Números y aritmética

Arquitectura de Computadores – IIC2343

La cantidad de bits disponibles para representar un número queda fija al momento de diseñar el computador:

números de precisión finita

Esto tiene consecuencias

(Esta no es una limitación de la base 2, sino de que la memoria del computador —independiente de la tecnología que se use para implementarla— es finita)

P.ej., consideremos el conjunto de números enteros positivos representables mediante tres dígitos decimales, sin punto decimal ni signo:

000, 001, 002, …, 999

P.ej., consideremos el conjunto de números enteros positivos representables mediante tres dígitos decimales, sin punto decimal ni signo:

000, 001, 002, …, 999

En este caso, es imposible representar ciertos números:

- mayores que 999
- negativos
- fracciones
- irracionales
- complejos

P.ej., consideremos el conjunto de números enteros positivos representables mediante tres dígitos decimales, sin punto decimal ni signo:

000, 001, 002, …, 999

En este caso, es imposible representar ciertos números:

- mayores que 999
- negativos
- fracciones
- irracionales
- complejos

Además, el conjunto no es cerrado con respecto a las operaciones aritméticas básicas:

- 600 + 600 = 1200 → muy grande
- 003 – 005 = –2 → negativo
- 050 x 050 = 2500 → muy grande
- 007 / 002 = 3.5 → no es un entero

P.ej., consideremos el conjunto de números enteros positivos representables mediante tres dígitos decimales, sin punto decimal ni signo:

000, 001, 002, …, 999

En este caso, es imposible representar ciertos números:

- mayores que 999
- negativos
- fracciones
- irracionales
- complejos

Finalmente, el álgebra de los números de precisión finita es diferente del álgebra "normal":

- p.ej., la ley de asociatividad $a + (b - c) = (a + b) - c$ no se cumple si $a = 700$, $b = 400$ y $c = 300$

  … porque al calcular $a + b$ en el lado derecho produce *overflow*

Además, el conjunto no es cerrado con respecto a las operaciones aritméticas básicas:

- 600 + 600 = 1200 → muy grande
- 003 – 005 = –2 → negativo
- 050 x 050 = 2500 → muy grande
- 007 / 002 = 3.5 → no es un entero

Estas mismas limitaciones se dan al representar números enteros positivos (en base 2) mediante, p.ej., 32 bits —típico en un computador (casi) moderno

Obviamente, no es que los computadores no sean adecuados para hacer aritmética, sino que

… **es importante entender cómo funcionan**

(lo que vamos a ver a continuación)

En computadores en que las palabras tienen 32 bits

… podemos representar 2^{32} patrones diferentes de bits

… los números desde el 0 hasta el $2^{32} – 1 = 4{,}294{,}967{,}295_{10}$

(Vamos a usar el subíndice 10 cuando escribamos números en base 10, y el subíndice 2 cuando escribamos números en base 2; p.ej., 421_{10} y 110100101_2)

Con 32 bits, el valor del número representado como

$$x_{31}\, x_{30} \cdots x_1\, x_0$$

… es

$$(x_{31} \times 2^{31}) + (x_{30} \times 2^{30}) + (x_{29} \times 2^{29}) + \cdots + (x_1 \times 2^1) + (x_0 \times 2^0)$$

Estos números positivos, incluyendo el 0, se llaman **números sin signo** (*unsigned numbers*) o **enteros sin signo** (*unsigned integers*)

También tenemos que representar **números negativos**

También tenemos que representar **números negativos**

P.ej., podríamos agregar un signo, representado en un bit adicional:

$0\ 1\ 1\ 0\ 1\ 0\ 0\ 1\ 0\ 1_2 = 421_{10}$ $\quad$ $1\ 1\ 1\ 0\ 1\ 0\ 0\ 1\ 0\ 1_2 = -421_{10}$

Esta representación se llama **signo y magnitud**

$0\,1\,1\,0\,1\,0\,0\,1\,0\,1_2 = 421_{10}$ $\quad$ $1\,1\,1\,0\,1\,0\,0\,1\,0\,1_2 = -421_{10}$

