Parte 7

Operatori bit a bit

Notazione posizionale

Ogni cifra assume un ***significato diverso a seconda della posizione*** in cui si trova

Rappresentazione di un numero su *n* cifre in base *b*:

Posizioni

$$a_{n-1}\ a_{n-2}\ a_{n-3} \quad \ldots \quad a_1\ a_0$$

$$a_i \in \{0, 1, \ldots, b-1\}$$

ESEMPIO ***Notazione decimale***:

$b = 10$, $a_i \in \{0, 1, 2, \ldots, 9\}$

$345 \Rightarrow a_2 = 3,\ a_1 = 4,\ a_0 = 5$

Definizione formale (1)

Si sceglie un qualsiasi numero naturale ***b*** (diverso da 0 e da 1) che chiameremo ***base***

Si scelgono ***b*** simboli diversi che chiameremo ***cifre*** {0, ... *b* - 1}

Si compongono i numeri tenendo presente che il valore di ogni cifra va moltiplicato per un ***peso***:

- b^0 cioè 1 (unità) se è l'ultima cifra alla destra del numero che stiamo considerando
- b^1 cioè b se è la seconda cifra da destra,
- b^2 se è la terza cifra da destra,
- così via, $b^{(n-1)}$ se è la n-esima cifra da destra

Definizione formale (2)

Valore:

$$[\ a_{n-1}\, a_{n-2}\, a_{n-3} \ldots a_1\, a_0\ ]_b =$$

$$a_0*1 + a_1*b + a_2*b^2 + a_3*b^3 + \ldots + a_{n-1}*b^{n-1} = \sum_{i=0,1,\ldots,n-1} a_i * b^i$$

Peso cifra i-esima

ESEMPIO

$b = 10,\ a_i \in \{0, 1, 2, \ldots, 9\}$

$[\ 345\ ]_{10} = 5 * 1 + 4 * 10 + 3 * 10^2$

Notazione binaria

Base 2 → 2 cifre:

- {0, 1}

La cifra nella posizione *i-esima* ha peso 2^*i*

ESEMPI (*configurazioni di bit*):

$[0]_{10} = [0]_2 = [0*1]_{10}$

$[1]_{10} = [1]_2 = [1*1]_{10}$

$[2]_{10} = [10]_2 = [1*2 + 0*1]_{10}$

$[3]_{10} = [11]_2 = [1*2 + 1*1]_{10}$

Notazione esadecimale

- Base 16 → 16 cifre:
  - {0, 1, 2, ..., 9, A, B, C, D, E, F}
- Valore corrispondente delle cifre in decimale:
  - 0, 1, 2, ..., 9, 10, 11, 12, 13, 14, 15
- La cifra nella posizione *i-esima* ha peso *16^i*

ESEMPI

$[0]_{10} = [0]_{16} = [0*1]_{10}$

$[10]_{10} = [A]_{16} = [10*1]_{10}$

$[18]_{10} = [12]_{16} = [1*16 + 2*1]_{10}$

Intervallo di rappresentabilità

- Con *N* cifre in base *b* si possono rappresentare numeri naturali nell'intervallo $[0, b^N - 1]$
- Il tipo naturale (*unsigned*) su *N* bit permette di rappresentare numeri naturali nell'intervallo $[0, 2^N - 1]$

Corrispondenza notazioni binaria ed esadecimale

Il sistema esadecimale è molto usato in informatica per la sua **relazione diretta tra una cifra esadecimale e quattro cifre binarie**:

una cifra in base 16 è rappresentata mediante 4 cifre in base 2

- $[0]_{16} = [0000]_2$ $[1]_{16} = [0001]_2$ $[2]_{16} = [0010]_2$ $[3]_{16} = [0011]_2$
- $[4]_{16} = [0100]_2$ $[5]_{16} = [0101]_2$ $[6]_{16} = [0110]_2$ $[7]_{16} = [0111]_2$
- $[8]_{16} = [1000]_2$ $[9]_{16} = [1001]_2$ $[A]_{16} = [1010]_2$ $[B]_{16} = [1011]_2$
- $[C]_{16} = [1100]_2$ $[D]_{16} = [1101]_2$ $[E]_{16} = [1110]_2$ $[F]_{16} = [1111]_2$

Da base 2 a base 16

Per ottenere la rappresentazione in base 16 di un numero in base 2:

- dividere le cifre in gruppi di 4 da destra a sinistra, aggiungendo eventualmente degli zeri (a sinistra)
- sostituire ad ogni gruppo la corrispondente cifra in base 16

