Organización del computador - Benitez

Franco Secchi

May 6, 2024

Buenas buenas, paso a aclarar que es un resumen hecho con mis palabras, entonces peco de informal en algunos momentos.

Muchos éxitos en la cursada máquina.

Contenidos

1 Numeración **3**
1.1 Teorema fundamental de la numeración 3
1.2 Pasaje de base X a base 10 3
1.3 Pasaje de base 10 a otra base 3
1.3.1 Números enteros 3
1.3.2 números con coma 4
1.4 Pasaje con raíz exacta 4
1.5 Pasaje con potencia 6

1.6 Aritmética 6
1.6.1 Suma + ... 6
1.6.2 Resta - 8

2 Formatos de representación en una computadora 9
2.1 Métodos de representación de números negativos 9
2.1.1 Bit de signo y valor absoluto 9
2.1.2 Complemento a 1 10
2.1.3 Complemento a la base 11
2.1.4 Complemento a 2 12
2.1.5 Exceso 13
2.2 Conceptos de Formato y Configuración 15
2.3 Conceptos de Expansión y Truncamiento 15
2.4 Formatos de representación de números enteros 16
2.4.1 Binario de punto fijo sin signo 16
2.4.2 Binario de punto fijo con signo 16
2.4.3 BCD Empaquetado 16
2.4.4 Zoneado 16
2.5 Formatos de representación de caracteres 16
2.5.1 ASCII 16
2.5.2 EBCDIC 16
2.5.3 UNICODE 16
2.6 Formatos de representación de números decimales 16
2.6.1 Binario punto Flotante IEEE 754 16

3 Maquina elemental 16

4 Arquitectura del conjunto de instrucciones 16

5 Lenguaje ensamblador 16

6 Componentes de un computador 16

7 Almacenamiento secundario 16

1 Numeración

1.1 Teorema fundamental de la numeración

$$C = X * B = \sum_{i=-\infty}^{+\infty} X_i * bi$$

donde C es un número en cualquier base, X es el vector de dígitos, B es el vector de pesos y b es la base del sistema de numeración.

Esto te sirve para pasar cualquier número de base X a base 10.

1.2 Pasaje de base X a base 10

Por ejemplo:

Quiero pasar $1234\rfloor_4$ a base 10. Entonces lo que hago es utilizar el teorema:

$$1 * 4^3 + 2 * 4^2 + 3 * 4^1 + 4 * 4^0 = 64 + 32 + 12 + 4 = 112$$

Entonces podemos decir que $1234\rfloor_4$ es igual a $112\rfloor_{10}$ ¿Y con los que tienen coma? ¿Cómo se hace?

Supongamos que queremos pasar $3,21\rfloor_4$ a base 10. Vamos a seguir utilizando el teorema, y lo vamos a usar de esta manera:

$$3 * 4^0 + 2 * 4^{-1} + 1 * 4^{-2} = 3 + \frac{1}{2} + \frac{1}{16} = 3.5625$$

Entonces podemos decir que $3,21\rfloor_4$ es igual a $3,5625\rfloor_{10}$

1.3 Pasaje de base 10 a otra base

1.3.1 Números enteros

Acá vamos a usar las divisiones sucesivas. Te sirve para pasar cualquier número en base 10 a otra base. Supongamos que queremos pasar a base 2.

Tenés que hacer divisiones sucesivas hasta que el resto de 0. Vamos con un ejemplo:

Queremos pasar él $10\rfloor_{10}$ a base 2

Dividendo	Cociente
$10 \div 2$	$= 5,$ residuo 0
$5 \div 2$	$= 2,$ residuo 1
$2 \div 2$	$= 1,$ residuo 0
$1 \div 2$	$= 0,$ residuo 1

Entonces él $110\rfloor_{10}$ es $1010\rfloor_2$

1.3.2 números con coma

Y acá vamos a usar las multiplicaciones sucesivas. Vamos como hacer para pasar él $27.625\rfloor_{10}$ a base 4.

