2° PARCIAL DE ORGANIZACIÓN DE LAS COMPUTADORAS – RECUPERATORIO- 2009

1) En un sistema de Punto flotante con mantisa fraccionaria normalizada en BCS de 4 bits y exponente en CA2 de 4 bits (de izquierda a derecha), ¿Qué número representa la cadena 11010111? _____-80_____

Explicación:
1101 mantisa fraccionaria normalizada en BCS. Por suerte ya está normalizada, así que no nos preocupamos por moverlo. ¿Cuál es ese valor? Bueno, el 1 más significativo es el de signo, sabemos que es negativo (no lo olvidemos!!), luego vemos que no tiene bit implícito, por lo tanto el bit siguiente vale 2^-1 y el último 2^-3 (recordemos que es fraccionaria).
0111 es el exponente, en Ca2, que, como indica el 0 del bit más significativo, es positivo, por lo tanto lo leemos tal cual como si fuese BSS, es el 7!
Entonces tenemos -(2^-1 + 2^-3) x 2^7, lo que nos da como resultado, haciendo distributiva, -(2^6 + 2^4), o sea, -(64 + 16), que ¡Sí! da como resultado -80

2) ¿Cuál es la cadena, en el sistema del punto anterior, que representa el número +0,125? _____0100 1110_____

Explicación:
La forma de hacerlo es la siguiente:
Sabemos que 0,125 es 2^-3. Escribimos 2^-3 en BSS: ,001 y el exponente 0000 para que nos quede el mismo número
Como el resultado tiene que estar normalizado porque el enunciado lo especifica, debemos correr el 1 hasta que llegue al lado de la coma. Entonces lo corremos 2 lugares, agrandando la mantisa, por lo tanto debemos achicar el exponente para mantener el mismo valor. Se achica tantas unidades como lugares corrimos, por lo tanto si tenías 0, ahora tenés -2. El -2 en Ca2 se escribe 1110.
Y ¡Tarán! Nos queda el número: 0100 de mantisa y 1110 de exponente

3) ¿Cuál es el número decimal positivo más grande que se puede representar en el sistema anterior? _____112_____

Número decimal positivo más grande con mantisa fraccionaria normalizada BCS y exponente Ca2 de 4 bits cada uno.
Para saber el número más grande positivo, debemos representar el número más grande positivo posible para la mantisa. Si tenemos 4 bits y es BCS, sabemos que uno de esos bits es exclusivo para el signo, entonces ya nos quedan 3. Mantisa: 0111, ese número es +(2^-1+ 2^-2+2^-3)

El exponente que tenemos que usar es el que nos agrande más y más el número. Sabemos que si usamos un exponente positivo, agranda el número y si usamos uno negativo, lo achica.
Entonces como es Ca2 y de 4 bits, uno lo usamos para el signo y los otros tres para representar el número... tenemos que es 0111, que es el +7.

4) ¿Cuál es el resultado de sumar las cadenas 01100011 y 01000100, expresadas en el sistema del punto 1? _____01110100_____

...0110 0011 (a)
+ 0100 0100 (b)
Tenemos mantisa fraccionaria normalizada en BCS y exponente en Ca2.
Los pasos a seguir para hacer operaciones aritméticas en punto flotantes son los siguientes:
1) Igualar exponentes. Para eso debemos determinar si achicamos uno, agrandamos otro y movemos los dos, de acuerdo con lo que nos permita la matisa -está prohibido "perder" bits-. Vemos entonces que (a) tiene un exponente positivo, cuyo valor es 3 y que (b) también es positivo y vale 4. Entonces podemos sumarle uno a 3 ó restarle uno a 4. El tema de igualar exponentes viene de la mano de tener que achicar o agrandar también la mantisa, porque sino estaríamos modificando el valor del número. Por lo tanto, si achico el exponente debo agrandar la mantisa y si agrando el exponente debo achicar la mantisa. ¿Cómo se hace? Corriendo la coma de lugar TANTOS LUGARES como unidades modifiqué el exponente. Si la mantisa es fraccionaria y yo corro la coma hacia la derecha, entonces estaría agrandando la mantisa, ya que haría enteros valores que antes eran fraccionarios. Pasa lo contrario, claramente, si corro la coma hacia la derecha.

