

Trabajo: Aritmética en computadores

Hernán Darío Castaño Rueda

Fundación Universitaria Internacional de la Rioja

Tecnología de computadores

Edwin Eduardo Millan Rojas

27/03/2024

Índice

Introducción	1
Respuesta a los ejercicios propuestos	1
Monitor de recursos del computador	1
Preguntas adicionales al proceso	2
Niveles de batería en dispositivos IoT	3
Preguntas adicionales al proceso	10
Niveles de batería en dispositivos móviles	14
Preguntas adicionales al proceso	15
Conclusiones	16
Referencias	20

Índice de figuras

1.	Tabla de potencias base 2 con sus respectivos binarios	1
2.	Tabla de potencias base 2 con sus respectivos binarios	2

Índice de tablas

1.	Representación en binario de los niveles de batería para dispositivos móviles	14
2.	Representación de complemento a 2 en 16 bits para el número 3FA.BC . . .	17
3.	Representación de complemento a 2 en 16 bits para el número -1E4.5D . . .	18
4.	Representación de complemento a 2 en 16 bits para el número -A9F.07 . . .	19

Introducción

En este documento se entrega al docente las respuestas al trabajo de aritmética de computadores propuesto para la materia. Consta de tres ejercicios en los que se hace uso de los métodos de conversión de bases numéricas.

Respuesta a los ejercicios propuestos

Monitor de recursos del computador

Para este ejercicio vamos a construir una tabla como la que se presenta en la siguiente figura, en la cual se observa que la fila “Valor” corresponde al calculo de 2 elevado al elemento correspondiente de la fila “Potencia de base 2”. Para hacer la conversión a decimal, solo se necesita sumar los elementos de la fila “Valor” que posean un 1 en la columna correspondiente.

Potencia base 2	6	5	4	3	2	1	0	-1	-2	-3
Valor	64	32	16	8	4	2	1	0.5	0.25	0.125
Uso de CPU	1	0	0	1	1	0	0	1	0	1
Uso de Memoria	1	1	0	0	0	0	1	0	1	1
Uso de disco	0	1	0	1	0	1	1	0	0	1

Figura 1

Tabla de potencias base 2 con sus respectivos binarios

Efectuando la suma de los valores se tendría el siguiente resultado:

1. Uso de CPU: $64 + 8 + 4 + 0.5 + 0.125 = 76.625$
2. Uso de Memoria: $64 + 32 + 1 + 0.25 + 0.125 = 97.375$
3. Uso de disco: $32 + 8 + 2 + 1 + 0.125 = 43.125$

Para la conversión a hexadecimal se puede partir del numero binario, agrupando de a cuatro cifras, dado que este agrupamiento nos va a arrojar valores desde 0x0 hasta 0xF.

En caso de que el numero no tenga grupos de cuatro exactos, se pueden agregar ceros hacia la izquierda para la parte entera y ceros a la derecha para la parte decimal.

Grupo	8	4	2	1	8	4	2	1	8	4	2	1
Uso de CPU	0	1	0	0	1	1	0	0	1	0	1	0
Uso de Memoria	0	1	1	0	0	0	0	1	0	1	1	0
Uso de disco	0	0	1	0	1	0	1	1	0	0	1	0

Figura 2

Tabla de potencias base 2 con sus respectivos binarios

1. Uso de CPU: Grupo 1: 4 (0x4), Grupo 2: $8 + 4 = 12$ (0xC), Grupo 3: $8 + 2 = 10$ (0xA) = 0x4C.A
2. Uso de Memoria: Grupo 1: $4 + 2 = 6$ (0x6), Grupo 2: $1 = 1$ (0x1), Grupo 3: $4 + 2 = 6$ (0x6) = 0x61.6
3. Uso de disco: Grupo 1: 2 (0x2), Grupo 2: $8 + 2 + 1 = 11$ (0xB), Grupo 3: 2 (0x2) = 0x2B.2

Preguntas adicionales al proceso

¿Cómo estas conversiones numéricas podrían o no contribuir a la eficiencia y precisión del monitor de recursos?

