



Sistemas de Procesamiento de Datos – Unidad 5

- Números enteros y fraccionarios signados
- ALU
- Números reales
- Notación de punto fijo y punto Flotante

Profesor: Fabio Bruschetti
Ayudante: Pedro Iriso
Ver 2020



Números enteros SIN signo

- Los n bits son usados para la magnitud del número. No hay bit de signo.
 - Ejemplo: Número de 4-bits: 0000_b 1111_b , es decir, desde 0_d hasta 15_d
- Ejemplos
 - $11_b = 3_d = +3_d$
 - $1001_b = 9_d = +9_d$
 - Recordamos el rango de representación para n bits

$$\{0; 2^n - 1\}$$



Números enteros CON signo

- Convenios
 - Signo y Magnitud
 - Complemento a 1
 - Complemento a 2
- Se desplaza la recta de representación
 - Exceso 2^{n-1}



Convenio de Signo y Magnitud (SyM)

- Necesito usar 1 bit para el signo
- El bit más significativo será el del signo
 - 0 = positivo
 - 1 = negativo
- Los restantes bits codifican la magnitud
- ¿Cuál es la ventaja? → Tengo signo! → 1 bit
- ¿Cuál es la desventaja? → Perdí un dígito para la magnitud... El rango de representación cambiará. Tengo $n-1$ bits para la magnitud

Convenio de Signo y Magnitud (SyM)

■ Ejemplos

■ $+12_d = \mathbf{0}0001100_b$

■ $-1_d = \mathbf{1}0000001_b$

■ Problema: no sirve para las matemáticas

$$\begin{array}{rcl} & 0000\ 1100_b & +12 \\ + & 1000\ 0001_b & + -1 \\ \hline & 1000\ 1101_b & -13 \end{array}$$

| C | B | A | # |
|---|---|---|----|
| 0 | 0 | 0 | +0 |
| 0 | 0 | 1 | +1 |
| 0 | 1 | 0 | +2 |
| 0 | 1 | 1 | +3 |
| 1 | 0 | 0 | -0 |
| 1 | 0 | 1 | -1 |
| 1 | 1 | 0 | -2 |
| 1 | 1 | 1 | -3 |

NO SIRVE PARA
HACER CUENTAS



Convenio de Signo y Magnitud (SyM)

- Rango de representación
 - Con n bits se pueden representar 2^n combinaciones posibles
 - Al perder 1 bit por el signo, la mitad de los valores posibles serán número negativos y la otra mitad positivos
 - 2^{n-1} valores positivos y 2^{n-1} negativos
 - En este caso habrá un $+0$ y un -0 que duplican el valor de dos combinaciones distintas de bits
 - Rango de representación para n bits
 $\{-(2^{(n-1)}-1); +(2^{(n-1)}-1)\}$

| C | B | A | # |
|---|---|---|----|
| 0 | 0 | 0 | +0 |
| 0 | 0 | 1 | +1 |
| 0 | 1 | 0 | +2 |
| 0 | 1 | 1 | +3 |
| 1 | 0 | 0 | -0 |
| 1 | 0 | 1 | -1 |
| 1 | 1 | 0 | -2 |
| 1 | 1 | 1 | -3 |



Convenio de complemento a 1 (Ca1)

- Números positivos = Igual que el convenio de SyM
- Números negativos = Se representan con el Complemento a 1 (Ca1) de su correspondiente número binario positivo
- Complemento a 1 de un bit o “flipp”
 - Ca1 de $1 \rightarrow 0$
 - Ca1 de $0 \rightarrow 1$
- Ca1 de una variable A se puede decir el “Complemento” de A

Convenio de complemento a 1 (Ca1)

- Complemento a 1 de n-bits:
 - Complemento de cada bit por separado
- Ejemplo
 - $+12_d = 00001100_b$
 - $-1_d = \text{Ca1 de } (+1) \rightarrow +1 = 00000001_b$
 - $\text{Ca1} = 11111110_b = -1_d$
- Problema: no sirve para las matemáticas

$$\begin{array}{rcl} & 11111110_b & -1_d \\ + & \underline{11111110_b} & + \underline{-1_d} \\ & 11111100_b & -3_d \\ \text{Ca1} & 00000011_b & \rightarrow +3_d \rightarrow \text{Resultado} = -3 \end{array}$$

NO SIRVE PARA
HACER CUENTAS



Convenio de complemento a 1 (Ca1)

- Rango de representación
 - En este caso también tenemos dos combinaciones binarias que valen +0 y -0 que indican el mismo valor
 - Rango de representación para n bits

$$\{-(2^{(n-1)}-1);+(2^{(n-1)}-1)\}$$

| C | B | A | # |
|---|---|---|----|
| 0 | 0 | 0 | +0 |
| 0 | 0 | 1 | +1 |
| 0 | 1 | 0 | +2 |
| 0 | 1 | 1 | +3 |
| 1 | 0 | 0 | -3 |
| 1 | 0 | 1 | -2 |
| 1 | 1 | 0 | -1 |
| 1 | 1 | 1 | -0 |



Convenio de complemento a 2 (Ca2)

- Números positivos = Igual que el convenio de SyM
- Números negativos = Se representan con el Complemento a 2 de su correspondiente positivo
- La magnitud de los números negativos se codifica como “flip & add” (“dar vuelta y sumar”)
- Complemento a 2 de n-bits:
 - Complemento a 1 de los n-bits y luego...
 - Se suma 1 al bit menos significativo

- Ejemplo

- $+12_d = 00001100_b$

- $-1_d = \text{Ca2 de } (+1) \rightarrow +1 = 00000001_b$

$$\text{Ca1} = 11111110_b$$

$$\begin{array}{r} + 1_b \\ \hline \end{array}$$

$$\text{Ca2} = 11111111_b = -1_d$$

Convenio de complemento a 2 (Ca2)

- Hay una cuestión con el 100_b (ejemplo de 3 bits)
- Es un número negativo (empieza con 1)
- ¿Cuál es su valor decimal? ¿d?
- Hago el complemento a 2

- Nro = 100_b
- Ca1 = 011_b
- Sumo 1 + $\underline{\quad 1}_b$
- 100_b (¿Da de nuevo negativo?)

| C | B | A | # |
|---|---|---|----|
| 0 | 0 | 0 | +0 |
| 0 | 0 | 1 | +1 |
| 0 | 1 | 0 | +2 |
| 0 | 1 | 1 | +3 |
| 1 | 0 | 0 | ? |
| 1 | 0 | 1 | -3 |
| 1 | 1 | 0 | -2 |
| 1 | 1 | 1 | -1 |

→

| C | B | A | # |
|---|---|---|----|
| 0 | 0 | 0 | +0 |
| 0 | 0 | 1 | +1 |
| 0 | 1 | 0 | +2 |
| 0 | 1 | 1 | +3 |
| 1 | 0 | 0 | -4 |
| 1 | 0 | 1 | -3 |
| 1 | 1 | 0 | -2 |
| 1 | 1 | 1 | -1 |

- Se resuelve así: Al convertir un número de positivo a negativo, el valor será el que resulte de la magnitud de todos los bits como si no tuviese signo
- En el ejemplo 100_b es negativo y su Ca2 da 100_b que es 4_d entonces $\rightarrow 100_b$ es -4_d



Convenio de complemento a 2 (Ca2)

- Veamos:

$$\begin{array}{r} 110_b \\ + \quad \underline{110}_b \\ \hline 100_b \end{array} \quad \begin{array}{r} -2_d \\ + \quad \underline{-2}_d \\ \hline -4_d \end{array}$$

$$\begin{array}{r} 010_b \\ + \quad \underline{111}_b \\ \hline 001_b \end{array} \quad \begin{array}{r} +2_d \\ + \quad \underline{-1}_d \\ \hline +1_d \end{array}$$

SIRVE PARA
HACER CUENTAS

No se tiene en cuenta el acarreo o carry!



