3.- Representación de la información en formato digital (parte 2)

Diseño Lógico 2018

Representación numérica

- Números Naturales:
  - La forma más simple de representarlos es codificarlos con su equivalente binario.
  - Problema: las máquinas tiene numero finito de bits.
  - Tamaños usuales:
    - 1 byte = 8 bits → 0 a 255 (2^8-1)
    - 2 bytes = 16 bits → 0 a 65535 ($2^{16}-1$)
    - 4 bytes = 32 bits → 0 a $2^{32}-1$
    - 8bytes = 64 bits → 0 a $2^{64}-1$
  - El número de bits me indica el número más grande que puedo representar.
    - Para r bits puedo representar N / $0 \leq N \leq 2^r - 1$

Representación numérica

- Aritmética binaria:

  Antes de continuar veremos como operar con base 2.

  - Suma:

    Defino: 0 + 0 = 0
    0 + 1 = 1
    1 + 0 = 1
    1 + 1 = **1** 0

    Acarreo (carry)

    Resultado

    Ejemplo 1:

$$\begin{array}{r} {\scriptstyle 1} \\ 1\,0\,0\,1\,1 \\ +\;\; 0\,1\,0\,1\,0 \\ \hline 1\,1\,1\,0\,1 \end{array}$$

    Ejemplo 2:

$$\begin{array}{r} {\scriptstyle 1} \\ 1\,1\,0\,0\,1 \\ +\;\; 0\,1\,0\,1\,0 \\ \hline \mathbf{1}\,0\,0\,0\,1\,1 \end{array}$$

Representación numérica

- Aritmética binaria:
  - Multiplicación:

Defino: 0 . 0 = 0
0 . 1 = 0
1 . 0 = 0
1 . 1 = 1

Algoritmo: A x B = R

R ← 0

Para i = 0 hasta (largo(B) -1) hago:

Si B(i) = 1 entonces R ← R + A

Desplazo A 1 lugar a la izquierda

próximo i

Ejemplo:

```
        1
    1 0 1 1 0
x       1 0 1
-------------
    1 0 1 1 0
  0 0 0 0 0 _      <- Desplazo 1 lugar
1 0 1 1 0 _ _      <- Desplazo 1 lugar más
-------------
1 1 0 1 1 1 0
```

Se reduce a sumar el multiplicando por cada 1 del multiplicador pero desplazado la posición de los 1's

Representación numérica

- Aritmética binaria:
  - Resta:

Defino: 0 - 0 = 0
1 - 0 = 1
1 - 1 = 0
0 - 1 = - 1

Préstamo (borrow) → **1**0 - 1 = 1

Ejemplo:

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

Representación numérica

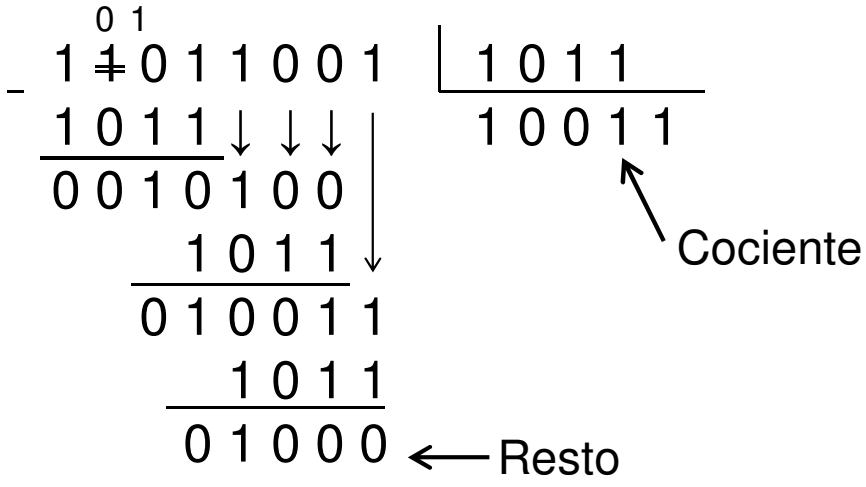
- Aritmética binaria:
  - División:

Será similar a la multiplicación, “pero al revéz”.

Algoritmo de desplazamiento y resta.

Es lo mismo que hacemos en base 10, pero aplicado a base 2.

Ejemplo:



Representación numérica

- Números Enteros:

  ¿Cómo representar los números negativos?

