Rappresentazione numeri interi con segno

Fondamenti di Informatica I
Corso di laurea in Ingegneria Informatica e Automatica
Sapienza Università di Roma

Domenico Lembo, Paolo Liberatore,
Alberto Marchetti Spaccamela, Marco Schaerf

Numeri negativi

Esistono diversi modi per rappresentare numeri interi con segno, ossia positivi e negativi. I due modi più comuni sono:

1. Bit di segno
2. Complementazione alla base

In entrambi i casi, occorre scegliere in anticipo il numero di bit da usare per rappresentare i numeri. Se non bastano per rappresentare il risultato di un'operazione, questa dà errore.

Modulo e segno

Questo metodo consiste nel riservare il primo dei bit a disposizione per indicare il segno del numero. Di norma si usa 0 per positivo e 1 per negativo.

Assumendo di avere a disposizione quattro bit totali, il primo viene quindi usato per rappresentare il segno del numero, e rimanenti tre per il valore assoluto. Per esempio:

0001 = positivo, valore assoluto 001 = +1

0101 = positivo, valore assoluto 101 = +5

1001 = negativo, valore assoluto 001 = -1

1111 = negativo, valore assoluto 111 = -7

0000 = positivo, valore assoluto 000 = +0

1000 = negativo, valore assoluto 000 = -0

Come si vede negli ultimi due casi, lo zero ha due rappresentazioni, come se +0 e -0 fossero valori diversi.

Modulo e segno

Per effettuare un'operazione di somma, occorre prima verificare il segno dei due addendi e poi decidere se fare la somma o la sottrazione:

- 0001 + 0101
  entrambi positivi, si fa la somma 001+101=110, il risultato è positivo: 0110
- 0101 + 1001
  il secondo è negativo ed in valore assoluto è minore del primo, quindi va fatta la sottrazione: 101-001=100 e il risultato è positivo: 0100
- 0001 + 1111
  va fatta ancora la sottrazione, ma il secondo è in valore assoluto maggiore del primo, quindi: 111-001=110 e il risultato è negativo: 1110
- 1101 + 1010
  entrambi negativi, si fa la somma 101+010=111, il risultato è negativo: 1111

Come in tutte le rappresentazioni a numero fisso di bit, è sempre possibile che il risultato non sia rappresentabile con quel numero di bit (overflow/underflow).

Ad esempio: 1111 + 1001: entrambi negativi, quindi si fa la somma 111+001=1000; il risultato sarebbe negativo, ma dato che richiede quattro bit per il valore assoluto e invece lo spazio per il valore assoluto è di tre bit, non è rappresentabile → *underflow*

Questo succede sempre quando tutti i numeri (con segno o senza) vanno rappresentati con una quantità prefissata di cifre.

Modulo e segno

Il vero svantaggio di questo sistema è che per fare una somma occorre distinguere fra vari casi a seconda del segno dei due operandi: se entrambi dello stesso segno si fa la somma, altrimenti la sottrazione tenendo conto di quali dei due è il maggiore in valore assoluto.

Questo problema si risolve con il sistema del complemento alla base, in cui i numeri si sommano sempre nello stesso modo a prescindere dal segno. A fronte di una maggiore complessità teorica, il sistema risulta più semplice in pratica. Questo porta anche a una minore complessità dei circuiti elettronici.

Esercizi

Sommare i due numeri 1011 e 0101, rappresentati entrambi con modulo e segno

Dato che i due numeri hanno segni opposti, bisogna fare una sottrazione fra i valori assoluti dei due numeri (il maggiore in valore assoluto meno il minore in valore assoluto). In questo caso il maggiore in valore assoluto è il numero positivo (0101), pertanto il risultato sarà positivo (aggiungiamo uno 0 per il segno):

```
 101 -
 011 =
------
 010
```

si ottiene quindi 0010

Esercizi

Sommare i due numeri 0010 e 1110, rappresentati con modulo e segno

in questo caso il secondo numero è negativo ed il primo positivo, il valore assoluto del secondo numero è maggiore del valore assoluto del primo per cui il risultato sarà negativo e si ottiene aggiungendo il bit 1 davanti al risultato della sottrazione 110 - 010:

```
 110 -
 010 =
------
 100
```

