

Q₂.

• Convertir un # binario a Two Complement, su versión negativa.

Se debe copiar el numero binario luego recorrerlo de derecha a izquierda y cuando se encuentre el primer 1, todos los bits a la izquierda del primer 1 deben ser invertidos. Este 1 se debe mantener.

$$0 = 00000$$

$$00000 =$$

$$-1 = 11111$$

$$00001 = 2^0$$

$$-2 = 00010 \rightarrow 11110$$

$$00010 = 2^1$$

$$-4 = 00100 \rightarrow 11100$$

$$00111 = 2^3 + 2^2 + 2^1$$

$$-7 = 00111 \rightarrow 11001 \rightarrow 7 \text{ in Two Complement}$$

$$01111 = 2^3 + 2^2 + 2^1 + 2^0$$

$$-16 = 10000 \rightarrow 10000 \rightarrow 10000$$

$$11111 = \overbrace{\quad}^{\downarrow}$$

$$2^4 = 16$$

$$2^3 = 8$$

$$2^2 = 4$$

$$2^1 = 2$$

$$2^0 = 1$$

$$2^4 = 16$$

$$2^3 = 8$$

$$2^2 = 4$$

$$2^1 = 2$$

$$2^0 = 1$$

$$-16 = 10000 \rightarrow 10000 \rightarrow 10000$$

↳ en 5 bits parecen ser lo mismo pero si se aumenta el numero de bits los que le siguen son 0's

$$16 = 00010000$$

$$-16 = 11110000$$

Pasarlo a Decimal

Two

$$11011 = -5$$

$$10100 = -11$$

$$01011 = -21$$

$$00001 = -31$$

$$10000 = -16$$

* No hay forma de transformar un Two' Complement a base 10.

Este tiene una interpretación directa a su negativo del # original.

Q₃.

De decimal a hexadecimal.

$$\begin{array}{r} 2^4=16 \\ 2^3=8 \\ 2^2=4 \\ 2^1=2 \\ 2^0=1 \end{array}$$

↳ La forma de transformar un numero de decimal

$$0 = 00000000 \quad 00_{16}$$

$$127 = \overbrace{0111}^{6} \overbrace{1111}^{0} = 7F_{16}$$

$$10 = \overbrace{0000}^{1} \overbrace{010}^{0} \rightarrow 0A_{16}$$

$$255 = \overbrace{1111}^{1} \overbrace{1111}^{0}$$

$$14 = \overbrace{0000}^{1} \overbrace{110}^{0} \rightarrow 0E_{16}$$

$$2^7+2^6+2^5+2^4+2^3+2^2+2^1+2^0 = FF_{16}$$

$$15 = \overbrace{0000}^{1} \overbrace{111}^{1} \rightarrow 0F_{16}$$

$$\begin{array}{r} 128 \\ 64 \\ 32 \\ 16 \\ 8 \\ 4 \\ 2 \\ 1 \end{array}$$

$$16 = \overbrace{0001}^{1} \overbrace{0000}^{0} \rightarrow 10_{16}$$

$$\begin{array}{r} 256 \\ 128 \\ 64 \\ 32 \\ 16 \\ 8 \\ 4 \\ 2 \\ 1 \end{array}$$

$$32 = \overbrace{0010}^{1} \overbrace{0000}^{0} \rightarrow 20_{16}$$

$$\begin{array}{r} 256 \\ 128 \\ 64 \\ 32 \\ 16 \\ 8 \\ 4 \\ 2 \\ 1 \end{array}$$

a hexadecimal es transformar el # a binario

y luego separarlo 4bit y cada 4 bits representan un dígito en hexadecimal.

$$\begin{array}{l} A=10 \\ B=11 \\ C=12 \\ D=13 \\ E=14 \\ F=15 \end{array}$$

Q4. $0x1 = 00000001 = 1_{10}$

$0x2 = 00000010 = 2_{10}$

$0x8 = \overbrace{0000}^1 \overbrace{000}^0 = 8_{10}$

$0x10 = \overbrace{0001}^1 \overbrace{0000}^0 = 16_{10}$

$0x7F = \overbrace{0111}^7 \overbrace{1111}^1 = 127_{10}$

$0xFF = \overbrace{1111}^F \overbrace{1111}^F = 255_{10}$

$0x80 = \overbrace{1000}^1 \overbrace{0000}^0 = 128$

Q5. 1. $11111111 = -1_{10}$

2. $11111110 = -2_{10}$

3. $11111000 = -8_{10}$

4. $11110000 = -16_{10}$

5. $10000001 = -127_{10}$

6. $\overbrace{00000001}^1 = -255_{10}$

7. $1\dots10000000 = -128$

↙

Q6. Unsigned Two' Complement

$[0; 2^4 - 1]$

$[0; 15]$

$[0; 31]$

$[0; 63]$

$[0; 127]$

$[0; 255]$

$[-2^{N-2}; 2^{N-1} - 1]$

$[-8; 7] \quad 4$

$[-16; 15] \quad 5$

$[-32; 31] \quad 6$

$[-64; 63] \quad 7$

$[-128; 127] \quad 8$

Q7.