Convenio de complemento a 2 (Ca2)

- Rango de representación
 - Rango de representación para n bits
$$\{-(2^{(n-1)}); +(2^{(n-1)}-1)\}$$
 - Ya no vemos el "-1" en la parte negativa

| C | B | A | # |
|---|---|---|----|
| 0 | 0 | 0 | +0 |
| 0 | 0 | 1 | +1 |
| 0 | 1 | 0 | +2 |
| 0 | 1 | 1 | +3 |
| 1 | 0 | 0 | -4 |
| 1 | 0 | 1 | -3 |
| 1 | 1 | 0 | -2 |
| 1 | 1 | 1 | -1 |

Convenio de complemento a 2 (Ca2)

■ Ejemplo:

- Representación de -1 en complemento a 2 con 8 bits.

| | |
|----------------|---------------------------------------|
| $+1_d$ | $0000\ 0001_b$ |
| Ca1 | $1111\ 1110_b$ |
| <u>Sumar 1</u> | <u>$+ \quad \quad 1_b$</u> |
| -1_d | $1111\ 1111_b = FF_h$ |

Operación de
complemento a 2

- Representación de 1 en complemento a 2 con 8 bits

- $+1_d$ $0000\ 0001_b = 01_h$

- Valor decimal del valor FE_h en complemento a 2

| | |
|----------------|--|
| FE_h | $1111\ 1110_b$ (Bit de signo = 1) |
| Ca1 | $0000\ 0001_b$ |
| <u>Sumar 1</u> | <u>$+ \quad \quad 1_b$</u> |
| Ca2 | $0000\ 0010_b = (\text{bit de signo})\ 2_d = -2_d$ |



Convenio de complemento a 2 (Ca2)

■ Ejemplo:

- Encuentre el valor decimal del valor 79_h en complemento a 2
 - Bit de signo = 0 $0111\ 1001_b = 64+32+16+8+1 = 121_d$
 - $79_h = +121_d$

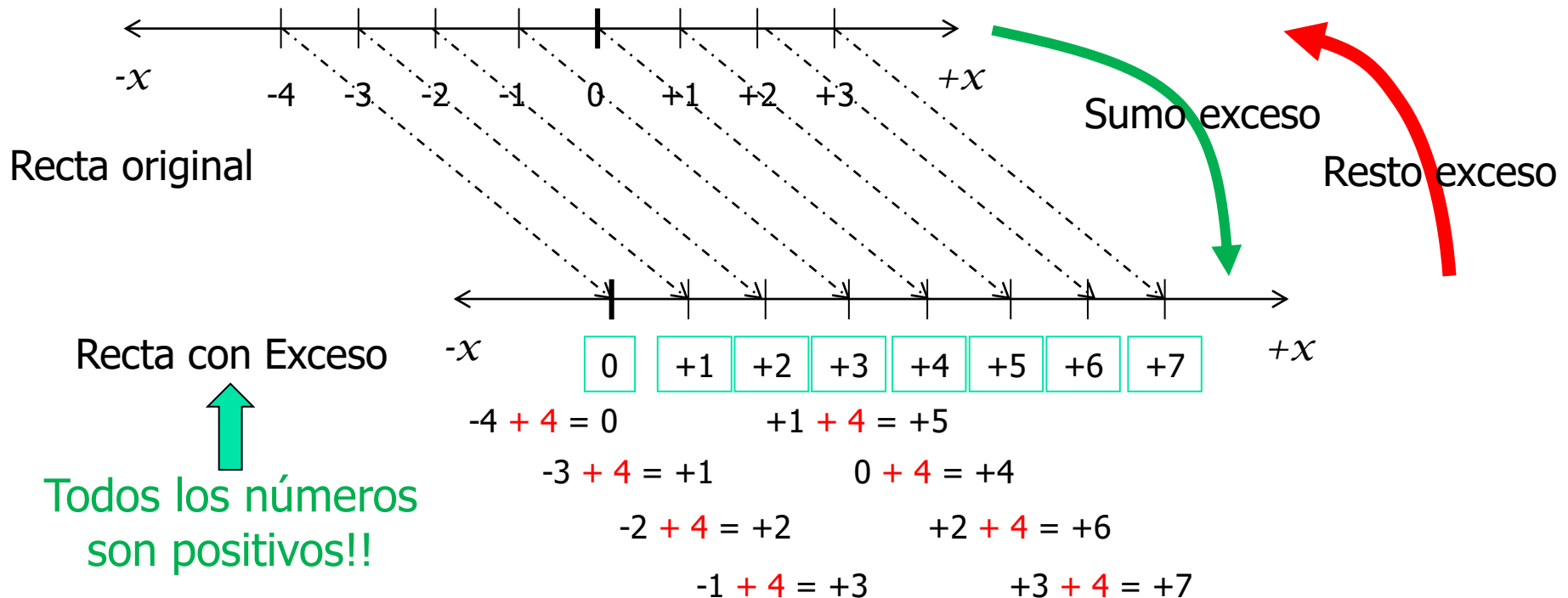
■ Ejercicios:

- Encuentre el valor decimal del valor 90_h en complemento a 2
- Encuentre el valor decimal del valor $8D_h$ en complemento a 2
- Encuentre el valor decimal del valor $6F_h$ en complemento a 2

Convenio de Exceso 2^{n-1}

■ Convenio de Exceso 2^{n-1}

- Se desplaza la recta de representación en 2^{n-1} , sumándole ese valor a cada punto a representar.
- Ejemplo $n = 3 \rightarrow 2^{n-1} = 2^{3-1} = 2^2 = 4 \rightarrow$ Se sumará 4 a cada punto

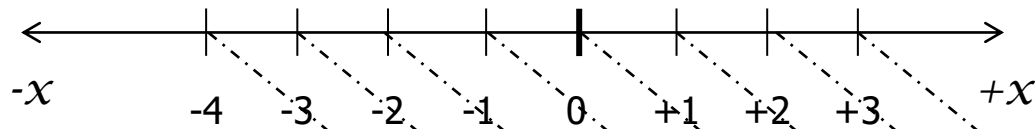


Convenio de Exceso 2^{n-1}

Todos enteros positivos


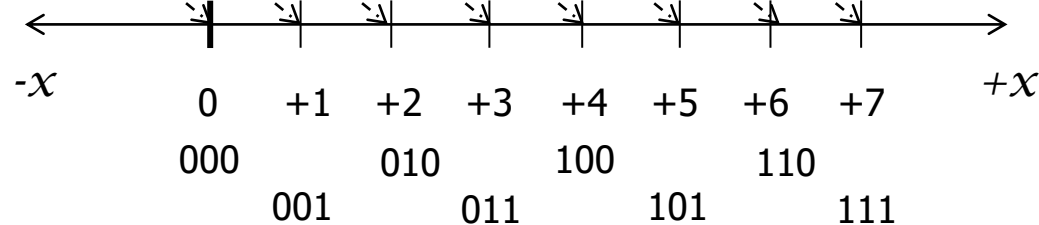
■ Rango de representación

- Para 3 bits va del -4...0...+3
- Rango para n bits: $-2^{(n-1)} \dots 0 \dots 2^{(n-1)}-1$



Recta original

Recta de representación



| C | B | A | # |
|---|---|---|----|
| 0 | 0 | 0 | -4 |
| 0 | 0 | 1 | -3 |
| 0 | 1 | 0 | -2 |
| 0 | 1 | 1 | -1 |
| 1 | 0 | 0 | +0 |
| 1 | 0 | 1 | +1 |
| 1 | 1 | 0 | +2 |
| 1 | 1 | 1 | +3 |

Convenio de Exceso 2^{n-1}

- Veamos:

$$\begin{array}{rcl} & 010_b & -2_d \\ + & \underline{010}_b & + \underline{-2}_d \\ & 100_b & +0_d \end{array}$$

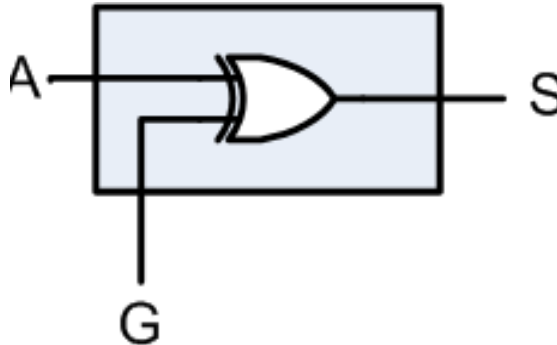
NO SIRVE PARA
HACER CUENTAS

| C | B | A | # |
|---|---|---|----|
| 0 | 0 | 0 | -4 |
| 0 | 0 | 1 | -3 |
| 0 | 1 | 0 | -2 |
| 0 | 1 | 1 | -1 |
| 1 | 0 | 0 | +0 |
| 1 | 0 | 1 | +1 |
| 1 | 1 | 0 | +2 |
| 1 | 1 | 1 | +3 |

