NÚMEROS CON SIGNO

1. Formato *Signo-Magnitude*

Para codificar números enteiros é necesario representar e diferenciar os positivos dos negativos. Existen varios sistemas, cada un coas súas vantaxes e inconvintes.

O formato *signo-magnitude* (S-M) é o máis intuitivo, xa que se asemella ao sistema utilizado habitualmente polos seres humáns: $-3, +5, -1, ...$

1.1. Descrición do formato *signo-magnitude*

A principal característica do formato *signo-magnitude* consiste en utilizar **un bit para representar o signo do número** (o bit máis significativo, é dicir, o situado máis a esquerda). **O resto de bits representan a magnitude do número**, é dicir, é un número positivo que se codifica coma sempre en binario puro.

Como en binario só existen dúas cifras (0 e 1), non se pode diferenciar o bit de signo do resto de bits polo seu valor, máis si pola súa posición (o signo é o bit máis significativo). Convenio:

Signo positivo: 0 **Signo negativo**: 1

Se empregamos 4 bits para representar un número $(X_3X_2X_1X_0)$ entón:

Signo: X_3 **Magnitude**: $X_2X_1X_0$

Exemplos:

0011 (+3),	0100 (+4),	0000 (+0),	1111 (−7),	0101 (+5).
1011 (−3),	1100 (−4),	1000 (−0),	0111 (+7),	1101 (−5).

1.2. Principais propiedades do formato *signo-magnitude*

- Dobre representación do número 0: 0000 (+0) e 1000 (−0).
- Rango simétrico: con n bits $(X_{n-1} \dots X_0)$ codifícanse valores entre $[-2^{n-1}+1, \dots, 2^{n-1}-1]$. Por exemplo, con 4 bits codifícanse valores entre $[-7, \dots, +7]$.
- É moi doado traducir de decimal a signo-magnitude e viceversa.
- As operacións aritméticas son bastantes complexas e engorrosas.

1.3. Operacións básicas no formato *signo-magnitude*

Número oposto (-X): basta con cambiar o signo de X, é dicir, para catro bits $X'_3 = \overline{X}_3$. Si X é positivo $(X_3 = 0)$ o seu oposto será negativo $(X'_3 = 1)$, e viceversa.

Exemplos: $-(0001) = 1001$, $-(1110) = 0110$, $-(0000) = 1000$.

Valor absoluto ($|X|$): como a magnitude xa está en valor absoluto só é necesario fixar o signo como positivo, é dicir, $X'_3 = 0$.

Exemplos: $|0101| = 0101$, $|1110| = 0110$, $|1001| = 0001$.

Suma ($X + Y$). Existen dous casos:

1. Se os dous operandos teñen o mesmo signo, basta con sumar as dúas magnitudes e poñer o mesmo signo.

   Exemplos: $0011 + 0001 = 0100$, $1101 + 1010 = 1111$, $1110 + 1000 = 1110$.

$$\begin{array}{cccccc} 0 & & 0 & 1 & 1 & (+3) \\ 0 & + & 0 & 0 & 1 & (+1) \\ \hline 0 & & 1 & 0 & 0 & (+4) \end{array} \qquad \begin{array}{cccccc} 1 & & 1 & 0 & 1 & (-5) \\ 1 & + & 0 & 1 & 0 & (-2) \\ \hline 1 & & 1 & 1 & 1 & (-7) \end{array} \qquad \begin{array}{cccccc} 1 & & 1 & 1 & 0 & (-6) \\ 1 & + & 0 & 0 & 0 & (-0) \\ \hline 1 & & 1 & 1 & 0 & (-6) \end{array}$$

   Se para representar o resultado de sumar as magnitudes se precisasen máis bits dos disponibeis para representar á magnitude, entón dise que existe un desbordamento ou *overflow* e o resultado é incorrecto.