Esercizi

Convertire in esadecimale:

$[11011010]_2 = [\ ?\ ]_2 = [\ ?\ ]_{16}$

$[1011010]_2 = [\ ?\ ]_2 = [\ ?\ ]_{16}$

$[11111111]_2 = [\ ?\ ]_2 = [\ ?\ ]_{16}$

Esercizi

Soluzione:

$[11011010]_2 = [\ 1101\ 1010\ ]_2 = [\ D\ A\ ]_{16}$

$[1011010]_2 = [\ 0101\ 1010\ ]_2 = [\ 5\ A\ ]_{16}$

$[11111111]_2 = [\ 1111\ 1111\ ]_2 = [\ F\ F\ ]_{16}$

Da base 16 a base 2

Per ottenere la rappresentazione in base 2 di un numero in base 16 sostituire ciascuna cifra con la sua corrispondente rappresentazione in base 2

Convertire in binario:

$[\,A\ B\,]_{16} = [\,1010\ 1011\,]_2 = [\,10101011\,]_2$

$[\,3\ 7\,]_{16} = [\,0011\ 0111\,]_2 = [\,110111\,]_2$

$[\,4\ F\,]_{16} = [\,0100\ 1111\,]_2 = [\,1001111\,]_2$

Operatori bit a bit

Operano sui numeri naturali intesi come ***vettori (o configurazioni) di bit***

Operatore	Simbolo	Arietà
AND bit a bit	&	Binario
OR bit a bit	|	Binario
XOR (OR esclusivo) bit a bit	^	Binario
complemento a 1	~	Unario
traslazione (shift) a sinistra	<<	Binario
traslazione (shift) a destra	>>	Binario

Operatori binari AND, OR, XOR

- Siano *x* ed *y* gli argomenti di un operatore binario bit a bit AND, OR o XOR
- Siano x_i e y_i le *i*-esime cifre di *x* ed *y* nella rappresentazione in base 2
- Il risultato sia il vettore di bit *z*
- Supponiamo di voler calcolare z = x & y
- Denotiamo con $(x \& y)_i$ l'*i*-esima cifra di z:

  $(x \& y)_i = z_i$

Tabella degli operatori

In generale, a seconda dell' operatore,usiamo la notazione:

$(x \,\&\, y)_i$, $(x \mid y)_i$ e $(x \,\hat{}\, y)_i$

per indicare l'*i*-esima cifra di ***z***

x_i	y_i	$(x\&y)_i$	$(x\|y)_i$	$(x\hat{}y)_i$
0	0	0	0	0
0	1	0	1	1
1	0	0	1	1
1	1	1	1	0

Esempi

	Binario	Hex	Dec
x	1 1 0 1 0 1	35	53
y	1 0 1 1 0 0	2C	44
x & y			

	Binario	Hex	Dec
x	1 1 0 1 0 1	35	53
y	1 0 1 1 0 0	2C	44
x ^ y			

	Binario	Hex	Dec
x	1 1 0 1 0 1	35	53
y	1 0 1 1 0 0	2C	44
x | y			

Esempi

	Binario	Hex	Dec
x	1 1 0 1 0 1	35	53
y	1 0 1 1 0 0	2C	44
x & y	1 0 0 1 0 0	24	36

	Binario	Hex	Dec
x	1 1 0 1 0 1	35	53
y	1 0 1 1 0 0	2C	44
x ^ y	0 1 1 0 0 1	19	25

	Binario	Hex	Dec
x	1 1 0 1 0 1	35	53
y	1 0 1 1 0 0	2C	44
x | y	1 1 1 1 0 1	3D	61

Operatore NOT

- Operatore unario
- Scambia tutti i bit di un vettore da 0 a 1 e viceversa

	Binario	Hex	Dec
x	1 1 0 1 0 1	35	53
~x	0 0 1 0 1 0	A	10

Configurazioni speciali

Come si scrive una configurazione con tutti i bit a 1, quale che sia la dimensione del tipo di dato su cui la configurazione è memorizzata?

Si può usare l'espressione: ~0

Traslazione (shift)

- L'operatore di traslazione (***shift***) a sinistra/destra sposta ogni bit del primo operando di tante posizioni a sinistra/destra quante sono specificate dal secondo operando
- Le posizioni lasciate libere in seguito alla traslazione sono riempite con bit a 0

Esempi di traslazione

	Binario	Hex	Dec
x	1 1 0 1 0 1	35	53
x << 1	1 0 1 0 1 0	2A	42
x << 2	? ? ? ? ? ?	?	?
x >> 2	0 0 1 1 0 1	D	13
x >> 3	? ? ? ? ? ?	?	?

