Advanced Data Structure

19. Bit Manipulation

Handwritten by pankaj kumar

Bit Manipulation

Date

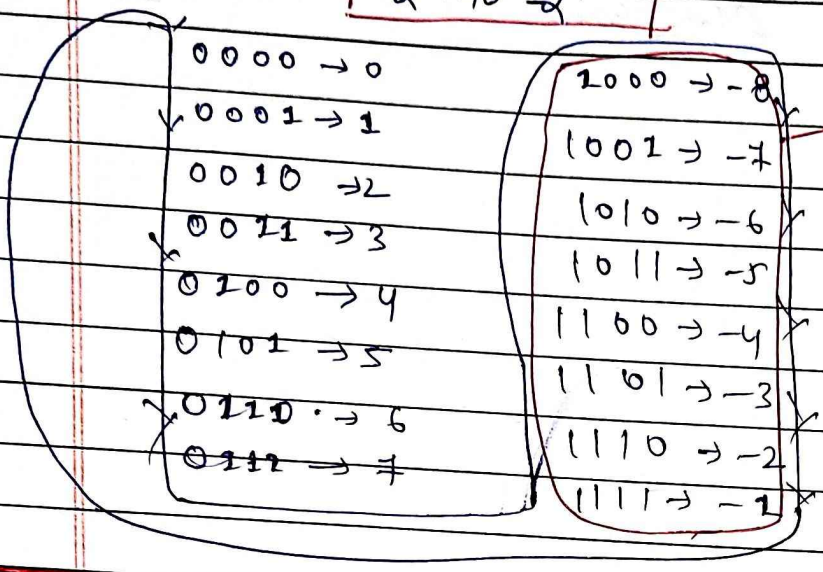
Page No. 295.

Date
Page No. 296

① Introduction

(*) let we have a data type which store 1 byte then it can store the value -8 to 7 i.e -2^3 to 2^3-1

in general -2^n to 2^n-1



→ when msB is 1 then,
→ 2's complement
→ convert to decimal
→ - sign,

ex:- (1010) ← in machine
0101 → 1's
+1
0110 → 2's
-6 → -ve sign.
↓ show to user

(*) Out of range

- in above data type if we will store no. out of range i.e if we will store 12

$(12)_{10} - (1100)_2$

i.e 1100 = -4

hence it will store -4

2	12
2	6 — 0
2	3 — 0
2	1 — 1
	0 — 1

ex:- if we will store 16

2	16
2	8 → 0
2	4 → 0
2	2 → 0
2	1 → 0
	0 → 1

$(16)_{10} = (10000)_2$

So, it will store only (0000) which is equal to 0.

(*) When user will store no. in machine (Decimal to Binary)

i.e +ve
→ convert to binary
→ Fit in bits

-ve
→ leave to sign
→ convert to binary
→ Fit in bits
→ 2's C

Number store in machine showing to user (Binary to Decimal)

msB = 0
→ convert to decimal
→ give + sign

msB = 1
→ 2's C
→ convert to decimal
→ - sign.

Date

Page No. 297

Note:- (1) Store -7 in memory.

-7 → binary → 0111 → convert to 2's c

1000
+1
1001 store in memory

(2) what is value of 1111

since here MSB = 1 ∴ it is -ve number.

so, 1111
1's
0000 → 0000 (2's) +1 → 0001 . i.e in decimal (1)

(1) with -ve sign

(2) Bit wise operator

*	|	&	^	<<	>>	>>>
	or	and	xor	left shift	right shift	for (triple right shift)

~x 1's complement

-x 2's complement.

(*) or (|)

0	1	1
1	1	1
0	0	0
1	0	1

a|1=1
a|0=a

use for switch bits on

(*) & (and)

0&0=0
1&0=0
a&0=0
a&1=a
0&1=0
1&1=1

0's

use for switch bits off

(*) ^ (xor)

toggle.

a^1 = ~a → 1^1=0, 0^1=1
a^0 = a → 1^0=1, 0^0=1

use for toggle

(*) << (left shift)

x = 00101011

x<<3 = 01011000

(*) >> (right shift)

y = 10010110

y>>3 = 11110100

same as MSB

(*) >>> (for)

if y = 10100110

y>>>3 = 00010100

always takes '0'

On (or)	Off (and)	toggle (xor)	Check (any)
x = 1011 * 101	x = 101 * 0101	x = 101 * 0101	x = 1101 * 0011
mask = 0000 | 000	mask = 1111 0 1111	mask = 0001 0000	y = 110 * 0011
x | mask = 1011 | 101	x & mask = 101 0 0101	x ^ mask = 101 0 0101	Using (f)
Note:- to create mask.	Note:- to create mask		mask = 000 1 0011
00000001 << 3 (1<<3)	~(1<<4)		x & mask = 0001 0000
	000 0000 1 <<4		(hence it is on)
0 → x	~(000 10000)	0 → x	x & mask = 0000 0000
1 → ✓	= 11101111	1 → ✓	(hence it is off)
	0 → ✓		Using (1)
	1 → x		

(3) Basic of Bit manipulation (on, off, toggle, check)

you are given a number n.