Problemas de la representación *signo y magnitud*:

- el bit de signo, ¿es el de más a la izquierda o el de más a la derecha?
- al sumar, se necesita un paso adicional para saber el valor de este bit
- hay dos ceros —uno positivo y otro negativo ($00\ldots00_2$ y $10\ldots00_2$)

Hay otras opciones, todas con sus pros y sus contras

No habiendo una mejor opción obvia

… los arquitectos computacionales eligieron finalmente la representación llamada ***complemento de 2*** que hacía más simple el hardware:

- 0s a la izquierda significa positivo
- 1s a la izquierda significa negativo

Veámoslo para 32 bits (próxs. diaps.)

[muchos computadores hoy día usan 64 bits y en el pasado usaban 16 bits —la representación en *complemento de 2* es independiente del número de bits: básicamente, lo único que se ve afectado es el rango de números representados]

$$0000\ 0000\ 0000\ 0000\ 0000\ 0000\ 0000\ 0000_2 = 0_{10}$$
$$0000\ 0000\ 0000\ 0000\ 0000\ 0000\ 0000\ 0001_2 = 1_{10}$$
$$0000\ 0000\ 0000\ 0000\ 0000\ 0000\ 0000\ 0010_2 = 2_{10}$$
$$\vdots$$
$$0111\ 1111\ 1111\ 1111\ 1111\ 1111\ 1111\ 1101_2 = 2{,}147{,}483{,}645_{10}$$
$$0111\ 1111\ 1111\ 1111\ 1111\ 1111\ 1111\ 1110_2 = 2{,}147{,}483{,}646_{10}$$
$$0111\ 1111\ 1111\ 1111\ 1111\ 1111\ 1111\ 1111_2 = 2{,}147{,}483{,}647_{10}$$

$$1000\ 0000\ 0000\ 0000\ 0000\ 0000\ 0000\ 0000_2 = -2{,}147{,}483{,}648_{10}$$
$$1000\ 0000\ 0000\ 0000\ 0000\ 0000\ 0000\ 0001_2 = -2{,}147{,}483{,}647_{10}$$
$$1000\ 0000\ 0000\ 0000\ 0000\ 0000\ 0000\ 0010_2 = -2{,}147{,}483{,}646_{10}$$
$$\vdots$$
$$1111\ 1111\ 1111\ 1111\ 1111\ 1111\ 1111\ 1101_2 = -3_{10}$$
$$1111\ 1111\ 1111\ 1111\ 1111\ 1111\ 1111\ 1110_2 = -2_{10}$$
$$1111\ 1111\ 1111\ 1111\ 1111\ 1111\ 1111\ 1111_2 = -1_{10}$$

$0000\ 0000\ 0000\ 0000\ 0000\ 0000\ 0000\ 0000_2 = 0_{10}$

$0000\ 0000\ 0000\ 0000\ 0000\ 0000\ 0000\ 0001_2 = 1_{10}$

$0000\ 0000\ 0000\ 0000\ 0000\ 0000\ 0000\ 0010_2 = 2_{10}$

$\vdots$

$0111\ 1111\ 1111\ 1111\ 1111\ 1111\ 1111\ 1101_2 = 2{,}147{,}483{,}645_{10}$

$0111\ 1111\ 1111\ 1111\ 1111\ 1111\ 1111\ 1110_2 = 2{,}147{,}483{,}646_{10}$

$0111\ 1111\ 1111\ 1111\ 1111\ 1111\ 1111\ 1111_2 = 2{,}147{,}483{,}647_{10}$

esta mitad positiva, de 0 a $2{,}147{,}483{,}647_{10}$ $(= 2^{31} - 1)$ usa la misma representación que antes

