Trabajo práctico

Representación de números reales en formato punto fijo

Ejercicio 1

Compile el siguiente código en C en su PC:

```
#include <math.h>
#include <stdio.h>

void main(void)
{
      signed char a, b, c, d, s1, s2;

      a = 127;
      b = 127;

      c = a + b;
      d = a * b;

      s1 = (-8) >> 2;
      s2 = (-1) >> 5;

      printf("c = %d \n", c );
      printf("d = %d \n", d );
      printf("s1 = %d \n", s1 );
      printf("s2 = %d \n", s2 );
}
```

1. Verifique el valor de las variables c y d. ¿Son estos valores correctos?
2. De no ser así, ¿qué soluciones propone?
3. Verifique el valor de las variables s1 y s2. ¿Son estos valores correctos?

Check 3rd answer

Ejercicio 2

Cree 2 funciones en C:

1. Una función para pasar de punto fijo a punto flotante, `fx2fp( )`.
2. Una función para pasar de punto flotante a punto fijo, `fp2fx( )`.
3. Verifique el correcto funcionamiento haciendo,

   ```
   b = fx2fp( fp2fx( 2.4515) )
   ```

4. Compare `b` con `2.4515` para Q23.8 y Q21.10.

Ejercicio 3

Multiplique dos números en Q21.10. El resultado debe ser ajustado según los dos esquemas de redondeos estudiados.

1. Cree una función que implemente redondeo por truncación.

   ```
   int32_t truncation(int64_t X)
   ```

2. Cree una función que implemente redondeo al valor más cercano.

   ```
   int32_t rounding(int64_t X)
   ```

Usaría int64_t para no perder parte entera cierto?

3. Compare el resultado de cada esquema de redondeo con el resultado que obtendría usando números en formato `double`.

Ejercicio 4

Implemente la suma de dos números con aritmética de saturación,

```
int32_t saturation(int32_t a, int32_t b)
```

1. Utilice las constantes `INT_MAX` e `INT_MIN` definidas en la biblioteca `limits.h` para la
2. Opere con una serie de número para verificar que `saturation()` funciona correctamente.

Ejercicio 5

1. Escriba un programa en C que implemente la operación MAC en punto fijo para los siguientes vectores:
```
float X[5] = {1.1, 2.2, 3.3, 4.4, 5.5 } ;
float Y[5] = {6.6, 7.7, 8.8, 9.9, 10.10};
```
2. Convierta ambos vectores a formato Q21.10.
3. Implemente la operación MAC con dos enfoques:
   1. Multiplique 2 números de 32 bits y redondee por truncación antes de sumar cada resultados.
   2. Multiplique 2 números de 64 bits y solo redondee luego de sumar todos los números.

```
for(i=0; i < 5; i++)
{
    acum_32a += (int32_t) ( truncation( (int64_t) ( A[i] * B[i] ) ) );
    acum_64  += (int64_t) ( (int64_t) ( A[i] * B[i] ) );
}

acum_32b += (int32_t) acum_64;
```

4. Compare ambos resultados en punto fijo con el que se obtiene al operar en formato `double`.

```
double X[5] = {1.1, 2.2, 3.3, 4.4, 5.5 };
double Y[5] = {6.6, 7.7, 8.8, 9.9, 10.10};

for(i=0; i < 5; i++)
{
      acum_db += A[i] * B[i];
}
```

5. Finalmente, compare los valores de `acum_32a`, `acum_32b` y `acum_db`.

Ejercicio 6

Suponga que debe implementar un filtro digital con un procesador DSP de 16 bits. El mismo cuenta con un acumulador de 40 bits en su ALU. Calcule cuántas sumas consecutivas puede realizar garantizando que no se producirá un overflow.

- The general rule is the sum of s individual m-bit can require an accumulator of as many as $m + log_2(s)$ bits.

16+log2(s)=40
s=2^24
s=16.777.216 consecutive sums without an overflow