Tarea 18

Elementos de ciencias de la computación

Tulio Muñoz Magaña

October 25, 2020

Inciso a) El programa "bitwise_operators.c" está hecho para mostrar el funcionamiento de las operaciones entre cadenas de bits imprimiendo el resultado, toma de ejemplo las cadenas de bits de los números 5 y 9, es decir $\cdots 000101$ y $\cdots 0001001$ respectivamente.

Primero muestra el funcionamiento del operador &, que compara posición por posición las dos cadenas de bits, haciendo la operación lógica AND entre los bits que están en la misma posición respectiva. La operación en este casi solo deja entonces un uno al final, que es la única posición donde ambas cadenas tienen un uno.

Después muestra el funcionamiento del operador |, que hace un OR lógico posición por posición, en este caso regresará una cadena que tiene un 1 en las posiciones donde al menos una de las cadenas tiene un 1, es decir $\cdots 0001101$.

Muestra luego el funcionamiento del operador ^, que es un XOR lógico, que regresa un 1 en las posiciones donde exactamente una de las cadenas tiene un 1, es decir $\cdots 0001100$.

Muestra luego el funcionamiento del operador unario $\sim$ operado a 5, que cambia todos los bits de la cadena por el opuesto, es decir que regresa $111111111111 \cdots 1111010$.

Luego muestra los operadores de corrimiento, haciendo un corrimiento a la izquierda a 5 y mostrando el resultado, y luego un corrimiento a la derecha a 5 y mostrando el resultado, lo que genera las cadenas $\cdots 0001010$

y $\cdots 000010$ respectivamente.

Todas las anteriores cadenas al ser mostradas en pantalla se traducen a su expresión decimal a partir de su expresión en bits, que es la expresión binaria.

Muestra el código luego la técnica que se usa para verificar si una posición específica de una cadena de bits tiene un 1, se hace recorriendo el entero 1 (representado por la cadena $\cdots 000001$) el número de veces necesarias para llevar el 1 de la cadena a la posición que queremos evaluar, realizando después una operación & con la cadena que queremos evaluar, si había un 1 en esa posición, entonces tendremos un 1 en esa posición como resultado y 0 en todas las demás posiciones, es decir que la cadena resultante representa un entero distinto de 0. Si hay un 0 en la posición, habrá 0 en todas las posiciones de la cadena resultante de la operación. Al meter el resultado de esta operación en un if, sabremos que había un 1 si el resultado es distinto de 0, y que no, si el resultado es 0. De esta manera podemos mostrar en pantalla si había un uno en la posición que estábamos evaluando.

inciso c) Cuando hacemos un corrimiento de 3 bits a la izquierda a un número n es como realizar en decimal, la operación

$$res(\frac{n}{2^{29}}) \cdot 8$$

donde res nos da el residuo de la división que hay dentro de los paréntesis. Al hacer la operación, los tres primeros bits de la izquierda se tirarán, que son los bits correspondientes a las potencias 2^{31}, 2^{30} y 2^{29}, por lo que si realizamos la operación a un número nos dará el mismo resultado independientemente de lo que haya en estos 3 primeros bits, solo nos importan los bits del cuarto en adelante, para deshacernos en decimal de estos primeros tres bits calculamos el residuo de dividir n entre 2^{29}, el cual es un número entre 0 y $2^{29} - 1$, el cual está representado justamente por los bits a partir del cuarto. Después de realizada esta operación, el mover los bits a la izquierda es como aumentar en 3 la potencia de cada 2 con coeficiente 1, es decir, si el número que tenemos hasta ahora es $res(\frac{n}{2^{29}}) = 2^{a_1} + 2^{a_2} + \cdots + 2^{a_k}$ con $28 \geq a_1 > a_2 > \cdots > a_k$, entonces la operación corresponde al número

$$2^{a_1+3} + 2^{a_2+3} + \cdots + 2^{a_k+3} = 2^3(2^{a_1} + 2^{a_2} + \cdots + 2^{a_k}) = 8 \cdot res(\frac{n}{2^{29}})$$

Cuando hacemos un corrimiento de 2 bits a la derecha en un unsigned int, es como tomar la parte entera de dividir el número entre 4 en decimal, por lo que si el número original es n, el resultado de la operación es

$$[\frac{n}{4}].$$

Al realizar la operación, los dos bits de hasta derecha se tirarán, por lo que no importa lo que haya en ellos, el resultado es el mismo que si hubiera ceros en esos dos últimos bits. Si hay ceros en estos dos últimos bits, tenemos un número de la forma $2^{a_1} + 2^{a_2} + \cdots + 2^{a_k} = n$ con $a_1 > a_2 > \cdots > a_k \geq 2$. Si corremos todos los unos 2 espacios a la derecha, es como decrecer en 2 la potencia de cada 2, por lo que nos queda el número

$$2^{a_1-2} + 2^{a_2-2} + \cdots + 2^{a_k-2} = 2^{-2}(2^{a_1} + 2^{a_2} + \cdots + 2^{a_k}) = 2^{-2}n = \frac{n}{4}$$

Si alguno de los dos últimos bits o ambos son diferentes de cero, a lo más agregan 3 unidades al número que acabamos de calcular, por lo que al tomar la parte entera de la división entre 4 nos da este número.

Los incisos b y d están en el proyecto Tarea18, la captura de pantalla del funcionamiento de los algoritmos es captura.png