Q8. En teoria

Big endian: Cuando el mayor valor comienza por byte mas significativo.

Little endian: Cuando el menor valor comienza por el byte menos significativo.

0x 1000 →

0x 1001 →

0x 1002 →

0x 1003 →

↓
posiciones
en memoria

$$EF_{16} \rightarrow \overbrace{1110}^1 \overbrace{1111}^2 = 2^0 + 2^1 + 2^2 + 2^3 + 2^4 + 2^5 + 2^6 + 2^7 = 239_{10}$$

$$BE_{16} \rightarrow 11 \times 16 + 14 = 11 \times 16 + 14 = 190_{10}$$

$$AD_{16} \rightarrow \overbrace{1010}^2 \overbrace{1101}^3 = 2^2 + 2^3 + 2^4 + 2^5 + 2^6 = 173_{10}$$

$$DE_{16} \rightarrow \overbrace{1101}^1 \overbrace{1110}^2 = 2^7 + 2^8 + 2^9 + 2^{10} = 220_{10}$$

valores en Decimal de los valores hexadecimales.

- Número Original: 0x DE AD BE EF
- (MSB) = DE contiene los bits de mayor peso.
- (LSB) = EF contiene a los bits de menor peso.

* Realmente little endian o big endian no depende de un numero en particular sino que en base a cualquier numero ya creado es que tu computadora los ordena en base al endian que pertenece.

* Si tengo 0x 10 20 30 40 como # guardado en memoria. Si tu pc guarda este valor como 40 30 20 10, es little endian pq toma los bytes mas significativos y los coloca en los bytes mas significativos del nuevo orden. En el caso de tener un orden 10 20 30 40, estamos ante un big endian ya que el MSB esté ubicado en el MSB del nuevo orden.

Q9. Zero-extend y Sign-extend
de 4 bits a 8 bits \rightarrow byte

Unsigned $[0 ; 2^n - 1]$

	Zero-extend	Decimal	Sign-extend	TWO Complement	Decimal
0001	$0000\ 0001$	1	$0000\ 0001$	$0000\ 0001$	1
1111	$0000\ 1111$	15	$1111\ 1111$	$0000\ 0001$	-1
1010	$0000\ 1010$	10	$1111\ 1010$	$\overbrace{0000}^1 \overbrace{0110}^0$	-6
1000	$0000\ 1000$	8	$1111\ 1000$	$\overbrace{0000}^1 \overbrace{1000}^0$	-8
0111	$0000\ 0111$	7	$0000\ 0111$	$0000\ 0111$	7

$$[-2^{n-1}, 2^{n-1} - 1]$$

Decimal en Two Complement

$0001 \rightarrow 2^0 = +1$	$1111 \rightarrow -2^3 + 7 = -1$
$\underbrace{1111}_{4\text{ bits}} \rightarrow -2^{n-1} = -8 + 2^2 + 2^1 + 2^0 = -1$	$0001 \rightarrow 2^0 = 1$
$1010 \rightarrow -2^{4-1} = -8 + 2 = -6$	$0110 \rightarrow 2^1 + 2^2 = 4 + 2 = 6$
$1000 \rightarrow -2^4 = -8 + 0 = -8$	$1000 \rightarrow 2^3 = 8$
$0111 \rightarrow 2^0 + 2^1 + 2^2 = 7$	$1001 \rightarrow -8 + 1 = -7$

\downarrow	$\text{TWO}'C$
0000 0001	1111 1111
1111 1111	0000 0001
1111 1010	0000 0110
1111 1000	0000 1000
0000 0111	1111 1001

Extensión en

* Hay una diferencia enorme entre Unsigned y Two Complement.
La diferencia radica en la forma en como interpreta mas un conjunto de bits. Las dimensiones del rango de valores se reduce a un exponente menor.

Cuando comienza por 0 , le suman potencias de la forma tradicional.

Cuando comienza por 1 se entiende que el numero es negativo en TwC . Por ello, se agrega el "límite negativo" que representaría el primer bit , despues se hace una suma del resto de bits.

Q 10.

a	b	ab	a	b	a+b	a	b	a+b
0	1	0	0	1	1	0	1	1
0	1	0	0	1	1	0	1	1
1	0	0	1	0	1	1	0	1
1	0	0	1	0	1	1	0	1