

OPERATORI PENTRU LUCRU LA NIVEL DE BIT

Limbajul **C** oferă un număr de operatori pentru lucrul la nivel de bit. Acești operatori se aplică numai variabilelor de tip **int** și **char** și conduc la expresii ale căror rezultate depind de reprezentarea internă a întregilor și au caracteristici care depind de implementarea tipurilor cu semn.

Operatorii binari de lucru la nivel de bit sunt:

- <<** deplasare stânga cu un anumit număr de biți
- >>** deplasare dreapta cu un anumit număr de biți
- &** **ȘI (AND)** la nivel de bit;
- ^** **SAU EXCLUSIV (XOR)** la nivel de bit;
- |** **SAU (OR)** la nivel de bit.

În afară de acești operatori binari, limbajul ne mai pune la dispoziție un operator unar care permite determinarea complementului față de 1 al operandului. Acest operator este **~** și are aceeași prioritate ca și ceilalți operatori unari existenți în limbaj.

Operatorii **&**, **^** și **|** sunt folosiți, de obicei, pentru a realiza operațiile dorite asupra primului operand în conformitate cu o configurație de biți indicată de cel de-al doilea operand. Din acest motiv, de foarte multe ori, cel de al doilea operand mai este numit și „mască” pentru că nu face decât să „mascheze” anumiți biți din primul operand. Acești operatori se aplică fiecărui bit din reprezentarea internă (binară, în virgulă fixă) a numărului.

Operatorii de deplasare **<<** și **>>**

<< Deplasează la stânga numărul din stânga operatorului cu un număr de poziții egal cu numărul existent în dreapta operatorului.

Exemplu: rezultatul expresiei

$n \ll 3$

este valoarea numărului **n** deplasat cu 3 (trei) poziții la stânga.

Deplasarea la stânga completează pozițiile rămase libere cu zerouri. Deplasarea cu o poziție a numărului **n** este echivalentă cu o înmulțire cu 2 a valorii inițiale a numărului **n** și în exemplu dat ($n \ll 3$) expresia este echivalentă cu $n * 2 * 2 * 2$.

>> Deplasează la dreapta numărul din stânga operatorului cu un număr de poziții egal cu numărul existent în dreapta operatorului.

Exemplu: rezultatul expresiei

$n \gg 3$

este valoarea numărului n deplasat cu 3 (trei) poziții la dreapta.

Deplasarea spre dreapta completează pozițiile rămase libere cu o extensie a semnului (dacă numărul este pozitiv, completarea se face cu zero; dacă numărul este negativ completarea se face cu 1). Deplasarea la dreapta cu un bit (o poziție) este echivalentă cu o împărțire la 2 a numărului inițial. Astfel în exemplul considerat expresia $n \gg 3$ este echivalentă cu expresia $n/(2*2*2)$.

În cazul în care considerăm n fiind un întreg reprezentat pe 2 octeți (de tip short int) și a cărei valoare este 38, reprezentarea sa internă este următoarea:

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

După deplasarea cu trei poziții la stânga, conținutul celor doi octeți asociați variabilei n are următoarea reprezentare binară:

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

Rezultatul deplasării spre dreapta cu trei poziții a valorii inițiale a variabilei n are următoarea reprezentare binară:

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

Operatorul & (&)

Lucrează după următorul tabel de adevăr:

&	0	1
0	0	0
1	0	1

și este folosit pentru a pune pe zero (a reseta) anumiți biți din variabila luată în considerare (biții respectivi „se șterg”). Datorită acestui mod de lucru, folosirea operatorului $\&$ permite scoaterea în evidență a biților care ne interesează.

Exemple:

a). $c = n \& 0100$;

pune pe zero toți biții lui **n** mai puțin bitul 6 (numerotarea începe de la 0 (zero) și din dreapta) care rămâne la valoarea inițială.

$$n = 67_{10} = 2^6 + 2^1 + 2^0 = 01000011_2 = 0103_8$$

$$c = n \& 0100_8 = 0103_8 \& 0100_8 = 0100_8$$

$$\text{b). } c = n \& 0177400_8 = n \& \underbrace{11111111}_{\text{octet superior}} \underbrace{00000000}_{\text{octet inferior}}_2;$$

face octetul inferior egal cu zero, cel superior rămâne neschimbat;

$$\text{c). } c = n \& 0377_8 = n \& \underbrace{00000000}_{\text{octet superior}} \underbrace{11111111}_{\text{octet inferior}}_2;$$

face octetul superior egal cu zero, cel inferior rămânând la fel ca în **n**.

Operatorul | (SAU)

Lucrează după următorul tabel de adevăr:

	0	1
0	0	1
1	1	1

și este folosit pentru a pune pe 1 (unu) (a seta) toți biții care au valoarea 1 în "mască".