Entonces en este caso particular tengo, como decía antes, dos opciones... agrando el exponente de (a) y achico su mantisa (corro la coma a la derecha) o achico el exponente de (b) y agrando su mantisa.
En este caso en particular, no podría correr la coma hacia la izquierda de (b), porque estaría usando el bit de signo como bit del número y eso sería incorrecto, por lo tanto la opción más viable sería agrandar de 3 a 4 el exponente de (a) y correr la coma un lugar hacia la derecha de la mantisa, lo que quedaría:

...0011 0100 (a)

+ 0100 0100 (b)

...0111 0100 (c)

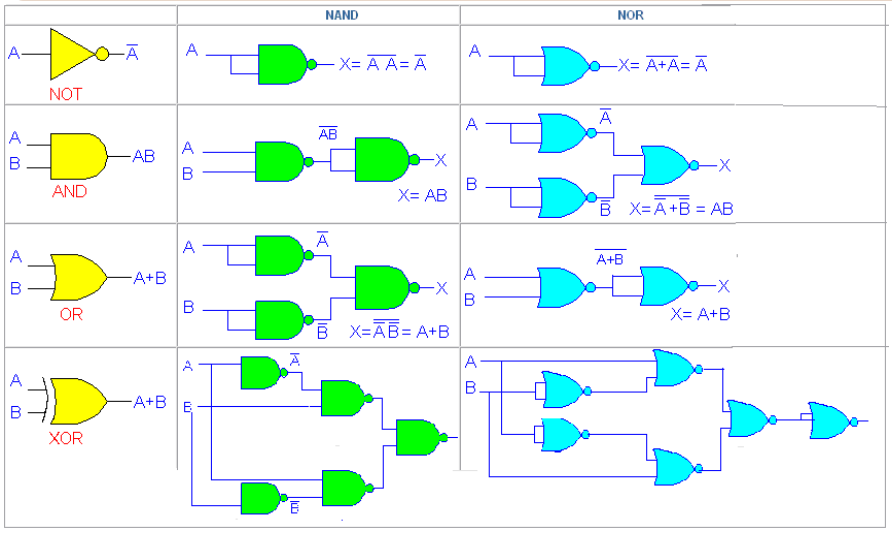
2) El 2° paso es sumar las mantisas (los exponentes no se suman, para eso los igualamos, es igual que en la aritmética ordinaria)
3) El 3° paso es convertir el resultado a lo que te pide en el enunciado. En este caso (c) quedó normalizado, por lo tanto no hay que hacer corrimiento de bits pero podría pasar

5) ¿Cuál es el resultado de la operación XNOR entre los bytes 1101 0110 y 0110 1100? _____01000101___

1101 0110
0110 1100
Sabemos que XOR nos da 0 cuando los bits son iguales, así que primero preferentemente hacemos el XOR de los bytes, que nos da:
1011 1010
Luego negamos todo el número, así hacemos el XNOR
0100 0101

6) Dibujar un circuito lógico correspondiente a la ecuación F = (~A+B).C utilizando sólo compuertas NOR.

Sólo debemos recordar qué representa cada símbolo y su correspondiente pasaje a compuerta NOR.
F= (~ A +B). C es lo mismo que decir F = (NOT A OR B) AND C



7) ¿Cuántas de las 8 posibles combinaciones de entradas A, B, C, dan como resultado un 1 lógico a la salida del circuito anterior? __3_____

Acá lo que tenemos que hacer es una tabla de verdad con la fórmula F = (NOT A OR B) AND C. Vemos que tiene 3 entradas (A, B y C) por lo tanto va a tener 2^3 salidas, o sea 8. Luego calculamos los resultados de las compuertas y vemos en F la cantidad de 1 que nos queda en la salida.