Soluzioni

	Binario	Hex	Dec
x	1 1 0 1 0 1	35	53
x << 1	1 0 1 0 1 0	2A	42
x << 2	0 1 0 1 0 0	14	20
X >> 2	0 0 1 1 0 1	D	13
x >> 3	0 0 0 1 1 0	6	6

Utilizzo della traslazione (1)

- Come è fatta la rappresentazione in base 2 del numero **1<<n** ?

Ha tutti i bit a zero, tranne quello in posizione n

Ad esempio, su 32 bit, 1<<14 è:

00000000 00000000 01000000 00000000

Utilizzo della traslazione (2)

- Come è fatta la rappresentazione in base 2 del numero **~(1<<n)** ?

Ha tutti i bit a 1, tranne quello in posizione n

Ad esempio, su 32 bit, **~(n<<14)** è:

11111111 11111111 10111111 11111111

Maschere di bit

Maschere di bit o ***maschere binarie***

- Configurazioni di bit usate per:
  - Controllare il valore di alcuni bit
  - Settare/resettare il valore di alcuni bit
  - Indicare quali bit sono significativi e quali non lo sono

Controllo dello stato di un bit

- Come si può fare per controllare lo ***stato di un bit in un vettore***?
- Idea di base:
  - Utilizzare una maschera binaria e un operatore bit a bit
  - Verificare in base al risultato lo stato del bit all'interno del vettore di bit
- Proposte?

Controllo dello stato di un bit

- Per scoprire se il bit *i*-esimo di un vettore è o meno settato, basta fare l'AND con una maschera in cui solo tale bit è settato
- Il risultato sarà diverso da 0 se e solo se l'*i*-esimo bit è effettivamente settato
- Controllo se il bit di posizione 2 è settato in x

	Binario
x	0 1 0 1 0 1
y	0 0 0 1 0 0
x & y	0 0 0 1 0 0

	Binario
x	1 1 1 0 0 1
y	0 0 0 1 0 0
x & y	0 0 0 0 0 0

Esempio di controllo (1)

```
main(){
unsigned int x = 53, maschera = 4;
if ((x & maschera) != 0)
    cout<<"Bit settato"<<endl;
else
    cout<<"Bit non settato"<<endl;
}
```

Esempio di controllo (2)

Sarebbe stato corretto scrivere il programma così?

```
main(){
unsigned int x = 53, maschera = 4 ;
if ((x & maschera) == 1)
    cout<<"Bit settato"<<endl ;
else
    cout<<"Bit non settato"<<endl ;
}
```

Esempio di controllo (2)

Sarebbe stato corretto scrivere il programma così?

```
main(){
unsigned int x = 53, maschera = 4 ;
if ((x & maschera) == 1)
    cout<<"Bit settato"<<endl ;
else
    cout<<"Bit non settato"<<endl ;
}
```

No, perché il risultato è 1 solo nel caso particolare in cui il bit da controllare sia quello di posizione 0

Esercizio

- Scrivere un programma che effettui il controllo sullo stato di un bit in un vettore
- Programma che legga da stdin due numeri naturali x ed n, e dica se il bit in posizione n è settato o meno nella rappresentazione del numero x
- Attenzione alla ***priorità*** degli operatori bit a bit
- Per la soluzione, **vedi programma** ***controlla_bit.cc***

Controllo di più bit

- Come facciamo per scoprire se più di un bit è settato con una sola operazione?

- Basta usare una ***maschera y con tutti i bit da controllare settati***

- Se il risultato dell'AND è uguale ad y, i bit sono tutti settati

- Se il risultato è diverso da 0 almeno un bit è settato

Settare un bit

- Come si fa a settare un bit in una specifica posizione di un vettore di bit?
- Per settare il bit *i*-esimo:
  - Effettuare l'OR con una maschera in cui solo tale bit è settato
  - Assegnare il risultato al vettore originario
- Ovviamente una maschera con più di un bit settato permetterà di settare più bit
- I bit già settati nel vettore originario rimangono invariati

Esempio di settaggio

- Setto i bit di posizione 2 e 3

	Binario	Hex	Dec
x	1 1 0 1 0 1	35	53
y	0 0 1 1 0 0	C	12
x = x | y	1 1 1 1 0 1	3D	61

- Il bit di posizione 2 – già settato – resta invariato

Esercizio

- Programma che setta i bit di posizione 2 e 3 (es. precedente)

```
main(){
unsigned int x=53, maschera=12;
x |= maschera;
cout<<x<<endl;
}
```