Representación en magnitud y signo

- Es la forma más intuitiva de representar los enteros

S	Magnitud

S=0 números positivos
S= 1 números negativos
Magnitud idem número natural

2 formas de representar el "0"

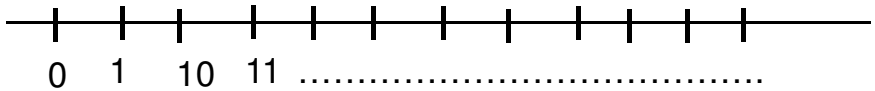
3	011
2	010
1	001
0	000
0	100
-1	101
-2	110
-3	111

- Si bien es fácil y se corresponde con lo que sabemos de números decimales, realizar operaciones con esta representación resulta difícil para las máquinas.

Representación numérica

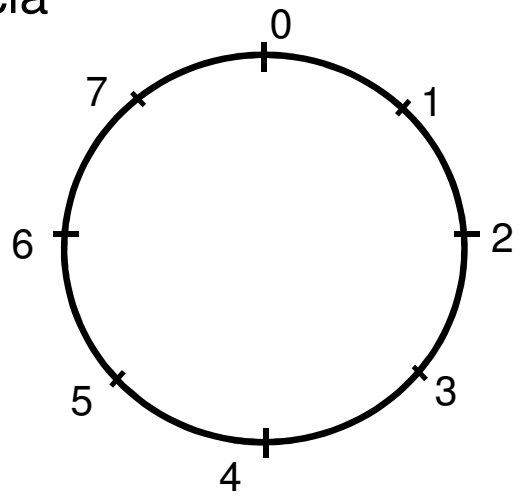
Sistema Normal

La suma la podemos representar sobre una recta



Sistema Modular

La suma lo podemos representar en una circunferencia



Ejemplo módulo 2^3

Observar que $N + 2^3 = N$

Definición:

A y B son equivalentes 2^N si Resto$(A/2^N)$=Resto$(B/2^N)$

Representación numérica

- Números Enteros:

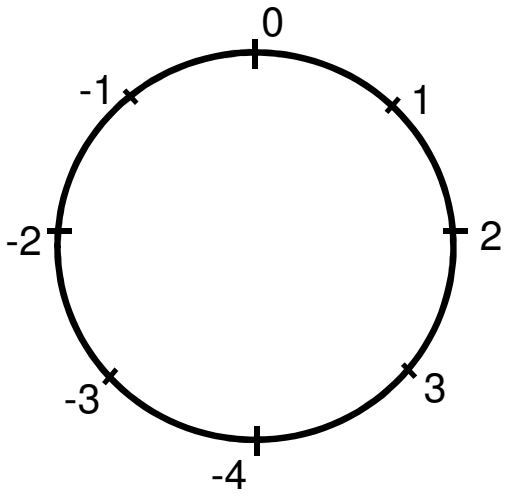
Representación en complemento a 2

- Numero fijo de dígitos
- Única representación para el "0" (código asimétrico)
- El bit más significativo representa el signo
- Las operaciones se "simplifican".
- Rango: Con r bits $-2^{r-1} \leq N \leq 2^{r-1} - 1$
- Ejemplo para 3 bits:

Rango para 3 bits:

$-2^{3-1} \leq N \leq 2^{3-1} - 1$

$-4 \leq N \leq 3$



3	011
2	010
1	001
0	000
−1	111
−2	110
−3	101
−4	100

Representación numérica

- Números Enteros:

Representación en complemento a 2

- Opuesto:
  - Aritmética normal: $A + A' = 0 \rightarrow A' = -A$
  - Aritmética Modulo 2^r: $A + A' = k.\ 2^r$

    Tomamos $k = 1 \rightarrow A' = 2^r - A$

- Por tanto "el opuesto de A" en módulo 2^r es $(2^r - A)$

- Ejemplo:

0 0 1	→ 1
1 1 1	→ - 1
1 0 0 0	0

Se va, pues equivale al módulo 2^3

Representación numérica

- Números Enteros:

Representación en complemento a 2

- Calculo del opuesto de A:

$A = a_{r-1} \ldots\ldots\ldots\ldots a_0$

$2^r = \underbrace{1\ 0\ldots\ldots\ldots\ldots 0}_{r+1\ bit} = \underbrace{11\ \ldots..\ 1}_{r\ bit} + \underbrace{00\ \ldots.\ 01}_{r\ bit}$

$1\text{-}a_i = \begin{cases} 1 \text{ si } a_i=0 \\ 0 \text{ si } a_i=1 \end{cases} = \overline{a}_i$

$2^r - A = 11\ \ldots..\ 1\ + 00\ \ldots.\ 01 - a_{r-1} \ldots\ldots\ldots\ldots a_0 =$

