VNiVERSiDAD D SALAMANCA

Técnicas Informáticas en Física
Grado en Física – Primer curso

Parte I: Introducción

Representación y almacenamiento de la Información

1. Introducción
2. Códigos numéricos
3. **Representación de datos numéricos**
   - **3.1 Punto fijo**
   - 3.2 Coma flotante

Dpto. de Informática y Automática
Facultad de Ciencias

3. Representación de datos numéricos

- Los números enteros, racionales o reales forman conjuntos **infinitos pero su representación** de forma digital **es finita**
- Se codifican mediante **cadenas binarias** de una cierta longitud finita *n* y normalmente en notación posicional
- La representación finita supone una aproximación que puede producir **errores** (desbordamiento, precisión…) que hay que tener en cuenta
- Existen diferentes métodos de representación, de los que destacan:

  Punto fijo: utilizan una cantidad fija de cifras para la **parte entera** y otra fija para la **parte fraccionaria.** Es la forma habitual para representar datos enteros

  Punto flotante (o **coma flotante**): similar a la notación científica, se basa en almacenar tres campos: **signo**, **mantisa** y **exponente**

3.1. Representación de datos enteros

- Los números **naturales** (enteros sin signo) se pueden representar directamente *en codificación binaria*
- Para representar **enteros** (positivos y negativos) pueden usarse **diferentes criterios:**
  - *Signo-magnitud*
  - *Complemento a 1*
  - *Complemento a 2*
  - *Representación sesgada*
- Se utilizan cadenas de **longitud** fija: 8, 16, 32 o 64 bits
- El **rango** (cantidades máxima y mínima representable) depende del número de bits
- Su **resolución** (mínima distancia entre dos números) es 1

3.1. Representación de datos enteros

Ejemplo con n=3

BINARIA	SIN SIGNO	SIGNO-MAGNITUD	C-1	C-2	SESGADA $S=2^{n-1}-1=3$
000	**0**	**0**	**0**	**0**	**-3**
001	**1**	**1**	**1**	**1**	**-2**
010	**2**	**2**	**2**	**2**	**-1**
011	**3**	**3**	**3**	**3**	**0**
100	**4**		**-3**	**-4**	**1**
101	**5**	**-1**	**-2**	**-3**	**2**
110	**6**	**-2**	**-1**	**-2**	**3**
111	**7**	**-3**		**-1**	**4**

26

3.1.1. Enteros *sin signo*

Consultar en Matlab
>> doc uint32
(*unsigned integer*)

- Se representan con la codificación en ***binario puro***

$$X = \sum_{i=0}^{\infty} x_i \cdot 2^i \text{ con } x_i = 0 \text{ ó } 1$$

- Usando cadenas de longitud ***n*** se pueden representar los números naturales del **rango** *[0, 2^n-1]*

  Ej. *Con n=8 el rango es [0 , 255] y con n=32 es [0 , 4294967295]*
- Para **extender** la representación de un número a una palabra de mayor número de bits basta con añadir ceros a la izquierda
- Las **operaciones aritméticas** son sencillas en este método, pero si los resultados caen fuera del rango de representación se produce error por desbordamiento

  Ej. *Con n=2 la suma 11+01 proporcionaría un resultado 00*

3.1.2. Enteros en *signo-magnitud*

- De los *n* bits disponibles se usa uno para representar el **signo** y los restantes para el **valor absoluto** o módulo del número en binario
- El **bit de signo** se coloca a la izquierda y es:
  - 0 si el número es positivo
  - 1 si el número es negativo

signo módulo

Ej. *Con n=8* **$36_{(10}$ → 0 0 1 0 0 1 0 0**

$-36_{(10}$ → 1 0 1 0 0 1 0 0

- El **rango** de representación que se consigue con *n* cifras es simétrico *[-2^{n-1}+1, 2^{n-1}-1]* (***Ej***. *Con n=8 el rango es [-127, 127])*

  Ej. *Con n=4 ¿el rango es...?*
- Además del desbordamiento otros **inconvenientes** son:
  - Presenta **ambigüedad** para representar el **0**
  - No es adecuado para realizar **operaciones** de suma y resta (ya que la operación depende de los signos y magnitudes de los operandos)
  - No es fácil **extender** una representación a una palabra de mayor longitud *n*

Representación de datos enteros

BINARIA	SIN SIGNO	SIGNO-MAGNITUD	C-1	C-2	SESGADA
000	0	0			
001	1	1			
010	2	2			
011	3	3			
100	4	¿?			
101	5	-1			
110	6	-2			
111	7	-3			

$3 + (-1) = 011 + 101 = 1\,000 =$ ¿2?
$3 + (-3) = 011 + 111 = 1\,010 =$ ¿0?
Suma en signo-magnitud

29

3.1.3. *Complemento a uno*

- El bit de la izquierda indica el **signo** y también es *1* para el signo –
- El **rango** de representación es simétrico $[-2^{n-1}+1, 2^{n-1}-1]$
- En los **números positivos**, los otros *n-1* bits representan el módulo en código binario (igual que en representación *signo-magnitud*)
- El **negativo** de un número se obtiene **complementando** todos sus dígitos (cambiando ceros por unos y viceversa), incluso el bit de signo.

