Auxiliatura INF-143 "A"
Manipulación de Bits

Univ. Miguel Angel Quispe Mamani
Universidad Mayor de San Andrés
Carrera de Informática

I/2022

1 Introducción

El sistema de numeracion decimal (Base 10) esta compuesto por 10 digitos: 0, 1, 2, 3, 4, 5, 6, 7, 8, 9 y todos los números estan compuestos para la combinación de estos, por otro lado el sistema Binario(Base 2) esta compuesto unicamente por dos digitos: 0 y 1, entonces un bit representa uno de estos valores 0 o 1, se puede interpretar como un foco que tiene dos estados: Encendido(1) y Apagado(0).

Nota: *Los datos que usamos en nuestros programas, internamente, estan representados en Binario, por ende muchas veces se requiere hacer operaciones a nivel bits.*

2 Operadores Bit a Bit

Una operación bit a bit opera sobre números binarios a nivel de sus bits individuales.

2.1 NOT

El operador NOT bit a bit, es una operación unaria que realiza la negación lógica en cada bit, invirtiendo los bits del número, de tal manera que los ceros se convierten en 1 y viceversa.

a	**~a**
0	1
1	0

Ejemplo:

```cpp
#include <bits/stdc++.h>
using namespace std;

int main(){
    int a;
    cin >> a;//10 = 1010 en base 2
    a = ~a;//a invertira sus bits, 0101
    cout << a << "\n";
    return 0;
}
```

2.2 AND

El AND bit a bit, toma dos números enteros y realiza la operación AND lógica en cada par correspondiente de bits. El resultado en cada posición es 1 si el bit correspondiente de los dos operandos es 1, y 0 de lo contrario. La operacion AND se representa con el signo "&".

a	b	a & b
0	0	0
0	1	0
1	0	0
1	1	1

Ejemplo:

```cpp
#include <bits/stdc++.h>
using namespace std;

int main(){
    int a, b;
    cin >> a >> b;//13 y 10. En binario 1101 y 1010
    /*
    1101
    1010
    ----
    1000
    */
    cout << (a & b) << "\n";//1000
    return 0;
}
```

2.3 OR

Una operación OR de bit a bit, toma dos números enteros y realiza la operación OR inclusivo en cada par correspondiente de bits. El resultado en cada posición es 0 si el bit correspondiente de los dos operandos es 0, y 1 de lo contrario. La operacion OR se representa con el signo "|".

a	b	a | b
0	0	0
0	1	1
1	0	1
1	1	1

Ejemplo:

```cpp
#include <bits/stdc++.h>
using namespace std;

int main(){
    int a, b;
    cin >> a >> b;//13 y 10. En binario 1101 y 1010
    /*
    1101
    1010
    ----
    1111
    */
    cout << (a | b) << "\n";//1111
    return 0;
}
```

2.4 XOR

El XOR bit a bit, toma dos números enteros y realiza la operación OR exclusivo en cada par correspondiente de bits. El resultado en cada posición es 1 si el par de bits son diferentes y 0 si el par de bits son iguales. La operacion XOR se representa con el signo "^".

a	b	a ^ b
0	0	0
0	1	1
1	0	1
1	1	0

Ejemplo:

```cpp
#include <bits/stdc++.h>
using namespace std;

int main(){
    int a, b;
    cin >> a >> b;//13 y 10. En binario 1101 y 1010
    /*
    1101
    1010
    ----
    0111
    */
    cout << (a ^ b) << "\n";//1111
    return 0;
}
```

3 Operaciones de Desplazamiento

Se tiene dos operaciones de desplazamiento: El desplazamiento a la derecha y el desplazamiento a la izquierda. Al desplazar los bits los espacios faltantes son rellenados con ceros.