Implementación de la codificación Ca1

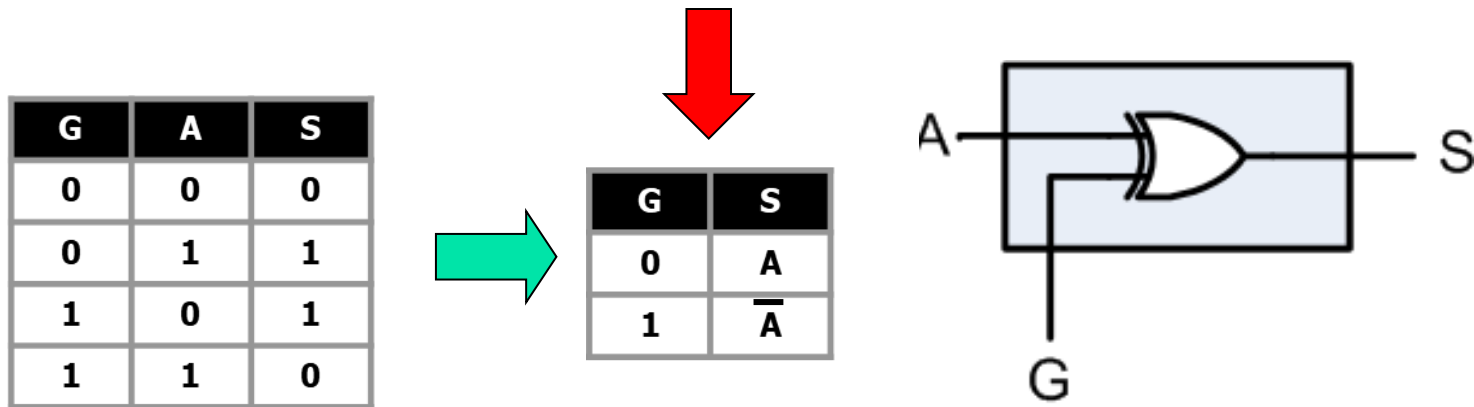
- Recordemos la compuerta "o" exclusiva

| G | A | S |
|---|---|---|
| 0 | 0 | 0 |
| 0 | 1 | 1 |
| 1 | 0 | 1 |
| 1 | 1 | 0 |



Implementación de la codificación Ca1

- Voy a reacomodar la tabla de verdad



- Si la señal "G" es 0, $S=A$
- Si la señal "G" es 1, $S=\overline{A}$, o también !A
- Es un negador controlado por la señal "G"!!! → Si le sumamos un sumador, ¿podremos hacer un complementador a 2? Mmmm.....veremos



Conceptos para Programador

■ Representación Números Enteros (Resumen)

■ Módulo y Signo ($-2^{n-1} + 1 \leq X \leq 2^{n-1} - 1$)

- Por ejemplo, representar los números 10 y -10 con 8 bits.

| | |
|-----------------------|---------------------------|
| ■ 0 (+) 0 0 0 1 0 1 0 | representa al número 10, |
| ■ 1 (-) 0 0 0 1 0 1 0 | representa al número -10. |

■ Complemento a 1 ($-2^{n-1} + 1 \leq X \leq 2^{n-1} - 1$)

- Por ejemplo, representar los números 10 y -10 con 8 bits.

| | |
|-----------------------|---------------------------|
| ■ 0 (+) 0 0 0 1 0 1 0 | representa al número 10, |
| ■ 1 (-) 1 1 1 0 1 0 1 | representa al número -10. |

■ Complemento a 2 ($-2^{n-1} \leq X \leq 2^{n-1} - 1$) [Representación única del 0]

- Por ejemplo, representar los números 10 y -10 con 8 bits.

| | |
|-----------------------|---------------------------|
| ■ 0 (+) 0 0 0 1 0 1 0 | representa al número 10, |
| ■ 1 (-) 1 1 1 0 1 1 0 | representa al número -10. |

■ Exceso a 2^{n-1} ($-2^{n-1} \leq X \leq 2^{n-1} - 1$) [Representación única del 0]

- Por ejemplo, representar los números 10 y -10 con 8 bits.

| | |
|-------------------|---------------------------|
| ■ 1 0 0 0 1 0 1 0 | representa al número 10, |
| ■ 0 1 1 1 0 1 1 0 | representa al número -10. |

- Se codifican sumando el exceso al número, para $n=8$, el exceso es de 128 :

| |
|--------------------------|
| ■ $10 = 128 + 10 = 138$ |
| ■ $-10 = 128 - 10 = 118$ |



Conceptos para Programador

- Importante

- Valores binarios de longitud fija son utilizados para representar la información en computadores
- El computador trabaja con la representación binaria (no con la información)
- El mismo valor binario puede ser usado para representar información diferente

- Ejemplo 8-bits: 11110000_2

- Sin Signo (unsigned) 240_{10}
- Con Signo (signed) -16_{10}
- ASCII de 8 bits `""`
- Otro ??



Aritmética y Lógica

■ Concepto de Overflow

- Es el resultado de una operación fuera de rango que puede ser representado:
 - Se produce debido al rango limitado de una representación de tamaño fijo
 - Se genera un resultado, pero es sin sentido.
 - $255_{10} = 1111\ 1111_b$
 - $+ 1_{10} = \underline{0000\ 0001}_b$
 - $256_{10} = (1)\ 0000\ 0000_b \rightarrow$ Pero esto es $0_{10}!!$
 - En este caso se necesitan 9 bits para representar el resultado
 - Hemos sumado dos cifras binarias enteras sin signo
- Con un ancho fijo de 8-bits: Se produce OVERFLOW
 - $CF = 1$ ($CF \rightarrow$ En 0 = "NC", en 1 = "CY" para el DEBUG)
- En el caso de estar con números sin signo, el Carry es fundamental para la interpretación del resultado. Si hay Carry, hay overflow
 - $CF = 1$



Aritmética y Lógica

■ Concepto de Overflow

■ Suma de números con signo:

$$\begin{array}{rcl} \text{■} & -1_{10} & = & 1111\ 1111_b \\ \text{■} & +\ 1_{10} & = & \underline{0000\ 0001}_b \\ \text{■} & 0_{10} & = & (1)\ 0000\ 0000_b \end{array}$$

- El resultado es correcto ($-1 + 1 = 0$), sin embargo vemos que hay Carry pero no hay overflow!
 - $OF=0\ CF=1$
- En el caso de estar con números con signos, el Carry en el bit más significativo, no es importante para la interpretación del resultado.

■ Overflow:

- Carry Flag (CF) para números sin signo
 - $CF \rightarrow$ En 0 = "NC", en 1 = "CY" para el DEBUG
- Overflow Flag (OF) para números con signo
 - $OF \rightarrow$ En 0 = "NV", en 1 = "OV" para el DEBUG

Aritmética y Lógica

■ Concepto de Overflow

$$\begin{array}{rcl} \blacksquare & 32_{10} & = \quad 0010\ 0000_b \\ \blacksquare & -\ 65_{10} & = \quad 0100\ 0001_b \\ \hline \blacksquare & -\ 33_{10} & = \quad (1)1101\ 1111_b \text{ (223 sin signo)} \end{array}$$

- La resta de valores sin signo, “pedir prestado” para realizar la resta implica la existencia de un overflow. El resultado es erróneo.
- Si los valores son tomados con signo, no hay overflow; se ignora el “pedir prestado” y el resultado es correcto.
- Si el resultado de una operación arroja un número con el bit más significativo en 1, el SF=1 indicando que es negativo. En caso contrario dará SF=0.
- **En este ejemplo: CF = 1, OF = 0, SF=1**
 - SF → En 0 = “PL”, en 1 = “NG” para el DEBUG

Aritmética y Lógica

■ Concepto de Overflow

■ Otro ejemplo

| | Sin signo | Con signo |
|------------------------|-----------|------------|
| 0111 1111 _b | 127 | 127 |
| 0000 0001 _b | + 1 | + 1 |
| 1000 0000 _b | 128 | - 128 |
| | CORRECTO | INCORRECTO |

■ Hay overflow aún cuando no hay Carry y el resultado es negativo

■ **CF=0, OF=1, SF=1**

Aritmética y Lógica

■ Concepto de Overflow (Resumen)

■ Sin Signo:

- Carry o Borrow (pedir prestado) implican Overflow



■ Con Signo:

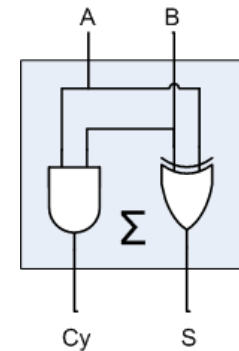
- Se ignoran el Carry o Borrow
- Overflow es posible:
 - Positivo + Positivo = Negativo
 - (positivo – negativo = negativo)
 - Negativo + Negativo = Positivo
 - (negativo – positivo = positivo)
- Overflow es imposible:
 - Positivo + negativo (positivo – positivo)
 - Negativo + positivo (negativo – negativo)

ALU (Unidad Aritmético Lógica)

■ Sumador aritmético

- $S = A \oplus B$
- $Cy = A.B$

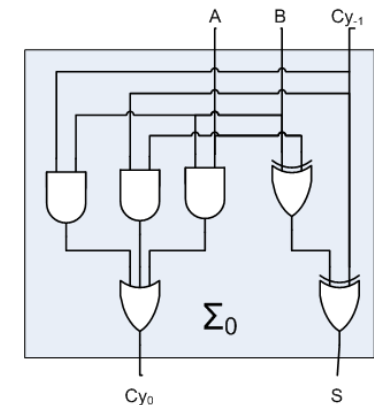
| A | B | S | Cy |
|---|---|---|----|
| 0 | 0 | 0 | 0 |
| 0 | 1 | 1 | 0 |
| 1 | 0 | 1 | 0 |
| 1 | 1 | 0 | 1 |



■ Sumador completo

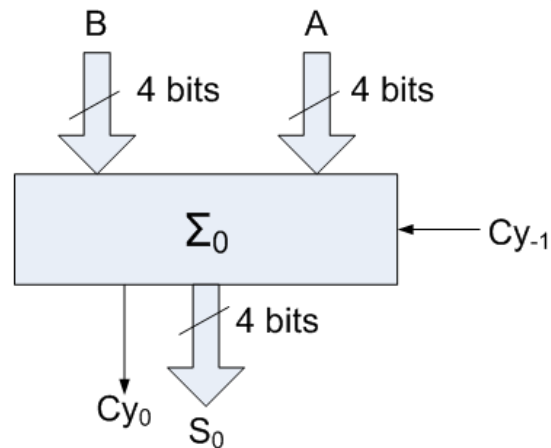
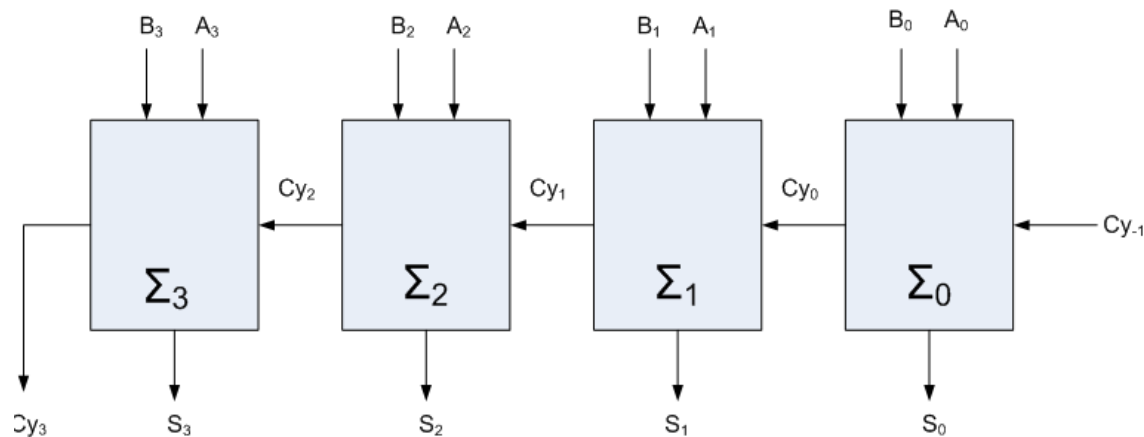
- $S_0 = A \oplus B \oplus Cy_{-1}$
- $Cy_0 = A.B + Cy_{-1}.A + Cy_{-1}.B$

| Cy_{-1} | A_0 | B_0 | S_0 | Cy_0 |
|-----------|-------|-------|-------|--------|
| 0 | 0 | 0 | 0 | 0 |
| 0 | 0 | 1 | 1 | 0 |
| 0 | 1 | 0 | 1 | 0 |
| 0 | 1 | 1 | 0 | 1 |
| 1 | 0 | 0 | 1 | 0 |
| 1 | 0 | 1 | 0 | 1 |
| 1 | 1 | 0 | 0 | 1 |
| 1 | 1 | 1 | 1 | 1 |



ALU (Unidad Aritmético Lógica)

- Sumador de 4 bits



ALU (Unidad Aritmético Lógica)

- Aritmética sin signo

- $117_{10} = 0111\ 0101_b$
- $+ 99_{10} = 0110\ 0011_b$
- $216_{10} = 1101\ 1000_b$

- Aritmética con signo

- $-117_{10} = 1000\ 1011_b$
- $+ 99_{10} = 0110\ 0011_b$
- $-18_{10} = 1110\ 1110_b\ (0001\ 0001 + 1) = 12_h = 18_d$

- Restas en binario:

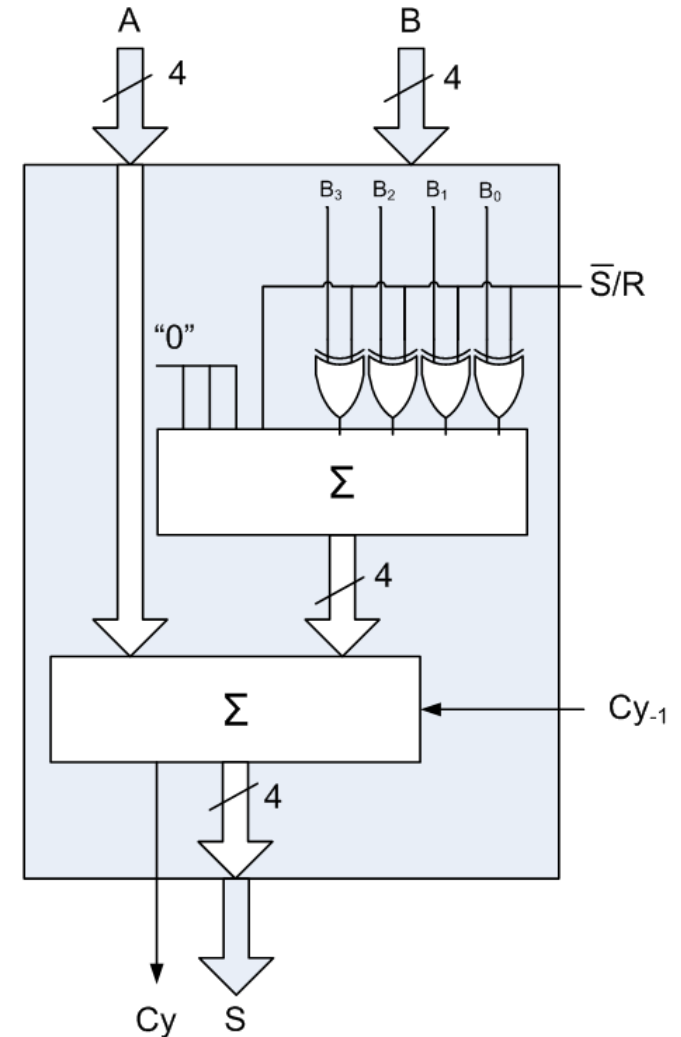
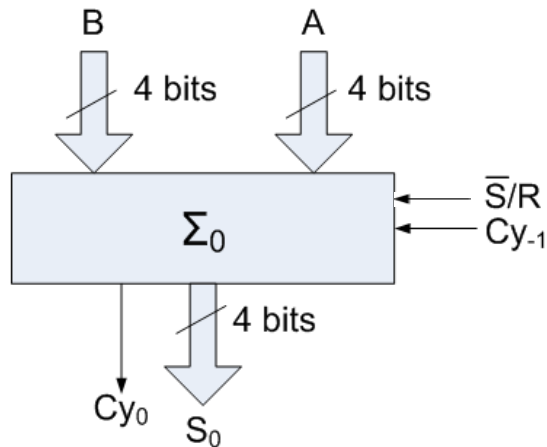
- Negar y sumar: $X - Y = X + (-Y)$