Exemplu:

$$z = x | \text{MASK};$$

$$x = 66_{10}; \text{MASK} = 011_8;$$

$$z = 66_{10} | 011_8 = 0102_8 | 011_8 = 0113_8 = 75_{10}$$

$$x = 66_{10}; \text{MASK} = 077_8;$$

$$z = 66_{10} | 077_8 = 0102_8 | 077_8 = 0177_8 = 127_{10}$$

$$x = 66_{10}; \text{MASK} = 68_{10};$$

$$z = 66_{10} | 68_{10} = 0102_8 | 0104_8 = 01000010_2 | 01000100_2 = 01000110_2 = 0106_8 = 70_{10}$$

Operatorii pe biți **&** și **|** sunt diferiți de operatorii logici **&&** și **||**. Operatorii logici implică o evaluare de la stânga la dreapta a valorii de adevăr a expresiilor implicate, pe când operatorii care lucrează pe bit execută operațiile ca în algebra Booleană.

Exemplu: $x = 1; y = 2 \rightarrow x \& y = 0$, pe când $x \&\& y = 1$.

Operatorul ^ (SAU EXCLUSIV – XOR)

Lucrează în conformitate cu tabelul de adevăr următor:

$\wedge$	0	1
0	0	1
1	1	0

Operatorul unar ~

Aplicarea acestui operator are ca rezultat complementul față de 1 al operandului (biții egali cu zero devin unu și cei egali cu unu devin zero).

Exemplu, în instrucțiunea:

$x = x \& \sim 077;$

este folosit pentru complementarea valorii octale 077 și astfel putem realiza resetarea (punerea pe zero) a ultimilor 6 biți ai lui x.

Exemple:

a). $x = 66_{10}$ și $\sim 077 = 111111111000000 = 177700_8$, deci

$x = 66_{10} \& \sim 077 = 0102_8 \& 177700_8 = 0100_8$

b). $\sim 1234 = \sim 0000010011010010_2 = 1111101100101101_2 = 0175455_8$

$\sim -1234 = \sim 1111101100101110_2 = 0000010011010001_2 = 02321_8$

$\sim \sim 1234 = \sim 1111101100101101_2 = 0000010011010011_2 = 02323_8$

TEMA

În cele ce urmează se vor folosi numai operatori de lucru pe bit pentru toate operațiile cerute.

Se consideră că bitul cel mai semnificativ al unui număr este bitul din stânga, iar cel mai puțin semnificativ este cel din dreapta.

Problema 1

Să se scrie un program care folosește numai operatorii de lucru la nivel de bit pentru a realiza „împachetarea” unei date calendaristice într-un întreg de doi octeți (astfel biții 0-4 reprezintă ziua, biții 5-8 reprezintă luna, biții 9-15 reprezintă anul numărat de la 1900) și afișarea reprezentării binare a rezultatului.

ATENȚIE: NU se folosesc vectori.

Data se citește sub forma: **zz/ll/aaaa**.

Se vor scrie obligatoriu funcții pentru: împachetare dată, afișare bit cu bit (afișarea se va face astfel încât bitul cel mai semnificativ să apară în partea stângă a reprezentării).

Problema 2

Să se scrie un program care realizează rotirea la dreapta sau la stânga (în funcție de opțiunea utilizatorului) cu un anumit număr de biți a unui număr întreg lung.

Se vor scrie funcții de rotire dreapta, rotire stânga, și afișare bit cu bit (afișarea se va face astfel încât bitul cel mai semnificativ să apară în partea stângă a reprezentării).

Problema 3

Cea mai simplă verificare a corectitudinii transmiterii unui șir de biți este să ne asigurăm că numărul biților egali cu 1 din șirul de biți este par sau impar. Pentru a realiza acest lucru se atașează respectivului șir de biți un bit numit bit de paritate. Folosirea biților de paritate este cea mai simplă formă de "cod de detectare a erorilor" folosit pentru verificarea corectitudinii transmiterii unui mesaj în formă binară.

Sunt două variante de biți de paritate: bit de paritate pară și bit de paritate impară. Când se utilizează paritatea pară, bitul de paritate este setat pe 1 dacă numărul de biți egali cu 1 dintr-un șir de biți (neincluzând bitul de paritate) este impar astfel încât numărul total de biți (incluzând bitul de paritate) să fie par. Când se utilizează paritatea impară, bitul de paritate este setat pe 1 dacă numărul de biți egali cu 1 dintr-un șir de biți (neincluzând bitul de paritate) este par astfel încât numărul total de biți (incluzând bitul de paritate) să fie impar.

Paritatea pară este un caz particular de cod CRC (Cyclic Redundancy Check) care este un cod de detectare a erorilor.

Să se scrie un program care determină bitul de paritate pară asociat unui număr întreg lung citit de la tastatură și îl înscrie în bitul cel mai semnificativ (bitul de paritate va fi 1 sau 0 după cum numărul biților egali cu 1 din ceilalți 31 de biți este impar sau par). De asemenea, operatorii de lucru pe bit se vor folosi pentru a determina dacă un număr este par sau nu (cum?). Se va afișa numărul inițial și reprezentarea binară a numărului cu paritatea setată.