   Exemplos: $1111 + 1001 = \textit{overflow}$, $0100 + 0100 = \textit{overflow}$.

$$\begin{array}{cccccccc} 1 & & 1 & 1 & 1 & (-7) & & \\ 1 & + & 0 & 1 & 1 & (-3) & & \\ \hline 1 & 1 & 0 & 1 & 0 & (-10) & \Rightarrow & \textit{overflow} \end{array} \qquad \begin{array}{cccccccc} 0 & & 1 & 0 & 0 & (+4) & & \\ 0 & + & 1 & 0 & 0 & (+4) & & \\ \hline 0 & 1 & 0 & 0 & 0 & (+8) & \Rightarrow & \textit{overflow} \end{array}$$

2. Se teñen signos distintos entón débese restar a magnitude máis pequena da magnitude máis grande e o signo final será o signo do número con maior magnitude. Neste caso o resultado sempre será máis pequeno ou igual que os operandos e non existirá nunca desbordamento.

   Exemplos: $1111 + 0001 = 1110$, $0100 + 1100 = 0000$, $0011 + 1110 = 1011$.

$$\begin{array}{cccccc} 1 & & 1 & 1 & 1 & (-7) \\ 0 & - & 0 & 0 & 1 & (+1) \\ \hline 1 & & 1 & 1 & 0 & (-6) \end{array} \qquad \begin{array}{cccccc} 0 & & 1 & 0 & 0 & (+4) \\ 1 & - & 1 & 0 & 0 & (-4) \\ \hline 0 & & 0 & 0 & 0 & (+0) \end{array} \qquad \begin{array}{cccccc} 1 & & 1 & 1 & 0 & (-6) \\ 0 & - & 0 & 1 & 1 & (+3) \\ \hline 1 & & 0 & 1 & 1 & (-3) \end{array}$$

Resta ($X - Y$): basta cambiar o subtraendo de signo e sumar, é dicir, $X + (-Y)$

Extensión do formato: se queremos aumentar o número de bits na representación, o procedemento consiste en angadir máis bits (ceros) á magnitude, **deixando sempre o bit máis significativo (o signo) á esquerda**.

Exemplos (extensión de 4 a 6 bits):

$1111 \Rightarrow 100111$ (-7), $0110 \Rightarrow 000110$ $(+6)$, $1001 \Rightarrow 100001$ (-1).

NÚMEROS CON SIGNO

1. Formato *Complemento a 1*

Para codificar números enteiros é necesario representar e diferenciar os positivos dos negativos. Existen varios sistemas, cada un coas súas vantaxes e inconvintes.

1.1. Descrición do formato *complemento a 1*

A principal característica do formato *complemento a 1* é codificar os **números negativos** utilizando a operación denominada *complemento a 1* (por iso o seu nome), é dicir, **negando todos os bits un a un**. Os números positivos represéntanse igual que en binario puro.

O bit máis significativo (o situado máis a esquerda) representa o signo do número. Como en binario só existen dúas cifras (0 e 1), non se pode diferenciar o bit de signo do resto de bits polo seu valor, mais si pola súa posición (o signo é o bit máis significativo). Convenio:

Signo positivo: 0 **Signo negativo**: 1

Se empregamos 4 bits para representar un número ($X_3X_2X_1X_0$) entón: **Signo**: X_3

Exemplos:

0011 (+3), 0100 (+4), 0000 (+0), 1000 (−7), 0101 (+5).
1100 (−3), 1011 (−4), 1111 (−0), 0111 (+7), 1010 (−5).

1.2. Principais propiedades do formato *complemento a 1*

- Dobre representación do número 0: 0000 (+0) e 1111 (−0).
- Rango simétrico: con n bits ($X_{n-1} \dots X_0$) codifícanse valores entre $[-2^{n-1}+1, \dots, 2^{n-1}-1]$. Por exemplo, con 4 bits codifícanse valores entre $[-7, \dots, +7]$.
- A traducción entre decimal e complemento a 1 non é directa para os números negativos.
- As operacións aritméticas son bastantes sinxelas.