El algoritmo es el siguiente:

- Parte Entera: Utilizas las divisiones sucesivas que vimos anteriormente
- Parte fraccionaria
    1. Multiplica la parte fraccionaria del número decimal por la base a que quieras pasar (en este caso 4).
    2. La parte entera del resultado será el dígito más significativo en la parte fraccionaria de la nueva base.
    3. Toma la parte fraccionaria del resultado y repite el proceso hasta alcanzar la precisión deseada o hasta que la parte fraccionaria sea 0.

Entonces apliquemos el algoritmo.

- Parte entera:

Dividendo	Residuo (dígito base 4)
$27 \div 4$	$= 6,$ residuo 3
$6 \div 4$	$= 1,$ residuo 2
$1 \div 4$	$= 0,$ residuo 1

  por lo tanto, la parte entera seria $123\rfloor_4$.

- Parte fraccionaria:

$$0.625 \times 4 = 2.5$$
$$0.5 \times 4 = 2.0$$

  Por lo tanto, la parte fraccionaria es $22\rfloor_4$

Entonces, él $27.625\rfloor_{10}$ es equivalente a $123.22\rfloor_4$.

1.4 Pasaje con raíz exacta

Hay casos que cuando queremos pasar de una base a otra, la base a es raíz exacta (x) de b. ¿Qué caso podría ser? Sí quiero pasar de base 2 a base 16. ¿Cómo sería eso?

$$2 = \sqrt[x]{16}$$
$$x = 4$$

¿Y qué se hace con ese dato? Bueno la idea es agrupar de a x dígitos de la base de origen y genero un dígito de la base destino. Vamos con un ejemplo (de la guía).

Ejemplo 1

$$1001\rfloor_2 \rightarrow \rfloor_{16}$$

Como vimos anteriormente, la raíz exacta es 4, entonces vamos a agrupar de a cuatro dígitos y ese numero lo vamos a pasar a base 16.

$$\overbrace{1001}\rfloor_2$$

Ahora lo que hacemos es pasar ese grupo a base 16. Para hacerlo claro, lo pasamos a base 10 y de 10 a 16.

$$1001\rfloor_2 = 1 * 2^3 + 0 * 2^2 + 0 * 2^1 + 1 * 2^0 = 9\rfloor_{10}$$

$$9\rfloor_{10} = 9\rfloor_{16}$$

Entonces nos queda qué $1001\rfloor_2 = 9\rfloor_{16}$.

Ejemplo 2 Vamos a ir con otro ejemplo que es importante. Queremos hacer este pasaje.

$$101\rfloor_2 \rightarrow \rfloor_{16}$$

Como vimos anteriormente, la x es 4, pero el problema es que no podemos agrupar de a 4 dígitos porque solo tenemos 3 dígitos, ¿cómo hacemos entonces? Lo que hay que hacer es **agregar 0's a la izquierda** hasta que podamos agrupar de a x dígitos.

Entonces, agregando 0's a la izquierda, nos quedaría de la siguiente manera:

$$0101$$

Y con eso hacemos la misma lógica que vimos, y concluimos que $101\rfloor_2 = 5\rfloor_{16}$.

Hasta acá vimos solamente cuando hay 1 grupo de dígitos, ¿cómo hay que hacer cuando hay más?

Ejemplo 3

$$110101\rfloor_2 \rightarrow \rfloor_{16}$$

Sabemos que $x = 4$. Como vemos que no podemos agrupar de a 4, agregamos 0's a la izquierda.

$$\overbrace{0011}\overbrace{0101}\rfloor_2$$

Entonces la lógica es que el primer grupo va a representar al primer dígito de la base destino (en este caso 16), el segundo grupo representa al segundo dígito, y así.

$$0011\rfloor_2 = 3\rfloor_{16}$$

$$0101\rfloor_2 = 5\rfloor 16$$

Entonces concluimos que $110101\rfloor_2 = 35\rfloor_{16}$

1.5 Pasaje con potencia

$$a = b^x$$

La idea es expandir un dígito de la base origen y generar x dígitos de la base destino.

