Auxiliar 2 - Bits

Profesores: Luis Mateu
Alexandra Ibarra
Rodrigo Urrea
Auxiliares: Luciano Márquez
Tomás Vergara C.

Resumen

Hexadecimal y Binario Habitualmente para hablar de números utilizamos el sistema decimal, utilizando un total de 10 cifras (0, 1, 2, 3, 4, 5, 6, 7, 8, 9). En el contexto de las ciencias de la computación, es frecuente utilizar otros sistemas para los números, hexadecimal, octal y binario.
Para el binario, se representan los números en base 2, utilizando únicamente 0 y 1, para transformarlo en una expresión decimal, se debe realizar la potencia de dos según la posición de la cifra y sumar los resultados, por ejemplo 1010 en binario, $1 \cdot 2^3 + 0 \cdot 2^2 + 1 \cdot 2^1 + 0 \cdot 2^0 = 1 \cdot 2^3 + 1 \cdot 2^1 = 8 + 2 = 10$. De decimal a binario se puede pasar, simplemente dividiendo por 2 guardando el resto en una "pila", luego se extraen de la pila para formar el binario, por ejemplo $14 \Rightarrow 14/2 = 7$, resto 0, $7/2 = 3$, resto 1, $3/2 = 1$, resto 1, $1/2 = 0$, resto 1. La pila se guardó como 0, 1, 1, 1 y entonces el número en binario resulta 1110. En C se representan los binarios utilizando el prefijo $0b$, así que en C sería $0b1110$ (versión C2X).
En Hexadecimal, se utiliza la base 16, aunque como solo tenemos 10 cifras para la representación numérica, se utilizan también las primeras 6 letras del abecedario, utilizando los símbolos $\{0, 1, 2, 3, 4, 5, 6, 7, 8, 9, A, B, C, D, E, F\}$. Luego, cada cifra su valor se obtiene a partir de la potencia de 16 con el exponente según su posición, de esta forma, $0xA5F0 = 10 \cdot 16^3 + 5 \cdot 16^2 + 15 \cdot 16^1 + 0 \cdot 16^0 = 10 \cdot 16^3 + 5 \cdot 16^2 + 15 \cdot 16 = 10 \cdot 4096 + 5 \cdot 256 + 15 \cdot 16 = 40960 + 1280 + 240 = 42480$. Lo interesante para este ramo, es que la representación hexadecimal es muy útil también para números binarios, esto pues con solo una cifra hexadecimal es capaz de tener el rango entre 0 y 15, es decir hasta 4 cifras en binario (tabla 1).

Hexadecimal	Decimal	Binario	Hexadecimal	Decimal	binario
0	00	0000	1	01	0001
2	02	0010	3	03	0011
4	04	0100	5	05	0101
6	06	0110	7	07	0111
8	08	1000	9	09	1001
A	10	1010	B	11	1011
C	12	1100	D	13	1101
E	14	1110	F	15	1111

Table 1: Representación Hexadecimal, decimal y binaria

En C, se utiliza el prefijo $0x$ para números hexadecimales. Luego para un byte, que tiene 8 cifras binarias, basta utilizar 2 en hexadecimal para su representación, $0b\underbrace{1100}_{C}\underbrace{1001}_{9} = 0xC9$ y así, para un `int` de 4 bytes, se requieren de 8 cifras en hexadecimal.

Función	Símbolo	Significado	Ejemplo
and	&	y lógico bit a bit	011011 & 001110 = 001010
or	|	o lógico bit a bit	011011 | 001110 = 011111
xor	^	o exclusivo bit a bit	011011 ^ 001110 = 010101
shift left	<<	desplazamiento a la izquierda	011011 << 1 = 110110
shift right	>>	desplazamiento a la derecha	011011 >> 1 = 001101
complement	∼	Negación bit a bit	∼011011 = 100100

Table 2: Resumen de operaciones bit a bit, todas se hacen en 1 ciclo.

Tener en cuenta que el desplazamiento a la derecha mantiene el signo del número, es decir, que si el número es negativo, el bit de más a la izquierda (el más significativo) será un 1, si se desplaza a la derecha, el nuevo bit que aparecerá en su lugar también será 1, veamos un ejemplo para números de 5 bits (donde el primer bit representa el signo y los otros 4 bits el número, teniendo un rango entre -16 y 15):

$$10100 >> 2 = 11101 \iff -12 >> 2 = -3 \iff -12/2^2 = -3$$

Ejercicios

P1. Cree una función llamada `bits1` que dado un número n, retorna el número de bits en 1 que tiene n. Como pista, su declaración debería ser: `int bits1(unsigned int n)`.

P2. (P2a C1 Otoño 2014) Programe la siguiente función:

```
int posicionBits(int x, int p, int n);
```

Esta función busca en x el patrón p de n bits, entregando su posición expresada como la posición del bit menos significativo de p en x, o -1 si no se encuentra.

P3. (P2b C1 Otoño 2013) Programe la siguiente función:

```
unsigned repBits(unsigned x, int i, int k, unsigned val);
```

Esta función retorna el resultado de reemplazar k bits en x, a partir del i-ésimo bit de x, por el valor val.

P4. **(Propuesto) Variable-Length Quantity** es una codificación que representa número enteros usando número variable de bytes. Cada byte almacena 7 bits del número (partiendo desde la derecha), y el bit más significativo indica si es byte es el último de la cadena (0) o si el número continúa en el siguiente byte (1). Implemente la función:

```
int decodeVLQ( unsigned int x);
```

la cual recibe un entero sin signo representando un número en formato **VLQ** y retorna su valor decodificado.