Arquitectura de ordenadores

4-2. Aritmética II

Ignacio Calles González ignacio.gonzalez@ext.live.u-tad.com

Tiago Manuel Louro Machado de Simas tiago.louro@u-tad.com

Francisco Javier García Algarra javier.algarra@u-tad.com

Carlos M. Vallez Fernández carlos.vallez@u-tad.com

2023-2024

Índice

1. Operaciones aritméticas básicas bin

En apartados posteriores se tratará como se realizan internamente las operaciones dentro de un ordenador en función de la representación que empleen.

En este apartado nos centraremos en recordar las operaciones aritméticas binarias básicas (suma y resta) entre números codificados en binario puro.

Este capítulo es, por tanto, la base para lo que explicaremos en posteriores apartados.

1.1 Suma binaria

En cierto modo es semejante a la suma con números decimales con la salvedad de que en binario solo tenemos dos símbolos el 0 y el 1. Recordad que también existe el concepto de “acarreo”.

Tablas de sumar:

Tabla del 0: 0+0 =0

0+1 =1

Tabla del 1:

1+0 =1

1+1 =10 (equivale a decir que a 0 con un acarreo de 1)

1.2 Resta binaria

También es semejante a la resta con números. Solo tiene dos símbolos el 0 y el 1 y al ir realizando las restas parciales entre dos dígitos, si el sustraendo fuese mayor que el minuendo se sustrae una unidad del dígito o posición situado más a la izquierda en el minuendo. Es el concepto justamente opuesto al acarreo.

Tablas de restar:

Tabla del 0: 0-0 =0

0-1 = No cabe, habrá que sustraer a la izquierda

Tabla del 1:

1-0 =1

1-1 =0

Índice

2. Otras representaciones binarias

Con el sistema binario, cualquier número puede representarse mediante 0 y 1, la coma (o punto) para los números decimales y un signo menos (-).

No obstante, un procesador es únicamente capaz de manejar 0's y 1's. No es posible representar comas ni signos.

Para solucionar este problema, se usan diferentes sistemas de representación que trabajan sobre los números en binario, añadiendo información de su signo y coma.

2.1 Signo-Magnitud

Con este sistema de representación, el bit situado más a la izquierda es el signo (= para e signo + y 1 para el -). El resto de bits (N-1 siendo N el número de cifras o dígitos de la cantidad) representan la magnitud del número entero.

Signo (1 bit)	Magnitud (mantisa) (n-1 bits)

$+18 = 00010010$
$-18 = 10010010$ (sign magnitude)

000	+0
001	+1
010	+2
011	+3
100	-0
101	-1
110	-2
111	-3

Codificación en S-M de tres bits

2.1 Signo-Magnitud

El valor del número A (a_{n-1}, a_{n-2}, a_{n-3}, …, a_1, a_0) en decimal, se calcula mediante:

$$A = \begin{cases} \sum_{i=0}^{n-2} 2^i a_i & \text{if } a_{n-1} = 0 \\ -\sum_{i=0}^{n-2} 2^i a_i & \text{if } a_{n-1} = 1 \end{cases}$$

$4785 = 4x10^3 + 7\,x10^2 + 8x10^1 + 5x10^0$

$-4785 = -(4x10^3 + 7\,x10^2 + 8x10^1 + 5x10^0)$

2.1 Signo-Magnitud

Limitaciones que tiene esta representación:

- El signo va por separado de la magnitud. Hace difíciles los cálculos en un computador.
- Hay dos representaciones para el número 0. Ej. Una representación de 8 bits:
  - + 0 = 0 0000000
  - - 0 = 1 0000000

Como ventaja que presenta este sistema de representación frente a otros está la de poseer un rango de representación simétrico. El rango de representación es el conjunto de números representables. Así por ejemplo:

Con 8 bits el rango va de -127 a + 127

Para 16 bits de -32767 a 32767

2.2 Representación modular

Acabamos de comentar que los ordenadores representan los números enteros con una cantidad finita de bits. Cuando programamos en C, si usamos una variable unsigned short, ocupa 16 bits de memoria y su valor varía entre 0 y 65535, que es 2^{26} -1. Si intentamos guardar el número 100000 en esa variable, lo que obtendremos es un 34465 que equivale a 100000 en módulo 2^{26} -1.

Toda la aritmética en los ordenadores es modular. Eso supone una limitación, Pero también tiene ventajas. En aritmética modular, la resta y la suma son la misma operación si se elige de una forma adecuada la representación de los números negativos.

En aritmética modular, con módulo N, dos números A y B se dice que son equivalentes si se cumple que : A mod N = B mod N, y se escribe como (A=B) mod N. Si se cumple esta condición entonces existe un número entero tal que A= B + KN.

2.2 Representación modular

.

Ejemplo: pensemos en los números módulo 100. Disponemos de un ordenador que solo puede representar en un tipo de variable decimal las cifras de 00 a 99. En estas condiciones los números 72 y 472 son equivalentes módulo 100, puesto que el 72 módulo 100 = 472 módulo 100 (72 =472)mod 100. Se cumple que 72= 472 + K *100 => K =-4.

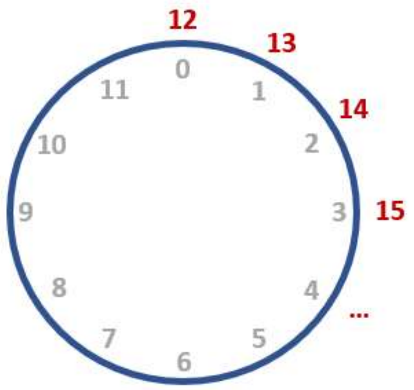
Definimos ahora la suma modular de 2 números enteros como la función siguiente f (A, B) = (A+B) mod N.

Existe un ejemplo cotidiano de suma modular, el reloj analógico. El reloj tiene una esfera que es un sumador módulo 12.

2.2 Representación modular

.

