1111 1111 1110 =
1100 0101 0010 0001 1001 0111 0110 1110 (-987654322)
```

Solo adesso sarà possibile effettuare la divisione intera vista in precedenza...

```
1100 0101 0010 0001 1001 0111 0110 1110 \ 2
1110 0010 1001 0000 1100 1011 1010 0111 (-493827161)
```

Resta soltanto da mettere a punto il bit all'estrema sinistra impostandolo su 0 mediante la solita operazione di **AND**.

```
1110 0010 1001 0000 1100 1011 1010 0111 AND
0111 1111 1111 1111 1111 1111 1111 1111 =
0110 0010 1001 0000 1100 1011 1011 0111 (+1653656487)
```

Queste tre operazioni effettuano l'operazione elementare di Shift a destra di 1 bit soltanto. Se dovessimo spostare più bit dovremmo ripetere lo stesso procedimento svariate volte oppure modificare i valori delle operazioni:

```
1100 0101 0010 0001 1001 0111 0100 1111 AND
1111 1111 1111 1111 1111 1111 1111 0000 =
1100 0101 0010 0001 1001 0111 0100 0000 (-987654336)

1100 0101 0010 0001 1001 0111 0100 0000 \ 16 (2^4)
1111 1100 0101 0010 0001 1001 0111 0100 (-61728396)

1111 1100 0101 0010 0001 1001 0111 0100 AND
0000 1111 1111 1111 1111 1111 1111 1111 =
0000 1100 0101 0010 0001 1001 0111 0100 (206707060)
```

In questo caso abbiamo effettuato uno Shift a destra di 4 bit del numero -987654321.

Una soluzione alternativa proposta da VB Simple consiste nell'utilizzo di una [DLL](#) non [ActiveX](#) creata mediante Assembly. In un'altra sezione di questo sito verrà spiegato come creare una DLL in linguaggio Assembly compilando tramite **NASM** (disponibile presso

l'[angolo VBUP2](#)) e come utilizzare del codice in linguaggio macchina precompilato sempre con Assembly per eseguire operazioni difficoltose o troppo lente per essere eseguite con Visual Basic puro.

Non ci soffermeremo sulla produzione e compilazione della DLL in questione. Il codice sorgente e quello precompilato possono essere scaricati dalla sezione [Downloads](#).

La nostra DLL per effettuare lo Shift dei bit di un numero si chiamerà BitShift.DLL e conterrà due semplici funzioni: *FBIShiftLeft* ed *FBIShiftRight* definite come segue:

1. Private Declare Function SHL Lib "BitShift.dll" Alias "FBIShiftLeft" (ByVal Numero As Long, ByVal Bits As Long, ByVal Reserved1 As Long, ByVal Reserved2 As Long) As Long
2. Private Declare Function SHR Lib "BitShift.dll" Alias "FBIShiftRight" (ByVal Numero As Long, ByVal Bits As Long, ByVal Reserved1 As Long, ByVal Reserved2 As Long) As Long

Entrambe richiedono il passaggio di due parametri fondamentali: Numero e Bits, più due parametri inutilizzati ma aggiunti soltanto per compatibilità con certe funzioni [API](#). Le funzioni richiedono e restituiscono dati di [tipo Long](#) ovvero numeri interi a 32 bit.

Nel progetto in questione sono proposte, oltre alle funzioni sviluppate in Assembly, anche le stesse due funzioni scritte in Visual Basic puro il cui utilizzo è sconsigliato per l'estrema lentezza intrinseca al linguaggio stesso.

Si ringrazia in particolare [Dave Foley \(davef@speakeasy.org\)](#), autore della funzione originale *ShiftLeft* qui leggermente modificata ed adattata.

[Fibia FBI](#)

26 Gennaio 2002



[Torna all'indice degli Articoli](#)

---