$1000\ 0000\ 0000\ 0000\ 0000\ 0000\ 0000\ 0000_2 = -2{,}147{,}483{,}648_{10}$

$1000\ 0000\ 0000\ 0000\ 0000\ 0000\ 0000\ 0001_2 = -2{,}147{,}483{,}647_{10}$

$1000\ 0000\ 0000\ 0000\ 0000\ 0000\ 0000\ 0010_2 = -2{,}147{,}483{,}646_{10}$

$\vdots$

$1111\ 1111\ 1111\ 1111\ 1111\ 1111\ 1111\ 1101_2 = -3_{10}$

$1111\ 1111\ 1111\ 1111\ 1111\ 1111\ 1111\ 1110_2 = -2_{10}$

$1111\ 1111\ 1111\ 1111\ 1111\ 1111\ 1111\ 1111_2 = -1_{10}$

$0000\ 0000\ 0000\ 0000\ 0000\ 0000\ 0000\ 0000_2 = 0_{10}$

$0000\ 0000\ 0000\ 0000\ 0000\ 0000\ 0000\ 0001_2 = 1_{10}$

$0000\ 0000\ 0000\ 0000\ 0000\ 0000\ 0000\ 0010_2 = 2_{10}$

⋮

$0111\ 1111\ 1111\ 1111\ 1111\ 1111\ 1111\ 1101_2 = 2{,}147{,}483{,}645_{10}$

$0111\ 1111\ 1111\ 1111\ 1111\ 1111\ 1111\ 1110_2 = 2{,}147{,}483{,}646_{10}$

$0111\ 1111\ 1111\ 1111\ 1111\ 1111\ 1111\ 1111_2 = 2{,}147{,}483{,}647_{10}$

esta mitad positiva, de 0 a $2{,}147{,}483{,}647_{10}$ $(= 2^{31} - 1)$ usa la misma representación que antes

$1000\ 0000\ 0000\ 0000\ 0000\ 0000\ 0000\ 0000_2 = -2{,}147{,}483{,}648_{10}$

$1000\ 0000\ 0000\ 0000\ 0000\ 0000\ 0000\ 0001_2 = -2{,}147{,}483{,}647_{10}$

$1000\ 0000\ 0000\ 0000\ 0000\ 0000\ 0000\ 0010_2 = -2{,}147{,}483{,}646_{10}$

⋮

$1111\ 1111\ 1111\ 1111\ 1111\ 1111\ 1111\ 1101_2 = -3_{10}$

$1111\ 1111\ 1111\ 1111\ 1111\ 1111\ 1111\ 1110_2 = -2_{10}$

$1111\ 1111\ 1111\ 1111\ 1111\ 1111\ 1111\ 1111_2 = -1_{10}$

secuencia de números negativos de magnitud decreciente, desde $-2{,}147{,}483{,}647_{10}$ $(= 1000\ldots001_2)$ hasta -1_{10} $(= 1111\ldots111_2)$

$0000\ 0000\ 0000\ 0000\ 0000\ 0000\ 0000\ 0000_2 = 0_{10}$

$0000\ 0000\ 0000\ 0000\ 0000\ 0000\ 0000\ 0001_2 = 1_{10}$

$0000\ 0000\ 0000\ 0000\ 0000\ 0000\ 0000\ 0010_2 = 2_{10}$

⋮

$0111\ 1111\ 1111\ 1111\ 1111\ 1111\ 1111\ 1101_2 = 2{,}147{,}483{,}645_{10}$

$0111\ 1111\ 1111\ 1111\ 1111\ 1111\ 1111\ 1110_2 = 2{,}147{,}483{,}646_{10}$

$0111\ 1111\ 1111\ 1111\ 1111\ 1111\ 1111\ 1111_2 = 2{,}147{,}483{,}647_{10}$

esta mitad positiva, de 0 a $2{,}147{,}483{,}647_{10}$ $(= 2^{31} - 1)$ usa la misma representación que antes