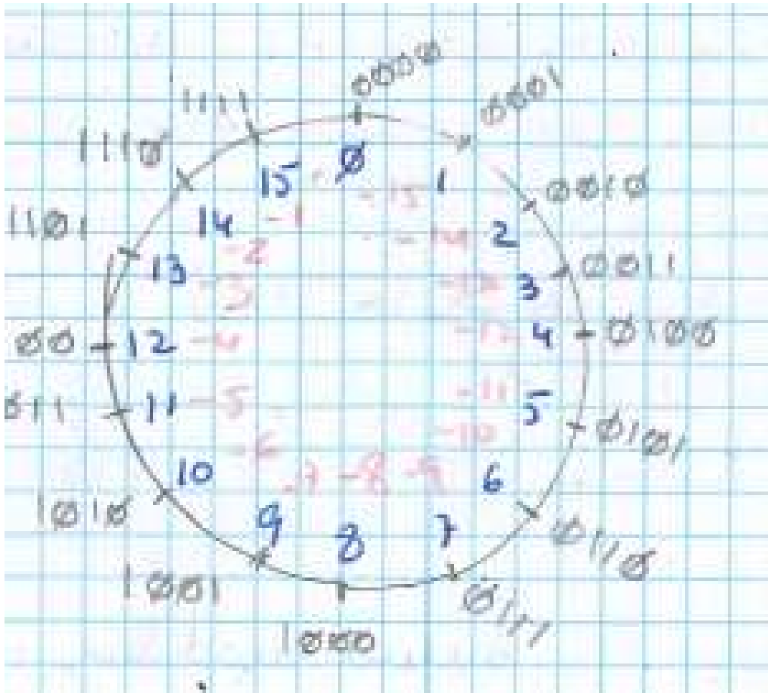
El reloj analógico:



Si ahora son las 10:00 h de la mañana y queremos saber qué hora será cuando pasen otras 5 podemos indicarlo en la métrica sin módulo como las 15 o en la manera Tradicional como las 15:00 h de la tarde. esto ocurre porque (10 +5) mod 12= 3 y por tanto (15=3) mod 12.

Se dice que 15 y 3 son congruentes módulo 12.

2.2 Representación modular



Sin pérdida de generalización pensemos ahora en N bits Que son capaces de representar 2^n números. Estamos trabajando en módulo 2^n. Para el caso particular en el que N=4, entonces el módulo es 16.

2.2 Representación modular

En el “reloj” hemos representado los valores binarios y las posiciones de los números 0 a 15 en azul y - 1 a - 15 en rojo. Puede comprobarse que (-3 mod 16)=13.

El problema ahora consiste en representar un conjunto de enteros (positivos y negativos) alrededor de cero. Para n bits, podemos representar 2n enteros, La mitad positivos incluyendo el cero en este conjunto y la mitad negativos.

En C, si definimos una variable como short int, podemos guardar números desde -32767 a 32767. Los positivos irán desde 0 a 2^{n-1} -1, para el caso particular de n= 4, si llamamos K a ese número se cumple que :

0 < k <= 2n-1 => 0<k <=7

Queremos encontrar una representación negativa del número (-k). Por la definición de la suma modular existen dos enteros m>=0 y p>0 tales que:

(-k+m =p+m) mod 2n

2.2 Representación modular

Y a todos los efectos podemos usar p como la representación modular de -k. Restamos m de ambos lados de la ecuación:
$(-k = p) \bmod 2^n$

Y ahora recurrimos a un truco. Como trabajamos con mod 2n, Se cumple que (2n =0) mod 2n, Así que sumamos en el lado izquierdo de la ecuación:

$$(2^n - k = p) \bmod 2^n$$

Por tanto, la representación modular de -k es 2^n-k trabajando con módulo 2^n.

Hemos dicho que $0<k<=2^{n-1}$ y $p = 2^n - k$ => $k = 2^n - p$

$0 < 2^n - p <= 2^{n-1}$

$0 > p - 2^n >= -2^{n-1}$

$2^n > p >= 2^{n-1}$ Estos son precisamente los valores del "reloj" que no hemos usado y cuyo bit más pesado o significativo es 1.

2.3 Complemento a 2

Sea K un número positivo, se dice que p es su número complemento a dos si se cumple que :

$(k+p=0) \bmod 2^n$

Este es el número que encontramos en el apartado anterior:

$P= 2^n -k$

Donde n es el número de bits que emplearemos

2.3 Complemento a 2

En la representación Complemento a 2 para representar números enteros con signo, al igual que en S-M, se utiliza el primer bit para codificar el signo. Varía la forma de codificar la magnitud: y con ello conseguimos ofrecer sólo una codificación para el 0.

2.3 Complemento a 2

Ejemplo Codificación de 4 bits:

Decimal	S-M	Complemento a dos
+8	-	-
+7	0111	0111
+6	0110	0110
+5	0101	0101
+4	0100	0100
+3	0011	0011
+2	0010	0010
+1	0001	0001
0	0000	0000
-0	1000	-
-1	1001	1111
-2	1010	1110
-3	1011	1101
-4	1100	1100
-5	1101	1011
-6	1110	1010
-7	1111	1001
-8	-	1000

Además de la ventaja de tener un solo 0, hay que notar que en este caso el rango no es simétrico (es por tanto asimétrico).

Para 8 bits: desde-128 a + 127 (nótese que el 0 cae en los positivos)

2.3 Complemento a 2

¿Cómo calculamos el valor decimal de un número binario expresado en complemento a 2?

El valor decimal se calcula mediante:

$$A = -2^{n-1}a_{n-1} + \sum_{i=0}^{n-2} 2^i a_i$$

Cuando el número es positivo:

$$A = -2^{n-1}\cancelto{0}{a_{n-1}} + \sum_{i=0}^{n-2} 2^i a_i$$

2.3 Complemento a 2

Aunque resulte engorrosa, esta representación facilita la realización de las operaciones aritméticas más importantes: la suma y la resta.
Para facilitar la conversión entre un número decimal y complemento a dos, se puede usar una caja de n posiciones:

−128	64	32	16	8	4	2	1

Codificación en Ca2 de ocho bits

2.3 Complemento a 2

En las n-1 posiciones (desde la derecha), se coloca en primera fila las potencias de 2 asociadas a esas posiciones.
En la posición n, se coloca en la primera fila la potencia 2^n-1, con el signo menos.
En la fila inferior se coloca el número en complemento a dos. El resultado decimal se obtiene sumando los valores de la primera fila que tengan un 1 en la segunda fila:

−128	64	32	16	8	4	2	1
1	0	0	0	0	0	1	1

−128 +2 +1 = −125

2.4 Si se desea convertir un número de longitud n a longitud m

- **En S-M**: Se traslada el bit de signo hasta la nueva posición más a la izquierda y se rellena con 0s.

+18	=	00010010
+18	=	0000000000010010
−18	=	10010010
−18	=	1000000000010010

- **En complemento a dos**: Se traslada el bit de signo hasta la nueva posición más a la izquierda y:
  - **Si el número es positivo**: se rellena con 0s.
  - **Si el número es negativo**: se rellena con 1s.

+18	=	00010010
+18	=	0000000000010010

−18	=	11101110
−18	=	1111111111101110

La extensión de signo es necesaria para realizar las operaciones matemáticas de dos Números. Los signos tienen que estar alineados, en la misma posición.

