Operatori logici pe biți

Operatorii logici pe biți se aplică numai operanzilor întregi şi au ca efect aplicarea operațiilor logice cunoscute (negație, conjuncție, disjuncție şi disjuncție exclusivă) **bit cu bit**. Operatorii logici pe biți sunt:

Operator	Denumire	Tip	Prioritate
~	complementariere (negația pe biți)	unar	2
<<, >>	deplasare stânga, deplasare dreapta	binari	6
&	conjuncție logică pe biți	binar	9
^	disjuncție exclusivă pe biți	binar	10
|	disjuncție logică pe biți	binar	11

Efectul operatorilor pe biți este:

x	0	1
~x	1	0

&	0	1
0	0	0
1	0	1

^	0	1
0	0	1
1	1	0

|	0	1
0	0	1
1	1	1

Operatorii de deplasare au ca efect deplasarea reprezentării binare a primului operand spre stânga (<<) sau spre dreapta (>>). Numărul de poziții care se deplasează este specificat de cel de-al doilea operand. La deplasarea la stânga, pozițiile rămase libere în dreapta se completează cu `0`. La deplasarea la dreapta, pozițiile rămase libere în stânga se completează cu `0` (dacă operandul stâng este un întreg pozitiv) sau cu `1` (dacă operandul este întreg negativ).

Exemple

Să considerăm următoarele declarații de variabile:

```
int n=3, a=0X1F8A, b=0XF0F5;
```

Pentru a determina cu uşurință reprezentările în memorie ale variabilelor `a` şi `b`, am exprimat valorile lor în hexazecimal. Reprezentarea în memorie a acestor variabile este:

a=	0	0	0	1	1	1	1	1	1	0	0	0	1	0	1	0
	15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0

b=	1	1	1	1	0	0	0	0	1	1	1	1	0	1	0	1
	15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0

~a=	1	1	1	0	0	0	0	0	0	1	1	1	0	1	0	1
	15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0

a&b=	0	0	0	1	0	0	0	0	1	0	0	0	0	0	0	0
	15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0

a^b=	1	1	1	0	1	1	1	1	0	1	1	1	1	1	1	1
	15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0

a|b=	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1
	15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0

a>>3=	0	0	0	0	0	0	1	1	1	1	1	1	0	0	0	1
	15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0

a<<3=	1	1	1	1	1	1	0	0	0	1	0	1	0	0	0	0
	15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0

Dacă `a` ar fi fost declarat de tip `unsigned`, prin deplasare la dreapta, s-ar obține acelaşi rezultat, deoarece valoarea lui `a` este pozitivă (bitul semn este `0`). Valoarea lui `b` este negativă (bitul 15 – bitul semn – este `1`), prin deplasare la dreapta se propagă semnul, deci se completează cu `1`.

b>>3=	1	1	1	1	1	1	1	0	0	0	0	1	1	1	1	0
	15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0

Observații

1. Expresia `x<<n` are ca efect înmulțirea operandului `x` cu 2^n. Expresia `x>>n` are ca efect împărțirea întreagă a operandului `x` cu 2^n.
2. Operațiile care se efectuează pe biți (deci acționează direct asupra reprezentării interne a operanzilor) sunt foarte performante (se execută foarte rapid).

Aplicații cu operații la nivel de biți

Transformarea unui bit în 1

Pornim de la valoarea întreagă **X=50**. Reprezentarea acestuia pe **8** biți este **00110010**. Presupunem că dorim setarea bitului **2** la valoarea **1**. Pentru aceasta vom folosi o mască logică în care doar bitul **2** este **1** restul biților fiind **0**, adică **M=00000100**. Valoarea lui **M** este **1 shl 2** (**2^2**). Operația de disjuncție **SAU** aplicată asupra lui **X** şi a lui **M**, conduce la obținerea rezultatului dorit.

```
X   00110010 |
M   00000100
Rez 00110110
```

Generalizând, dacă se doreşte ca valorii X, să i se seteze la valoarea **1**, bitul B (**0≤B≤7**), atunci masca logică este **1 shl B**.

```
X | (1 << B)
```

Transformarea unui bit în 0

Să luăm ca exemplu **X= 109**, pentru a vedea cum se setează un bit la valoarea **0**. Reprezentarea internă a lui este **01101101**. Se cere să se seteze bitul **5** la valoarea **0**. De data aceasta masca va conține toți biții de **1**, excepție bitul **5**. Aspura lui **X** şi **M** vom aplica **ŞI** logic.

```
X   01101101 &
M   11011111
Rez 01001101
```

Presupunem că dorim să setăm la **0** valoarea bitului **B** (**0≤B≤7**).

```
X & ~(1 << B)
```

Testarea valorii unui bit

Plecăm de la valoarea **X=47**. Reprezentarea internă a lui este **00101111**. Presupunem că dorim să cunoaştem valoarea bitului **3** şi bitului **6**. Vom folosi măştile M_1=**00001000** şi M_2=**01000000**. Vom aplica de fiecare dată **ŞI** logic între **X** şi cele două măşti:

```
X   00101111 &
M1  00001000
Rez 00001000
```

