Pontificia Universidad Católica de Chile
Escuela de Ingeniería
Departamento de Ciencia de la Computación

IIC2343 – Arquitectura de Computadores (I/2013)

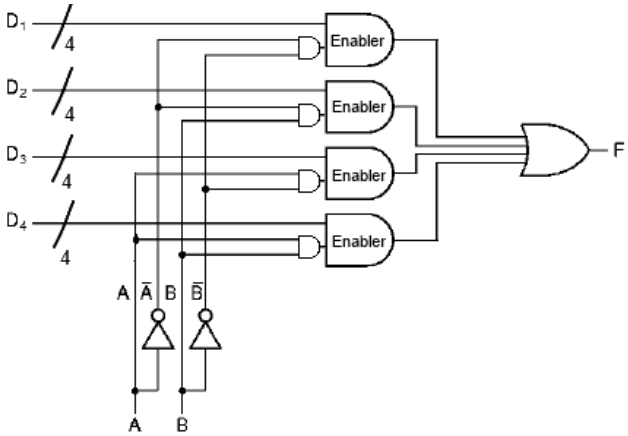
Interrogación 1

Pregunta 1

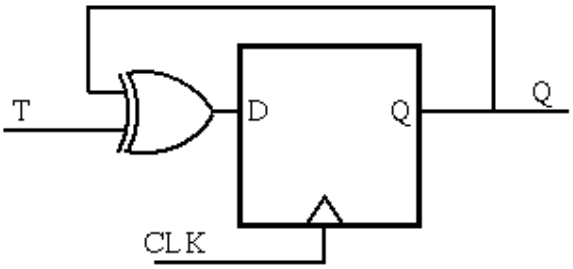
a) ¿Cuál es el valor del número 11000001100000000000000000000000, representado mediante el tipo de dato **float**? **(1 pto.)**
Solución: A simple vista se puede apreciar que el significante es cero y el signo es negativo, por lo que el número tiene la forma $-1 \times 2^{exp-127}$. Luego, como el exponente es igual a 131, el valor del número es $-1 \times 2^{131-127} = -1 \times 2^4 = -16$.

b) ¿Cuál es el mayor número positivo que puede representarse mediante un complemento a 2 de 11 bits? **(1 pto.) (1 pto.)**
Solución: Si se tienen 11 bits disponible, el mayor número positivo será $01111111111 = 2^{10}-1 = 1023$.

c) Diseñe un Mux de 4 entradas de 4 bits cada una. **(1 pto.)**
Solución:



d) Diseñe usando compuertas lógicas y flip-flops **D**, un flip-flop **T**. El comportamiento de este flip-flop consiste en invertir el valor de su salida **Q** si su señal de entrada **T** está en 1 y la señal de control **C** pasa de 0 a 1 (flanco de subida). En cualquier otro caso, la salida **Q** se mantiene igual. **(1 pto.)**
Solución:



e) ¿Cuántas entradas y salidas tiene un display controller de una calculadora que utiliza números de 4 bytes? **(1 pto.)**
Solución: Un display de siete segmentos puede mostrar númerosd de hasta 4 bits. Luego, necesitamos 8 displays para mostrar números de 4 bytes, por lo tanto, el display controller necesita 56 salidas y 32 entradas.

f) ¿Qué secuencia de números enteros es generada al realizar 8 operaciones **rotate left** a un registro de 8 bits que inicialmente almacena el número 79? **(1 pto.)**
Solución: La secuencia generada es la siguiente: -98, 61, 122, -12, -23, -45, -89, 79.

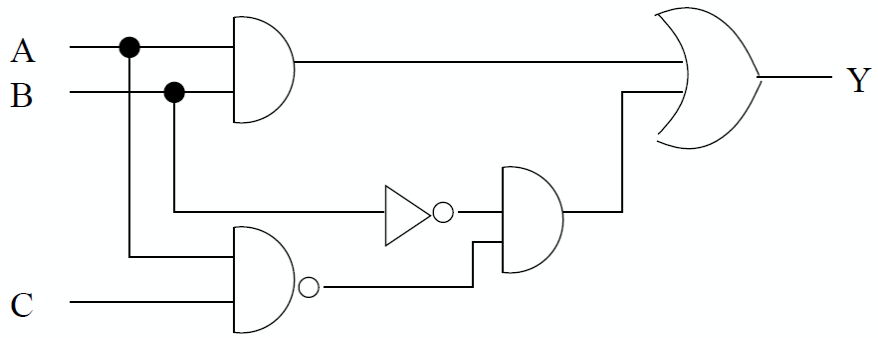
Pregunta 2

a) Considere la siguiente fórmula en lógica booleana:

$$Y = (A \wedge B) \vee \neg(A \wedge C) \wedge \neg B$$

i) Escriba la tabla de verdad correspondiente a la función anterior y diseñe un circuito lógico que la implemente. **(1 pto.)**
Solución: Esta es una de muchas posibles soluciones, dependiendo de la distribución (factorización) utilizada en la fórmula