$(11\ \ldots..\ 1 - a_{r-1} \ldots\ldots\ldots\ldots a_0) + 00\ \ldots.\ 01$

$2^r - A = \overline{a}_{r-1} \ldots\ldots\ldots\ldots \overline{a}_0 + 00\ \ldots.\ 01$

➔ Opuesto de A = $2^r - A = \overline{A} + 1$

REGLA:
Para hallar el opuesto de A:
1) Invierto los bits de A
2) Sumo 1

Representación numérica

- Números Enteros:

Representación en complemento a 2

- Calculo del opuesto de A:

Ejemplo:

r= 3

$A = 011 = 3_d$

$\overline{A} = 100$

$\overline{A} + 1 = 100 + 001$

$\overline{A} + 1 = 101$

➔ $(A)_{2c} = 101 = -3_d$

Notación: Opuesto de A en complemento a 2.

REGLA:
Para hallar el opuesto de A:
1) Invierto los bits
2) Sumo 1

$B = 101 = -3_d$

$\overline{B} = 010$

$\overline{B} + 1 = 010 + 001$

$\overline{B} + 1 = 011$

➔ $(B)_{2c} = 011 = 3_d$

Representación numérica

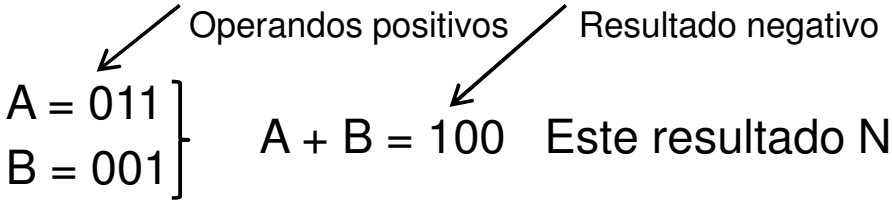
- Números Enteros:

Representación en complemento a 2

Overflow:

Rebase aritmético en código complemento a 2

Ejemplo: r= 3



¿Cómo identificamos estos problemas?

Hay overflow si:
1) Los operandos tienen = signo
2) El resultado tiene signo opuesto a los operandos

Pero esta forma de detectar un resultado no válido no es muy práctica para las máquinas digitales.

Representación numérica

- Números Enteros:

Representación en complemento a 2

Overflow (OV)

Hay overflow si $Cy_{in} \neq Cy_{out}$ ó lo que es lo mismo: $OV = Cy_{in}$ xor Cy_{out}

Donde:

Cy_{in}: acarreo que ingresa al bit de signo

Cy_{out}: acarreo que sale del bit de signo

Cy_{in}	Cy_{out}	OV
0	0	0
0	1	1
1	0	1
1	1	0

Ejemplo:

Cy_{out} Cy_{in}

0 1 1	+3
0 1 0	+2
1 0 1	-3

➔ $Cy_{in} = 1$ y $Cy_{out} = 0$ ➔ Hay Overflow

Representación numérica

- Números Enteros:

Representación en complemento a 2

Representación:

Si $0 \le N \le 2^{r-1} - 1$ (positivo) ➔ Binario sin signo

Si $2^{r-1} \le N < 0$ (negativo) ➔ Opuesto al binario sin signo.

Opuesto: $(A)_{2c} = A + 1$

Extensión del signo

Pare convertir un número A en 2C con r bits a s bits (s>r):

- Si A≥0 ➔ bit signo es 0 ➔ extiendo con 0,s
- Si A<0 ➔ bit signo es 1 ➔ extiendo con 1,s

Pesos

Representación numérica

- Números Enteros:

Representación en complemento a 2

Ejemplo r=4 ➔ $-8 \leq N \leq 7$

- Se quiere hacer la resta R = 6 - 5

- Represento el 6 : 6>0 ➔ **0110**
- Represento el -5 : -5 < 0 ➔ hallo el opuesto de +5 ➔ $+5 = 0101$

$\overline{+5} = 1010$

$\overline{+5} + 1 = \mathbf{1011} = -5$

6 - 5 = 6 + (-5) ➔ 0110

1011

1 0001

$OV = Cy_{in}$ xor Cy_{out}

OV = 1 xor 1 =0

No hay overflow

Representación numérica

- Números Enteros:

Representación por desplazamiento

La idea es desplazar el 0 de forma de obtener valores negativos pero con un código ordenado, lo que nos va a permitir hacer comparaciones de enteros fácilmente.