Las conversiones numéricas solo se efectúan para que el ser humano pueda identificar los valores y hacer cálculos aparte con ellos. Para la maquina es más eficiente mantener todo en binario que es su representación original. En el caso de que se necesite efectuar la transmisión de los datos, la única conversión necesaria, debería ser la conversión que se adapte al protocolo de transmisión requerido. Hacer conversiones puede generar un impacto a nivel de procesamiento y esta decisión debería ser tomada solo cuando sea estrictamente necesario.

Para el tema de precisión, depende del rango de valores que se necesite registrar, puede ser necesario ampliar el tamaño de palabra que puede registrar la máquina, pero esto implica un cambio físico y puede ser costoso en temas de sustitución de los equipos que se encuentran en uso.

¿Las cifras leídas podrían generar alguna alerta?

En este caso todo va a depender de lo que se considere como umbral de alertamiento, que es una medida acordada con el experto del sistema, ya que conoce cual es el comportamiento normal de este. Pero en el caso de que se vaya a hacer una suposición, el caso de memoria podría indicar que este elemento está cerca de su límite y se podría presentar problemas de llenado de memoria.

Niveles de batería en dispositivos IoT

Para la conversión a decimal se efectúa el siguiente procedimiento: Se separa la parte entera de la decimal y a cada parte se ejecutan las siguientes operaciones

Parte entera: Se hacen divisiones sucesivas por dos hasta que ya no sea posible seguir efectuando las divisiones que den un resultado entero. Los residuos pasan a ser el número binario, ordenado de atrás hacia adelante.

Parte decimal: Se hacen multiplicaciones sucesivas por dos, si el resultado da uno (1), se separa el uno y se repite la operación hasta que se encuentra un 1.0 o un 0.0. El número binario es la concatenación de los unos que se quitaron mas ceros en las operaciones que no dieron como resultado un uno.

Los resultados para los números planteados fueron los siguientes:

Int N: 3 cociente: 1 resto: 1

Int N: 1 cociente: 0 resto: 1

✓

Frac N: 0.7519999999999998 {:integer 1, :decimal 0.5039999999999996}

Frac N: 0.5039999999999996 {:integer 1, :decimal 0.007999999999999119}

Frac N: 0.007999999999999119 {:integer 0, :decimal 0.015999999999998238}

Frac N: 0.01599999999998238 {:integer 0, :decimal 0.03199999999996476}
 Frac N: 0.03199999999996476 {:integer 0, :decimal 0.0639999999999295}
 Frac N: 0.0639999999999295 {:integer 0, :decimal 0.127999999999859}
 Frac N: 0.127999999999859 {:integer 0, :decimal 0.255999999999718}
 Frac N: 0.255999999999718 {:integer 0, :decimal 0.511999999999436}
 Frac N: 0.511999999999436 {:integer 1, :decimal 0.023999999999887223}
 Frac N: 0.023999999999887223 {:integer 0, :decimal 0.047999999999774445}
 Frac N: 0.047999999999774445 {:integer 0, :decimal 0.09599999999954889}
 Frac N: 0.09599999999954889 {:integer 0, :decimal 0.19199999999909778}
 Frac N: 0.19199999999909778 {:integer 0, :decimal 0.38399999999819556}
 Frac N: 0.38399999999819556 {:integer 0, :decimal 0.7679999999963911}
 Frac N: 0.7679999999963911 {:integer 1, :decimal 0.5359999999927823}
 Frac N: 0.5359999999927823 {:integer 1, :decimal 0.07199999999855645}
 Frac N: 0.07199999999855645 {:integer 0, :decimal 0.1439999999971129}
 Frac N: 0.1439999999971129 {:integer 0, :decimal 0.2879999999942258}
 Frac N: 0.2879999999942258 {:integer 0, :decimal 0.5759999999984516}
 Frac N: 0.5759999999984516 {:integer 1, :decimal 0.15199999999769032}
 Frac N: 0.15199999999769032 {:integer 0, :decimal 0.30399999999538064}
 Frac N: 0.30399999999538064 {:integer 0, :decimal 0.60799999999076128}
 Frac N: 0.60799999999076128 {:integer 1, :decimal 0.215999999998152256}
 Frac N: 0.215999999998152256 {:integer 0, :decimal 0.431999999996304512}
 Frac N: 0.431999999996304512 {:integer 0, :decimal 0.863999999992609024}
 Frac N: 0.863999999992609024 {:integer 1, :decimal 0.72799999999852180481}
 Frac N: 0.72799999999852180481 {:integer 1, :decimal 0.45599999999704360962}
 Frac N: 0.45599999999704360962 {:integer 0, :decimal 0.91199999999408721924}
 Frac N: 0.91199999999408721924 {:integer 1, :decimal 0.82399999999817443848}
 Frac N: 0.82399999999817443848 {:integer 1, :decimal 0.64799999999634887695}