  Ej. Con n=8 $36_{(10}$ → 0 0 1 0 0 1 0 0

  $-36_{(10}$ → 1 1 0 1 1 0 1 1

  binario que correspondería a **128+64+16+8+2+1 = 219**

- La complementación de un número X es (-X) = $2^n - 1 - X$

  En el caso ***n=8 el -36 es*** **256-1-36 = 219**

30

3.1.3. *Complemento a uno*

Las ventajas son:

- Las **sumas y restas** se realizan directamente en binario y, si se produce un acarreo superior (*posición n+1*), el resultado final deberá incrementarse en una unidad
- La **extensión** a palabras de longitud mayor es sencilla, se consigue repitiendo el bit de la izquierda

  Ej. **- $36_{(10}$** es *1101 1011* en 8 bits y en 12 será *1111 1101 1011*

También presenta algunos inconvenientes:

- El **cero** admite dos representaciones: *000...00* o bien *111...1*
- Hay que tener en cuenta el efecto del **desbordamiento** al operar cuando el resultado excede el rango

  Ej. **127 + 2** *= 0111 1111 + 0000 0010 = 1000 0010* **= -126 ?**

31

Representación de datos enteros

BINARIA	SIN SIGNO	SIGNO-MAGNITUD	C-1	C-2	SESGADA
000	0	0	**0**	0	
001	1	1	**1**		
010	2	2	**2**		
011	3	3	**3**		
100	4		**-3**		
101	5	-1	**-2**		
110	6	-2	**-1**		
111	7				

3 + (-1) *= 011+110 = 001+* $\mathbf{1}_{ac}$ *= 010 =* **2**
3 + (-3) *= 011+100 = 111 =* **0**
Suma en C-1

3.1.4. *Complemento a dos*

- Se utiliza el mismo criterio que en complemento a uno para el bit de **signo** y para los números **positivos**
- El **rango** es $[-2^{n-1}, 2^{n-1}-1]$
- La representación del número **negativo** correspondiente a una cantidad X es (-X) = $\mathbf{2^n - X}$
- Los números **negativos** también se pueden calcular obteniendo primero el complemento a uno del positivo e incrementando después el resultado en una unidad, es decir, ***C2 = C1 + 1***

Ejemplo *Representar el número –25 con 8 bits usando s-m, C1 y C2*

El número positivo es **25** = *0 0 0 1 1 0 0 1* (en cualquier repr.)
En signo-magnitud es **- 25** =
En C1 será **- 25** = ???
En C2 será **- 25** =

33

3.1.4. *Complemento a dos*

- La ventaja de este método es que **simplifica las operaciones** de suma y resta con mezcla de números positivos y negativos
- En este caso hay que **ignorar** los bits de acarreo superior
- De nuevo si hay desbordamiento y el valor absoluto del resultado es mayor o igual que 2^{n-1} la interpretación del binario resultante no es correcta

Ejemplo *Suma en C2 los números 53 y -25 (para n=8)*

53 = *0 0 1 1 0 1 0 1* (***53*** = 32+16+4+1)
-25 = + *1 1 1 0 0 1 1 1* (128+64+32+4+2+1 = ***231*** = ***256-25***)
28 *1 0 0 0 1 1 1 0 0* (16+8+4 = ***28***)

El acarreo se desprecia

34

Representación de datos enteros

BINARIA	SIN SIGNO	SIGNO-MAGNITUD	C-1	C-2	SESGADA
000	0	0	0	**0**	
001	1	1	1	**1**	
010				**2**	
011				**3**	
100	4		-3	**-4**	
101	5	-1	-2	**-3**	
110	6	-2	-1	**-2**	
111	7	-3		**-1**	

$3 + (-3) = 011 + 101 = 000 = 0$
$3 + (-1) = 011 + 111 = 010 = 2$
Suma en C-2

3.1.5. Representación *sesgada*

- Una cantidad *X* se representa con el binario que resulta al codificar el número $N = X + S$, siendo *S* un **sesgo** que suele valer $S=2^{n-1}$ *o* $S=2^{n-1}-1$
- La cadena *000…000* corresponde a la cantidad menor y la *111..111* a la mayor. El valor **0** ocupará la posición central del rango
- El **rango** de representación es $[-2^{n-1}+1, 2^{n-1}]$ *o bien* $[-2^{n-1}, 2^{n-1}-1]$
- **Ejemplo** *Con n=8 y suponiendo un sesgo* $S= 2^7 -1 = 127$ *sería:*

  cantidad $X = \mathbf{0}_{(10}$ → $N = 127 = 0111\ 1111$

  cantidad $X=\mathbf{36}_{(10}$ → $N = 36 + 127 = 163 = 1010\ 0011$

  cantidad $X = \mathbf{-36}_{(10}$ → $N = -36 + 127 = 91 = 0101\ 1011$

- En este caso los binarios de los enteros negativos empiezan por 0 y los de los positivos empiezan por 1
- Al quedar los números correlativos las **operaciones** son sencillas