2.5 Complemento a 1

Sea k un número positivo, se dice que q es su complemento, a 1 si se cumple que:

$$(k+q=0) \bmod 2^n - 1$$

y siguiendo el razonamiento de la deducción vista en apartados anteriores:

$$\mathbf{q = 2^n - 1 - k}$$

De forma resumida, el negativo de un número se obtiene cambiando (complementando) todos sus dígitos (ceros por unos y viceversa) incluido el bit de signo:

Número 10 : 00001010 donde el cero de más ala izquierda indica el signo

Número -10: 11110101

Este sistema de numeración tiene la ventaja de ser simétrico como el S-M pero también tiene la desventaja de tener 2 ceros (el +0 00000000 y el -0 11111111 para 8 bits por ejemplo).

El rango es como en el caso del S-M

Índice

3. Aritmética con enteros y Sist. de Num.

En este apartado vamos a revisar las distintas operaciones y cómo se realizan dependiendo de cada representación. En esta unidad llegaremos exclusivamente hasta la multiplicación sin signo de enteros, dejando para la siguiente unidad la multiplicación con signo, división y operaciones con decimales.

3.1 Negación

Esta operación es básicamente lo que ya hemos visto: Obtener el número negativo equivalente a otro dado.

- En S-M es trivial: basta con cambiar el bit del signo

  Número 10 : 00001010

  Número -10: 10001010

- En complemento a 1 hemos visto que hay que cambiar los 0 por 1 incluidos el del signo:

  Número 10 : 00001010

  Número -10: 11110101

- En complemento a 2:

  Primero: Obtener el complemento (negación) de cada bit (0->1, 1->0).

  Segundo: Sumar 1 al resultado.

3.1 Negación

Esta operación es básicamente lo que ya hemos visto: Obtener el número negativo equivalente a otro dado.

- En S-M es trivial: basta con cambiar el bit del signo

  Número 10 : 00001010

  Número -10: 10001010

- En complemento a 1 hemos visto que hay que cambiar los 0 por 1 incluidos el del signo:

  Número 10 : 00001010

  Número -10: 11110101

- En complemento a 2:

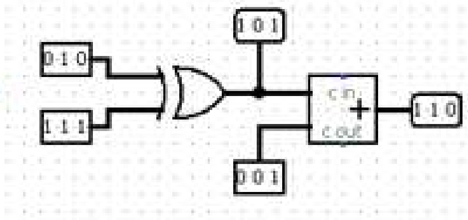
  Primero: Obtener el complemento (negación) de cada bit (0->1, 1->0).

  Segundo: Sumar 1 al resultado.

3.1 Negación

- En complemento a 2:

+18	00010010	
	11101101	
	+ 1	
	11101110	-18



Este proceso se llama **transformación a complemento a dos** u **obtención del complemento a dos de un entero**.

Es equivalente negar todos los dígitos haciendo XOR contra un número con la misma cantidad de dígitos binarios pero lleno de 1s y sumar 1 al resultado.

Haremos algún ejercicio con esto

3.2 Suma y Resta

Primeramente, vamos a demostrar que la suma y resta pueden ser la misma operación en aritmética modular, si se escoge la representación adecuada de los números negativos.

Una vez lo hayamos demostrado procederemos a ver cómo se realiza dicha operación en cada una de las representaciones (complemento 1 y complemento a 2).

Para el caso de S-M se trata de una suma binaria o resta tradicional en la cual habrá que tener en cuenta el valor del signo.

3.2 Suma y Resta

Para demostrarlo volveremos a nuestro ejemplo del reloj módulo 16. queremos hacer la operación 6 – 3. primero encontramos la representación modular 16 del número -3:

p = 16 -3 = 13

A continuación, sumamos modularmente:

(6+13) mod 16 = (19) mod 16=3

- Hay que observar que 13 y -3 son componentes congruentes módulo 16.

3.2 Suma y Resta

Con representación en complemento a 1

Recordamos que si N es un número entero positivo, su representación en complemento 1 a la cual llamaremos $\overline{N}$ es el número que cumple la igualdad:

Ca1(N)= $\overline{N}$ =(2^n-1) -N

Tomemos el caso en el que n=4 (dígitos) Y queremos representar en complemento a 1 el número 6 : Ca1(6) = (2^4 -1) -6 = 15 -6 =9.

3.2 Suma y Resta

Con representación en complemento a 1

El procedimiento es muy simple en binario:

$2^4 - 1 = 1111$

$-6 = -0110$

$Ca1(6) = 9 = 1001$

Resta binaria pura y dura

bit de mayor peso = 1. Indica que es representación de negativo. Recordemos que los positivos van de 0 a $2^{n-1}-1$ y los negativos de 2^{n-1} a $2^{n-1}-1$

3.2 Suma y Resta

Con representación en complemento a 1

Lo único que hay que hacer para complementar a 1, como ya hemos visto, es cambiar cada bit por su complementario y la operación funciona en ambos sentidos. Así pues el número 6 es el 0110 el Ca1(6) es el 1001 y el Ca1(Ca1(6))= 0110 que es el propio 6. La implementación electrónica es muy simple basta poner un negador para cada bit. como hemos comentado en la próxima unidad trabajaremos nuevamente con puertas lógicas y haremos estas comprobaciones.

La resta se convierte en suma del complementario. Esto es A-B = A + Ca1(B).

3.2 Suma y Resta

Con representación en complemento a 1

Veamos primeramente cómo es una suma del número 4 y el 3 cuyo resultado como sabemos será 7:

4 es 0100

3 es 0011

Si sumamos bit a bit obtenemos 0111 que es 7.

Ahora vamos a hacer 4 - 3 que sabemos igualmente que tiene que dar 1

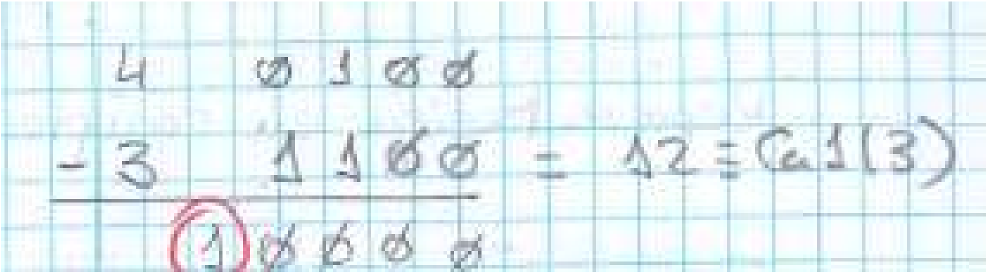
4 es 0100

Ca1(3) = 12= 1100

3.2 Suma y Resta

Con representación en complemento a 1

Sumamos y tenemos:



Se suma el bit de acarreo (el 1 de la izquierda) y obtenemos 0001 =1. Esta propiedad se da porque es una suma mod 2^n -1 , siendo en este caso 16.