→ print the number (i) after setting ith bit
(ii) after unsetting jth bit
(iii) after tossing kth bit

→ also, check if its mth bith is on or off. print 'true' if on else 'false'

example :-

I/P	O/P
57	57
3	49
3	
3	49
3	true

57 = 111*0001

after setting same

after unsetting 4g

after tossing 4g

2 | 57
2 | 28 → 1
2 | 14 → 0
2 | 7 → 0
2 | 3 → 1
2 | 1 → 1
0 → 1

```
void checkbit(int n, int i, int j, int k, int m){
    int onmask = (1<<i);
    int offmask = ~(1<<j);
    int tmask = (1<<k);
    int cmask = (1<<m);
    cout << n|onmask << endl;
    cout << n&offmask << endl;
    cout << n^tmask << endl;
    cout << ((n&cmask)==0 ? false : true);
}
```

(4) Right most set bit (RSB)

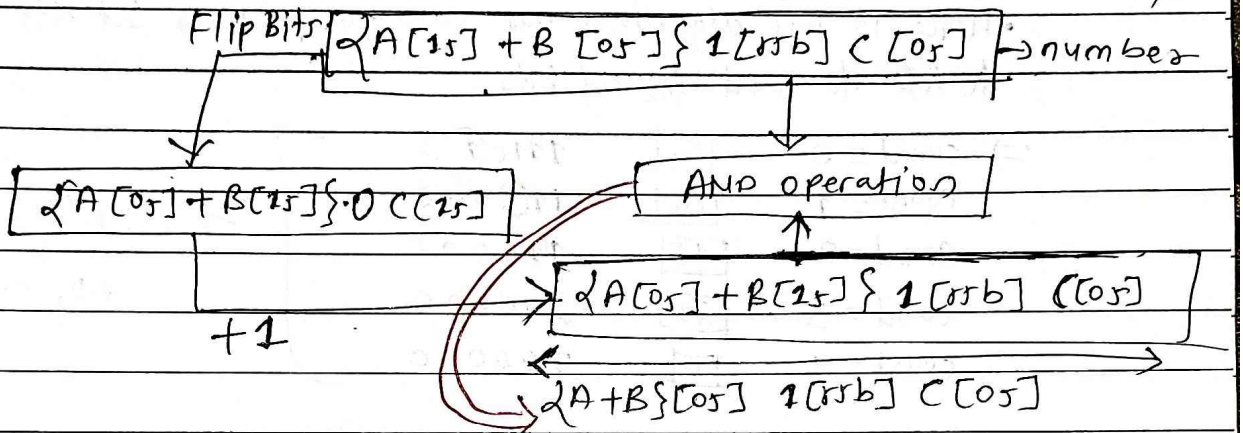
- you are given a number
- you have to print the right-most set bit mask.

I/P:- 58

O/P:- 000010 (2) :- 10 (in binary)

Approach :-

58 = [1 1 1 0 1 0], answer 2= [0 0 0 0 1 0] as right most set bit for n is at 1st index



2's C
- Step 1:- flip bit of number store into mark ⇒ (000101)
- Step 2:- add 1 to get our desired mark ⇒ (000110)
- Step 3:- AND operation number and mark ⇒ (000110) & (111010) = 000010

i.e `n & n+1` or `n & -n`

code :-

```
void RSB(int n){
    int rsbm = n & -n;
    cout<< rsbm<<endl;
}
```

(5) Kernighans Algorithm:

- you are given a number n.
- you have to count the number of set bits in the given number.

ex:- I/P: 58 (111010)
O/P: 4

Approach :-

Date
Page No. 300

① traverse over the entire 32 bits and check if it's set or not.

TC: 32 * constant

② Kernighan's Algorithm

- by using RSB mask.
- so with every iteration, we calculate the RSB mask of the number and then subtract it from the number until the number become zero.
- Hence in this algorithm our code jumps over set bits from right to left to count the set bits.

⇒ count = 0 [58] 1110(1)0
count = 1 [56] 11(1)000
count = 2 [48] (1)00000
count = 3 [32] (1)00000
count = 4 [0] 000000

TC: O(k) → no. of set bits.

Code:-

```
int count = 0;
while(n != 0){
    int rsbm = n & -n;
    n -= rsbm;
    count++;
}
```