$N = b - D$

- b: número binario sin signo
- D : desplazamiento

Nosotros vamos a utilizar desplazamientos ($2^{r-1} - 1$) (r cantidad de bits). En particular:

r	D
5	15
8	127
11	1023

Ejemplo:
3 bits y D = 3

4	111
3	110
2	101
1	100
0	011
-1	010
-2	001
-3	000

Representación numérica

- Números Reales:

  2 representaciones: → punto fijo
  → punto flotante

Representación en punto fijo

$$N = (-1)^S . (b_{n-1} 2^{n-1} + b_{n-2} 2^{n-2} ..... b_0 2^0 + b_{-1}2^{-1} + ... + b_{-m} 2^{-m})$$

Debe quedar predefinido cuantos bits son parte entera y cuantos de parte fraccionaria. Con esto queda preestablecido el lugar donde va la coma.

Ejemplo: 3 bits de parte entera y 4 bits de parte fraccionaria

$$N = 1010\ 0110 \rightarrow N = -\ 010{,}0110 \rightarrow N = -2{,}375_d$$

Representación numérica

- Números Reales:

Representación en punto flotante

Es igual que la notación científica sustituyendo la base 10 por base 2.

En base 10 : $N = \pm d,f \times 10^n$

- d: 1 dígito ≠ 0
- f : parte fraccionaria con n dígitos
- n: exponente, número entero.

Ejemplo: $N = -7,234 \times 10^4$

En base 2: $N = (-1)^S \cdot 1,f \times 2^n$

- 1: pues d ≠ 0 siempre es 1
- f : parte fraccionaria con m dígitos
- n: exponente, entero por desplazamiento.

Facilita las operaciones

Representación numérica

- Números Reales:

Representación en punto flotante

Estándar IEEE para punto flotante: *IEEE Standard 754* (1985):

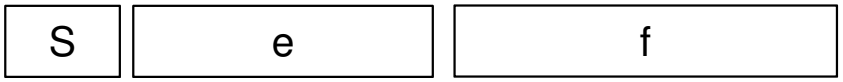
$$N = (-1)^S \cdot 1,f \cdot 2^{e-d} \quad ; \qquad 000\ldots00 < e < 1111\ldots11$$

Tipo de dato	S (bits)	e (bits)	f (bits)	d	Total (bytes)	Total (bits)
Media precisión	1	5	10	15	2	16
Simple Precisión (short real)	1	8	23	127	4	32
Doble Precisión (long real)	1	11	52	1023	8	64
Extended real	1	15	64	16383	10	80

← Procesadores de hoy

Para *Simple Precisión* seria: $N = (-1)^S \cdot 1,f \cdot 2^{e-127}$; $0 < e < 255$

Representación:



Representación numérica

- Números Reales:

Representación en punto flotante

Estándar IEEE para punto flotante: *IEEE Standard 754* (1985):

Existen excepciones a la ecuación anterior:

Tipo	Valor numérico	Detalle
Normalizado	$N= (-1)^S . 1,f . 2^{e-127}$	$0 < e < 255$
Subnormalizado	$N= (-1)^S . 0,f . 2^{-126}$	$e = 0$
Cero	$N = 0$	$e = 0 ; f = 0$
Infinito	$N = \pm\infty$	$e = 255 ; f = 0$
Not a number (NaN)	N no es un número	$e = 255 ; f \neq 0$

Tabla para Simple Precisión.

Representación numérica

- Números Reales:

Representación en punto flotante

Ejemplo:

Dado N = -584,2723 ➔ Expresarlo en punto flotante Simple precisión

$N=(-1)^S \cdot 2e^{-127} \cdot 1,f$ (S:1 bit; e: 8 bits: f: 23 bits)

Se halla el equivalente en punto fijo:

Parte entera: $I_d = 584 = 512 + 64 + 8$ → **$I_2 = 10\ 0100\ 1000$**

Parte fraccionaria: $F_{10} \cdot 2^{14} = 0,2723 \cdot 2^{14} = 4461,3632$

$4461 = 4096 + 256 + 64 + 32 + 8 + 4 + 1$ → **$F_2 = 0100\ 0101\ 1011\ 01$**

→ **Error = $0,3632 \cdot 2^{-14}$**

Nota: $Bits(F_2) = 23 + 1 - Bits\ (I_2) = 14$

Representación numérica

- Números Reales:

Representación en punto flotante

Ejemplo: N = -584,2723 → $N=(-1)^S \cdot 2^{e-127} \cdot 1,f$

N_2 = **- 10 0100 1000 , 0100 0101 1011 01**

S=1

Corro la coma hasta que quede de la forma 1,f

10 0100 1000,0100 0101 1011 01 = 1,0 0100 1000 0100 0101 1011 01. 2^9

1,f = 1,0 0100 1000 0100 0101 1011 01 → f = 00100100001000101101101

e-127 = 9 → e = 136 → e = 128 + 8 → e = 10001000

Representación: 11000100000100100001000101101101

Códigos binarios de nos decimales

Decimal	BCD	2421	Exceso 3	1 de 10
0	0000	0000	0011	0000000001
1	0001	0001	0100	0000000010
2	0010	0010	0101	0000000100
3	0011	0011	0110	0000001000
4	0100	0100	0111	0000010000
5	0101	1011	1000	0000100000
6	0110	1100	1001	0001000000
7	0111	1101	1010	0010000000
8	1000	1110	1011	0100000000
9	1001	1111	1100	1000000000

Reflejado

Siempre tiene 0 y 1

BCD: Binary Coded Decimal

Los códigos 1010 al 1111 NO SON PALABRAS DEL CÓDIGO

Códigos binarios de nos decimales

- BCD empaquetado

En 1 byte representar 2 dígitos BCD (unidades y decenas)

Decenas	Unidades

Ejemplo de BCD empaquetado:

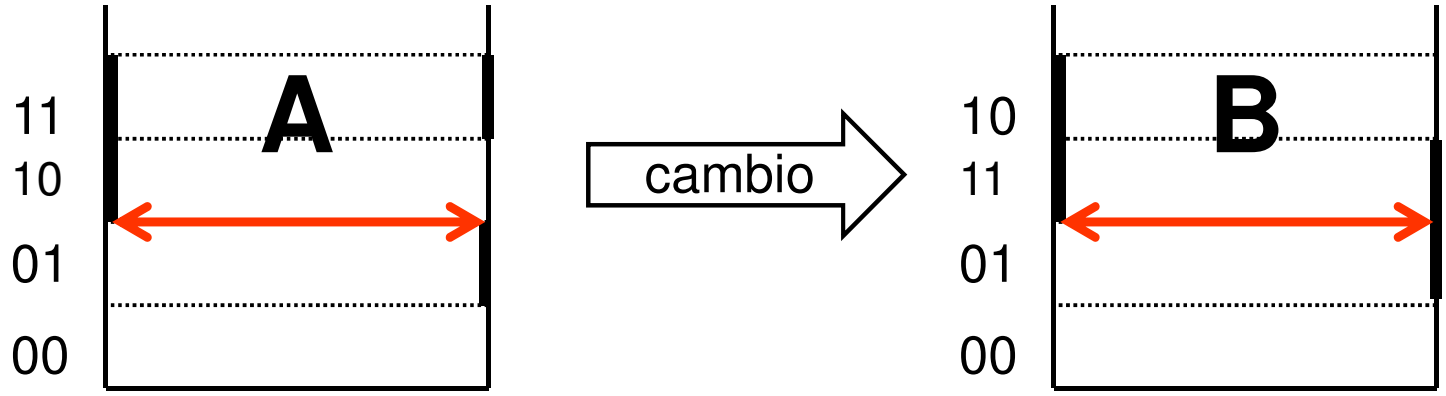
Dado 25 → en BDC empaquetado queda: 0010 0101

2 5

Código Gray

Ejemplo:

Necesito leer el nivel de un tanque de agua. Para ello ubico sensores de la forma siguiente:



De 01 a 10 cambian 2 bits

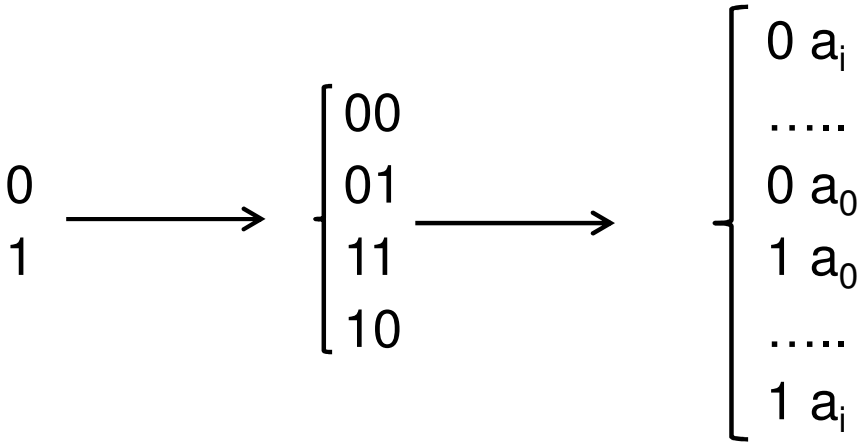
Las palabras sucesivas difieren en 1 solo bit

Código Gray

Construcción del código

:

Las palabras sucesivas difieren en 1 solo bit.



Este código tiene la característica de ser reflejado.