Ejemplo 1 Queremos hacer este pasaje $74321\rfloor_8 \rightarrow \rfloor_2$.

$$8 = 2^x$$

$$x = 3$$

Esto significa que el 7 va a estar generado por 3 dígitos en base 2, 4 con 3, 2 con 3 y 1 con 3.

$$7\rfloor_8 = 111\rfloor_2$$
$$4\rfloor_8 = 100\rfloor_2$$
$$3\rfloor_8 = 011\rfloor_2$$
$$2\rfloor_8 = 010\rfloor_2$$
$$1\rfloor_8 = 001\rfloor_2$$

Entonces $74321\rfloor_8 = \overbrace{111}^{7}\overbrace{100}^{4}\overbrace{011}^{3}\overbrace{010}^{2}\overbrace{001}^{1}\rfloor_2$

1.6 Aritmética

En la cursada se ven solamente lo que son sumas y restas. Todo lo que es multiplicar y dividir no se ve.

1.6.1 Suma +

La idea es muuuuy similar a lo que hacemos cotidianamente con el sistema decimal. Supongamos que tenemos la suma de $109 + 19$ ¿qué haciamos? Bueno, $9 + 9 = 18$ entonces dejo el 8 y paso el 1 que me sobra. Bueno, en otras bases es igual.

Ejemplo base 2 Queremos hacer la siguiente suma $1001 + 101\rfloor_2$

$$\begin{array}{r} 1001 \\ +\ 101 \\ \hline \end{array}$$

Sumamos como sabemos hacerlo, $1 + 1\rfloor_2 = 10$, entonces bajo el 0 y paso el uno

$$\begin{array}{r} {\scriptstyle 1} \\ 1001 \\ +0101 \\ \hline 0 \end{array}$$

Ahora hacemos $1 + 0\rfloor_2 = 1$ y asi sucesivamente. Y nos quedaria que

$$\begin{array}{r} {\scriptstyle 001} \\ 1001 \\ +0101 \\ \hline 1110 \end{array}$$

Ejemplo base 16 Queremos hacer la siguiente suma $CDDE + 1F1F\rfloor_{16}$

$$\begin{array}{r} CDDE \\ +1\,F\,1\,F \\ \hline \end{array}$$

Cuando hacemos cuentas con hexadecimal, siempre vamos a pensar en decimal y despues lo pasamos a hexa. Por ejemplo, empecemos haciendo $E + F$

$$E + F = 14 + 15\rfloor_{10} = 29\rfloor_{10}$$

Entonces agarramos el 29 y lo pasamos a base 16

$$29\rfloor_{10} = 1D\rfloor_{16}$$

¿Cómo sé que da 1D? Haciendo divisiones sucesivas. Entonces el 1 lo acarreamos y el D lo dejamos abajo

$$\begin{array}{r} {\scriptstyle 1} \\ CDDE \\ +1\,F\,1\,F \\ \hline D \end{array}$$

Como tenemos acarreado el 1, hay que hacer $1 + D + 1 = E + 1 = F$

$$\begin{array}{r} {\scriptstyle 1} \\ CDDE \\ +1\,F\,1\,F \\ \hline FD \end{array}$$

Para sumar $D + F$ hacemos lo mismo de antes, lo pensamos en decimal y después lo pasamos.

$$D + F = 13 + 15 = 28$$

$$28\rfloor_{10} = 1C\rfloor_{16}$$

$$\begin{array}{r} \color{blue}{1\quad 1} \\ CDDE \\ +1F1F \\ \hline CFD \end{array}$$

Y por último tenemos qué $1 + C + 1 = D + 1 = E$.

$$\begin{array}{r} \color{blue}{1\quad 1} \\ CDDE \\ +1F1F \\ \hline ECFD \end{array}$$

Entonces podemos concluir que $CDDE + 1F1F = ECFD\rfloor_{16}$

1.6.2 Resta -

Con la resta, la idea es la misma que en el sistema decimal, solamente que hay un concepto que puede abrir la cabeza (o tal vez ya lo sabías y no es para tanto).