3.2 Suma y Resta

Con representación en complemento a 1

Veamos un ejemplo en el cual el resultado sea negativo:

Ejemplo en el que el resultado es negativo

2	0	0	1	0		
− 7	1	0	0	0	=	8 = Ca1 (7)
	1	0	1	0	=	12

negativo

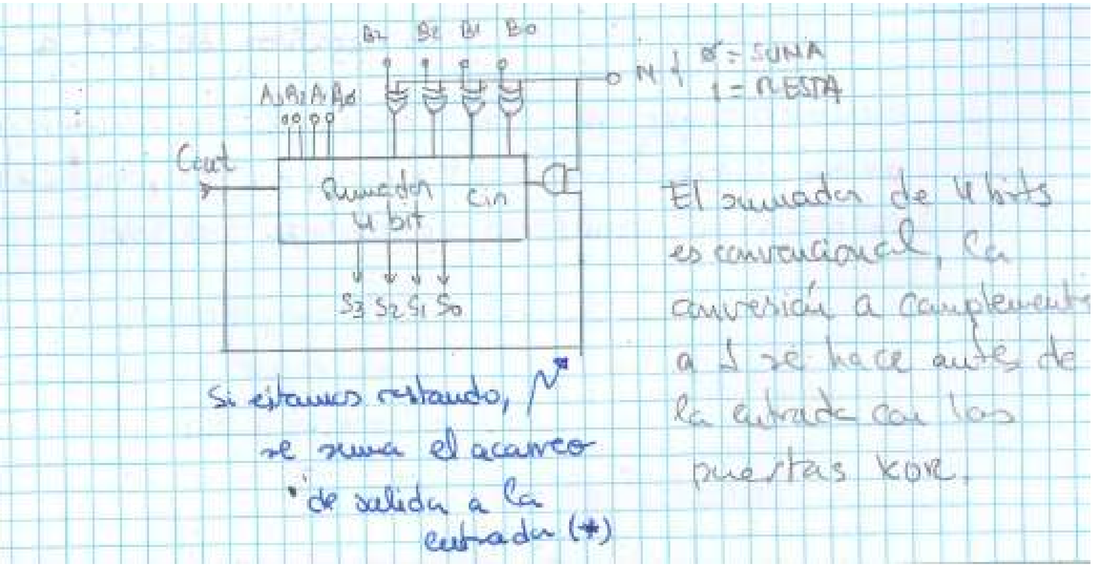
⇓ Ca1

0 1 0 1 = −5

3.2 Suma y Resta

Con representación en complemento a 1

La construcción de un sumador/restador en complemento a 1 es muy simple. basta tener un sumador de n bits, convertir opcionalmente el operando B en negativo complementando a 1 y sumar el acarreo de salida:



3.2 Suma y Resta

Con representación en complemento a 2

Recordamos que si N es un número entero positivo, su representación en complemento a 2 con n bits es: $Ca2(N) = 2^n - N$

De nuevo tomamos n= 4 y queremos representar el -6:

$Ca2(6) = 2^4 - 6 = 16 - 6 = 10$

El procedimiento para complementar a 2 un número es muy simple si nos fijamos en la igualdad:

$$Ca2(N) = Ca1(N) + 1$$

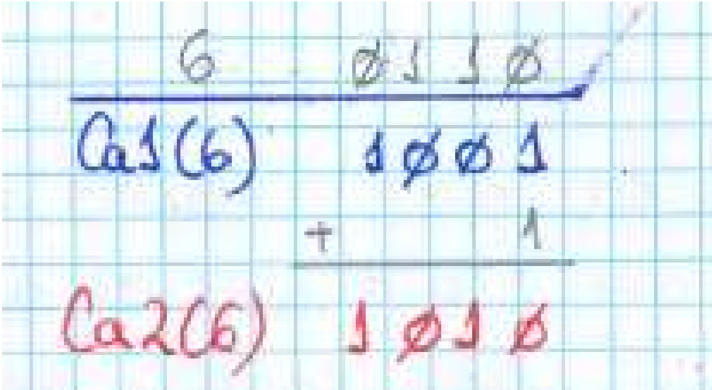
$$2^n - N = (2^n - 1) - N + 1$$

Por tanto, para obtener el complemento a dos de un número, obtenemos su complemento a 1 cambiando cada vez por su complementario y sumamos 1 a esa cifra.

3.2 Suma y Resta

Con representación en complemento a 2

Ejemplo

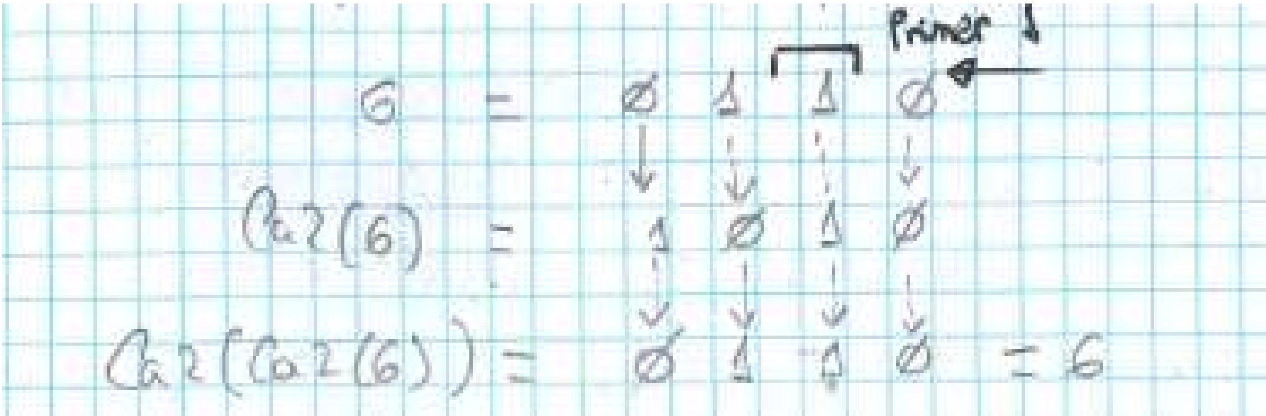


Hay una propiedad importante de complemento a dos de la que carece complemento a 1. en complemento a dos el número cero no tiene una representación complementaria mientras que en complemento a 1 cero puede ser 0000 o 1111. Esto permite, como ya hemos visto, representar un número negativo adicional.