si ottiene quindi 1100

Esercizi

Effettuare la sottrazione 11000 - 00001, dove i due numeri sono entrambi rappresentati con modulo e segno

il primo numero è negativo mentre il secondo è positivo; il risultato (se rappresentabile) è sicuramente negativo, e si ottiene facendo la somma (dei valori assoluti) e aggiungendo il segno meno (primo bit pari a 1):

```
 1000 +
 0001 =
-------
 1001
```

il risultato è quindi 11001

Complemento a due

È un sistema a numero fisso di bit che permette di rappresentare numeri sia positivi che negativi. Ha il vantaggio che addizione e inversione di segno sono molto semplici. L'inversione di segno permette di ricondurre la sottrazione all'addizione, dato che basta invertire il segno del secondo operando prima di sommarlo: a-b = a+(-b).

I numeri positivi si rappresentano nello stesso modo che nel sistema con modulo e segno, cioè con un bit 0 per il segno e i restanti per il valore assoluto. Usando quattro bit totali, si ha per esempio:

0001 = +1

0101 = +5

0000 = 0

I numeri positivi che non rientrano nell'intervallo 0000 - 0111 non sono rappresentabili in complemento a due con quattro bit. Il massimo numero positivo rappresentabile è quindi 7.

I numeri negativi hanno tutti il primo bit che vale uno, ma i restanti bit non sono il valore assoluto del numero!

Come si rappresentano i numeri negativi in complemento a due viene spiegato più avanti.

Complementazione

Dato un qualsiasi numero, positivo o negativo, l'inversione di segno si chiama *complementazione*, e si realizza così:

- si invertono tutti i bit (si sostituisce 0 con 1 e viceversa)
- si somma 1 (usando la normale operazione di somma)

Questa operazione consente quindi di passare da 4 a -4, ma anche da -3 a 3, ecc. Per esempio, il complemento di 0010 si ottiene invertendo i bit e ottenendo 1101 e poi sommando uno, cosa che produce 1110:

```
0010 ← numero di partenza

1101 ← inversione dei bit
   1 ← si somma 1
-----
1110 ← complemento di 0010
```

Complementazione

Altri esempi:

- complemento di 1110:

```
0001 ← inversione dei bit
   1 ← si somma 1
-----
0010 ← complemento di 1110
```

- complemento di 1010:

```
 0101 ← inversione dei bit
   1 ← si somma 1
-----
0110 ← complemento di 1010
```

- complemento di 0001:

```
1110 ← inversione dei bit
   1 ← si somma 1
-----
1111 ← complemento di 0001
```

- complemento di 0110:

```
1001 ← inversione dei bit
   1 ← si somma 1
-----
1010 ← complemento di 0110
```

Complementazione

Un metodo alternativo per ottenere lo stesso risultato è il seguente:

- partendo da destra, si lasciano inalterati tutti i bit fino al primo uno, incluso
- si invertono i restanti (ossia si sostituisce 0 con 1 e viceversa)

Questo metodo è equivalente al precedente perché quando sommiamo 1, invertiamo tutti i bit a partire da destra fino al primo 0 incluso. A questo punto termina la propagazione del riporto.

Per esempio, per complementare 0110:

0 1 1 0

↑ si lasciano invariati fino a questo

↑↑

questi si invertono

1 0 1 0

Notiamo che la complementazione di zero, qualunque sia il numero di bit che si sta usando per rappresentarlo, è sempre zero (cioè una sequenza di 0).
Inoltre, la complementazione del numero 10…0 è ancora 10…0 (qualunque sia il numero di bit usati!

Complementazione con tre bit

zero	000	←→	000	zero
uno	001	←→	111	- uno
due	010	←→	110	- due
tre	011	←→	101	- tre
- quattro	100	←→	100	- quattro

La complementazione funziona in entrambi i versi.
Notiamo che il complemento di 000 è sempre 000: lo zero ha un'unica rappresentazione
Notiamo che il complemento di 100 è sempre 100: infatti 100 rappresenta – quattro, ed il complemento (+ quattro) non è rappresentabile con 3 bit in complemento a due.