Problema 4

Să se scrie o funcție care folosind numai operatori de lucru pe bit calculează valoarea în octal a unui număr întreg primit ca parametru și o depune într-un șir de caractere primit ca al doilea parametru. Numărul octal este reprezentat într-un șir de

caractere (care este al doilea parametru al funcției) și este precedat de caracterul 0. Se va ține seama de faptul că o cifră octală poate fi reprezentată pe 3 biți.

Să se scrie o funcție care folosind numai operatori de lucru pe bit calculează valoarea în hexazecimal a unui număr întreg primit ca parametru și o depune într-un șir de caractere primit ca al doilea parametru. Numărul hexazecimal este reprezentat într-un șir de caractere (care este al doilea parametru al funcției) și este precedat de caracterele 0x. Se va ține seama de faptul că o cifră hexazecimală poate fi reprezentată pe 4 biți.

Să se scrie un program care citește de la tastatură un întreg fără semn și folosind cele două funcții de mai sus, afișează reprezentarea hexazecimală sau octală a numărului. Opțiunea de afișare este dată de utilizator.

Problema 5

Să se scrie o funcție care are ca parametri un număr întreg și un număr real și returnează un număr real și care realizează ridicarea unui număr oarecare la o putere întreagă, folosind următorul algoritm cu aplicarea operatorilor de lucru pe bit:

Observație: orice număr întreg se poate exprima în baza 2.

De exemplu $23_{10} = 10111_2 = 2^4 + 2^2 + 2^1 + 2^0$.

Pentru ridicarea unui număr a la puterea 23 se poate face următorul calcul:

$$a^{23} = a^{(16 + 4 + 2 + 1)} = a^{16} \cdot a^4 \cdot a^2 \cdot a^1$$

Ca urmare, pentru ridicarea unui număr la o putere întreagă trebuie să se genereze un șir care va include valorile a având ca exponenți puteri ale lui 2, adică trebuie să se genereze următorul șir:

$$a, a^2, a^4, a^8, a^{16}, a^{32}, \dots$$

Termenii acestui șir se generează printr-un număr relativ mic de operații, astfel:

$$f = a;$$

$$f = f * f;$$

Ținând cont de observația de mai sus, a^n se poate calcula făcând apel la termenii acestui șir, alegându-i numai pe acea ai căror exponenți însumați dau puterea n . Pentru aceasta se face operația $n \& 1$, și dacă rezultatul este 1, primul termen este a . Se deplasează n spre dreapta cu o poziție. Dacă $n \& 1 = 1$, se selectează a^2 , s.a.m.d.

Să se scrie un program care citește un număr real și un număr întreg și afișează rezultatul ridicării numărului real la numărul întreg.

Problema 6

Să se scrie un program care determină și afișează toate numerele prime inferioare unui număr dat (prestabilit), folosind ciurul lui Eratostene. Metoda constă din eliminarea succesivă a numerelor neprime prin parcurgerea următorilor pași:

1. Se elimina numărul 1 (prin definiție nu este prim).

2. Apoi se caută (pornind de la ultimul număr prim considerat - inițial 1) primul număr neeliminat (acesta este prim) și se elimina toți multiplii săi (din intervalul 1-n).

3. Se repetă pasul 2 până când numărul prim considerat este mai mare decât $\sqrt{n}$ adică radical din n. Funcția `sqrt` are prototipul în `math.h`.

Pentru rezolvare nu se vor folosi vectori, fiecărui număr natural fiindu-i asociat un bit a cărei valoare este 1 dacă numărul este prim și 0 dacă numărul nu este prim.

Problema 7

Să se scrie o funcție care, folosind numai operatori de lucru pe bit, verifică dacă un număr natural este o putere a lui 2 (un număr natural este o putere a lui 2 dacă, în reprezentare binară, numărul are exact un bit egal cu 1). Funcția primește ca parametru numărul de analizat și returnează 1 sau 0 după cum numărul este sau nu putere a lui 2. (NU se vor folosi operatorii aritmetici `/` sau `%`).

Să se scrie o funcție care, folosind numai operatori de lucru pe bit, inversează cei doi octeți ai unui număr întreg reprezentat pe doi octeți. Funcția primește ca parametru un număr întreg și returnează numărul cu octeții inversați. (NU se vor folosi operatorii aritmetici `/` sau `%`).

Să se scrie o funcție care primește ca parametri un vector de întregi și dimensiunea acestui vector. Funcția asociază fiecărui element din vector un bit dintr-un întreg și pune bitul respectiv pe 1 dacă elementul are o valoare pară și pe 0 dacă elementul respectiv are o valoare impară și returnează numărul astfel construit. (NU se vor folosi operatorii aritmetici `+`, `-`, `*`, `/` sau `%`).

Să se scrie un program care face una din prelucrările de mai sus, în funcție de alegerea utilizatorului.