Hagamos la siguiente resta:

$$\begin{array}{r} 111 \\ -\quad 3 \\ \hline \end{array}$$

¿Qué hacemos ahí? Bueno, como $1 - 3$ es negativo, el 1 de la derecha le tiene que pedir 1 al de la izquierda.

$$\begin{array}{r} \color{blue}{1} \\ 101 \\ -\quad 3 \\ \hline \end{array}$$

Entonces, ¿ahí qué decís? Bueno, ahora tengo $11 - 3$... pero ¿por qué decimos que tiene 11? ¿Por qué le sumamos 10? Esto ocurre porque vos al 3 no le estás pasando una unidad, sino que le estás pasando **la base** que trabajas, por eso le pasas 10!!!!

Entonces, por ejemplo si queremos restar $10 - 1\rfloor_8$, al 0 no le pasas 10, sino que le pasas 8!

$$\begin{array}{r} \color{blue}{8} \\ 10 \\ -\ 1 \\ \hline \end{array}$$

$$\begin{array}{r} \color{blue}{8} \\ 10 \\ -\ 1 \\ \hline 07 \end{array}$$

Entonces el resultado de la resta es $7\rfloor_8$. ¿Y en base 2?

$$\begin{array}{r} {\scriptstyle 2} \\ 10 \\ -\ 1 \\ \hline \end{array}$$

Pero para, el 2 no existe en base 2, entonces hay que pasarlo a base 2.

$$\begin{array}{r} {\scriptstyle 10} \\ 1\,0 \\ -\ \ 1 \\ \hline \ \ 1 \end{array}$$

2 Formatos de representación en una computadora

2.1 Métodos de representación de números negativos

En general, los números negativos en cualquier base b se representan del modo habitual, precediéndolos con un signo $-$. En cambio, en una computadora (sistema binario) existen 4 métodos posibles que son:

1. Bit de signo y valor absoluto
2. Complemento a 1
3. Complemento a la base
4. Complemento a 2

2.1.1 Bit de signo y valor absoluto

Para representar un número signado de n-bits usando éste método, consiste en:

1. Un bit para representar el signo. Ese bit a menudo es el bit más significativo y, por convención, se usa un 0 para números positivos, y 1 para los negativos.
2. Los $n-1$ bits restantes para representar él significando que es la magnitud del número en valor absoluto.

Ejemplo Representar $-97\rfloor_{10}$ en 8 bits

Con 8 bits, tenemos 1 bit para el signo y 7 bits para la magnitud. Entonces, para representar el número -9710 procedemos a:

1. Mirar el signo del número $-97\rfloor_{10}$, apreciamos que es negativo, llevará como bit de signo un 1

2. Realizar la conversión: el valor absoluto de $-97\rfloor_{10}$ es $|-97\rfloor_{10}| = 97\rfloor_{10}$. Que en binario es: $1100001\rfloor_2$;

3. Colocar todo junto $\rightarrow$ el número -9710 queda representado de la siguiente manera usando **Signo y Magnitud**: $11100001\rfloor_2$. Donde el 1 en el bit más significativo indica un número negativo, y $1100001\rfloor_2$ es el significando en valor absoluto.

Ventajas y desventajas: Posee un rango simétrico de $n-$bits, el rango decimal va de -2^{n-1} hasta 2^{n-1}, usando $n = 8$, los números van del $-127\rfloor_{10} = -\mathbf{0}1111111\rfloor_2$ hasta $127\rfloor_{10} = \mathbf{1}1111111\rfloor_2$ (Acordate que el primer byte es para el signo, por eso lo puse en negrita).

Con esta ventaja podemos apreciar fácilmente las desventajas que se tienen:

1. No permite operar aritméticamente ya que se obtienen resultados incorrectos. Ej. $00010101_2 + 11100001_2 = 11110110_2 (-118\rfloor_{10}) \neq -76\rfloor_{10}$. Pasando cada número a decimal seria $+21\rfloor_{10} + -97\rfloor_{10} = -76\rfloor_{10}$

2. Posee doble representación del cero. Al representar en Signo y Magnitud, aparece el cero signado: $\mathbf{0}0000000\rfloor_2 (+0\rfloor_{10})$ y $\mathbf{1}0000000\rfloor_2 (-0\rfloor_{10})$.