Complemento a due

La complementazione inverte il segno, quindi consente di passare da n a -n e viceversa. Questo permette di convertire i numeri negativi da decimale a complemento a due e viceversa:

-5 si converte effettuando prima la trasformazione di +5 in binario, che produce 0101; questo numero va poi complementato, ottenendo 1011;

1100 si converte in decimale tenendo conto che è negativo (primo bit vale uno): la complementazione produce 0100, che è 4; il numero è quindi -4.

In generale, per convertire un numero decimale negativo -n in complemento a due si passa al suo valore assoluto n e si converte questo in binario (usando il metodo delle divisioni successive); il risultato si complementa. La stessa cosa al contrario consente di convertire un numero negativo rappresentato in complemento a due in decimale: si complementa (ottenendo così un valore positivo) e si converte.

Complemento a due

Un sistema alternativo per convertire un negativo da complemento a due a decimale consiste nell'assumere che il peso della prima cifra sia negativo.

Nel caso di quattro bit, il peso dei bit normalmente sarebbe: 8, 4, 2, 1. Nella rappresentazione in complemento a due invece è -8, 4, 2, 1. Questo sistema si può usare anche per i positivi: dato che il primo bit è zero assumere che il suo peso sia negativo non cambia niente.

Alcune conversioni di esempio:

il numero in complemento a due 1100 si converte in decimale con la somma $1\times(-8) + 1\times4 + 0\times2 + 0\times1 = -8+4 = -4$

0110 è $0\times(-8) + 1\times4 + 1\times2 + 0\times1 = 4+2 = 6$

1010 è $1\times(-8) + 0\times4 + 1\times2 + 0\times1 = -8+2 = -6$

Esercizi

Convertire il numero decimale -12 in complemento a due con sei bit.

si converte prima il numero +12 usando il meccanismo delle divisioni successive:

```
12 | 0
 6 | 0
 3 | 1
 1 | 1
```

in questo modo si ottiene 1100, che è la rappresentazione di +12; dal momento che il numero di bit devono essere sei, si aggiungono due bit a sinistra: 001100; questa è la rappresentazione del valore positivo dodici; per ottenere meno dodici occorre effettuare la complementazione

```
001100 (inversione bit)
110011 +
     1 =
--------
110100
```

la rappresentazione di meno dodici è quindi 110100

Esercizi

un numero x si rappresenta in complemento a due a cinque bit come 10011; determinare la rappresentazione in complemento a due di -x

Un modo di procedere è convertire 10011 in decimale, come abbiamo discusso in precedenza (si complementa, si converte), e successivamente convertire il decimale in binario (metodo delle divisioni successive). Non è però necessario procedere in tal modo, dato che l'inversione di segno consiste in una semplice operazione di complementazione

```
 10011 (inversione bit)
 01100 +
     1 =
--------
 01101
```

la rappresentazione di -x è quindi 01101

Esercizi

Convertire -289 in complemento a due a dieci bit

Si converte 289 e poi si inverte il segno. Usando il metodo delle divisioni successive si vede che 289=100100001.

Questo numero contiene solo nove cifre, per cui prima di complementarlo va esteso a dieci: 0100100001.

Questo numero si può ora invertire di segno: l'ultima cifra è uno, che si lascia così com'è; le altre si invertono, generando 1011011111

Operazioni in complemento a due

La rappresentazione in complemento a due, anche se più complessa per le conversioni da e verso il decimale, permette di effettuare somme e sottrazioni in modo semplice. Questo si riflette nella dimensione dei circuiti elettronici che realizzano queste operazioni.

Somma

si sommano i due numeri come fossero positivi;

nella somma si include anche il bit di segno come se facesse parte del numero;

Sottrazione

si inverte il segno del secondo operando (complementandolo) e poi si somma al primo.

