ĐẠI HỌC QUỐC GIA THÀNH PHỐ HỒ CHÍ MINH TRƯỜNG ĐẠI HỌC CÔNG NGHỆ THÔNG TIN

KHOA MẠNG MÁYTÍNH VÀ TRUYỀN THÔNG

BÁO CÁO: LAB01– BIT MANIPULATION

MÔN HỌC: LẬP TRÌNH HỆ THỐNG

Giảng viên thực hành: ThS. ĐỖ THỊ HƯƠNG LAN

Lớp: NT209.Q12.ANTT

Sinh viên thực hiện: - Nguyễn Minh Hiếu (24520506)

-Huỳnh Như Hoàng (24520535)

Mục lục

Bài làm

1.1 BitOr

Mục tiêu: Thực hiện phép OR giữa hai số nguyên, chỉ được sử dụng & và ~.

Đáp án:

```
int bitOr(int x, int y)
{
   return ~(~x & ~y); // Sử dụng định luật De Morgan
}
```

Ý tưởng : sử dụng định luật de morgan: A And B có thể biến đổi thành not (not A v not B) vì vậy có thể thay thế and bằng not và or

Test case 1: số dương

Ta có x = 5 và y = 3

- 5 = 0b0101 → ~5 = 0xFFFFFFFA
- 3 = 0b0011 → ~3 = 0xFFFFFFFC
- ~x & ~y = 0xFFFFFFFA & 0xFFFFFFFC = 0xFFFFFFF8
- ~(0xFFFFFFF8) = 0x00000007 → 7

Test case 2: số âm

-4 = 0xFFFFFFFC, ~ = 0x00000003

-8 = 0xFFFFFFF8, ~ = 0x00000007

-& = 0x00000003

-~(0x00000003) = 0xFFFFFFFC → -4

1.2 negative

Mục tiêu: Tính giá trị-x mà không dùng dấu trừ.

Đáp án code:

```
int negative(int x)
{
   return ~x + 1;  // Sử dụng bù 2
}
```

Ý tưởng: Sử dụng công thức bù 2 của 1 số : Đảo bit sau đó cộng 1

Test Case 1: x = -1

-1 = 0xFFFFFFFF

-~(-1) = 0x00000000

-+1 = 0x00000001 → 1

negative(-1) = 1 → đúng

Test Case 2: x = 5

-~5 = 0xFFFFFFFA

-+1 = 0xFFFFFFFB → -5

negative(5) = -5 → đúng

1.3getHexcha

Mục tiêu: Trả về ký tự hex thứ n của x (từ phải sang trái, 0–7).

Đáp án code:

```
nt getHexcha(int x, int n)
{
   return (x & (0xf << (n << 2))) >> (n << 2);
   /*
   B1: Tìm mask: Sử dụng 0xf=1111 dịch trái với n*4.
   B2: and x với mask -> ta được hexchar ở vị trí n.
```

B3: dịch phải lại n lần ta được hexchar.

*/}

Ý tưởng: tìm mask = 0xf = 1111 sau đó dịch trái n*4(Dịch trái nó để tạo **mặt nạ (mask)** cho nibble thứ n), x và **lọc ra nibble thứ n** bằng AND với mask, Sau khi lọc, dịch phải về lại vị trí 0 để lấy giá trị nibble dưới dạng số.

Testcase 1: x = 0x12345678, n = 0 // Lấy nibble thứ 0 (bit 0–3).

- n << 2 = 0
- 0xf << 0 = 0x0000000F
- x & 0xF = 0x12345678 & 0x0000000F = 0x00000008
- >> 0 = 0x08

Testcase2: x = 0x12345678, n = 1 // Lấy nibble thứ 1 (bit 4–7)

- n << 2 = 4
- 0xf << 4 = 0x000000F0
- x & 0xF0 = 0x70
- 0x70 >> 4 = 0x07

1.4 flipbyte

Mục tiêu: Lật byte thứ n của số nguyên x (0–3, từ phải sang trái).

Đáp án code:

return (x ^ (0xff << (n << 3)));

/*

B1: Tìm mask: 0xff=11111111 dịch trái với 8*n.

B2: Lấy mask xor với x -> ta được giá trị flipbyte

Giải thích: 1 số khi xor 0xff -> flip các bit.

1 số khi xor 0 -> thành chính nó.

*/

Ý tưởng: tìm mask là 0xff sau đó dịch trái với 8*n(n << 3) để lấy được byte vị trí cần lấy sau đó xor với x để được giá trị flipbyte

Testcase 1:

x = 0x12345678

n = 0

- (n << 3) = 0 << 3 = 0

- mask = 0xff << 0 = 0x000000ff

- x ^ mask = 0x12345678 ^ 0x000000ff = 0x12345687

Testcase 2:

x = 0x12345678

n = 0

-(n << 3) = 0 << 3 = 0

- mask = 0xff << 0 = 0x000000ff

- x ^ mask = 0x12345678 ^ 0x000000ff = 0x12345687

1.5. divpw2

Mục tiêu: Tính x / 2n với n có thể âm hoặc dương.

Đáp án code:

```
int divpw2av(int x, int n)
{
    return ((x >> n) & ~(n >> 31)) | (x << (~n + 1)) & (n >> 31);
}
```

Chia thành 2 trường hợp:

TH 1: chia cho 2^n với n dương.