La Fuente

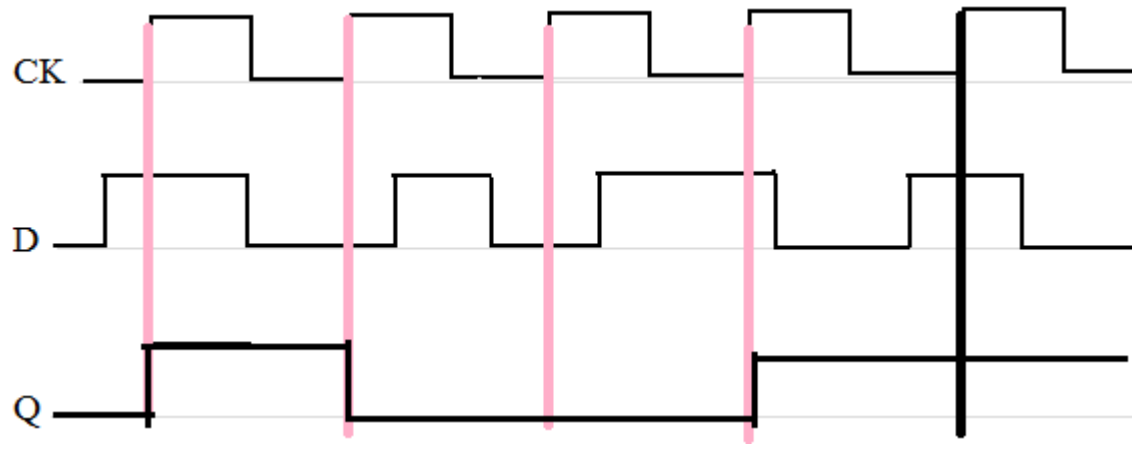
A	NOT A	NOT A OR B	B	C	F=(~ A OR B) * C
0	1	1	0	0	0
0	1	1	0	1	1
0	1	1	1	0	0
0	1	1	1	1	1
1	0	0	0	0	0
1	0	0	0	1	0
1	0	1	1	0	0
1	0	1	1	1	1

8) ¿Qué valor tomará la salida Q de un Flip Flop tipo JK, si inicialmente Q= 1 y las entradas se actualizan a J=1 y K=1? ___Q=0___

En un FF JK, cuando J y K son 0, Q toma el mismo valor que tenía inicialmente. Cuando J y K son 1, Q tiene el valor negado.
Cuando J y K son distintos, Q toma el valor de J.
En este caso, J=K=Q=1, así que la salida producirá una negación de Q, que claramente es 0

9) Dibuje cómo queda la salida Q en el siguiente diagrama de tiempo:

Es un FF D, así que sabemos que D copia su valor a Q en el flanco de subida de CK (quedó más lindo que nunca)



<u>El siguiente programa cuenta la cantidad de bits coincidentes entre DATO1 y DATO2</u>

```
                ORG   1000H
DATO1           DB    135
DATO2           DB    240
BITS            DB    ?

                ORG 3000H
SUB1:           MOV CH, 0
                XOR AL, AH
```

```
ORG 2000H
        MOV AL, DATO1
        MOV AH, DATO2
        CALL SUB1
    <Instrucción a agregar>
        HLT
        END
```

```
                <Instrucción faltante>
                MOV CL, 8
SALTO:          ADD AL, AL
                JNC SEGUIR
                INC CH
SEGUIR:         DEC CL
                JNZ SALTO
                RET
```

10) ¿Cuál debería ser la <Instrucción faltante> para que el programa funcione correctamente? _NOT AL

Es NOT AL porque en la instrucción anterior, XOR AL, AH, se guarda en AL la cantidad de bits iguales. Sabemos que XOR da 0 cuando son iguales, así que si queremos sumar debemos convertirlos a 1.

11) El pasaje de parámetros que se utiliza entre el programa principal y la subrutina SUB1 ¿es 'por referencia' o 'por valor'? __por valor___

Los parámetros DATO1 y DATO2 se guardan en AL y AH por valor, ya que se hace una copia de los mismos en los registros. Para que se pase por referencia deberían almacenarse mediante la instrucción OFFSET y luego acceder a ellos en la subrutina mediante el registro [BX].

12) ¿Qué valor queda almacenado en el registro CH al finalizar la ejecución del programa?__CH_

El registro CH se incrementa cada vez que la suma de AL con AL presente carry.
Acá tenemos que fijarnos qué valores son en binario DATO 1 Y DATO 2
DATO1 = 135 = 10000111
DATO2= 240 = 11110000
Luego vemos que XNOR entre los dos valores nos da como resultado: 10001000. El primer ADD entre este número y sí mismo nos da carry, ya que corre un lugar hacia la izquierda todos los bits. Así que ahí ya tendremos 1 en CH.
La instrucción SALTO se hace exactamente 8 veces, porque lo dice el registro CL. Si yo muevo 8 veces hacia la izquierda los bits de ese número, voy a tener dos veces carry, así que el resultado es 2.

13) ¿Cuál es el modo de direccionamiento de la instrucción MOV AL, DATO1? ___directo___

El campo de direcciones contiene la dirección efectiva del operando

14) ¿Qué instrucción debería agregarse <Instrucción a agregar> al final del programa para que el resultado quede almacenado en la celda BITS? _____MOV BITS, CH____

En CH se guarda el resultado, así que sólo es necesario moverlo a la celda