- Ejemplo: $32 - 65 = 32 + (-65)$

- $32_{10} = 0010\ 0000_b$
- $+ -65_{10} = 1011\ 1111_b\ (0100\ 00002 + 1 = 41_h = 65_d)$
- $-33_{10} = 1101\ 1111_b\ (0010\ 00002 + 1 = 21_h = 33_d)$

ALU (Unidad Aritmético Lógica)

- Sumador/restador de 4 bits



Aritmética y Lógica

■ Multiplicación binaria

- Procedimiento de resolución del cálculo de forma idéntica al de la multiplicación decimal

$$\begin{array}{r} 1101 \leftarrow \text{Multiplicando} \\ \times 101 \leftarrow \text{Multiplicador} \\ \hline 1101 \\ + 0000 \\ \hline 1101 \\ \hline 100001 \end{array}$$

- Por cada cifra del multiplicador, me desplazo en la suma final
- Por cada 1 del multiplicador, repito el multiplicando en la suma final
- La multiplicación es una “sucesión de sumas y desplazamientos”
- Multiplicar 2 cifras binarias de n bits darán como resultado otra cifra binaria de 2.n bits! ($1111_b \times 1111_b = 11100001_b$)

Aritmética y Lógica

■ División binaria

- Procedimiento de resolución del cálculo de forma idéntica al de la división decimal

Dividendo →

$$\begin{array}{r} 11011 \overline{) 11011} \\ \underline{101} \\ 0011 \\ \underline{101} \\ 00111 \\ \underline{101} \\ 10 \end{array}$$

Divisor

Cociente

Resto

$$\begin{array}{r} 11011 \\ \underline{101} \\ 10110 \\ \underline{101} \\ 10001 \\ \underline{101} \\ 1100 \\ \underline{101} \\ 111 \\ \underline{101} \\ 10 \end{array}$$

Resté el divisor 5 veces → Cociente

Me sobró 2 → Resto

- Ejemplo: $27 / 5 = 5$ con resto 2
- ¿Cómo se implementa esto?
- Se implementa como una sucesión de restas y desplazamientos

Números Reales

---, _d

000, _d } 1
001, _d } 1
002, _d

...

...

999, _d

{0 a 999}
De a 1

$\pi \rightarrow 003$

--, -_d

00, 0_d } 1/10=0,1
00, 1_d } 1/10=0,1
00, 2_d

...

...

99, 9_d

{0 a 99,9}
De a 0,1

03,1

-, --_d

0, 00_d } 1/100=0,01
0, 01_d } 1/100=0,01
0, 02_d

...

...

9, 99_d

{0 a 9,99}
De a 0,01

3,14

, ---_d

, 000_d } 1/1000=0,001
, 001_d } 1/1000=0,001
, 002_d

...

...

, 999_d

{0 a 0,999}
De a 0,001

,XXX

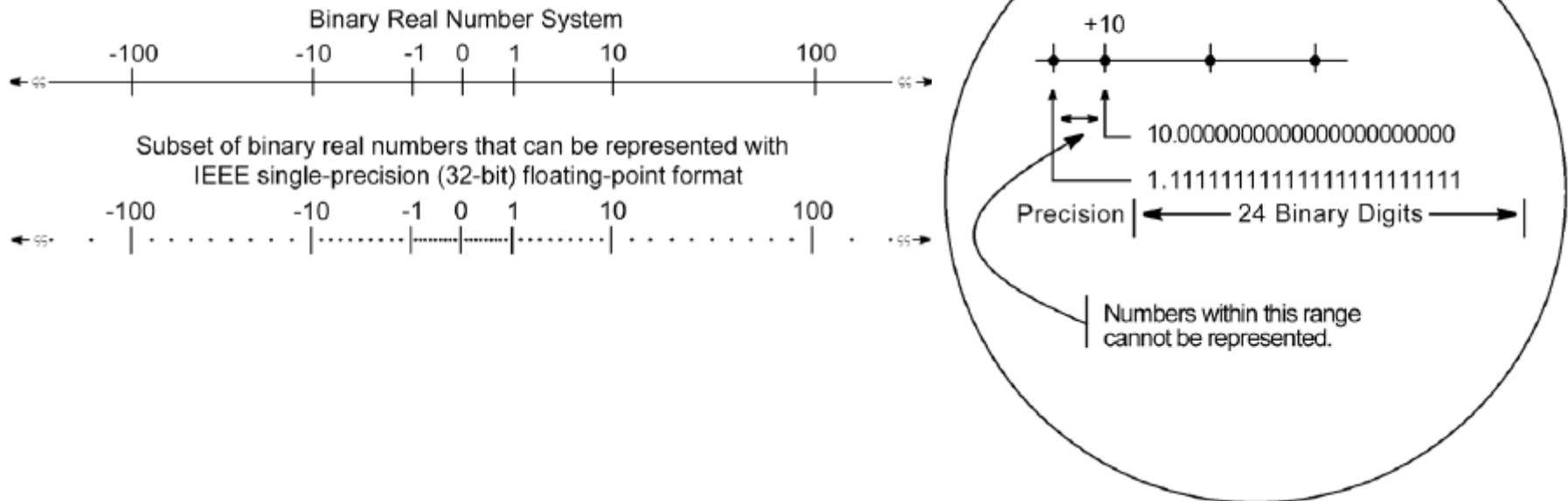
{RANGO DE REPRESENTACIÓN}
DISTANCIA O PRECISIÓN

A mayor rango de representación,
menor precisión (o viceversa)

Con una cantidad fija de dígitos \rightarrow Rango de representación vs. Precisión

Números Reales

- El rango de los números reales comprende desde $-\infty$ hasta $+\infty$.
- Los registros de un procesador tienen resolución finita.
- Por lo tanto un computador solo puede representar un subconjunto de \mathbb{R} . (No es solo un tema de magnitud sino de resolución)





Números Reales

- En general se puede formalizar la representación de un número real expresado en los siguientes formatos:
 - Punto Fijo: SIN signo y CON signo
 - Con Módulo y signo
 - Con complemento a 1
 - Con complemento a 2
 - Con exceso a 2^{n-1}
 - Punto Flotante
- Punto Fijo con signo
 - Se representan mediante una expresión del tipo
 - $(a_n a_{n-1} \dots a_0 \cdot a_{-1} a_{-2} \dots a_{-m})_2 = (-1)^s (a_n 2^n + \dots + a_0 2^0 + a_{-1} 2^{-1} + a_{-2} 2^{-2} + \dots + a_{-m} 2^{-m})$
 - Donde: $s=0$ si el número es positivo o $=1$ si el número es negativo
 - a_i es un entero y $0 \leq a_i \leq 1$, para todo $i = -m, \dots, -1, 0, 1, \dots, n$
 - Distancia entre dos números consecutivos es 2^{-m}
 - Deja de ser un rango continuo de números y pasa a ser un rango discreto.

Números Reales

m = cantidad de dígitos fraccionarios. Distancia = 2^{-m}

$m = 0$
Distancia
 $2^{-0} = 1$

$---, _b$
 $000, _b$
 $001, _b$
 $010, _b$
 $011, _b$
 $...$
 $111, _b$

{0 a 7}
De a 1

$m = 1$
Distancia
 $2^{-1} = 0,5$

$--, _b$
 $00, 0_b$
 $00, 1_b$
 $01, 0_b$
 $01, 1_b$
 $...$
 $11, 1_b$

{0 a 3,5}
De a 0,5

$m = 2$
Distancia
 $2^{-2} = 0,25$

$-, --_b$
 $0, 00_b$
 $0, 01_b$
 $0, 10_b$
 $0, 11_b$
 $...$
 $1, 11_b$

{0 a 1,75}
De a 0,25

$m = 3$
Distancia
 $2^{-3} = 0,125$

$, ---_b$
 $, 000_b$
 $, 001_b$
 $, 010_b$
 $, 011_b$
 $...$
 $, 111_b$

{0 a 0,875}
De a 0,125



Números Reales

- Punto Fijo con notación complemento a 2
 - El problema del punto fijo con signo consiste en representar los números negativos
 - La solución: el mismo criterio utilizado con los enteros, el complemento a 2.
 - Para representar el número $-N$ en punto fijo con notación complemento a 2 se hace $2^n - N$.
- Como convertir un número decimal fraccionario a binario:
 - 0,828125 * 2 = 1,65625
 - 0,65625 * 2 = 1,3125
 - 0,3125 * 2 = 0,625
 - 0,625 * 2 = 1,25
 - 0,25 * 2 = 0,5
 - 0,5 * 2 = 1

$$0,828125 = 0,110101$$

Números reales

■ Pasar a binario $29,5625_d$

1. Parte entera a binario $\rightarrow 29_d = \mathbf{11101}_b$
2. Parte fraccionaria a binario $\rightarrow 0,5625_d = ?$