Operazioni in complemento a due

Per esempio, 0001 + 1111 si ottiene facendo la normale somma come fossero numeri senza segno, nonostante il primo sia positivo e il secondo negativo:

```
 0001 +
 1111 =
-------
10000
```

Si elimina il bit di troppo (in questo caso 1), e si ottiene il risultato, ossia 0000. Questo si può verificare essere il risultato corretto, dato che i due addendi erano 0001=1 e 1111=-8+4+2+1=-1.

Operazioni in complemento a due

Per la sottrazione, bisogna prima invertire il segno del secondo operando, a prescindere se è positivo o negativo. Per esempio, per 0001 - 0101:

si complementa il secondo, cosa che produce 1011;

si somma il primo con il complemento del secondo: 0001 + 1011.

Come si vede,

- sia per la somma che per la sottrazione non c'è bisogno di guardare il segno degli operandi.
- Non c'è bisogno di vedere quale dei due operandi è il maggiore in valore assoluto.
- La somma si effettua sempre come fossero numeri senza segno, la sottrazione come una somma invertendo il segno del secondo operando.

Operazioni in complemento a due

Da notare che esiste un'unica rappresentazione dello zero, al contrario del sistema con modulo e segno. Infatti, +0=0000; volendolo invertire, la complementazione produrrebbe 1111+1=(1)0000, ossia sempre 0000.

Il massimo numero rappresentabile con quattro bit è 0111, ossia 7. Il minimo è 1000, ossia -8. L'intervallo non è del tutto simmetrico, dato che esiste un numero negativo in più rispetto ai positivi.

Più avanti verrà spiegato il perchè della asimmetria e quali sono i valori massimo e minimo per un numero generico di bit.

Operazioni in complemento a due

La somma può non essere rappresentabile nei bit a disposizione. Si può verificare se questo è successo in due modi alternativi:

- i due addendi hanno lo stesso segno ma il risultato ha segno diverso (se due addendi hanno segno diverso non c'è mai overflow o undeflow, anche detti *trabocco*);
- l'ultimo riporto è diverso dal penultimo.

Lo stesso vale per la sottrazione (per esempio, 7-(-5) non è rappresentabile con 4 bit).

La verifica si effettua in uno qualsiasi dei due modi quando si fa la somma.

Esempio: $n = 5, \quad \pm 9 \pm 3, \quad \pm 9 \pm 8$

intervallo di rappresentazione: da -2^4 a $2^4 - 1$ (ovvero, da -16 a 15)

riporto		00011		11001		00111		11111
	$+9$	01001	$+9$	01001	-9	10111	-9	10111
	$+3$	00011	-3	11101	$+3$	00011	-3	11101
	$+12$	01100	$+6$	00110	-6	11010	-12	10100

In questo caso non si ha trabocco.

riporto		01000		10000
	$+9$	01001	-9	10111
	$+8$	01000	-8	11000
	-15	10001	$+15$	01111
	(e non $+17$)		(e non -17)	

In questo caso si ha trabocco.

Si ha **trabocco** quando il riporto sul bit di segno è diverso dal riporto dal bit di segno verso l'esterno.

Esercizi

Convertire -1341 e -1123 in complemento a due a dodici bit. Sommare i due valori, controllando se si è verificato un overflow. In caso contrario, convertire il risultato in decimale.

Il valore assoluto del primo numero è 1341, che in binario a dodici bit è 010100111101; il complemento produce 101011000011. Per il secondo numero, 1123 è 010001100011, il cui complemento è 101110011101. La somma è:

```
 1 111  11111  ← riporti
  101011000011 +
  101110011101 =
--------------
 1011001100000
```

Il bit in più a sinistra va eliminato, ottenendo quindi 011001100000. Questo sarebbe un valore positivo (primo bit a zero) mentre gli operandi sono negativi (primo bit a uno). Si è quindi verificato un overflow. Questo si può verificare anche guardando i riporti: il penultimo non c'è ma l'ultimo sì; sono diversi, quindi si è verificato un overflow.

Notare che la presenza del bit in più a sinistra è irrilevante ai fini dell'overflow, come si vede dal prossimo esempio.

Esercizi

Convertire 45 e -10 in complemento a due a otto cifre, sommare i due valori binari e riconvertire il risultato in decimale se non si è verificato overflow.