A	B	C	Y
F	F	F	F
F	F	V	F
F	V	F	V
F	V	V	V
V	F	F	F
V	F	V	V
V	V	F	F
V	V	V	F



ii) A partir de la tabla de verdad, re-escriba la función como conjuntos de **ANDs** de las variables (o sus negaciones), combinados mediante **ORs**. **(1 pto.)**
Hint: $A \oplus B = (A \wedge \neg B) \vee (\neg A \wedge B)$
Solución: De la tabla de verdad, basta tomar los casos en que el valor de Y es V de la siguiente manera:

$$Y = (\neg A \wedge B \wedge \neg C) \vee (\neg A \wedge B \wedge C) \vee (A \wedge \neg B \wedge C)$$

b) Si dos números enteros (con signo), A y B, son sumados (restados) mediante un **adder** (**subtractor**) de n bits, es posible que el resultado sea incorrecto: la suma de dos números positivos puede ser un número negativo, la suma de dos números negativos puede ser un número positivo, etc. Estos errores son llamados *overflow*.
Para detectar automáticamente estas situaciones, se define una variable binaria **overflow** que toma el valor 1 cuando ocurre el error y 0 cuando no ocurre. Esta variable depende de otras

cuatro variables binarias: A_{n-1} (bit más significativo de A), B_{n-1} (bit más significativo de B), S_{n-1} (bit más significativo del resultado de la operación), Op (operación: suma o resta).
Escriba la tabla de verdad para la variable **overflow**, en base a los valores de las otras cuatro variables. **(2 ptos.)**
Solución: Consideraremos los casos de suma y resta de manera separada. Primero, para la suma (Op) tebemos tres casos:

- si los dos números son de signo opuesto, NO puede existir overflow.
- si los dos números son positivos ($A_{n-1} = 0$, $B_{n-1} = 0$) , sólo existe overflow el resultado es negativo ($S_{n-1} = 1$).
- si los dos números son negativos, ($A_{n-1} = 1$, $B_{n-1} = 1$), sólo existe overflow si el resultado no es negativo ($S_{n-1} = 0$).

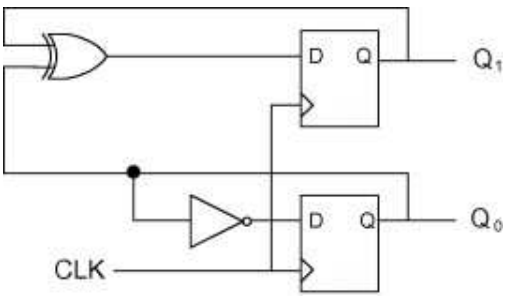
Para la resta el análisis es similar:

- si los dos números son del mismo signo, NO puede existir overflow.
- si A es positivo ($A_{n-1} = 0$) y B negativo ($B_{n-1} = 1$), sólo existe overflow cunado el resultado es negativo ($S_{n-1} = 1$).
- si A es negativo ($A_{n-1} = 1$) y B positivo ($B_{n-1} = 0$), sólo existe overflow si el resultado es positivo ($S_{n-1} = 0$).

Luego, la tabla toma la siguiente forma:

Op	A_{n-1}	B_{n-1}	S_{n-1}	overflow
0	0	0	0	0
0	0	0	1	1
0	0	1	0	0
0	0	1	1	0
0	1	0	0	0
0	1	0	1	0
0	1	1	0	1
0	1	1	1	0
1	0	0	0	0
1	0	0	1	0
1	0	1	0	0
1	0	1	1	1
1	1	0	0	1
1	1	0	1	0
1	1	1	0	0
1	1	1	1	0

c) Diseñe utilizando todos los elementos de circuitos lógicos vistos en clases, un contador secuencial de 2 bits, que se incrementa con cada flanco de subida de la señal de control. **(2 ptos.)**
Solución:



Pregunta 3

Para los siguientes ejercicios, escriba su respuesta en el lenguaje de programación que más le acomode. Deje por escrito claramente cual es su elección.

a) Escriba el código para el método `int[] sumaBinarios(int[] num1, int[] num2)`. Este método recibe dos números binarios de igual largo y retorna la suma. Los número se asumen que serán positivos y que la suma no sobrepasará la cantidad de bits de los sumandos. Los números binarios son representados como arreglos de enteros y cada posición del arreglo podrá tomar un valor 0 o 1 lo cual debe cumplirse también en el retorno. **(3ptos.)**
Solución:

```
public static int[] sumaBinarios(int[] num1, int[] num2)
{
    int[] resultado = new int[num1.length];
    int carry = 0;
    for (int i = num1.Length - 1; i >= 0; i--)
    {
        // Caso 3 unos.
        if (carry == 1 && num1[i] == 1 && num2[i] == 1)
        {
            carry = 1;
            resultado[i] = 1;
        }
        // Caso 2 unos.
        else if ((carry == 1 && num1[i] == 1) ||
                 (carry == 1 && num2[i] == 1) ||
                 (num1[i] == 1 && num2[i] == 1))
        {
            carry = 1;
            resultado[i] = 0;
        }
        // Caso 3 ceros.
        else if (num1[i] == 0 && num2[i] == 0 && carry == 0)
        {
            carry = 0;
            resultado[i] = 0;
        }
        // Caso 1 uno.
        else
        {
            carry = 0;
            resultado[i] = 1;
        }
    }
    return resultado;
}
```

b) Escriba el código para el método `float binarioAFloat(int[] single)`, el cual convierte un número binario (representado como arreglo de enteros) en su valor tipo `float`. El número binario representa un número del tipo «single precision floating point» (float de 32 bits) según las características del estándar IEEE754 visto en clases. En su conversión **no** debe considerar los casos especiales mencionados en los apuntes y en clases. **(3ptos.)**
Solución:

```
public static float BinarioAFloat(int[] single)
{
    //Posicion 0: signo
    //Posicion 1-8 = exponente
    //Posicion 9-31 = significante
    int exp = -127;
    float significante = 1;
    float resultado;
    int signo = single[0]*-1;

    for (int i = 7; i >= 0; i--)
    {
        exp += single[8-i] * (int)Math.pow(2, i);
    }
    for (int i = 9; i < 32; i++)
    {
        significante += single[i]*(float)Math.pow(2, -(i - 8));
    }
    resultado = significante * (float)Math.pow(2, exp);
    if (signo == -1)
    {
        resultado *= signo;
    }
    return resultado;
}
```