- $0,5625 * 2 = \mathbf{1},1250 \rightarrow 0,\mathbf{1}\dots_b$
- $\mathbf{0},125 * 2 = \mathbf{0},25 \rightarrow 0,\mathbf{10}\dots_b$
- $\mathbf{0},25 * 2 = \mathbf{0},5 \rightarrow 0,\mathbf{100}\dots_b$
- $\mathbf{0},5 * 2 = \mathbf{1},0 \rightarrow 0,\mathbf{1001}\dots_b$
- $\mathbf{0} * 2 = \mathbf{0},0 \rightarrow 0,\mathbf{10010}\dots_b$
- $\mathbf{0} * 2 = \mathbf{0},0 \rightarrow 0,\mathbf{100100}\dots_b$

Parte fraccionaria a binario $\rightarrow 0,5625_d = \mathbf{0,1101}_b$

3. Unimos $\rightarrow 29,5625_d = \mathbf{11101,1101}_b$

- $$\begin{array}{r}
 \blacksquare \quad \quad \quad 011101,1101_b \\
 \blacksquare \quad \text{Ca1} \quad 100010,0010_b \\
 \blacksquare \quad \quad \quad + \quad \underline{\quad \quad \quad 1_b} \\
 \blacksquare \quad \quad \quad 100010,0011_b
 \end{array}$$

Números reales

■ Pasar a binario $12,2_d$

1. Parte entera a binario $\rightarrow 12_d = \mathbf{1100}_b$

2. Parte fraccionaria a binario $\rightarrow 0,2 = ?$

■ $0,2 * 2 = \mathbf{0},4 \rightarrow 0,\mathbf{0}..._b$

■ $\mathbf{0},4 * 2 = \mathbf{0},8 \rightarrow 0,0\mathbf{0}..._b$

■ $\mathbf{0},8 * 2 = \mathbf{1},6 \rightarrow 0,00\mathbf{1}..._b$

■ $\mathbf{0},6 * 2 = \mathbf{1},2 \rightarrow 0,001\mathbf{1}..._b$

■ $\mathbf{0},2 * 2 = \mathbf{0},4 \rightarrow 0,0011\mathbf{0}..._b \rightarrow \text{Empieza de nuevo...}$

■ $\mathbf{0},4 * 2 = \mathbf{0},8 \rightarrow 0,00110\mathbf{0}..._b$

■ $\mathbf{0},8 * 2 = \mathbf{1},6 \rightarrow 0,001100\mathbf{1}..._b$

■ $\mathbf{0},6 * 2 = \mathbf{1},2 \rightarrow 0,0011001\mathbf{1}..._b$

Parte fraccionaria a binario $\rightarrow 0,00110011..._d = \mathbf{0,0011}_b$
periódico!

3. Unimos $\rightarrow 12,2_d = 1100,\overbrace{0011}_b$



ALU con binarios SIN signo

$$\begin{array}{r} 0010 \\ + 1011 \\ \hline 1101 \end{array} \quad \begin{array}{r} 2 \\ + 11 \\ \hline 13 \end{array}$$

$$\begin{array}{r} 001,0 \\ + 101,1 \\ \hline 110,1 \end{array} \quad \begin{array}{r} 1,0 \\ + 5,5 \\ \hline 6,5 \end{array} \quad \begin{array}{r} 00,10 \\ + 10,11 \\ \hline 11,01 \end{array} \quad \begin{array}{r} 0,50 \\ + 2,75 \\ \hline 3,25 \end{array}$$

$$\begin{array}{r} 0,010 \\ + 1,011 \\ \hline 1,101 \end{array} \quad \begin{array}{r} 0,250 \\ + 1,375 \\ \hline 1,625 \end{array} \quad \begin{array}{r} ,0010 \\ + ,1011 \\ \hline ,1101 \end{array} \quad \begin{array}{r} 0,1250 \\ + 0,6875 \\ \hline 0,8125 \end{array}$$

ALU con binarios CON signo

$$\begin{array}{r} 0010 \\ + 1011 \\ \hline 1101 \end{array} \quad \begin{array}{r} +2 \\ + -5 \\ -3 \end{array}$$

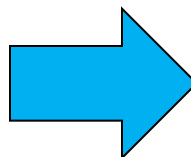
$$\begin{array}{r} 001,0 \\ + 101,1 \\ \hline 110,1 \end{array} \quad \begin{array}{r} +1,0 \\ + -2,5 \\ -1,5 \end{array}$$

$$\begin{array}{r} 00,10 \\ + 10,11 \\ \hline 11,01 \end{array} \quad \begin{array}{r} +0,50 \\ + -1,25 \\ -0,75 \end{array}$$

$$\begin{array}{r} 0,010 \\ + 1,011 \\ \hline 1,101 \end{array} \quad \begin{array}{r} +0,250 \\ + -0,625 \\ -0,375 \end{array}$$

$$\begin{array}{r} ,0010 \\ + ,1011 \\ \hline ,1101 \end{array} \quad \begin{array}{r} +0,1250 \\ + -0,3125 \\ -0,1875 \end{array}$$

La ALU suma
binario puro, la
interpretación
es del usuario



$$\begin{array}{r} 0010 \\ + 1011 \\ \hline 1101 \end{array}$$



Números Reales – Punto Fijo

- Cuando la cantidad de dígitos disponible no alcanza para representar el número ...
 - Problema: Representar un número de n dígitos fraccionarios en un sistema con m dígitos fraccionarios, siendo $m < n$
- Truncamiento:
 - Descarta los dígitos fraccionarios de orden mayor a m . El error máximo es de 1 bit en el dígito fraccionario m , o sea 2^{-m} .
- Redondeo:
 - Descarta los dígitos fraccionarios de orden mayor a m pero se suma 1 al menos significativo en caso que el bit inmediato descartado valga 1. Equivale a sumarle $0,5 \cdot 2^{-m}$ y truncar. El error es de $\frac{1}{2}$ bit.
- Para Complemento a 2
 - Cuando se desee hallar el complemento de una cifra binaria con punto fijo, primero se deberá:
 - Calcular la aproximación por truncamiento o redondeo
 - Calcular el complemento del número aproximado

Números Reales – Punto Fijo

- Números **positivos** truncados y redondeados. 11 bits, 4 fraccionarios

Sea el siguiente número: $+31,906025_d$

$$31,906025_{10} = 0011111.111010_2$$

- Si solo se trunca en 4 bits

$$0011111.1110\cancel{10}_2 \rightarrow 0011111.1110_2 = 31,875_{10}$$

- Si se redondea en 4 bits y luego se trunca

$$\begin{array}{r} 0011111.111010_2 \\ + \\ \hline 0011111.11110_2 \end{array} = 31,921875_{10}$$

- Haciendo cuentas:

$$31,906025 - 31,875 = 0,031025 < 0,0625 = 2^{-4}$$

$$31,921875 - 31,906025 = 0,015850 < 0,01953125 = 2^{-5}$$

- El número $31,906025$ no es representable en 11 bits con 4 fraccionarios. Se podrán representar o el $31,875_d$ o el $31,921875_d$.

Números Reales – Punto Fijo

- Números **negativos** truncados y redondeados. 11 bits, 4 fraccionarios
Sea el siguiente número: $-31,96875_d$

$$+31,96875_{10} = 0011111.1111100_2$$

$$-31,96875_{10} = 1100000.0000100_2$$

- Si solo se trunca en 4 bits

$$1100000.0000\cancel{100}_2 \rightarrow 1100000.0000_2 = -32_{10}$$

- Si se redondea en 4 bits y luego se trunca

$$\begin{array}{r} 1100000.0000100_2 \\ + \\ \hline \end{array}$$

$$1100000.00010_2 = 1100000.0001_2 = -31,9375_{10}$$

- Haciendo cuentas:

$$-31,9453125 - (-32) = 0,0546875 < 0,0625 = 2^{-4}$$

$$-31,9375 - (-31,9453125) = 0,0078125 < 0,01953125 = 2^{-5}$$

- El número $-31,9453125$ no es representable en 11 bits con 4 fraccionarios. Se podrán representar o el -32_d o el $-31,9375_d$.