Il primo numero si converte con il solito meccanismo delle divisioni successive, ottenendo 00101101. Per il secondo numero, prima si converte 10 in 00001010, poi si complementa: gli ultimi due bit rimangono inalterati, gli altri si invertono: 11110110. La somma produce:

```
 111111    ← riporti
  00101101 +
  11110110 =
  ----------
 100100011
```

Il risultato si ottiene scartando il bit in più. È quindi 00100011.

Esercizi (cont. sol.)

Notare che nonostante il risultato della somma avesse apparentemente un bit di troppo, non si è verificato un overflow.

Infatti, l'esistenza di un nono bit in una rappresentazione a otto bit indica overflow solo per numeri senza segno. Nel complemento a due invece l'overflow è rilevabile dai segni: i due operandi erano di segno diverso, e quindi l'overflow non si può verificare. In alternativa, si vede come i due ultimi riporti a sinistra erano entrambi presenti; essendo uguali, non si è verificato l'overflow.

È quindi possibile convertire il risultato in decimale, ottenendo:

$$-0\times2^7 + 0\times2^6 + 1\times2^5 + 0\times2^4 + 0\times2^3 + 0\times2^2 + 1\times2^1 + 1\times2^0 =$$
$$= -0 + 0 + 32 + 0 + 0 + 0 + 2 + 1 = 35$$

Esercizi

Convertire 12 e 24 in binario, complemento a due a otto cifre. Sottrarli e convertire il risultato in decimale.

La conversione dei due numeri produce 00001100 e 00011000. La sottrazione si realizza invertendo il segno del secondo e sommando poi i due valori. 00011000 cambiato di segno è 11101000. Si fa la somma:

```
    1        ← riporti
 00001100 +
 11101000 =
-----------
 11110100
```

Dato che i nessuno dei due riporti più a sinistra è presente, non si è verificato overflow. Come verifica alternativa, notiamo che i primi due bit dei due addendi sono 0 e 1: essendo diversi, l'overflow è impossibile.

Esercizi (cont. sol.)

La conversione in decimale produce:

$-1\times2^7 + 1\times2^6 + 1\times2^5 + 1\times2^4 + 0\times2^3 + 1\times2^2 + 0\times2^1 + 0\times2^0 =$
$= -128 + 64 + 32 + 16 + 0 + 4 + 0 + 0 = -12$

In alternativa, 11110100 si può convertire notando che il primo bit è uno, quindi negativo. Si complementa, ottenendo 00001100. Questo valore si converte come al solito: 8+4=12. Dato che il primo bit era negativo, il risultato era -12.

Esercizi

Convertire -2 e 4 in complemento a due a quattro cifre, sottrarli e convertire il risultato in decimale.

Con quattro bit, 2 è 0010, che complementato diventa 1110. Il secondo numero è positivo: 4 diventa 0100. I due numeri sono quindi:

-2 = 1110

4 = 0100

Dato che vanno sottratti, il secondo numero va complementato, ottenendo -4=1100. La somma produce:

```
11      ← riporti
 1110 +
 1100 =
-------
11010
```

Dato che i due riporti sono uguali non si è verificato overflow. Il risultato è 1010, che in decimale è -8+2=-6.

Massimo e minimo numero rappresentabile in complemento a due

Come abbiamo già visto, metà dei numeri rappresentabili sono zero o positivi, l'altra metà sono negativi. Questo significa che con quattro bit:

sedici numeri totali;

otto fra zero e positivi;

otto negativi.

Questo fa sì che i numeri rappresentabili siano 0, …, 7 per il gruppo "zero o positivo" e -1, …, -8 per in negativi. Il fatto che lo zero sia raggruppato nella metà dei positivi fa sì che il massimo positivo sia 7, mentre il minimo dei negativi sia -8.

Massimo e minimo numero rappresentabile in complemento a due

La regola generale per massimo e minimo rappresentabile con n bit è:

- in totale, sono rappresentabili 2^n valori;
- di questi, la metà sono $2^n/2=2^{n-1}$;
- una metà viene usata per i numeri zero o positivi; dato che c'è anche lo zero, il massimo è $2^{n-1}-1$;
- l'altra metà è per i negativi; questa volta lo zero non c'è, quindi il minimo è -2^{n-1}.