3.1 Desplazamiento a la Izquierda

El desplazamiento a la izquierda (left shift) es la operación de mover todos los bits una cantidad determinada de espacios hacia la izquierda. Esta operación esta representado por los signos "<<".
Por cada desplazamiento a la izquierda se agregara un cero a la derecha de todo el número.
Ejemplo:

```cpp
#include <bits/stdc++.h>
using namespace std;

int main(){
    int a;
    cin >> a;//11(Decimal)

    //1011 << 1 = 10110
    cout << (a << 1) << "\n";

    //1011 << 2 = 101100
    cout << (a << 2) << "\n";
    return 0;
}
```

3.2 Desplazamiento a la Derecha

El desplazamiento a la derecha (rigth shift) es la operacion de mover todos los bits una cantidad determinada de espacios hacia la derecha. Esta operacion esta representada por los signos ">>".
Al dezplazar el número hacia la derecha los bits menos significativos (los ultimos) se perderan.
Ejemplo:

```cpp
#include <bits/stdc++.h>
using namespace std;

int main(){
    int a;
    cin >> a;//11(Decimal)

    //1011 >> 1 = 101
    cout << (a >> 1) << "\n";

    //1011 << 2 = 10
    cout << (a >> 2) << "\n";
    return 0;
}
```

4 Aplicaciones

Con todas las operaciones anteriormente vistas podemos hacer muchas cosas interesantes a la hora de programar. Algunas aplicaciones son:

4.1 Encender un bit en una posición

Si queremos encender un bit (hacerlo 1) en alguna posición podemos aplicar un corrimiento a la izquierda de un 1 seguido por una operación OR. Por ejemplo, si queremos encender el bit x de un número a podemos hacer algo como sigue:

```cpp
#include <bits/stdc++.h>
using namespace std;

int main(){
    int a, x;//a es el numero, x el numero de bit
    //los bits se enumeran de derecha a izquierda, iniciando desde 0:
    /*
    Ej: n = 8
    Binario:     1  0  0  0
    Posicion:    3  2  1  0
    */
    cin >> a >> x;
    a = a | (1 << x); // Encendemos el bit x de a
    cout << a << "\n";
    return 0;
}
```

4.2 Apagar un bit en una posición

Si queremos apagar un bit (hacerlo 0) en alguna posición podemos aplicar un corrimiento de un 1 seguido por una operación NOT y una AND. Por ejemplo, si queremos apagar el bit x de un número a podemos hacer:

```cpp
#include <bits/stdc++.h>
using namespace std;

int main(){
    int a, x;//a es el numero, x el numero de bit
    //los bits se enumeran de derecha a izquierda iniciando desde cero
    /*
    Ej: n = 8
    Binario:     1  0  0  0
    Posicion:    3  2  1  0
    */
    cin >> a >> x;
    a = a & ~(1 << x); // Apagamos el bit x de a
    cout << a << "\n";
    return 0;
}
```

4.3 Revisar si un bit está encendido

Podemos saber si un bit x está encendido en una variable a aplicando un corrimiento de un 1 seguido de una operación AND. Por ejemplo:

```cpp
#include <bits/stdc++.h>
using namespace std;

int main(){
    int a, x;//a es el numero, x el numero de bit
    cin >> a >> x;
    if (a & (1 << x)){ // Si esto se cumple entonces el bit es 1
        cout << "Encendido\n";
    }else{
        cout << "Apagado\n";
    }
    return 0;
}
```

4.4 Máscara de Bits

Mascara de bits o BitMask es un algoritmo sencillo que se utiliza para calcular todos los subconjuntos de un conjunto. Este algoritmo tiene complejidad de $O(2^n)$ por lo que su uso se limitara a conjuntos que tengan a lo mucho 20 elementos.
El algoritmo funciona de esta manera, si se tiene n elemenos en un conjunto entonces generara todos los numeros desde 0 hasta 2^n, junto a sus representaciones binarias, se tomara en cuenta las n cifras menos significativas de cada numero en binario y cada columna representara a un elemento del conjunto, entonces si el bit esta encendido tomaremos ese elemento en el subconjunto.

```cpp
#include <bits/stdc++.h>
using namespace std;

int main(){
    int n;
    cin >> n;
    int v[n];
    for(int i = 0; i < n; i++)
        cin >> v[i];
    for(int subset = 0; subset < (1 << n); subset++){//generar numeros de 0 a (2^n - 1)
        cout << "Subconjunto\n";
        for(int i = 0; i < n; i++){
            if(subset & (1 << i)){//verificamos i-esimo bit encendido
                cout << v[i] << " ";
            }
        }
        cout << "\n";
    }
    return 0;
}
```

References

[1] Carvajal G., Castillo R, et al., (2015), *Introducción en C++ a la Programación Competitiva*, La Paz - Bolivia, Hivos

[2] Anónimo, (2020), *Operaciones con bits*, include <poetry>, https://www.include-poetry.com/Code/C++/Estructuras/Operaciones-con-bits/