Respectiv

```
X   00101111 &
M2  01000000
Rez 00000000
```

Generalizând, testarea se va realiza prin :

```
X & (1 << B)
```

Testarea valorilor ultimilor biţi

Pornim de la valoarea întreagă **X=50**. Reprezentarea acestuia pe **8** biţi este **00110010**. Presupunem că dorim să cunoaştem restul la împărţirea întreagă a lui **X** la **8**, adică **x%8**. Valoarea ultimilor **3** biţi din reprezentarea internă a lui, reprezintă tocmai acest rest. Pentru aceasta vom folosi o mască în care ultimii trei biţii sunt **1** restul **0**. Aceasta mască are valoarea **7**, adică **00000111**. Vom aplica operaţia **ŞI** logic.

```
X   00110010 &
M   00000111
Rez 00000010
```

Pe caz general, dacă dorim să cunoaştem valoarea ultimilor **B** biţi (care este egal cu restul împărţirii lui **X** la 2^B) vom exprima astfel:

```
X & (1 << B-1)
```

Reprezentarea mulţimilor prin vector caracteristic implementat pe biţi

Fie n un număr natural (n<VMAX).

O metodă eficientă de a reprezenta o submulţime S a mulţimii {0, 1, ..., n} este vectorul caracteristic. Mai exact, asociem fiecărui element x din mulţimea {0, 1, ..., n} un bit. Bitul corespunzător elementului x are valoarea 1 dacă $x \in S$, respectiv valoarea 0 dacă $x \notin S$.

Descrieţi funcţii care să implementeze operaţiile elementare cu mulţimi (reuniune, intersecţie, diferenţă, incluziune, test de apartenenţă) folosind o astfel de reprezentare.

Ciurul lui Eratostene

Fie n un număr natural ($n \leq 10000$). Să se genereze toate numerele prime mai mici decât n.

Soluţie

O primă idee ar fi să parcurgem toate numerele naturale din intervalul [2, n], pentru fiecare număr să verificăm dacă este prim şi să afişăm numerele prime determinate.

O idee mai eficientă provine in antichitate şi poartă numele ciurul lui Eratostene[1]. Ideea este de a „pune în ciur" toate numerele mai mic decât , apoi de a „cerne" aceste numere până rămân în ciur numai numerele prime. Mai întâi „cernem" (eliminăm din ciur) toţi multiplii lui 2, apoi cernem multiplii lui 3, ş.a.m.d.

Vom reprezenta ciurul ca un vector cu 10000 de componente care pot fi 0 sau 1, cu semnificaţia ciur[i]=1 dacă numărul i este în ciur şi 0 altfel.

Vom parcurge vectorul ciur de la 2 (cel mai mic număr prim), până la $\sqrt{n}$).

1. Eratostene (276–196 î.e.n) a fost un important matematician şi filosof al antichităţii. Deşi puţine dintre lucrările lui s-au păstrat, a rămas celebru prin metoda rapidă de determinare a numerelor prime şi prin faptul că a fost primul care a estimat cu acurateţe diametrul Pământului.

Dacă `ciur[i]` este `1` deducem că `i` este prim, dar toți multiplii lui nu vor fi (deci eliminăm din ciur toți multiplii lui `i`).

În final în vectorul `ciur` vor avea valoarea `1` doar componentele de pe poziții numere prime.

```
#include <iostream.h>
#define NMax 10000
int main()
{int ciur[NMax], n, i, j;
 cout<<"n= "; cin>>n;
 //initial toate numerele sunt in ciur
 for (i=2; i<n; i++) ciur[i]=1;
 for (i=2; i*i<=n; i++)
     if (ciur[i])//i este prim
        //elimin toti multiplii lui i
        for (j=2; j*i<n; j++)
            ciur[i*j]=0;
 for (i=2; i<n; i++)
     if (ciur[i]) cout<<i<<' ';
 return 0; }
```

Aplicații

1. Implementați ciurul lui Eratostene utilizând ciurul reprezentat ca vector caracteristic implementat pe biți!
2. Scrieți o funcție care determină numărul de cifre de 1 din reprezentarea binară a unui număr natural nenul n mai mic ca 2000000000.
3. Scrieți o funcție care identifică cea mai mare putere a lui 2 care îl divide pe n, unde n este număr natural nenul. (Problema se reduce la determinarea celui mai nesemnificativ bit de 1 din reprezentarea binară a lui n)
4. Cod
   http://campion.edu.ro/arhiva/index.php?page=problem&action=view&id=163
5. Paritate
   http://campion.edu.ro/arhiva/index.php?page=problem&action=view&id=843

Bibliografie

- E. Cerchez, M. Şerban - Programarea în limbajul C/C++. Volumul I. Editura Polirom
- D. Lica - Operații pe biți

  http://campion.edu.ro/arhiva/index.php?page=paper&action=view&id=21