1.3. Operacións básicas no formato *complemento a 1*

Número oposto ($-X$): basta con negar todos os bits do número X (aplicando a operación complemento a 1), é dicir, para catro bits $X' = \overline{X}_3\overline{X}_2\overline{X}_1\overline{X}_0$. Se X é positivo ($X_3 = 0$) o seu oposto será negativo ($X'_3 = 1$), e viceversa.

Exemplos: $-(0001) = 1110$, $-(1110) = 0001$, $-(0000) = 1111$.

Valor absoluto ($|X|$): se o número é positivo queda como está e se é negativo débese calcular o seu oposto (ver sección anterior).

Exemplos: $|0101| = 0101$, $|1110| = 0001$, $|1001| = 0110$.

Suma $(X + Y)$: súmanse bit a bit todos os bits de ambos os dous números (incluidos os bits de signo). Se existe carrexo de saída tamén se suma ao resultado.

Exemplos: $0011 + 0001 = 0100$ $(3 + 1 = +4)$, $1101 + 0010 = 1111$ $(-2 + 2 = -0)$, $0110 + 1000 = 1110$ $(6 + -7 = -1)$, $0110 + 1011 = 0010$ $(6 + -4 = +2)$.

	1	1	1	1	(-0)
+	0	0	0	1	$(+1)$
1	0	0	0	0	
+				1	
	0	0	0	1	$(+1)$

	0	1	0	0	$(+4)$
+	1	1	0	0	(-3)
1	0	0	0	0	
+				1	
	0	0	0	1	$(+1)$

	0	1	1	1	$(+7)$
+	1	1	1	0	(-1)
1	0	1	0	1	
+				1	
	0	1	1	0	(-6)

- Se os operandos teñen signos diferentes o resultado da suma sempre é correcto.
- Se os operandos teñen o mesmo signo pode existir desbordamento ou *overflow* (o resultado é demasiado grande e sería preciso un bit máis para ser codificado). **Hai desbordamento cando o signo do resultado non coincide co signo dos operandos**.

Exemplos: $1101 + 1001 = \textit{overflow}$ (0111), $0100 + 0100 = \textit{overflow}$ (1000).

Resta $(X - Y)$: basta cambiar o subtraendo de signo e sumar, é dicir, $X + (-Y)$

Extensión do formato: para aumentar o número de bits utilizados ao representar un número basta engadir máis bits á esquerda **repetindo o bit de signo**. Dese modo o valor do número permanece inalterado.

Exemplos (extensión de 4 a 6 bits):

$1111 = 111111$ (-0), $0110 = 000110$ $(+6)$, $1001 = 111001$ (-6).

NÚMEROS CON SIGNO

1. Formato *Complemento a 2*

Para codificar números enteiros é necesario representar e diferenciar os positivos dos negativos. Existen varios sistemas, cada un coas súas vantaxes e inconvintes.

1.1. Descrición do formato *complemento a 2*

A principal característica do formato *complemento a 2* é codificar os **números negativos** utilizando a operación denominada *complemento a 2* (por iso o seu nome), é dicir, **negando todos os bits un a un e sumando 1**. Os números positivos represéntanse igual que en binario puro.

O bit máis significativo (o situado máis a esquerda) representa o signo do número. Como en binario só existen dúas cifras (0 e 1), non se pode diferenciar o bit de signo do resto de bits polo seu valor, mais si pola súa posición (o signo é o bit máis significativo). Convenio:

Signo positivo: 0 **Signo negativo**: 1

Se empregamos 4 bits para representar un número $(X_3X_2X_1X_0)$ entón: **Signo**: X_3

Exemplos:

0011 (+3),	0100 (+4),	0000 (+0),	1001 (−7),	0101 (+5).
1101 (−3),	1100 (−4),	1000 (−8),	0111 (+7),	1011 (−5).