3.2 Suma y Resta

Con representación en complemento a 2

Procedimiento alternativo: se empieza por el bit menos significativo (derecha) y se deja tal cual hasta encontrar el primer 1, que también se respeta. a partir de ese bit todos a su izquierda se complementan. Ejemplo:



U-tad
CENTRO UNIVERSITARIO DE TECNOLOGÍA Y ARTE DIGITAL

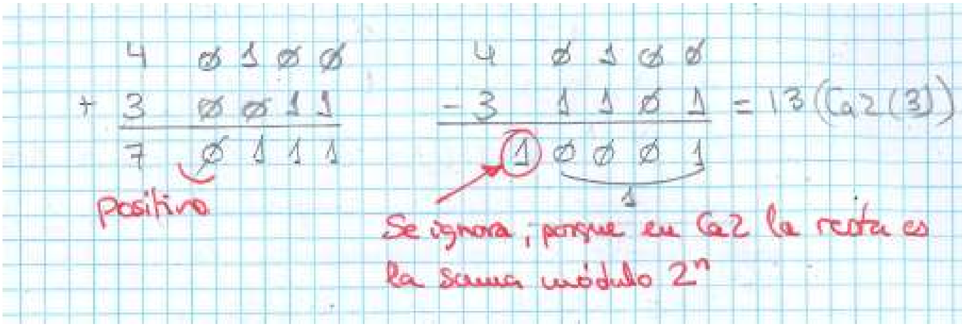
3.2 Suma y Resta

Con representación en complemento a 2

Al igual que pasa en complemento a 1, la resta en complemento a dos se convierte en:

$$A - B = A + Ca2(B)$$

Pero en este caso el acarreo se ignora:



3.2 Suma y Resta

Con representación en complemento a 2

Veamos otro ejemplo, pero con resultado negativo:

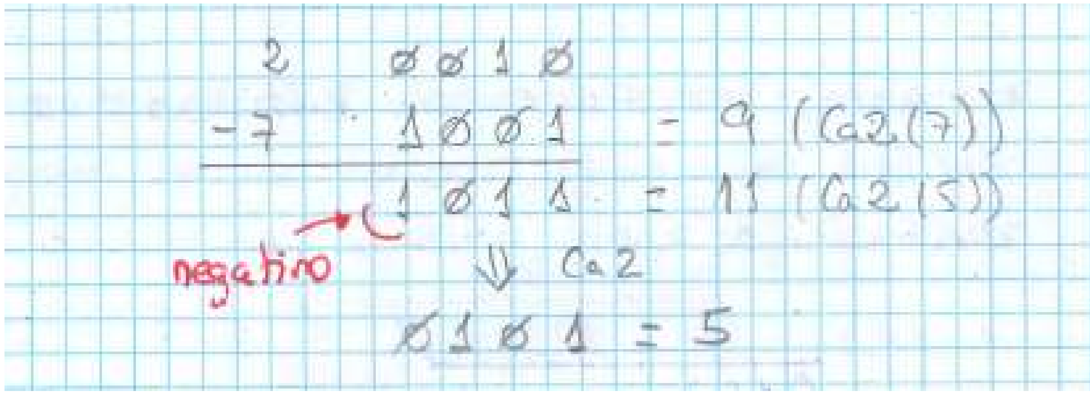
$$
\begin{array}{rcl}
2 & 0010 & \\
-7 & 1001 & = 9\ (Ca2(7)) \\
\hline
 & 1011 & = 11\ (Ca2(5)) \\
 & \Downarrow\ Ca2 & \\
 & 0101 & = 5
\end{array}
$$

negativo

3.2 Suma y Resta

Con representación en complemento a 2

Veamos otro ejemplo, pero con resultado negativo:



El sumador/restador en complemento a 2 es muy simple porque no hay que añadir el acarreo. por eso es la representación más popular en los sistemas digitales. el precio que se paga es que el circuito para pasar un número a complemento a dos es más complejo que el circuito para pasar a complemento a 1.

3.3 Desbordamiento (Overflow)

Desbordamiento (overflow):

Al trabajar con n bits (1 de signo y n-1 de magnitud) los resultados de las operaciones tienen que estar entre: -2^{n-1} <=resultado < 2^{n-1} -1

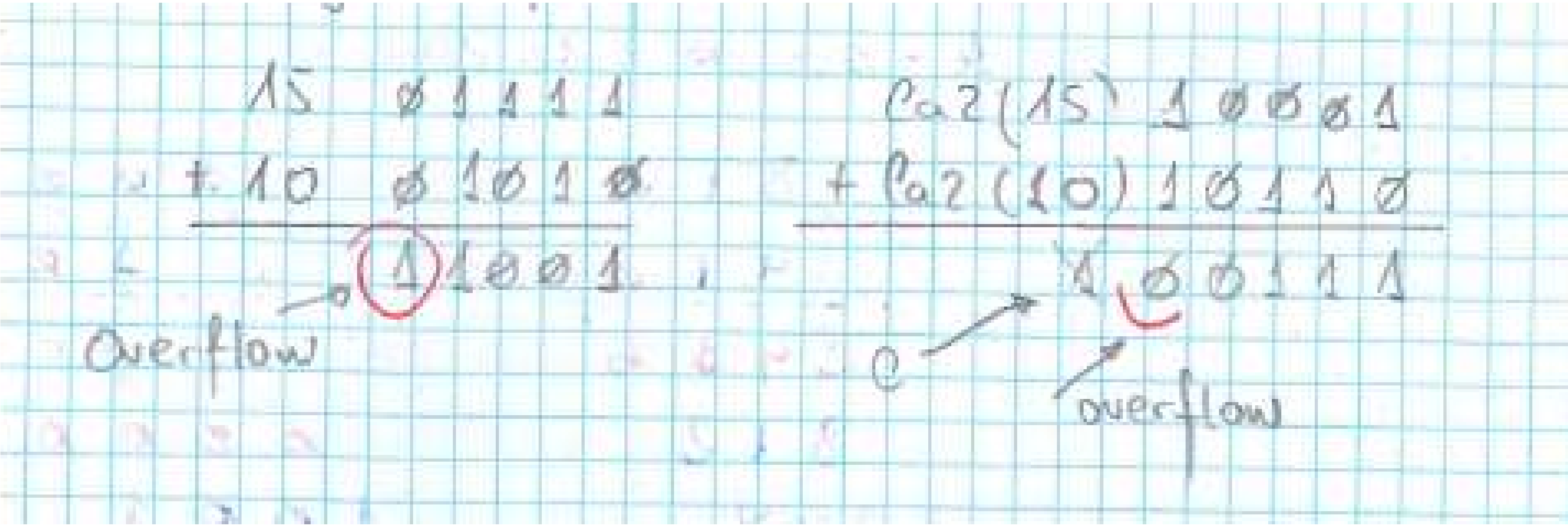
Así por ejemplo para n=5 bit: -16<= resultado <15.

Si salimos de este rango hay overflow y el resultado no será correcto.

