Representación de la información: Introducción a las representaciones numéricas

Escuela Técnica Superior de Ingeniería Informática
Universidad de La Laguna

Febrero, 2016

Esquema de la lección

1. Introducción
2. Representación de números
   - La notación posicional
   - Representación binaria, octal y hexadecimal.
   - Cambio de representación entre bases
3. Números faccionarios y representación posicional
4. Formatos de representación en el computador
   - Características de los formatos
   - Representación de números enteros
5. Operaciones aritméticas básicas con enteros
   - Introducción
   - La suma binaria
   - La suma binaria y la interpretación binario natural
   - La suma binaria y la suma de enteros con signo

Introducción

- **¿Qué es la información?:** Idea intuitiva: se asocia al proceso de conocer mejor el estado del "entorno", a reducir la incertidumbre sobre él.
- **¿Cómo se mide?:** El **bit** es la unidad básica de información. Representa una variable que sólo puede tomar uno de dos valores posibles.
- Supongamos que deseo reducir mi desconocimiento del universo, pero quiero la menor reducción posible. ¿Qué debo hacer?: localizar un bit y averiguar su valor.

Introducción (II)

- La tecnología de los computadores utiliza sistemas con dos estados posibles como elementos básicos para representar la información.
- Por esto, la representación de la información en las computadoras se basa en el bit. Este tipo de representación se denomina **binaria**.

Introducción (III)

- Observen que varios bits juntos pueden servirnos para representar variables que pueden tomar más de dos valores posibles: **n bits, logran 2^n estados posibles.**
- Una forma muy habitual de hacerlo es en grupos de 8. Esto genera una variable que puede estar en 1 de $2^8 = 256$ estados posibles. Esta agrupación se denomina **byte**.
- Abreviatura para bit: **b**
- Abreviatura para byte: **B**

Introducción (IV)

- **Nomenclatura de múltiplos en el Sistema Internacional (SI):** 10^3 es Kilo, 10^6 es Mega, 10^9 es Giga, 10^{12} es Tera, 10^{15} es Peta, 10^{18} es Exa, 10^{21} es Zetta y 10^{24} es Yotta.
- **Nomenclatura de múltiplos en el sistema binario:** 2^{10} es Kibi, 2^{20} es Mebi, 2^{30} es Gibi, 2^{40} es Tebi, 2^{50} es Pebi, 2^{60} es Exbi, 2^{70} es Zebi y 2^{80} es Yobi.
- Aunque esto no es exactamente correcto, es habitual usar la nomenclatura de múltiplos del SI incluso en el sistema binario, pero no olvidemos que incluo así, los múltiplos van con el factor $2^{10} = 1024$ y no con el factor $10^3 = 1000$.

Introducción (V)

- Así, cuando en ciencias de la computación o en telecomunicaciones decimos 1 Megabyte, en realidad nos referimos a 1 Mebibyte o 2^{20} bytes (que no es igual a 10^6 bytes).
- Para hacernos una idea: capacidades de almacenamiento típicas: en disco duro, de 250GB a varios TB (ordenador personal), CD 650 MB, DVD 4,7 GB, memoria principal: varios GB.
- Total de información del mundo según cálculos recientes: 295 exabytes.
- Información enviada en 2007 en sistemas de comunicación de dos vías: 65 exabytes, y por sistemas de broadcast 1,9 zettabytes.

Representación de los números

- **Principio de valor relativo o notación posicional:** Los números se representan en dígitos sucesivos, teniendo cada dígito un peso o valor que depende del lugar donde se encuentra.
- En un sistema de representación posicional existe un valor denominado **base**, que sirve para generar los pesos que multiplican el dígito. El valor del peso dependerá por tanto de la base y de la posición donde se desea aplicar.