1.2. Principais propiedades do formato *complemento a 2*

- Representación única do número 0: 0000 (+0).
- Rango asimétrico: con n bits $(X_{n-1} \dots X_0)$ codifícanse valores entre $[-2^{n-1}, \dots, 2^{n-1} - 1]$. Por exemplo, con 4 bits codifícanse valores entre $[-8, \dots, +7]$.
- A traducción entre decimal e complemento a 2 non é directa para os números negativos.
- As operacións aritméticas son bastantes sinxelas.

1.3. Operacións básicas no formato *complemento a 2*

Número oposto $(-X)$: basta con negar todos os bits do número X e sumar 1 (aplicando a operación complemento a 2), é dicir, para catro bits $X' = \overline{X}_3\overline{X}_2\overline{X}_1\overline{X}_0 + 1$. Se X é positivo $(X_3 = 0)$ o seu oposto será negativo $(X'_3 = 1)$, e viceversa, excepto o número -8 (o negativo máis pequeno non se pode poñer en positivo).

Exemplos: $-(0001) = 1111$, $-(1110) = 0010$, $-(0000) = 0000$.

Valor absoluto ($|X|$): se o número é positivo queda como está e se é negativo débese calcular o seu oposto (ver sección anterior).

Exemplos: $|0101| = 0101$, $|1110| = 0010$, $|1001| = 0111$.

Suma ($X + Y$): súmanse bit a bit todos os bits de ambos os dous números (incluidos os bits de signo). Sempre se desprecia o carrexo de saída.

Exemplos: $0011 + 0001 = 0100$ $(3 + 1 = +4)$, $1101 + 0010 = 1111$ $(-3 + 2 = -1)$, $0110 + 1000 = 1110$ $(6 + -8 = -2)$, $0110 + 1011 = 0001$ $(6 + -5 = +2)$.

$$\begin{array}{rrrrrr} & 1 & 1 & 1 & 1 & (-1) \\ + & 0 & 0 & 0 & 1 & (+1) \\ \hline 1 & 0 & 0 & 0 & 0 & (0) \end{array} \qquad \begin{array}{rrrrrr} & 0 & 1 & 0 & 0 & (+4) \\ + & 1 & 1 & 0 & 0 & (-4) \\ \hline 1 & 0 & 0 & 0 & 0 & (0) \end{array} \qquad \begin{array}{rrrrrr} & 0 & 1 & 1 & 1 & (+7) \\ + & 1 & 1 & 1 & 0 & (-2) \\ \hline 1 & 0 & 1 & 0 & 1 & (+5) \end{array}$$

- Se os operandos teñen signos diferentes o resultado da suma sempre é correcto.
- Se os operandos teñen o mesmo signo pode existir desbordamento ou *overflow* (o resultado é demasiado grande e sería preciso un bit máis para ser codificado). **Hai desbordamento cando o signo do resultado non coincide co signo dos operandos**.

Exemplos: $1101 + 1001 = \textit{overflow}$ (0110), $0100 + 0100 = \textit{overflow}$ (1000).

$$\begin{array}{rrrrrll} & 1 & 1 & 0 & 1 & (-3) & \\ + & 1 & 0 & 0 & 1 & (-7) & \\ \hline & 0 & 1 & 1 & 0 & (+6) & \Rightarrow \textit{overflow} \end{array} \qquad \begin{array}{rrrrrll} & 0 & 1 & 0 & 0 & (+4) & \\ + & 0 & 1 & 0 & 0 & (+4) & \\ \hline & 1 & 0 & 0 & 0 & (-8) & \Rightarrow \textit{overflow} \end{array}$$

Resta ($X - Y$): basta cambiar o subtraendo de signo e sumar, é dicir, $X + (-Y)$

Extensión do formato: para aumentar o número de bits utilizados para representar un número basta engadir máis bits á esquerda **repetindo o bit de signo**. Dese modo o valor do número permanece inalterado.

Exemplos (extensión de 4 a 6 bits):

$1111 = 111111$ (-1), $0110 = 000110$ $(+6)$, $1001 = 111001$ (-7).