La ALU detecta fácilmente el overflow. Si sumamos dos números positivos y obtenemos un negativo o viceversa hay overflow:

3.3 Desbordamiento (Overflow)

Desbordamiento (overflow):



3.3 Desbordamiento (Overflow)

Siempre podemos complementar a dos el resultado para averiguar qué número es como hemos visto en los ejemplos. Existe otro método que permite pasar a decimal en un solo paso: Supongamos que trabajamos con 6 bits:

010001 →

	0	1	0	0	0	1	
×	32	16	8	4	2	1	
		16				+ 1	= 17

positivo. bit de signo

100101 →

	1	0	0	1	0	1	
×	32	16	8	4	2	1	
	-32			4		1	= -27

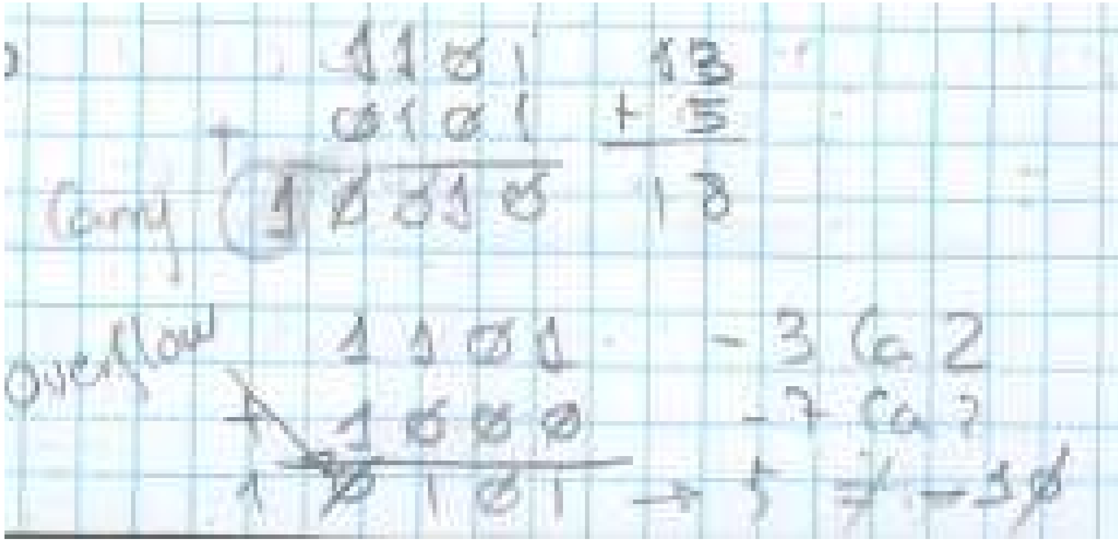
La posición del bit de signo se resta

Comprobemos $27_{(10} = 011011$

$Ca2(27) = 100101$ ok!

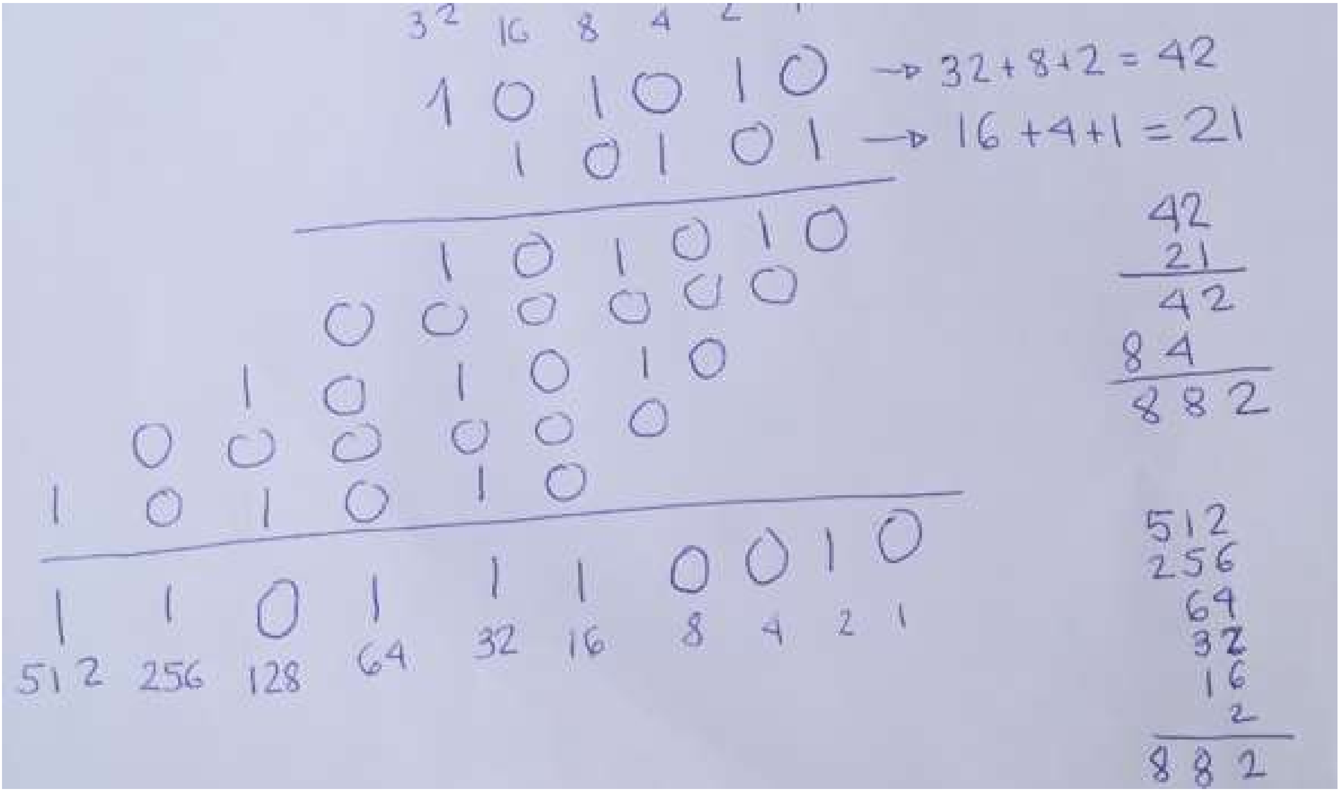
3.4 Diferencia entre Carry y Overflow

Carry es cuando hay desbordamiento del bit más significativo y trabajamos sin signo. overflow es cuando el signo queda es incoherente. Ejemplo:



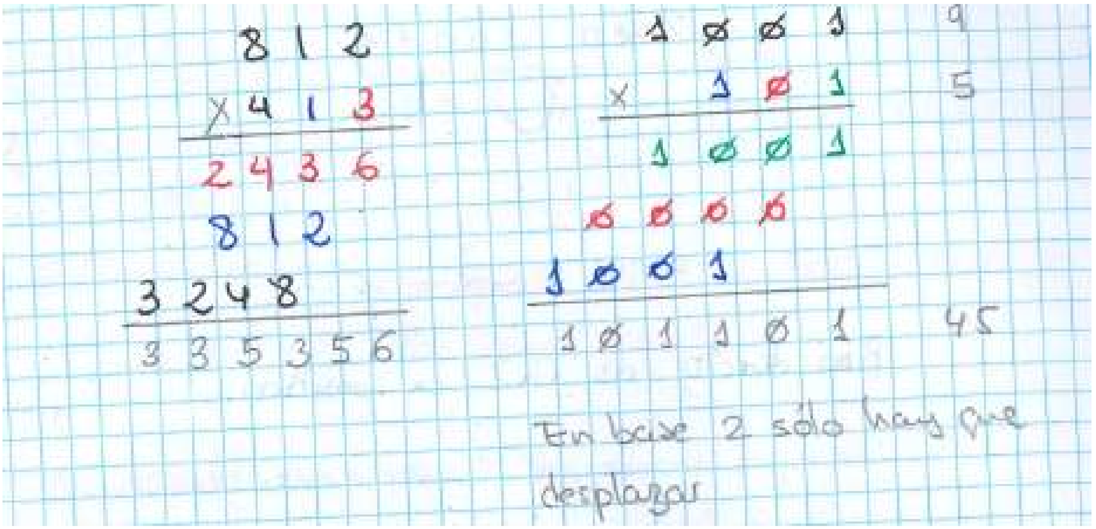
3.5 Multiplicación sin signo

El algoritmo de multiplicación de números positivos binarios es el mismo que en base 10.



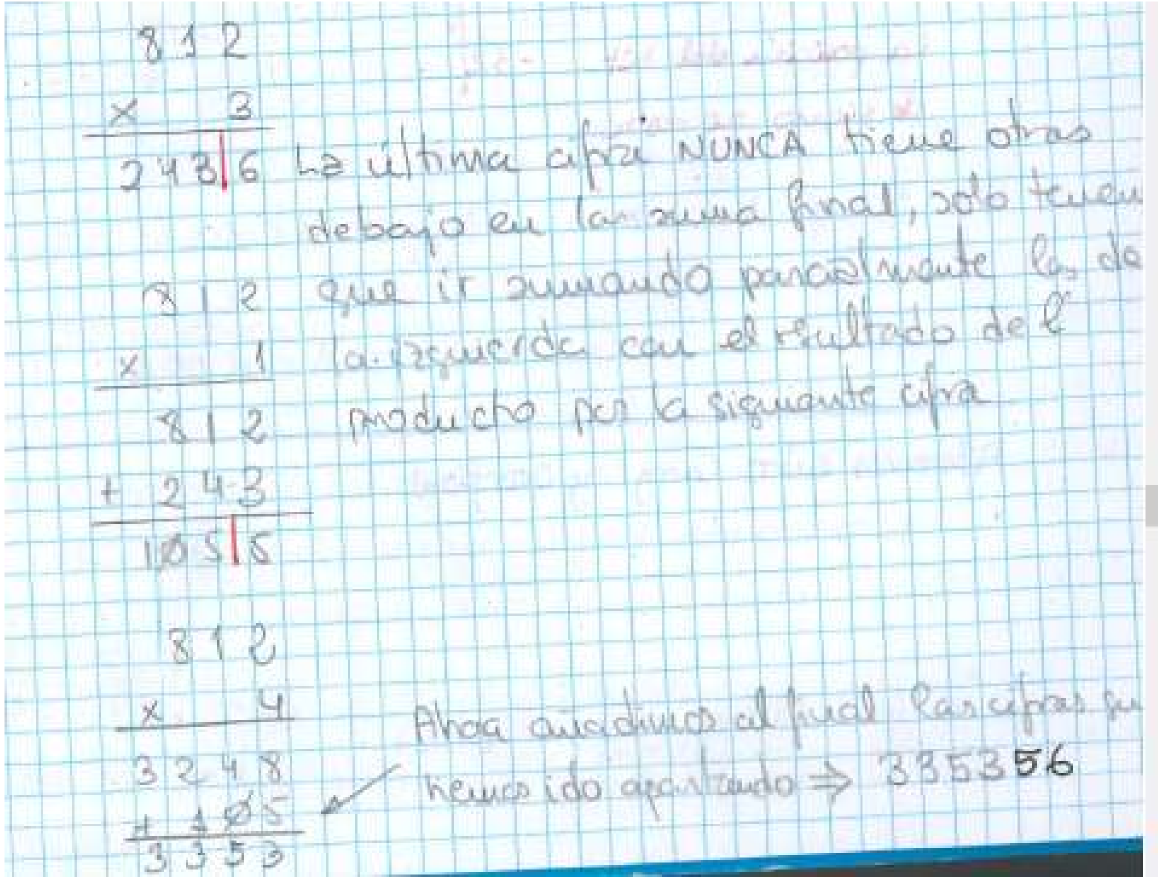
3.5 Multiplicación sin signo

Si multiplicamos un número de m cifras por otro de k cifras necesitamos un registro de salida de m+k. Esto es un problema a la hora de construir una ALU porque en general necesitaríamos un acumulador con el doble de bits que la arquitectura general. para evitarlo se usa una versión modificada que usa varios registros, pero todo es del tamaño de la arquitectura de la ALU.