Con n bit i valori rappresentabili vanno quindi da -2^{n-1} a $2^{n-1}-1$.

Esercizi

1. Dire il minimo e il massimo numero rappresentabile in complemento a due con cinque bit

il massimo numero rappresentabile è quello positivo in cui tutti i bit valgono uno a parte il bit di segno: 01111; convertendo questo numero in decimale si ottiene $2^3+2^2+2^1+2^0=15$; il minimo numero rappresentabile si ottiene ponendo il primo bit a uno, con valore $-2^4=-16$; dal momento che qualsiasi altro bit uno avrebbe valore positivo e aumenterebbe quindi il numero, gli altri si pongono a zero, ottenendo quindi -16

2. dire il minimo e il massimo numero rappresentabile con cinque bit usando la rappresentazione con modulo e segno

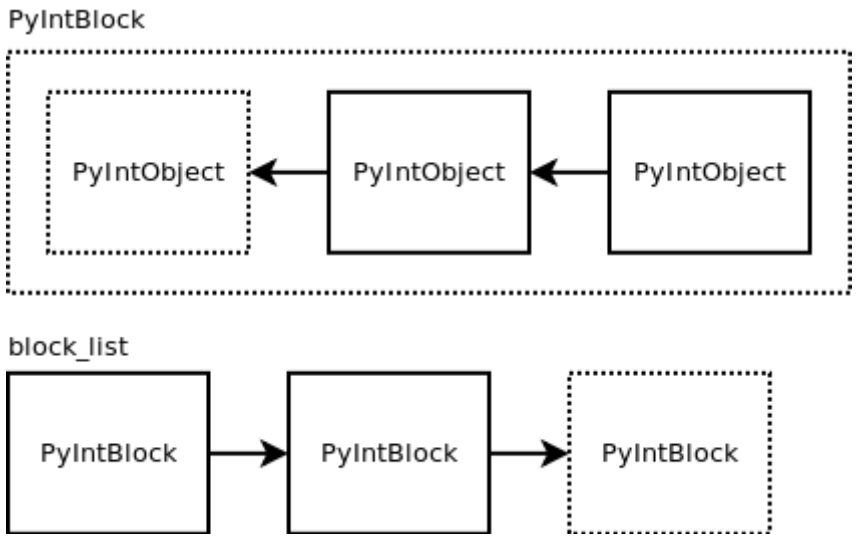
in questo caso basta trovare il massimo valore assoluto con quattro bit, dal momento che il quinto serve per il segno: 1111=15; il minimo e il massimo si ottengono poi ponendo il segno negativo o positivo, rispettivamente; quindi si ottiene -15 e +15

Rappresentazione Interi in Python

- Python usa una rappresentazione degli interi a precisione arbitraria. In pratica usa un numero arbitrario di locazioni di memoria da 32 bits ciascuna per rappresentare un intero

Ogni blocco da 32 bits contiene un numero binario senza segno da 30 bit, il segno è contenuto a parte in un bit specifico.

Rappresentazione in modulo e segno



Rappresentazione Interi in Python

La rappresentazione di 123456789101112131415 è composta da una variabile che denota il numero di blocchi e dalla serie di blocchi da 30 bit contenenti i valori (dal meno al più importante):

ob_size 3

ob_digit $(437976919)_2$ $(87719511)_2$ $(107)_2$

Per convertirlo nel numero decimale corrispondente:

$(437976919 * 2^{30*0}) + (87719511 * 2^{30*1}) + (107 * 2^{30*2})$

Rappresentazione Interi in Python

Le operazioni sono realizzate spezzandole in operazioni su numeri di 30 bits (con riporto tra un blocco ed un altro).

Ad esempio, per sommare due numeri si sommano prima i due blocchi meno significativi, si calcola il risultato ed il riporto e poi si passa ai secondi due blocchi (aggiungendo il riporto).

Altre operazioni anche più complesse vengono sempre ridotte a sequenze di operazioni su blocchi da 30 bits