Representación de datos enteros

BINARIA	SIN SIGNO	SIGNO-MAGNITUD	C-1	C-2	SESGADA $S=2^{n-1}-1$
000	0	0	0	0	**-3**
001	1	1	1	1	**-2**
010	2	2	2	2	**-1**
011	3	3	3	3	**0**
100	4		-3	-4	**1**
101	5	-1	-2	-3	**2**
110	6	-2	-1	-2	**3**
111	7	-3		-1	**4**

Exceso a 3

Representación de datos enteros

- https://es.wikipedia.org/wiki/Representación_de_números_con_signo

Representación de enteros de 4 bits

Decimal	Entero sin signo	Signo y Magnitud	Complemento a uno	Complemento a dos	En exceso a 7
+8	1000	n/d	n/d	n/d	1111
+7	0111	0111	0111	0111	1110
+6	0110	0110	0110	0110	1101
+5	0101	0101	0101	0101	1100
+4	0100	0100	0100	0100	1011
+3	0011	0011	0011	0011	1010
+2	0010	0010	0010	0010	1001
+1	0001	0001	0001	0001	1000
+0	0000	0000	0000	0000	0111
-0	n/d	1000	1111	n/d	n/d
-1	n/d	1001	1110	1111	0110
-2	n/d	1010	1101	1110	0101
-3	n/d	1011	1100	1101	0100
-4	n/d	1100	1011	1100	0011
-5	n/d	1101	1010	1011	0010
-6	n/d	1110	1001	1010	0001
-7	n/d	1111	1000	1001	0000
-8	n/d	n/d	n/d	1000	n/d

3.1.6. Representación *BCD (Binary Coded Decimal)*

- La representación en ***Decimal Codificado Binario (BDC)*** usa codificación por campos, es decir, **cada cifra** del número se codifica como **un campo** de longitud fija (como mínimo n=4 bits ya que debe ser $2^n>10$)
- En *BCD con signo* se usa un campo más para el signo (en números positivos es 0000, y en negativos 0001)

  Ej. $\mathbf{9\ 1\ 7\ 5)_{10}} = 1001\ 0001\ 0111\ 0101)_{BCD} =$

  $= 10\ 0011\ 1101\ 0111)_{\text{binario puro}}$
- La conversión es sencilla pero requiere más espacio y los cálculos son más complicados que en las demás representaciones

Cifra	BCD
0	0000
1	0001
2	0010
3	0011
4	0100
5	0101
6	0110
7	0111
8	1000
9	1001

3.1.7 Ejercicios

1. Interpretar las cadenas binarias *111001* y *0011* considerando que representan:
- enteros sin signo
- enteros en signo/magnitud
- enteros en complemento a uno (C1)

2. Escribir los números decimales 6.1 y –12.140625 en punto fijo (signo/magnitud) utilizando 7 bits para la parte entera y 10 para la parte fraccionaria

3. Pensar cuáles serán las cantidades máxima y mínima y la resolución (distancia mínima entre dos cantidades) que se consigue usando una representación en punto fijo con 7 bits para parte entera y 10 para parte fraccionaria

4. Representar el número entero 111_{10} en hexadecimal, octal y binario sin signo y el número -111_{10} en signo/magnitud y en C1 con 9 bits

5. ¿Cuál será el mínimo número de bits necesarios para representar el entero -50? ¿Y para el entero 2030?

6. Comprobar en MATLAB el rango de los enteros de 32 bits **`[intmin, intmax]`**

Documentation

CONTENTS

Integers

Enteros en Matlab

Here are the eight integer classes, the range of values you can store with each type, and the MATLAB conversion function required to create that type:

Class	Range of Values	Conversion Function
Signed 8-bit integer	-2^7 to 2^7-1	`int8`
Signed 16-bit integer	-2^{15} to $2^{15}-1$	`int16`
Signed 32-bit integer	-2^{31} to $2^{31}-1$	`int32`
Signed 64-bit integer	-2^{63} to $2^{63}-1$	`int64`
Unsigned 8-bit integer	0 to 2^8-1	`uint8`
Unsigned 16-bit integer	0 to $2^{16}-1$	`uint16`
Unsigned 32-bit integer	0 to $2^{32}-1$	`uint32`
Unsigned 64-bit integer	0 to $2^{64}-1$	`uint64`

Creating Integer Data

MATLAB stores numeric data as double-precision floating point (`double`) by default. To store data as an integer, you need to convert from `double` to the desired integer type. Use one of the conversion functions shown in the table above.

VNiVERSiDAD D SALAMANCA

Técnicas Informáticas en Física
Grado en Física – Primer curso

Parte I: Introducción

Representación y almacenamiento de la Información

Dpto. de Informática y Automática
Facultad de Ciencias