Números Reales – Punto Flotante

- Para el caso de los números reales se trabaja en notación científica.
 - $-725,832 = -7,25832 \cdot 10^2 = -725,832 \times 10^0$
 - $3,14 = 0,314 \cdot 10^1 = 3,14 \cdot 10^0$
 - $0,000001 = 0,1 \cdot 10^{-5} = 1,0 \cdot 10^{-6}$
 - $1941 = 0,1941 \cdot 10^4 = 1,941 \cdot 10^3$
- Para unificar la representación se recurre a la notación científica normalizada, en donde
 - $n = \pm m \cdot 10^{\pm e} \rightarrow m = \text{mantisa y } e = \text{exponente}$
 - $0,1 \leq m < 1$
- En el sistema binario la expresión de un número en notación científica normalizada es:
 - $n = \pm m \cdot 2^{\pm e}$
 - $0,1_b \leq m < 1_b \text{ --- } 0,5_d \leq f < 1_d$

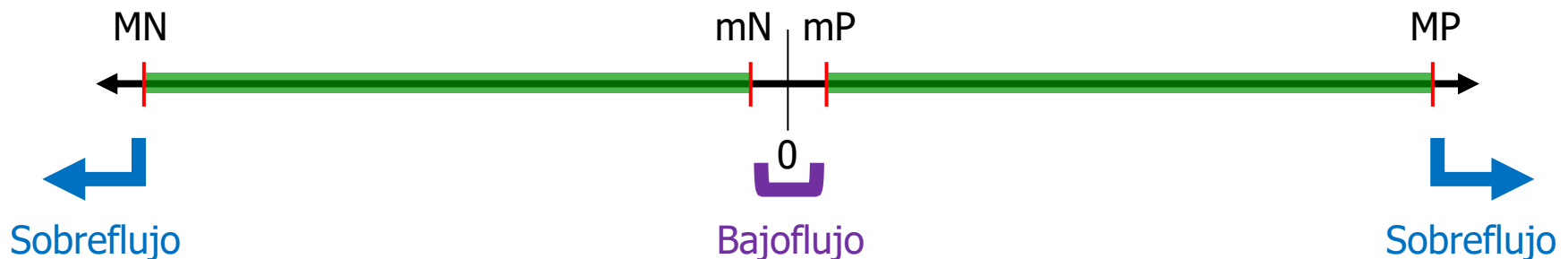


Números Reales – Punto Flotante

- Para que las representaciones sean únicas, la mantisa deberá estar normalizada.
- Normalización → Una cifra tiene:
 - Parte entera = 0 (cero)
 - Parte fraccionaria = su dígito más significativo es distinto de 0
- Ejemplos
 - $+1000 \rightarrow \text{Normalizado} = +0,1 \times 10^{+3}$
 - $-3,141592... \rightarrow \text{Normalizado} = -0,3141592... \times 10^{+1}$
 - $+0,00125 \rightarrow \text{Normalizado} = +0,125 \times 10^{-2}$

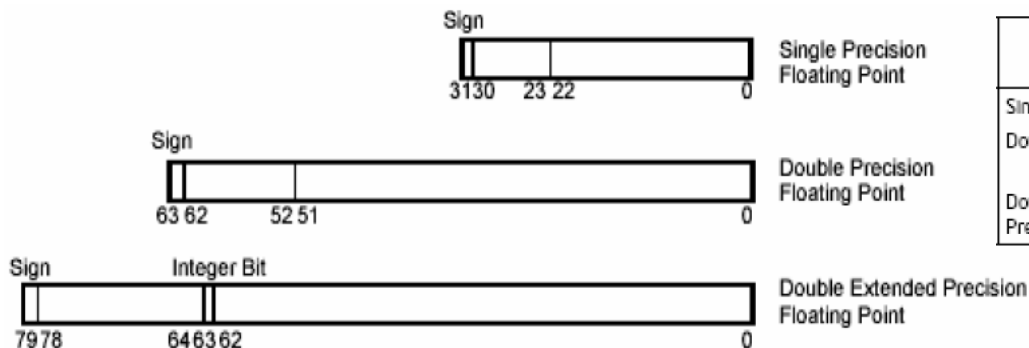
Números Reales – Punto Flotante

- Representación en Punto Flotante
- El rango de representación en punto flotante debe ser analizado teniendo en cuenta los máximos y mínimos valores representables tanto con signo positivo como negativo:
 - **mP (mín número positivo)** = + mantisa mín * base - **exponente máx**
 - **MP (máx número positivo)** = + mantisa máx * base + **exponente máx**
 - **mN (mín número negativo)** = - mantisa máx * base - **exponente máx**
 - **MN (máx número negativo)** = - mantisa mín * base + **exponente máx**



Números Reales – Punto Flotante

- Punto Flotante: Formato IEEE 754 (Institute of Electrical and Electronics Engineers, año 1985 y actualizada 2008)
- Base binaria
 - Precisión media (16 bits)
 - Precisión simple (32 bits)
 - Precisión simple extendida (≥ 43 bits), no usada
 - Precisión doble (64 bits)
 - Precisión cuádruple (128 bits)
 - Precisión óctuple (256 bits)



| Data Type | Length | Precision (Bits) | Approximate Normalized Range | |
|---------------------------|--------|------------------|------------------------------|---|
| | | | Binary | Decimal |
| Single Precision | 32 | 24 | 2^{-126} to 2^{127} | 1.18×10^{-38} to 3.40×10^{38} |
| Double Precision | 64 | 53 | 2^{-1022} to 2^{1023} | 2.23×10^{-308} to 1.79×10^{308} |
| Double Extended Precision | 80 | 64 | 2^{-16382} to 2^{16383} | 3.37×10^{-4932} to 1.18×10^{4932} |

Números Reales – Punto Flotante

■ Representación en Punto Flotante Precisión Simple:

- Máscara o plantilla



- bits 0 al 22 (b_0 a b_{22}) representa la parte fraccionaria de la mantisa normalizada
 - bits 23 al 30 (b_{23} a b_{30}) representa el exponente con exceso 127 (exponente obtenido luego de normalizar y sumándole 127)
 - bit 31 (b_{31}), será el signo de la mantisa, 0 = (+) y 1 = (-)
 - el 0_d se representa con los 32 bits en 0.
-
- Exponente sesgado en 127
 - El sesgo es $2^{n-1} = 128$ pero en este caso se suma 127 quedando los exponentes posibles desde 1_d a 254_d (01_h a FE_h)
 - Los exponentes 0_d a 255_d (00_h a FF_h) se utilizan para casos especiales

Números Reales – Punto Flotante

- Ej: Representar el número +12,25 en formato de 32 bits:
 - Paso 1 – Pasar a binario
 - $+12,25_d = +1100,01_b * 2^0 = +1,10001_b * 2^{+3}$
 - Paso 2 – Normalizar IEEE
 - El exponente de valor +3 en exceso a 127 es: $+3 + 127 = 130_d$ (número entero binario sin signo) = 10000010_b
 - Paso 3 – Signo de la mantisa
 - El signo de la mantisa es + \rightarrow **0**
 - Paso 4 – Completar la plantilla
 - **01000001010001000000000000000000**_b
 - Representándolo en hexadecimal será:
 - **0100-0001-0100-0100-0000-0000-0000-0000**_b
 - 4 - 1 - C - 4 - 0 - 0 - 0 - 0_h
 - **41C40000**_h

Números Reales – Punto Flotante

Ej: Determinar el valor decimal de $C3CC7000_h$ representado en IEEE 754 SP:

- Paso 1 – Pasar el IEEE de hexadecimal a binario

- $C3CC7000_h$
- $1100-0011-1100-1100-0111-0000-0000-0000_b$
- $11000011110011000111000000000000_b$

- Paso 2 – Aplicar la plantilla

- **1****10000111****100110001110000000000000**_b

- Paso 3 – Signo de la mantisa

- **1**_b → El signo de la mantisa es (–)