Representación decimal de los enteros sin signo (I)

- En la representación decimal, la base es 10 y los dígitos se corresponden con los símbolos del 0 al 9.
- Así, el valor del número se obtiene a partir de la suma de los valores $d_i \times 10^i$ donde d_i es el dígito que ocupa la posición i-ésima, contando de derecha a izquierda, siendo $i = 0$ el dígito situado más a la derecha:

$$y = \sum_{i=0}^{n-1} d_i \times 10^i$$

Representación decimal de los enteros sin signo (II)

- La fórmula anterior es equivalente a obtener la posición del número en una lista que comenzaría por los 10 dígitos:

$$0, 1, 2, 3, 4, 5, 6, 7, 8, 9$$

para continua añadiendo un 1 delante:

$$10, 11, 12, 13, 14, 15, 16, 17, 18, 19$$

ahora un 2:

$$20, 21, 22, 23, 24, 25, 26, 27, 28, 29$$

etcétera.

Bases usadas en el campo de la informática

- **Base 2 o binaria:** Las cifras son 0 y 1. El convenio por el cuál interpretamos una cadena de bits como un número positivo usando la representación posicional se denomina **binario natural**.
- **Base 8 u octal:** Las cifras son $0, 1, 2, 3, 4, 5, 6, 7$.
- **Base 16 o hexadecimal:** Las cifras son 0, 1, 2, 3, 4, 5, 6, 7, 8, 9, A, B, C, D, E y F.
- La base octal y hexadecimal tienen la virtud de obtener representaciones compactas fácilmente traducibles a la base binaria.

Recuerda

- El principio de valor relativo, dice que una cifra colocada a la izquierda de otra, representa unidades del orden inmediatamente superior.
- La diferencia entre un orden y el siguiente viene dada por un factor multiplicativo denominado **base**
- Las bases más usadas en el campo de los computadores son: la binaria (2), la octal 8 y la hexadecimal 16.
- Para distinguir un número representado en un sistema, se denota la base en el subíndice del número. Ejemplos:
  - $(010110)_2$ (representación binaria)
  - $(054172)_8$ (representación octal)
  - $(AE0F34B)_{16}$ (representación hexadecimal). Una notación alternativa mucho más utilizada es: $0xAE0F34B$.

El método de los residuos (I)

- Existen métodos para obtener la representación de un valor entero en una base b en la representación correspondiente en otra base p.
- **Método de los residuos:** si se divide un número entero expresado en un sistema de base b, por la base p, y el cociente se vuelve a divir por p y así sucesivamente hasta que el cociente sea inferior a la base, el último cociente y los restos obtenidos, tomados en orden inverso forman el número en el sistema de base p.

El método de los residuos (II)

Veamos por qué funciona el método de los residuos en base 2 (binario natural):

- $D = c \times d + r$ es la fórmula dividendo (D) igual cociente (c) por divisor (d) más el resto r.
- Si el divisor es 2, la primera división resulta en: $D = c \times 2 + b_0$.
- Si c es mayor que 2, volvemos a hacer la división: $D = (b_2 \times 2 + b_1) \times 2 + b_0$. Supongamos que b_2 es menor que 2.
- La última ecuación podemos reescribirla como: $D = b_2 \times 2^2 + b_1 \times 2^1 + b_0 \times 2^0$, que es precisamente la manera en la que obtenemos el valor del entero sin signo en binario natural.

Cambio de representación de binario natural a octal o hexadecimal

- **De binario natural a octal:** Los bits se agrupan de tres en tres de **derecha a izquierda**, y se sustituye cada grupo por la cifra octal que le corresponde. **Ejemplo:** Sea $(1010011001)_2$. Agrupamos de tres en tres de derecha a izquierda: (1)(010)(011)(001). Traducimos cada grupo a la cifra octal que le corresponde, interpretando cada grupo en binario natural: $(1231)_8$.
- **De binario natural a hexadecimal:** Los bits se agrupan de cuatro en cuatro de **derecha a izquierda**, y se sustituye cada grupo por la cifra hexadecimal que le corresponde. **Ejemplo:** Sea $(1010011001)_2$. Agrupamos de cutro en cuatro: (10)(1001)(1001). Traducimos cada grupo a la cifra hexadecimal que le corresponde: 0×255.