$1000\ 0000\ 0000\ 0000\ 0000\ 0000\ 0000\ 0000_2 = -2{,}147{,}483{,}648_{10}$

$1000\ 0000\ 0000\ 0000\ 0000\ 0000\ 0000\ 0001_2 = -2{,}147{,}483{,}647_{10}$

$1000\ 0000\ 0000\ 0000\ 0000\ 0000\ 0000\ 0010_2 = -2{,}147{,}483{,}646_{10}$

⋮

$1111\ 1111\ 1111\ 1111\ 1111\ 1111\ 1111\ 1101_2 = -3_{10}$

$1111\ 1111\ 1111\ 1111\ 1111\ 1111\ 1111\ 1110_2 = -2_{10}$

$1111\ 1111\ 1111\ 1111\ 1111\ 1111\ 1111\ 1111_2 = -1_{10}$

número más negativo, $-2{,}147{,}483{,}648_{10}$ $(= -2^{31})$; no tiene un número positivo correspondiente

secuencia de números negativos de magnitud decreciente, desde $-2{,}147{,}483{,}647_{10}$ $(= 1000\ldots001_2)$ hasta -1_{10} $(= 1111\ldots111_2)$

Todo computador hoy en día, y desde 1965, usa *complemento de 2* para representar **números con signo** (*signed numbers*)

string	unsigned	sign & magnitude	1's complement	2's complement
0000	0	0	0	**0**
0001	1	1	1	**1**
0010	2	2	2	**2**
0011	3	3	3	**3**
0100	4	4	4	**4**
0101	5	5	5	**5**
0110	6	6	6	**6**
0111	7	7	7	**7**
1000	8	–0	–7	**–8**
1001	9	–1	–6	**–7**
1010	10	–2	–5	**–6**
1011	11	–3	–4	**–5**
1100	12	–4	–3	**–4**
1101	13	–5	–2	**–3**
1110	14	–6	–1	**–2**
1111	15	–7	–0	**–1**

En *complemento de 2*, todos los números negativos —y sólo los números negativos— tienen un 1 en el bit más significativo (el de más a la izquierda):

- se lo puede considerar como un bit de signo
- basta examinar este bit para saber si un número es positivo o negativo (0 se considera positivo)
- aunque no es un bit que pueda realmente separarse de la magnitud del número (como se explica en la próx. diap.)

  p.ej., en 4 bits (diap. anterior), $1001_2 = -7_{10}$, pero $0001_2 \neq 7_{10}$

Así, para 32 bits en *complemento de 2*, el valor del número representado como

$$x_{31}\, x_{30} \cdots x_1\, x_0$$

… es

$$(x_{31} \times -2^{31}) + (x_{30} \times 2^{30}) + (x_{29} \times 2^{29}) + \cdots + (x_1 \times 2^1) + (x_0 \times 2^0)$$

Notar la diferencia en la interpretación del valor del bit 31 en el caso de los números sin signo (diap. 10):

$$(x_{31} \times 2^{31}) + (x_{30} \times 2^{30}) + (x_{29} \times 2^{29}) + \cdots + (x_1 \times 2^1) + (x_0 \times 2^0)$$

Complemento de 2
8 bits

$00000000_2 = 0_{10}$
$00000001_2 = 1_{10}$
$00000010_2 = 2_{10}$
$00000011_2 = 3_{10}$
⋮
$01111100_2 = 124_{10}$
$01111101_2 = 125_{10}$
$01111110_2 = 126_{10}$
$01111111_2 = 127_{10}$
$10000000_2 = -128_{10}$
$10000001_2 = -127_{10}$
$10000010_2 = -126_{10}$
$10000011_2 = -125_{10}$
⋮
$11111100_2 = -4_{10}$
$11111101_2 = -3_{10}$
$11111110_2 = -2_{10}$
$11111111_2 = -1_{10}$

$$\begin{array}{rll} & 0\,0\,0\,0\,0\,0\,1\,1_2 & (= 3_{10}) \\ + & 0\,1\,1\,1\,1\,1\,0\,0_2 & (= 124_{10}) \\ \hline & 0\,1\,1\,1\,1\,1\,1\,1_2 & (= 127_{10}) \end{array}$$