Frac N: 0.6479997634887695 {:integer 1, :decimal 0.29599952697753906}
 Frac N: 0.29599952697753906 {:integer 0, :decimal 0.5919990539550781}
 Frac N: 0.5919990539550781 {:integer 1, :decimal 0.18399810791015625}
 Frac N: 0.18399810791015625 {:integer 0, :decimal 0.3679962158203125}
 Frac N: 0.3679962158203125 {:integer 0, :decimal 0.735992431640625}
 Frac N: 0.735992431640625 {:integer 1, :decimal 0.47198486328125}
 Frac N: 0.47198486328125 {:integer 0, :decimal 0.9439697265625}
 Frac N: 0.9439697265625 {:integer 1, :decimal 0.887939453125}
 Frac N: 0.887939453125 {:integer 1, :decimal 0.77587890625}
 Frac N: 0.77587890625 {:integer 1, :decimal 0.5517578125}
 Frac N: 0.5517578125 {:integer 1, :decimal 0.103515625}
 Frac N: 0.103515625 {:integer 0, :decimal 0.20703125}
 Frac N: 0.20703125 {:integer 0, :decimal 0.4140625}
 Frac N: 0.4140625 {:integer 0, :decimal 0.828125}
 Frac N: 0.828125 {:integer 1, :decimal 0.65625}
 Frac N: 0.65625 {:integer 1, :decimal 0.3125}
 Frac N: 0.3125 {:integer 0, :decimal 0.625}
 Frac N: 0.625 {:integer 1, :decimal 0.25}
 Frac N: 0.25 {:integer 0, :decimal 0.5}
 Frac N: 0.5 {:integer 1, :decimal 0.0}

Nivel de batería 1:

3.C083126E978D4

11.11000000100000110001001001101110100101111000110101

Para el nivel de batería 2 el resultado fue el siguiente:

Int N: 12 cociente: 6 resto: 0

Int N: 6 cociente: 3 resto: 0

Int N: 3 cociente: 1 resto: 1

Int N: 1 cociente: 0 resto: 1

Frac N: 0.4879999999999955 {:integer 0, :decimal 0.975999999999991}

Frac N: 0.975999999999991 {:integer 1, :decimal 0.951999999999982}

Frac N: 0.951999999999982 {:integer 1, :decimal 0.903999999999964}

Frac N: 0.903999999999964 {:integer 1, :decimal 0.807999999999927}

Frac N: 0.807999999999927 {:integer 1, :decimal 0.615999999999854}

Frac N: 0.615999999999854 {:integer 1, :decimal 0.231999999999709}

Frac N: 0.231999999999709 {:integer 0, :decimal 0.463999999999418}

Frac N: 0.463999999999418 {:integer 0, :decimal 0.927999999998836}

Frac N: 0.927999999998836 {:integer 1, :decimal 0.855999999997672}

Frac N: 0.855999999997672 {:integer 1, :decimal 0.711999999995343}

Frac N: 0.711999999995343 {:integer 1, :decimal 0.423999999990687}

Frac N: 0.423999999990687 {:integer 0, :decimal 0.847999999981374}

Frac N: 0.847999999981374 {:integer 1, :decimal 0.695999999962747}

Frac N: 0.695999999962747 {:integer 1, :decimal 0.391999999925494}

Frac N: 0.391999999925494 {:integer 0, :decimal 0.7839999999850988}

Frac N: 0.7839999999850988 {:integer 1, :decimal 0.5679999999701977}

Frac N: 0.5679999999701977 {:integer 1, :decimal 0.13599999994039536}

Frac N: 0.13599999994039536 {:integer 0, :decimal 0.2719999998807907}

Frac N: 0.2719999998807907 {:integer 0, :decimal 0.5439999997615814}

Frac N: 0.5439999997615814 {:integer 1, :decimal 0.08799999952316284}

Frac N: 0.08799999952316284 {:integer 0, :decimal 0.17599999904632568}

Frac N: 0.17599999904632568 {:integer 0, :decimal 0.35199999809265137}

Frac N: 0.35199999809265137 {:integer 0, :decimal 0.7039999961853027}

Frac N: 0.7039999961853027 {:integer 1, :decimal 0.40799999237060547}
 Frac N: 0.40799999237060547 {:integer 0, :decimal 0.8159999847412109}
 Frac N: 0.8159999847412109 {:integer 1, :decimal 0.6319999694824219}
 Frac N: 0.6319999694824219 {:integer 1, :decimal 0.26399993896484375}
 Frac N: 0.26399993896484375 {:integer 0, :decimal 0.5279998779296875}
 Frac N: 0.5279998779296875 {:integer 1, :decimal 0.055999755859375}
 Frac N: 0.055999755859375 {:integer 0, :decimal 0.11199951171875}
 Frac N: 0.11199951171875 {:integer 0, :decimal 0.2239990234375}
 Frac N: 0.2239990234375 {:integer 0, :decimal 0.447998046875}
 Frac N: 0.447998046875 {:integer 0, :decimal 0.89599609375}
 Frac N: 0.89599609375 {:integer 1, :decimal 0.7919921875}
 Frac N: 0.7919921875 {:integer 1, :decimal 0.583984375}
 Frac N: 0.583984375 {:integer 1, :decimal 0.16796875}
 Frac N: 0.16796875 {:integer 0, :decimal 0.3359375}
 Frac N: 0.3359375 {:integer 0, :decimal 0.671875}
 Frac N: 0.671875 {:integer 1, :decimal 0.34375}
 Frac N: 0.34375 {:integer 0, :decimal 0.6875}
 Frac N: 0.6875 {:integer 1, :decimal 0.375}
 Frac N: 0.375 {:integer 0, :decimal 0.75}
 Frac N: 0.75 {:integer 1, :decimal 0.5}
 Frac N: 0.5 {:integer 1, :decimal 0.0}

Nivel de batería 2:

C.7CED916872B

1100.01111100111011011001000101101000011100101011

Para el nivel de batería 3 se encontraron los siguientes resultados

Int N: 114 cociente: 57 resto: 0

Int N: 57 cociente: 28 resto: 1

Int N: 28 cociente: 14 resto: 0

Int N: 14 cociente: 7 resto: 0

Int N: 7 cociente: 3 resto: 1

Int N: 3 cociente: 1 resto: 1

Int N: 1 cociente: 0 resto: 1

Frac N: 0.9150000000000063 {:integer 1, :decimal 0.8300000000000125}

Frac N: 0.8300000000000125 {:integer 1, :decimal 0.660000000000025}

Frac N: 0.660000000000025 {:integer 1, :decimal 0.32000000000005}

Frac N: 0.32000000000005 {:integer 0, :decimal 0.6400000000001}

Frac N: 0.6400000000001 {:integer 1, :decimal 0.2800000000002001}

Frac N: 0.2800000000002001 {:integer 0, :decimal 0.5600000000004002}

Frac N: 0.5600000000004002 {:integer 1, :decimal 0.12000000000080036}

Frac N: 0.12000000000080036 {:integer 0, :decimal 0.2400000000016007}

Frac N: 0.2400000000016007 {:integer 0, :decimal 0.4800000000032014}

Frac N: 0.4800000000032014 {:integer 0, :decimal 0.9600000000064028}

Frac N: 0.9600000000064028 {:integer 1, :decimal 0.9200000000128057}

Frac N: 0.9200000000128057 {:integer 1, :decimal 0.8400000000256114}

Frac N: 0.8400000000256114 {:integer 1, :decimal 0.6800000000512227}

Frac N: 0.6800000000512227 {:integer 1, :decimal 0.3600000001024455}

Frac N: 0.3600000001024455 {:integer 0, :decimal 0.720000000204891}

Frac N: 0.720000000204891 {:integer 1, :decimal 0.44000000040978193}

Frac N: 0.44000000040978193 {:integer 0, :decimal 0.8800000008195639}

Frac N: 0.8800000008195639 {:integer 1, :decimal 0.7600000016391277}

Frac N: 0.7600000016391277 {:integer 1, :decimal 0.5200000032782555}

Frac N: 0.5200000032782555 {:integer 1, :decimal 0.040000006556510925}
 Frac N: 0.040000006556510925 {:integer 0, :decimal 0.08000001311302185}
 Frac N: 0.08000001311302185 {:integer 0, :decimal 0.1600000262260437}
 Frac N: 0.1600000262260437 {:integer 0, :decimal 0.3200000524520874}
 Frac N: 0.3200000524520874 {:integer 0, :decimal 0.6400001049041748}
 Frac N: 0.6400001049041748 {:integer 1, :decimal 0.2800002098083496}
 Frac N: 0.2800002098083496 {:integer 0, :decimal 0.5600004196166992}
 Frac N: 0.5600004196166992 {:integer 1, :decimal 0.12000083923339844}
 Frac N: 0.12000083923339844 {:integer 0, :decimal 0.24000167846679688}
 Frac N: 0.24000167846679688 {:integer 0, :decimal 0.48000335693359375}
 Frac N: 0.48000335693359375 {:integer 0, :decimal 0.9600067138671875}
 Frac N: 0.9600067138671875 {:integer 1, :decimal 0.920013427734375}
 Frac N: 0.920013427734375 {:integer 1, :decimal 0.84002685546875}
 Frac N: 0.84002685546875 {:integer 1, :decimal 0.6800537109375}
 Frac N: 0.6800537109375 {:integer 1, :decimal 0.360107421875}
 Frac N: 0.360107421875 {:integer 0, :decimal 0.72021484375}
 Frac N: 0.72021484375 {:integer 1, :decimal 0.4404296875}
 Frac N: 0.4404296875 {:integer 0, :decimal 0.880859375}
 Frac N: 0.880859375 {:integer 1, :decimal 0.76171875}
 Frac N: 0.76171875 {:integer 1, :decimal 0.5234375}
 Frac N: 0.5234375 {:integer 1, :decimal 0.046875}
 Frac N: 0.046875 {:integer 0, :decimal 0.09375}
 Frac N: 0.09375 {:integer 0, :decimal 0.1875}
 Frac N: 0.1875 {:integer 0, :decimal 0.375}
 Frac N: 0.375 {:integer 0, :decimal 0.75}
 Frac N: 0.75 {:integer 1, :decimal 0.5}
 Frac N: 0.5 {:integer 1, :decimal 0.0}

Nivel de batería 3:

72.EA3D70A3D70C

1110010.1110101000111101011100001010001111010111000011

Preguntas adicionales al proceso

¿Cuál podría ser el impacto de estas conversiones numéricas en la eficiencia energética del dispositivo IoT?

La eficiencia energética del dispositivo se puede ver reducida porque el proceso de conversión implica una utilización adicional del procesador para efectuar dichas tareas, por lo tanto en dispositivos que presenten limitaciones de este tipo se debe ser selectivo en que las operaciones a efectuar sean pertinentes para la correcta operación e integración de dicho elemento de hardware.

¿Podrías representar el algoritmo de conversión en pseudocódigo o en algún lenguaje de programación?

El código que se deja a continuación está desarrollado en Clojure ¹ y permite la conversión de los valores con fracciones decimales de base 10 a base 2:

```
(ns decimals.binaries
  (:require [clojure.string :as str]))

(defn split-decimal
  "Converts a float to a map of integer and decimal part"
  [n]
  (let [integer-part (int n)
        decimal-part (- n integer-part)]
```