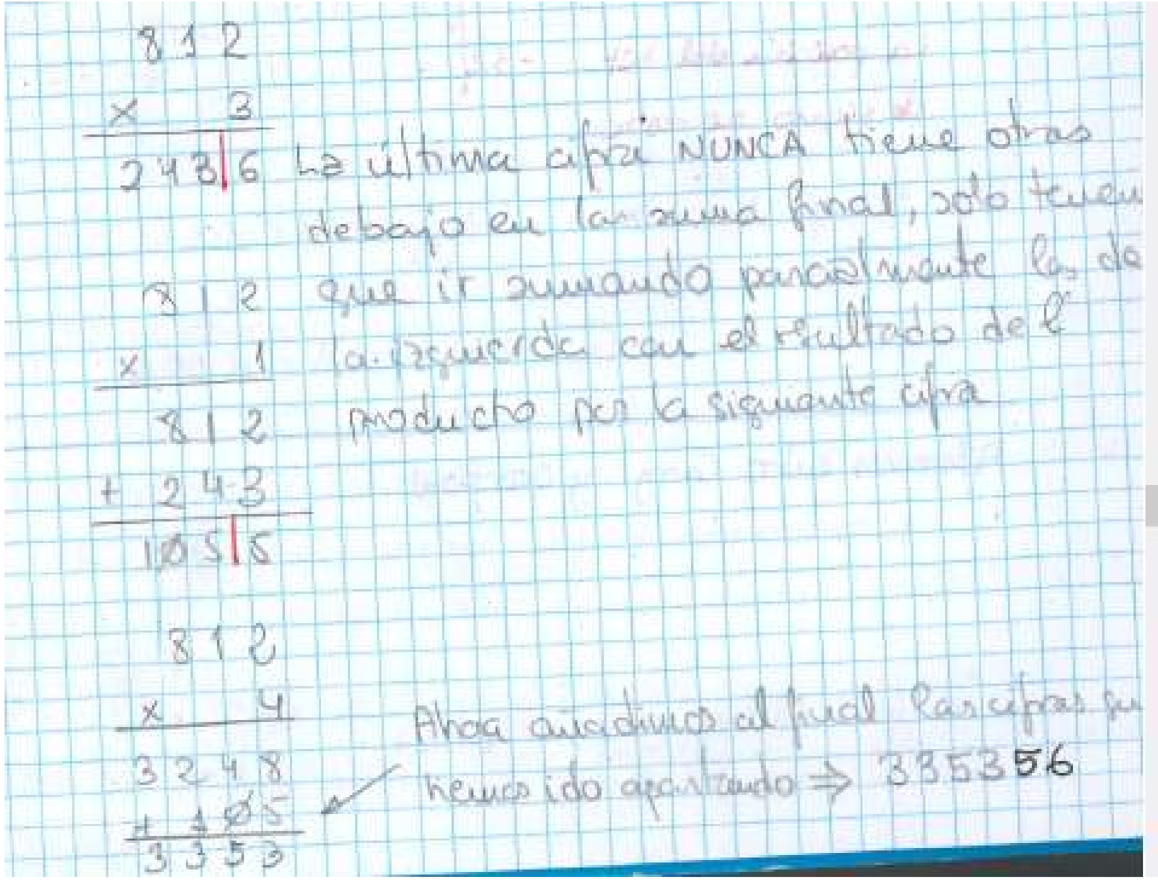
3.5 Multiplicación sin signo

Si lo realizamos por partes en el sistema decimal nos quedaría:



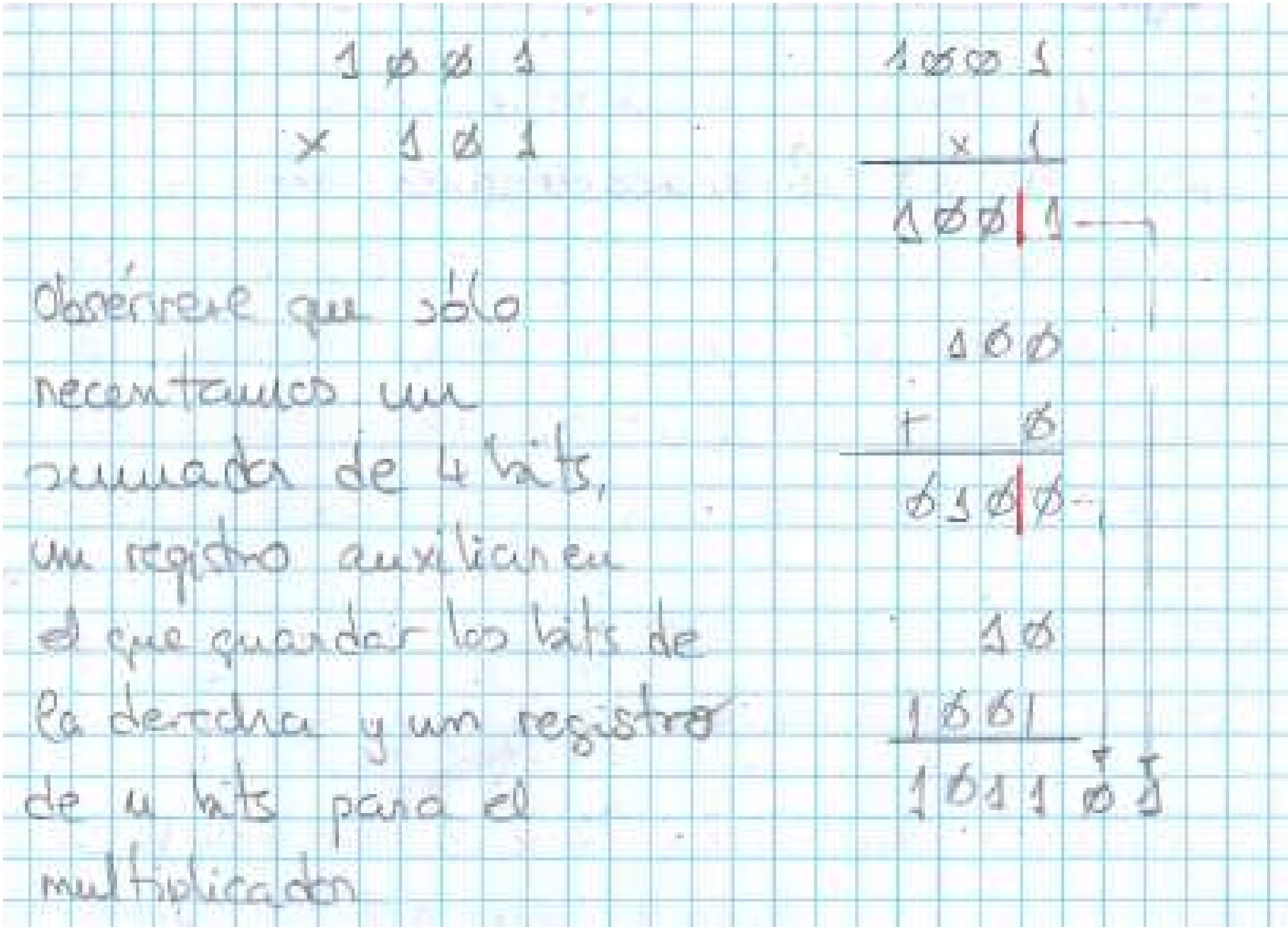
3.5 Multiplicación sin signo

Si lo realizamos por partes en el sistema decimal nos quedaría:



3.5 Multiplicación sin signo

Se puede aplicar el mismo procedimiento para números binarios:



3.5 Multiplicación sin signo

Este algoritmo se conoce como suma-desplazamiento y comprende los siguientes pasos:

Se inicializan dos registros Q y M con el multiplicador y el multiplicando.

Se inicializa un registro A a 0 y un registro de un único bit C a 0.

Si Q_0 es 1:

Se suma A + M y se almacena en A. Si hay acarreo, se anota en C.

Se desplazan todos los bits de C, A y Q una posición hacia la derecha.

$C \rightarrow A_{n-1}$

$A_0 \rightarrow Q_{n-1}$

Q_0 se pierde.

Si Q_0 es 0, sólo se realiza desplazamiento de C, A y Q.

Se vuelve a 3 hasta que se agoten los bits del multiplicador original.

El resultado del producto es la concatenación de A y Q.

Veremos un ejemplo el próximo día en ejercicios