Complemento de 2
8 bits

$00000000_2 = 0_{10}$
$00000001_2 = 1_{10}$
$00000010_2 = 2_{10}$
$00000011_2 = 3_{10}$
⋮
$01111100_2 = 124_{10}$
$01111101_2 = 125_{10}$
$01111110_2 = 126_{10}$
$01111111_2 = 127_{10}$
$10000000_2 = -128_{10}$
$10000001_2 = -127_{10}$
$10000010_2 = -126_{10}$
$10000011_2 = -125_{10}$
⋮
$11111100_2 = -4_{10}$
$11111101_2 = -3_{10}$
$11111110_2 = -2_{10}$
$11111111_2 = -1_{10}$

$$\begin{array}{rl} & 1\,0\,0\,0\,0\,0\,1\,1_2 \quad (= -125_{10}) \\ + & 1\,1\,1\,1\,1\,1\,1\,1_2 \quad (= -1_{10}) \\ \hline & 1\,1\,0\,0\,0\,0\,0\,1\,0_2 \quad (= -126_{10}) \end{array}$$

El 1 simplemente se descarta

Complemento de 2
8 bits

$00000000_2 = 0_{10}$
$00000001_2 = 1_{10}$
$00000010_2 = 2_{10}$
$00000011_2 = 3_{10}$
⋮
$01111100_2 = 124_{10}$
$01111101_2 = 125_{10}$
$01111110_2 = 126_{10}$
$01111111_2 = 127_{10}$
$10000000_2 = -128_{10}$
$10000001_2 = -127_{10}$
$10000010_2 = -126_{10}$
$10000011_2 = -125_{10}$
⋮
$11111100_2 = -4_{10}$
$11111101_2 = -3_{10}$
$11111110_2 = -2_{10}$
$11111111_2 = -1_{10}$

$$\begin{array}{rll} & 0\,0\,0\,0\,0\,0\,1\,0_2 & (=2_{10}) \\ + & 0\,1\,1\,1\,1\,1\,1\,1_2 & (=127_{10}) \\ \hline & 1\,0\,0\,0\,0\,0\,0\,1_2 & (=-127_{10}) \end{array}$$

El 1 indica que **hay un problema** con el resultado

Complemento de 2
8 bits

$00000000_2 = 0_{10}$
$00000001_2 = 1_{10}$
$00000010_2 = 2_{10}$
$00000011_2 = 3_{10}$
⋮
$01111100_2 = 124_{10}$
$01111101_2 = 125_{10}$
$01111110_2 = 126_{10}$
$01111111_2 = 127_{10}$
$10000000_2 = -128_{10}$
$10000001_2 = -127_{10}$
$10000010_2 = -126_{10}$
$10000011_2 = -125_{10}$
⋮
$11111100_2 = -4_{10}$
$11111101_2 = -3_{10}$
$11111110_2 = -2_{10}$
$11111111_2 = -1_{10}$

$$\begin{array}{rl} & 1\,0\,0\,0\,0\,0\,1\,1_2 \quad (= -125_{10}) \\ + & 1\,1\,1\,1\,1\,1\,0\,0_2 \quad (= -4_{10}) \\ \hline & 1\,0\,1\,1\,1\,1\,1\,1\,1_2 \quad (= 127_{10}) \end{array}$$

El 1 simplemente se descarta, pero el 0 indica que **hay un problema** con el resultado