TH 2: chia cho 2^n với n âm.

Rồi sử dụng biểu thức Shannon để chọn 1 trong 2 trường hợp với giá trị chọn là n âm hoặc dương.

Biểu thức Shannon: ac + b(~c).

Test case:

1. **x = 16.**

   n = 2.

   n >> 31 = 0

   ~(n >> 31) = 1

   (x >> n) & 1 = 4

   (x << (~n +1)) & 0 = 0

4 | 0 = 4

2. **x = 20**

   n = -2

   n >> 31 = 1

   ~(n >> 31) = 0

   (x >> n) & 0 = 5

   (x << (~n +1)) & 1 = 0

   0 | 5 = 5

2.1 equal

Mục tiêu : Kiểm tra hai số x và y có bằng nhau không.

Đáp án code: int isEqual(int x, int y)

```
{
   return !(x ^ y);
   /*
   Hai số bằng nhau khi xor với nhau sẽ bằng 0.
   */
}
```

Ý tưởng: Hai số bằng nhau khi xor với nhau sẽ bằng 0 vì vậy sẽ phủ định lại để ra đáp án đề mong muốn.

Testcase 1:

x = 5, y = 5

x ^ y = 5 ^ 5 = 0
!(0) = 1 → Hai số bằng nhau

Testcase 2:

x = 10, y = 12
x ^ y = 10 ^ 12 = 6
!(6) = 0 → Hai số không bằng nhau

2.2 is16x

Mục tiêu: Kiểm tra xem x có chia hết cho 16 không.

Đáp án code:

int is16x(int x)

{

return !(((x >> 4) << 4) ^ x);

/*

Một số nguyên khi chia hết cho 16 sẽ có 4 bit đầu là 0 (ex.0x********0000).

Dịch các bit sang trái 4 lần sau đó dịch phải lại 4 lần nếu hai số lúc đầu và lúc sau khác nhau

Thì số đó không chia hết cho 16 ngược lại chia hết cho 16

```
  */
}
```

Ý tưởng: Một số nguyên khi chia hết cho 16 thì sẽ có 4 bit đầu bằng 0 vì vậy ta dịch trái 4 bit sau đó dịch phải lại 4 bit nếu hai số khác nhau thì không chia hết cho 16

Testcase 1:

x = 16

Nhị phân: 0001 0000

(x >> 4) = 1, (1 << 4) = 16 → bằng x

Testcase 2:

x = 17

-Nhị phân: 0001 0001

-(x >> 4) = 1, (1 << 4) = 16, 16 ^ 17 = 1

2.3 isPositive

Mục tiêu: Kiểm tra xem x có phải số dương không.

Đáp án code:

```
int isPositive(int x)
{
   return (!((x >> 31) & 1)) ^ !x;
   /*
```

Chia thành 2 giai đoạn:

Giai đoạn 1: - Thử xem số đó là >=0 hay < 0.

- nếu >= 0 -> 0, ngược lại -> 1.

Giai đoạn 2: - Thử xem số có phải 0.

+ Ở giai đoạn 1 nếu số đó âm

*/

}

Ý tưởng là dịch phải 31 bit để lấy bit dấu sau đó and 1 rồi not nó lại để coi là số âm hay dương sau đó sẽ xor với not của x để xử lí trường hợp nó bằng 0.

Test case 1:

X = 0

- X >> 31 = 0
- 0 & 1 = 0
- !0 = 1
- 1 ^ !0 = 0

Đáp án là false vì 0 không phải là số dương

Testcase 2:

X = -1

- X >> 31 = 1
- 1 & 1 = 1
- !1 = 0
- 0 ^ !-1 = 0

Đáp án là false

Testcase 3:

X = 1

- X >> 31 = 0
- 0 & 1 = 0
- !0 = 1
- 1 ^ !1 = 1

  Đáp án là true

2.4 isGE2n

Mục tiêu: Kiểm tra xem x có lớn hơn hoặc bằng 2n hay không.

Đáp án code:

```
int isGE2n(int x, int n)
{
   return !((x + ((~(1 << n)) + 1)) >> 31);
   /*
   Để check 1 số x có lớn hơn 2^n không ta thực hiện x - 2^n .
   Rồi check giấu của kết quả lấy được.
   Nếu âm -> return 0 (nhỏ hơn).
   Nếu dương -> return 1 (lớn hơn).
   */
}
```

Ý tưởng: vì không dc sài 2^n vì vậy sẽ dùng 1 << n, sau đó bù 2 lại để là -2^n, x – 2^n kiểm tra xem là lớn hơn bằng 0 hay nhỏ

hơn 0 vì vậy dịch phải 31 bit để tìm dấu biểu thức sau đó not lại để ra đúng kết quả của đề bài

Test case 1:

X = 3 và n = 1

- 1 << 1 = 2
- Bù 2 của 2 là -2
- 3 – 2 = 1
- 1 >> 31 = 0
- Vì vậy !0 = 1

Đáp án là true

Testcase 2:

X = 1 và n = 2

- 1 << 2 = 4
- Bù 2 của 4 là -4
- 1 – 4 = -3
- -3 >> 31 =1
- Vì vậy !1 = 0
  Đáp án là false