¹ Para ver el código en funcionamiento, se puede obtener desde GitHub en

<https://github.com/dario-castano/decimals>

```
{:integer integer-part :decimal decimal-part}}))
```

```
(defn int-to-binary
  "Converts an integer value to a binary one"
  [n binarr]
  (if (or (= n 1) (= n 0))
      (cons n binarr)
      (recur (quot n 2) (cons (rem n 2) binarr))))

(defn fraction-to-binary
  "Converts a decimal value (zero dot) form to a binary"
  [n binarr]
  (if (or (== n 1.0) (== n 0.0))
      (reverse binarr)
      (let [split (split-decimal (* n 2))]
        (recur (:decimal split) (cons (:integer split) binarr)))))

(defn decimal-to-binary
  "Converts a full decimal value to a binary"
  [n]
  (let [splitted (split-decimal n)
        intval (:integer splitted)
        decval (:decimal splitted)]
    (str
      (reduce str (int-to-binary intval []))
```

```

      "."

      (reduce str (fraction-to-binary decval []))))))

(defn split-binary
  "Splits a string representation of a binary number"
  [binary]
  (let [splitted (str/split binary #"\.")]
    {:integer (first splitted) :decimal (second splitted) }))

(defn extend-left
  "Fill zeros to the left to complete spaces in products of 4"
  [binary-str]
  (let [len (count binary-str)]
    (if (= (mod len 4) 0)
        binary-str
        (recur (str/join "" (flatten ["0" binary-str]))))))

(defn extend-right
  "Fill zeros to the right to complete spaces in products of 4"
  [binary-str]
  (let [len (count binary-str)]
    (if (= (mod len 4) 0)
        binary-str
        (recur (str/join "" (flatten [(seq binary-str) \0]))))))

```

Para la conversión a hexadecimal se hizo uso de las siguientes funciones:

```

(ns decimals.hexadecimal
  (:require [decimals.binaries :as bins]

```



```
[clojure.string :as str]))
```

```
(defn group-to-hexa
  "Returns a hexadecimal code from a group of 4 binaries"
  [word]
  (let [hexa ["0" "1" "2" "3"
              "4" "5" "6" "7"
              "8" "9" "A" "B"
              "C" "D" "E" "F"]
        group [8 4 2 1]
        values (map int (seq word))
        bins (map #(- % 48) values)
        code (reduce + (map (fn [x y] (* x y)) bins group))]
    (get hexa code)))

(defn build-hex
  "Build an hexadecimal from a 4-product length binary string"
  [binary-str result]
  (if (empty? binary-str)
      (str/join (reverse result))
      (let [[a b c d & tail] binary-str
            group (seq [a b c d])]
        (recur tail (cons (group-to-hexa group) result))))))

(defn bin-to-hexa
  "Converts a binary decimal number to hexadecimal"
```

```
[binary-str]
(let [parts (bins/split-binary binary-str)
      int-part (bins/extend-left (:integer parts))
      dec-part (bins/extend-right (:decimal parts))]
  (str
    (build-hex int-part [])
    "."
    (build-hex dec-part []))))
```

Niveles de batería en dispositivos móviles

Cuadro 1

Representación en binario de los niveles de batería para dispositivos móviles

3				F				A				B				C			
8	4	2	1	8	4	2	1	8	4	2	1	8	4	2	1	8	4	2	1
11	10	9	8	7	6	5	4	3	2	1	0	-1	-2	-3	-4	-5	-6	-7	-8
0	0	1	1	1	1	1	1	1	0	1	0	1	0	1	1	1	1	0	0
1				E				4				5				D			
8	4	2	1	8	4	2	1	8	4	2	1	8	4	2	1	8	4	2	1
11	10	9	8	7	6	5	4	3	2	1	0	-1	-2	-3	-4	-5	-6	-7	-8
0	0	0	1	1	1	1	0	0	1	0	0	0	1	0	1	1	1	0	1
A				9				F				0				7			
8	4	2	1	8	4	2	1	8	4	2	1	8	4	2	1	8	4	2	1
11	10	9	8	7	6	5	4	3	2	1	0	-1	-2	-3	-4	-5	-6	-7	-8
1	0	1	0	1	0	0	1	1	1	1	1	0	0	0	0	0	1	1	1

Vamos a hacer uso de la segunda fila de cada número hexadecimal de la tabla, que

indica a que potencia de base 2 debo elevar cada bit que posea 1, para hacer la sumatoria y encontrar el numero en decimal:

Para $0x3FA.BC$

$$(2^9) + (2^8) + (2^7) + (2^6) + (2^5) + (2^4) + (2^3) + (2^1) + (2^{-1}) + (2^{-3}) + (2^{-4}) + (2^{-5}) + (2^{-6}) = 1018,734375$$