2.1.2 Complemento a 1

Este método es otra alternativa para representar números negativos y consiste en aplicarle un **NOT bit a bit** al número, es decir en otras palabras, la inversión de unos por ceros y ceros por unos. De esta forma, la representación por Complemento a Uno de un número signado de n-bits es:

1. un bit para representar el signo. Ese bit a menudo es el bit más significativo y, por convención un 0 es para los positivos, y un 1 para negativos

2. los $n-1$ bits restantes para representar el significando que es la magnitud del número en valor absoluto para el caso de números positivos, o bien, es el complemento a uno del valor absoluto del número, en caso de ser negativo.

Ejemplo Representar $-97\rfloor_{10}$ en 8 bits a complemento a 1

El ejemplo es el mismo del anterior, que sabemos que la representación es este $11100001\rfloor_2$. Ahora es sencillo, intercambiamos los 1's por 0's (excluyendo el bit para el signo) $\rightarrow$ $\mathbf{C1}(1100001) = 001111$. Entonces la representación es 1001111 en complemento a 1.

Ventajas y desventajas:

- Desventajas:

    1. Doble representación del 0: $00000000\rfloor_2 = +0\rfloor_{10}$, $11111111\rfloor_2 = -0\rfloor_{10}$

- Ventajas:

  1. Posee un rango simétrico de $n-$bits, el rango decimal va de -2^{n-1} hasta 2^{n-1}, usando $n = 8$, los números van del $-127\rfloor_{10} = -\mathbf{0}1111111\rfloor_2$ hasta $127\rfloor_{10} = \mathbf{1}1111111\rfloor_2$
  2. Permite operar aritméticamente y para obtener el resultado correcto al operar se debe sumar el acarreo obtenido al final de la suma/resta realizadas en caso de haberlo obtenido, este acarreo se lo conoce con el nombre de **end-around carry**. Por ejemplo:
     (a) $00010101\rfloor_2 + 10011110\rfloor_2 = 10110011\rfloor_2$ (+2110 + -9710 = -7610) end-around carry = 0;
     (b) $00000010\rfloor_2 + 11111110\rfloor_2 = \mathbf{1}00000000\rfloor_2$ (+2 + -1 = 0 $\neq$ +1), end-around carry = **1**; Y que pasó? Podemos ver que con la cuenta que hicimos, nos pasamos de bits, entonces usamos el carry para arreglarlo, y cómo? Asi $00000010\rfloor_2 + 11111110\rfloor_2 = 00000000\rfloor_2 + \mathbf{1}\rfloor_2 = 00000001\rfloor_2$, que es el resultado correcto de la suma.

2.1.3 Complemento a la base

El complemento de un número dado en una base es aquel que sumado al número original da la base a la n, siendo n la cantidad de dígitos que componen a ese número.

Formalmente, el complemento a b de un número r, representado en n dígitos en base b se define como:

$$C_b(r_b) = (10_n...0_1) - r_b$$

Fijemonos que el número que se utiliza como minuendo tiene un dígito más que los usados en la representación, es decir $n + 1$ dígitos, un 1 seguido de n ceros a la derecha. Por ejemplo: Usando el complemento a 10 de $13579\rfloor_{10} \rightarrow C_10(13579\rfloor_{10}) = 100000–13579 = 86421\rfloor_{10}$

Algo más es que el concepto de complemento puede ser usado para transformar una operación de resta de dos números (A y B), en la suma de uno de ellos más el complemento del otro. ¿cómo es esto? veamos un ejemplo:

$$A - B = d \tag{1}$$
$$A - B + b^n = d + b^n \tag{2}$$
$$b^n - B = B_{\text{comp}} \tag{3}$$
$$A + B_{\text{comp}} = d + b^n \tag{4}$$

d es solamente el resultado de la diferencia entre A y B.