Complemento de 2
8 bits

$00000000_2 = 0_{10}$
$00000001_2 = 1_{10}$
$00000010_2 = 2_{10}$
$00000011_2 = 3_{10}$
⋮
$01111100_2 = 124_{10}$
$01111101_2 = 125_{10}$
$01111110_2 = 126_{10}$
$01111111_2 = 127_{10}$
$10000000_2 = -128_{10}$
$10000001_2 = -127_{10}$
$10000010_2 = -126_{10}$
$10000011_2 = -125_{10}$
⋮
$11111100_2 = -4_{10}$
$11111101_2 = -3_{10}$
$11111110_2 = -2_{10}$
$11111111_2 = -1_{10}$

$$\begin{array}{rll} & 0\,0\,0\,0\,0\,0\,1\,1_2 & (=3_{10}) \\ + & 0\,1\,1\,1\,1\,1\,0\,0_2 & (=124_{10}) \\ \hline & 0\,1\,1\,1\,1\,1\,1\,1_2 & (=127_{10}) \end{array}$$

$$\begin{array}{rll} & 1\,0\,0\,0\,0\,0\,1\,1_2 & (=-125_{10}) \\ + & 1\,1\,1\,1\,1\,1\,1\,1_2 & (=-1_{10}) \\ \hline & 1\,1\,0\,0\,0\,0\,0\,1\,0_2 & (=-126_{10}) \end{array}$$

$$\begin{array}{rll} & 0\,0\,0\,0\,0\,0\,1\,0_2 & (=2_{10}) \\ + & 0\,1\,1\,1\,1\,1\,1\,1_2 & (=127_{10}) \\ \hline & 1\,0\,0\,0\,0\,0\,0\,1_2 & (=-127_{10}) \end{array}$$

$$\begin{array}{rll} & 1\,0\,0\,0\,0\,0\,1\,1_2 & (=-125_{10}) \\ + & 1\,1\,1\,1\,1\,1\,0\,0_2 & (=-4_{10}) \\ \hline & 1\,0\,1\,1\,1\,1\,1\,1\,1_2 & (=127_{10}) \end{array}$$

En complemento de 2, ocurre *overflow* (como en las dos últimas sumas de la diap. anterior)

… cuando el resultado de la operación produce un bit de signo incorrecto (bits rojos):

- un 0 en el bit de más a la izquierda (bit *n*–1 ésimo) cuando el número es negativo
- un 1 en el bit de más a la izquierda (bit *n*–1 ésimo) cuando el número es positivo

¿ Cómo determinamos el inverso aditivo de un número binario *Y* de n bits en complemento de 2 ?

… ya sea para pasar de positivo a negativo:

- p.ej., de 01111110_2 a 10000010_2

… o para pasar de negativo a positivo:

- p.ej., de 11111101_2 a 00000011_2

Complemento de 2
8 bits

$00000000_2 = 0_{10}$
$00000001_2 = 1_{10}$
$00000010_2 = 2_{10}$
$00000011_2 = 3_{10}$
⋮
$01111100_2 = 124_{10}$
$01111101_2 = 125_{10}$
$01111110_2 = 126_{10}$
$01111111_2 = 127_{10}$
$10000000_2 = -128_{10}$
$10000001_2 = -127_{10}$
$10000010_2 = -126_{10}$
$10000011_2 = -125_{10}$
⋮
$11111100_2 = -4_{10}$
$11111101_2 = -3_{10}$
$11111110_2 = -2_{10}$
$11111111_2 = -1_{10}$

Si miramos la diap. anterior (o la diap. 18 o la última columna de la diap. 20), vemos que

… $Y + \bar{Y}$ (invertimos cada bit de Y) = $111...111_2 = -1_{10}$

- p.ej., 01111110 + 10000001 = 11111111

  … 11111101 + 00000010 = 11111111

… por lo tanto, $Y + \bar{Y} = -1 \Rightarrow -Y = \bar{Y} + 1$

Es decir, el inverso aditivo $-Y$ se obtiene en dos pasos:

primero, invertimos cada bit de Y ($0 \rightarrow 1$, $1 \rightarrow 0$), lo que nos da $\bar{Y}$

… y luego, sumamos 1 al resultado

Por otra parte, si no tomamos en cuenta el signo,

$$Y + \bar{Y} = 111...111 = 2^n - 1$$

$$\Rightarrow Y + \bar{Y} + 1 = 2^n$$

$$\Rightarrow Y + (-Y) = 2^n$$

De aquí el nombre “complemento de 2”