Para $0x - 1E4,5D$

$$(2^8) + (2^7) + (2^6) + (2^5) + (2^2) + (2^{-2}) + (2^{-4}) + (2^{-5}) + (2^{-6}) + (2^{-8}) = -484,36328125$$

Para $0x - A9F,07$

$$(2^{11}) + (2^9) + (2^7) + (2^4) + (2^3) + (2^2) + (2^1) + (2^0) + (2^{-6}) + (2^{-7}) + (2^{-8}) = -2719,02734375$$

Si tenemos en cuenta que el registro de los datos se tiene que hacer en un registro de 16 bits con complemento a 2, se va a reservar el primer bit del registro para alojar el signo y los 15 bits restantes de la siguiente forma:

Bit 15: signo, Bit 14 a Bit 3: Parte entera del rango 2^{11} hasta 2^0 , Bits 2 a Bit 0:

Parte decimal del rango 2^{-1} a 2^{-3}

Para $0x3FA.BC$ se tendría la siguiente representación:

Dado que los siguientes números son negativos, se ejecuta el complemento a 2 para almacenarlo. Por favor observar las tablas en las que se detalla la conversión numérica.

Para $-0x1E4,5D$ tendríamos a 1111000011011.110 como representación en complemento a 2

Para $-0xA9F,07$ tendríamos a 1010101100001.000 como representación en complemento a 2

Preguntas adicionales al proceso

¿Cómo estas conversiones numéricas facilitan la gestión de la energía en los relojes inteligentes?

Estas conversiones numéricas facilitan la gestión de energía debido a que al ser un tamaño de registro mas pequeño, se requiere menor cantidad de pasos de iteración para efectuar dichas conversiones.

¿Qué ventajas proporciona el uso del sistema binario en complemento a 2 de 16 bits?

La ventaja es que en un mismo registro se pueden manejar números positivos y negativos conservando la integridad de estos, y permitiendo poder efectuar operaciones aritméticas sin una conversión previa antes de ejecutar cada operación. Se pueden operar directamente números positivos y negativos sin ningún problema.

Conclusiones

El ejercicio permitió poder explorar un poco más en detalle las razones de por qué se toman decisiones de diseño elaboradas en torno al almacenamiento de información que posee rangos infinitos, en un rango finito como es el registro de memoria de un dispositivo electrónico.

También permitió conocer las razones de por qué las máquinas trabajan en sistemas de numeración diferentes al que los seres humanos usan de forma habitual.

Cuadro 2

*Representación de complemento a 2 en 16 bits para el número
3FA.BC*

BIT	Exponente	3FA.BC	
15	Signo	0	0
14	11	0	0
13	10	0	0
12	9	1	512
11	8	1	256
10	7	1	128
9	6	1	64
8	5	1	32
7	4	1	16
6	3	1	8
5	2	0	0
4	1	1	2
3	0	0	0
2	-1	1	0.5
1	-2	0	0
0	-3	1	0.125
SUMA		1018.625	

Cuadro 3

*Representación de complemento a 2 en 16 bits para el número
-1E4.5D*

BIT	Exponente	1E4.5D		Complemento a 2
15	Signo	0	0	1
14	11	0	0	1
13	10	0	0	1
12	9	0	0	1
11	8	1	256	0
10	7	1	128	0
9	6	1	64	0
8	5	1	32	0
7	4	0	0	1
6	3	0	0	1
5	2	1	4	0
4	1	0	0	1
3	0	0	0	1
2	-1	0	0	1
1	-2	1	0.25	1
0	-3	0	0	0

Cuadro 4

*Representación de complemento a 2 en 16 bits para el número
-A9F.07*

BIT	Exponente	A9F.07		Complemento a 2
15	Signo	0	0	1
14	11	1	2048	0
13	10	0	0	1
12	9	1	512	0
11	8	0	0	1
10	7	1	128	0
9	6	0	0	1
8	5	0	0	1
7	4	1	16	0
6	3	1	8	0
5	2	1	4	0
4	1	1	2	0
3	0	1	1	1
2	-1	0	0	0
1	-2	0	0	0
0	-3	0	0	0

Referencias