Ejemplo 1: $A = 10376\rfloor_{10}$ y $B = 234\rfloor_{10}$ con $n = 5$

Para poder calcular esto es sencillo, los pasos son calcular B_{comp} y de ahí calcular d en la ecuación (4).

$$b^n - B = B_{\text{comp}} \tag{1}$$
$$10^5 - 234\rfloor_{10} = B_{\text{comp}} \tag{2}$$
$$99766\rfloor_{10} = B_{\text{comp}} \tag{3}$$

Ahora a B_{comp} la reemplazamos en la (4) y despejamos d.

$$10376 + 99766 = d + 10^5 \tag{4}$$
$$110142 = d + 10^5 \tag{5}$$
$$10142\rfloor_{10} = d \tag{6}$$

Ejemplo 2: $A = 101\rfloor_2$ y $B = 1000\rfloor_2$ con $n = 4$

¿Cuál es la diferencia entre este y el primero? La diferencia es que $B > A$ entonces la ecuacion ya no es $A - B$ sino $B - A$ y a A le vamos a buscar el complemento.

$$b^n - A = A_{\text{comp}} \tag{7}$$
$$10^5 - 101\rfloor_2 = A_{\text{comp}} \tag{8}$$
$$1011\rfloor_2 = A_{\text{comp}} \tag{9}$$

Ahora a A_{comp} la reemplazamos en la (4) y despejamos d.

$$1000 + 1011 = d + 10^4 \tag{10}$$
$$10011 = d + 10^5 \tag{11}$$
$$11\rfloor_2 = d \tag{12}$$

2.1.4 Complemento a 2

Permite representar números negativos en el sistema binario y la realización de restas mediante sumas. Tener en cuenta que estos números negativos están

representados a través de su complemento. Es decir un número más su negativo (inverso aditivo), nos da un único cero $\rightarrow a + (-a) = 0$

El complemento a 2 de un número X_2 se obtiene de sumar 1 al número negado bit a bit de X_2, en otras palabras $\rightarrow Not(X) + 1$

Ejemplo: $C_2(01101_2) = 10010 + 00001 = 10011\rfloor_2$

Supongamos que queremos representar el número -97_{10} usando 8 bits ($n = 8$). Procedemos a:

a) Tomar nota del signo del número -97_{10} que, siendo negativo, llevará como bit de signo un 1;

b) Como el signo es negativo, el número deberá expresarse en complemento a dos. Aplicamos el Not al valor absoluto y le sumamos 1, en otras palabras: $\text{Not}(97_{10}) + 1$. Que en binario es: $\text{Not}(01100001_2) + 1 = 10011110_2 + 1 = 10011111_2$ (complemento a dos);

Para el caso inverso, dado un número binario en Complemento a 2, por ejemplo, 10110101_2, y teniendo en cuenta que $n = 8$, lo que debemos hacer es:

1) Analizar el bit más significativo, que siendo un 1 indica que el número es negativo;

2) Convertir el significando a la base deseada, por ejemplo, en decimal, tomando en cuenta que: el valor obtenido está en valor absoluto, que la magnitud real estará dada por el bit de signo obtenido antes, y que en caso de ser bit de signo negativo (como es el caso) se deberá obtener el complemento a dos: $C2(10110101) = \text{Not}(10110101)+1 = 01001010_2+1 \rightarrow 01001011_2 = |75|_{10}$.

3) Siendo que el bit de signo es 1, el número real es -75_{10}. Si el bit de signo fuese 0, el número hubiese sido $00110101_2 = +53_{10}$ (sin complementar a dos).

Desventajas:

- Posee un rango asimétrico de números $\rightarrow$ los números van del $+127_{10}$ (01111111_2) pasando por el $+0_{10}$ (00000000_2). Pero el 11111111_2, ya no es -0_{10} como en la representación anterior, sino que es -128_{10} y al llegar al 10000000_2 nos encontramos con que el complemento a 2 de 10000000_2 es el mismo valor 10000000_2. Por lo tanto, por convención, se asigna a este número particular el valor de -128_{10} (para 8 bits). Generalizando el rango decimal para n-bits, usando el Complemento a 2 es $(-2^{n-1}; 2^{n-1} - 1)$.

Ventajas:

- No posee doble representación del cero.
- Permite operar aritméticamente.