- Paso 4 – Determinar el exponente sin exceso

- Exponente = **10000111**_b = $135_d \rightarrow 135_d - 127_d = 8_d$

- Armar el binario:

- **– 1,10011000111**_b × 2^8 = **– 110011000,111**_b × 2^0 = **– 408,875_d**



Números Reales – Punto Flotante

- El estándar IEEE 754 define los siguientes conjuntos de los valores posibles que pueden representarse:
 - Cero
 - Ceros signados
 - Valores finitos normalizados y desnormalizados
 - Valores especiales
 - Infinitos ($+\infty$ y $-\infty$)
 - NaN (Not a Number)
- Define 5 algoritmos de redondeo. Si un número cae en medio de dos representaciones, se puede redondear de la siguiente manera:
 - Se elige el más cercano que termine en cero (LSB = 0)
 - Se elige el más cercano hacia arriba (para +) o hacia abajo (para -)
 - Redondeos directos a:
 - cero (llamado truncamiento)
 - $+\infty$
 - $-\infty$



Números Reales – Punto Flotante

- El estándar define cinco excepciones que permitirán manejar los problemas que surgen en las operaciones con punto flotante. Estos son:
 - Operación inválida (x ej. $\sqrt{-2}$) \rightarrow NaN
 - Cero dividido cero \rightarrow NaN
 - Overflow (el resultado es tan grande no puede representarse correctamente dentro del rango de números finitos) \rightarrow Infinitos
 - Underflow (ídem para números muy pequeños) \rightarrow Desnormalización
 - Inexacto \rightarrow Redondeo



Números Reales – Punto Flotante

- Cero y ceros signados

- El cero es representado por el $+0$ (cero positivo) pero tiene el mismo valor que el -0 (cero negativo)
- El resultado puede ser $+0$ o -0 dependiendo de la operación.
Por ejemplo:
 - $-0/x = -0$ (si x es positiva)
 - $(-0).(-0) = +0$
- Los ceros signados ayudan a interpretar el intervalo aritmético en el que se ubicaría el resultado si la precisión aritmética fuese mayor (el resultado ha sido redondeado)
- Indican la dirección desde la cual ocurrió el redondeo a cero, o el signo de un infinito que fue invertido.



Números Reales – Punto Flotante

■ Números finitos normalizados

- El rango de éstos números se compone de todos los valores finitos distintos de cero codificables en formato de números reales entre 0 y $\pm\infty$
- En el formato de simple precisión (32 bits) incluyen números cuyos exponentes van de van de 2^{-126} a 2^{127} y se representan con exceso de 127 (exponentes van desde 1 a 254)
- La parte fraccionaria se normaliza de la siguiente forma:
1.xxxxxx... siendo xxxxxx... la mantisa → Permite ganar 1 bit más!
- Números menores a 2^{-126} no pueden expresarse en este formato ya que el rango del exponente no puede compensar el desplazamiento a izquierda del punto decimal, se pasa al rango de-normalizado

Números Reales – Punto Flotante

- Números finitos denormalizados (subnormal)
 - Su parte fraccionaria se representa como $0.xxxxxx...$, siendo $xxxxxx...$ la mantisa
 - El exponente es siempre 0 (2^{-126})
 - Si operamos con números normalizados, un underflow podrá expresarse en forma denormalizada
 - Ej:

| Operation | Sign | Exponent* | Significand |
|-----------------|------|-----------|--------------------|
| True Result | 0 | -129 | 1.01011100000...00 |
| Denormalize | 0 | -128 | 0.10101110000...00 |
| Denormalize | 0 | -127 | 0.01010111000...00 |
| Denormalize | 0 | -126 | 0.00101011100...00 |
| Denormal Result | 0 | -126 | 0.00101011100...00 |

* Expressed as an unbiased, decimal number.

Bits de pérdida de precisión



Números Reales – Punto Flotante

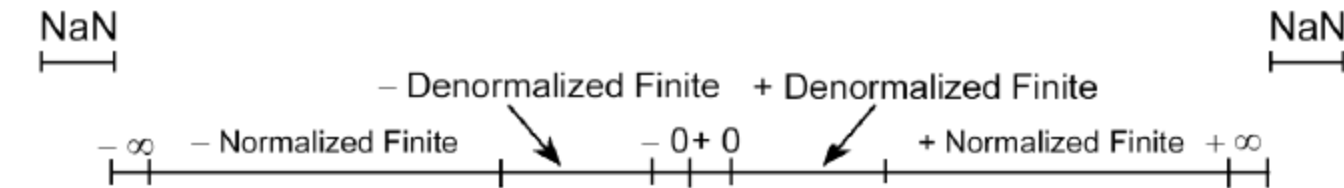
- Infinitos signados

- $+\infty$ y $-\infty$, representan los máximos números reales positivo y negativo representables en formato punto flotante
- La mantisa siempre es 1.000..00 y el máximo exponente desplazado representable (ej. 255 para simple precisión)

- NaNs (Not a Number)

- Se utilizan para expresar un resultado imposible de calcular como las raíces cuadradas negativas, las indeterminaciones ($0/0$, $0*\infty$, $\infty+[-\infty]$), $\log n$ siendo $n < 1$, etc.
- Existen 2 tipos:
 - QNaN: Quiet NaN tiene el bit más significativo fraccional en 1. Pueden propagarse por posteriores operaciones sin generar una excepción
 - SNaN: Signalling NaN. Tiene el bit más significativo fraccional en 0. Resulta de una operación inválida

Números Reales – Punto Flotante



Real Number and NaN Encodings For 32-Bit Floating-Point Format

| S | E | Sig ¹ | | | S | E | Sig ¹ |
|----------------|---------|-----------------------|-----------------------|----------------------|----------------|---------|-----------------------|
| 1 | 0 | 0.000... | - 0 | + 0 | 0 | 0 | 0.000... |
| 1 | 0 | 0.XXX... ² | - Denormalized Finite | +Denormalized Finite | 0 | 0 | 0.XXX... ² |
| 1 | 1...254 | 1.XXX... | - Normalized Finite | +Normalized Finite | 0 | 1...254 | 1.XXX... |
| 1 | 255 | 1.000... | - ∞ | + ∞ | 0 | 255 | 1.000... |
| X ³ | 255 | 1.0XX... ² | SNaN | SNaN | X ³ | 255 | 1.0XX... ² |
| X ³ | 255 | 1.1XX... | QNaN | QNaN | X ³ | 255 | 1.1XX... |

NOTES:

1. Integer bit of fraction implied for single-precision floating-point format.
2. Fraction must be non-zero.
3. Sign bit ignored.

Números Reales – Resumen

| Class | | Sign | Biased Exponent | Significand | |
|----------------------------|--------------------------------------|------|-----------------|----------------------|---------------------|
| | | | | Integer ¹ | Fraction |
| Positive | $+\infty$ | 0 | 11..11 | 1 | 00..00 |
| | +Normals | 0 | 11..10 | 1 | 11..11 |
| | | . | . | . | . |
| | | 0 | 00..01 | 1 | 00..00 |
| | +Denormals | 0 | 00..00 | 0 | 11..11 |
| | | . | . | . | . |
| | | 0 | 00..00 | 0 | 00..01 |
| | +Zero | 0 | 00..00 | 0 | 00..00 |
| Negative | −Zero | 1 | 00..00 | 0 | 00..00 |
| | −Denormals | 1 | 00..00 | 0 | 00..01 |
| | | . | . | . | . |
| | | 1 | 00..00 | 0 | 11..11 |
| | −Normals | 1 | 00..01 | 1 | 00..00 |
| | | . | . | . | . |
| | | 1 | 11..10 | 1 | 11..11 |
| | $-\infty$ | 1 | 11..11 | 1 | 00..00 |
| NaNs | SNaN | X | 11..11 | 1 | 0X..XX ² |
| | QNaN | X | 11..11 | 1 | 1X..XX |
| | QNaN Floating-Point Indefinite | 1 | 11..11 | 1 | 10..00 |
| Single-Precision: | | | ← 8 Bits → | | ← 23 Bits → |
| Double-Precision: | | | ← 11 Bits → | | ← 52 Bits → |
| Double Extended-Precision: | | | ← 15 Bits → | | ← 63 Bits → |

1. El bit entero está implícito y no se almacena para formatos single-precision y double-precision.

2. La fracción para codificación de SNaN debe ser distinta de cero, con el bit mas significativo en 0.