- Scrivere un programma che legga da stdin due numeri naturali x ed n e setti il bit in posizione n nel numero x.
- Per la soluzione, **vedi programma** *setta_bit.cc*

Reset di un bit

- Come si fa a resettare uno specifico bit in vettore di bit?
- Scrivere un programma che legga da stdin due numeri naturali x ed n, e resetti il bit in posizione n nel numero x
- Per la soluzione, **vedi programma** *resetta_bit.cc*

Flip

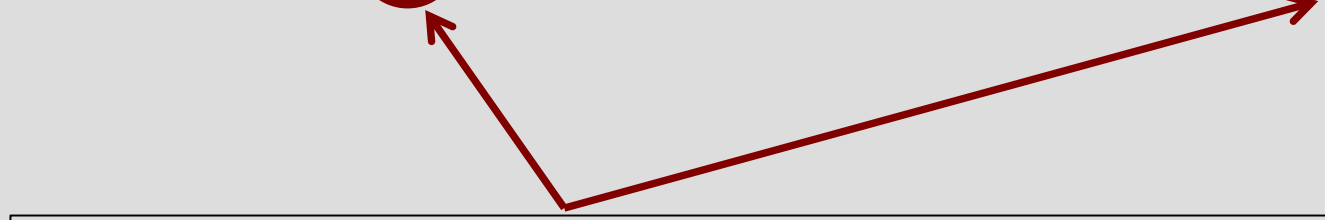
- Col verbo ***to flip*** si indica tipicamente il cambiamento di stato di un bit
- Come si fa per far cambiare stato ad un solo bit in un vettore di bit, con una sola operazione e senza cambiare valore a nessun altro bit?

Effettuare l'XOR con una maschera in cui solo tale bit è settato

Flip con XOR

	Binario
x	0 1 0 1 0 1
y	0 0 0 1 0 0
x ^ y	0 1 0 0 0 1

	Binario
x	1 1 1 0 0 1
y	0 0 0 1 0 0
x ^ y	1 1 1 1 0 1



L'unico bit a cambiare di stato è quello in corrispondenza del bit settato nella maschera binaria

Operatori e altri tipi di dato

- Gli operatori bit a bit non operano su float e double
- Possono essere utilizzati su numeri interi con segno (***signed int***)
- Per alcune operazioni tuttavia il risultato può essere diverso da quello che si ottiene con i numeri naturali (***unsigned int***)
- I numeri con segno hanno una rappresentazione diversa
  - Rappresentazione in complemento a 2
- Qui non approfondiremo oltre il problema

Esercizio

Bisogna gestire lo stato di illuminazione di una strada in termini di lampioni funzionanti.

La lista dei lampioni funzionanti è mantenuta attraverso una configurazione di bit (supponiamo 8 bit).

Ad ogni lampione è associato un bit della configurazione, che ne indica lo stato: 1 se il lampione è funzionante, 0 se il lampione è guasto.

Le funzioni previste sono:

1.aggiornamento dello stato di un lampione, che puo' guastarsi o essere riparato

2.stampa dei lampioni funzionanti

3.controllo se i lampioni possono funzionare in modalità risparmio energetico, ovvero quelli di indice pari accesi e quelli di indice dispari spenti.
Nota bene: per essere verificata, tutti i lampioni di indice pari devono essere accesi e tutti quelli di indice pari spenti.

Esercizio (cont.)

Utilizzare una sola funzione ausiliaria per guastare/riparare lampioni.

All'inizio tutti i lampioni gestiti dal programma sono funzionanti.
Il programma deve fornire il seguente **menù:**

Gestione lampioni

Comandi disponibili:
G Segnalare guasto ad un lampione
R Segnalare riparazione di un lampione
L Stampa lista lampioni funzionanti
K Controllo risparmio energetico
T Termina
Per la soluzione, **vedi programma** *lampioni.cc*

Esercizi

esercizi_bit1.txt

- Tutti i programmi vanno scritti usando solo istruzioni di assegnamento e di ingresso/uscita (niente if, for, while, ecc.)
- Non si devono utilizzare operatori di confronto (<, <=, >, >=, !=, ==)
- Non si devono utilizzare operatori logici (!, &&, ||)
- Per semplicità, si assume che i valori immessi dall'utente non portino mai ad effettuare traslazioni di un numero di posizioni maggiore o uguale del numero di bit su cui sono rappresentati i vettori di bit