2.1.5 Exceso

Otro método para representar un número signado de n-bits consiste en:

1. Tomar el valor real del número a representar;
2. Sumarle la base elevada según la cantidad de dígitos menos 1 que se tienen disponibles;

Esto se lo conoce como representación en Exceso a base (b^{n-1}), puesto que a cada número se le suma el mismo valor y está en exceso por dicho valor. El formato en exceso es habitual para la representación del exponente en números en punto flotante, por ejemplo, para la norma IEEE-754, con una salvedad ya que el exceso es calculado de la siguiente manera: $x + 2^{n-1} - 1$.

Ejemplo de Exceso 2^{n-1}: Siendo $n = 8$ la cantidad de bits disponibles, los números serán representados en Exceso de $2^{8-1} = 128_{10}$. Cabe destacar que, con 8 bits, podemos representar $2^8 = 256$ números.

Ahora, supongamos que tenemos que representar el número -97_{10} (decimal), la manera de representarlo usando Exceso 2^{n-1} es la siguiente:

a) Tomar el número -97_{10} y sumarle el exceso, en este caso $128_{10} \rightarrow -97_{10} + 128_{10} = 31_{10}$;

b) Convertimos a binario $\rightarrow 31_{10} = 00011111_2$.

Para el caso inverso, dado un número binario en Exceso 128_{10}, por ejemplo, 10110101_2, procedemos a:

1) Convertir el número a la base deseada $\rightarrow 10110101_2 = 181_{10}$;

2) Pero el valor obtenido está en exceso a 128, entonces debemos quitarle dicho exceso, restando 128 $\rightarrow 181_{10} - 128_{10} = 53_{10}$.

Ventajas y desventajas de Exceso 2^{n-1}:

- **Desventajas:**
  - Requiere de operaciones aritméticas intermedias para su obtención, y de cambiar el número de bits se deben actualizar dichas operaciones intermedias para reflejar el nuevo exceso.
  - Posee un rango asimétrico que va desde $+127_{10} = 11111111_2$ hasta el $-128_{10} = 00000000_2$. Generalizando el rango en decimal para exceso es $(-2^{n-1}; 2^{n-1} - 1)$.
- **Ventajas:**
  - No hay empaquetación del número. Esto significa que no hay que recordar que partes del número son signo y valor, sino que el número está formado por los n-bits.
  - Permite operar aritméticamente, teniendo en cuenta que cada operación lleva asociado su exceso y debemos restarlo al resultado final para obtener la correcta representación. Por ejemplo:
    - $00011111_2 + 10110101_2 = 11010100_2$
    - $-97_{10} + 53_{10} = 84_{10} \neq -44_{10}$ Al resultado obtenido debemos quitarle dicho exceso: $\rightarrow 84_{10} - 128_{10} = -44_{10}(1010100_2)$

2.2 Conceptos de Formato y Configuración

Es importante tener en claro los conceptos de Formato y Configuración. Cuando hablamos de **Formato** nos referimos a la representación computacional de un número. Ejemplos:

- Binario de Punto Fijo sin Signo
- Binario de Punto Fijo con Signo
- Empaquetado
- Zoneado
- Binario de Punto Flotante

En cambio, la **Configuración** es la representación en una determinada base de un número en un formato. Ejemplo: $15_{10} = 1111_2 \rightarrow$ Usando 16 bits, 0000000000001111_2 es la configuración binaria de un BPF sin signo.

2.3 Conceptos de Expansión y Truncamiento

Cuando hablamos de **Expandir formato**, nos referimos a completar su representación computacional sin alterar el número representado en el mismo. Mientras que, cuando hablamos de **Truncar formato**, nos referimos a descartar dígitos de su representación sin alterar el número representado en el mismo.

2.4 Formatos de representación de números enteros

2.4.1 Binario de punto fijo sin signo

2.4.2 Binario de punto fijo con signo

2.4.3 BCD Empaquetado

2.4.4 Zoneado

2.5 Formatos de representación de caracteres

2.5.1 ASCII

2.5.2 EBCDIC

2.5.3 UNICODE

2.6 Formatos de representación de números decimales

2.6.1 Binario punto Flotante IEEE 754

3 Maquina elemental

4 Arquitectura del conjunto de instrucciones

5 Lenguaje ensamblador

6 Componentes de un computador

7 Almacenamiento secundario