Cambio de representación de octal o hexadecimal a binario natural

- **De octal o hexadecimal a binario:** Traducimos cada cifra al código binario natural que le corresponde, usando 3 bits por cifra en octal y 4 bits por celda en hexadecimal. **Ejemplo:** Sea $0 \times AF03BC521$. Usando cuatro bits por cifra:

$$((1010)(1111)(0000)(0011)(1011)(1100)(0101)(0010)(0001))_2$$

dado que $(A)_{16} = (10)_{10} = (1010)_2$,
$(F)_{16} = (15)_{10} = (1111)_2$, etcétera.

Números fraccionarios: de representación decimal a binario natural (I)

- Sea F una fracción expresada en la base 10, con un número finito de cifras decimales. El objetivo que se pretende es convertirla a una base 2. Estos son los pasos a realizar:
  - Se separa la parte entera y la parte no entera.
  - La parte entera se convierte usando los métodos anteriormente expuestos.
  - La parte no entera se convierte multiplicando por 2. La parte entera del resultado será la nueva cifra. Con la parte no entera, se vuelve a realizar la operación descrita. El procedimiento se itera hasta obtener 0 en la parte no entera, hasta que se llegue al máximo número de cifras establecido, o hasta que la secuencia de cifras obtenidas empiece a repetirse (ver problemas).

Conversión de números fraccionarios de decimal a binario natural (II)

Ejemplo:

Sea el número $(10{,}23)_{10}$. Vamos a convertirlo a binario natural

- Convertimos la parte entera $(10)_{10} = (1010)_2$
- Convertimos la parte fraccionaria 0,23. Multiplicamos por la nueva base: $0{,}23 \times 2 = 0{,}46$. El bit resultante es 0 y la parte no entera es 0,46. Volvemos a multplicar: $0{,}46 \times 2 = 0{,}96$. El bit resultantes es 0 y la parte no entera es 0,96. Multiplicamos: $0{,}96 \times 2 = 1{,}92$. El bit e 1 y laparte no entera es 0,92. Multiplicamos: $0{,}92 \times 2 = 1{,}84$. Bit igual a 1 y parte no entera 0,84. Podría continuarse iterando pero paramos aquí.
- El resultado aproximado es por tanto: $(10{,}23)_{10} \approx (1010{,}0011)$.

Conversión de números fraccionarios de decimal a binario natural (III)

- Un número fraccionario en base decimal con un número finito de cifras decimales, puede requerir un número binario fraccionario con infinitos bits fraccionarios.
- Como al capacidad de almacenamiento de un ordenador no es infinita debemos truncar en algún punto.
- Este es el origen del error de precisión en la representación de números fraccionarios.

Caracerísticas de los formatos de representación en un computador

- **Dimensión:** es el número de bits utilizados para representar los números. Ejemplos: byte, palabra (uno o varios bytes) y doble palabra.
- **Palabra:** normalmente está asociada a la arquitectura del computador y es el número de bits de un dato que puede procesarse "de una vez" dadas las características de la máquina.
- **Convenios de representación:** junto con la dimensión o tamaño de la palabra, existen una serie de características que deben fijarse para poder comprender una representación.

Convenios de la representación

- Representación del signo (diferenciar números positivos y negativos).
- Orden de las partes de un número, si éste ocupa más de una palabra.
- Representación de los números reales. Existen dos tipos:
  - Formatos de punto fijo. En ste formato se establece un número de cifras que es fijo para representar la parte no entera.
  - Formatos de punto flotante. Se utiliza una técnica que equivale a considerar un número de cifras variable en la parte no entera que depende del número.

Formatos de representación de números enteros

- **Enteros sin signo**. El código binario se interpreta usando el convenio de la representación posicional sin signo (binario natural).
- **Enteros con signo**. Aquí tenemos tres posibilidades en cuanto a la interpretación del código:
  - Binario natural y un bit para el signo (también llamada representación módulo y signo).
  - Complemento a la base menos uno (también llamado complemento a uno).
  - Complemento a la base (también llamado complemento a dos).
- Lo tres sistemas anteriores representan los enteros positivos del mismo modo, pero de diferente manera los enteros negativos.
- El método de complemento a dos, como veremos, es el más eficiente y por tanto el más utilizado.

Representación módulo signo

- **Características del formato:**
  - El bit más a la izquierda (más sigificativo) es el bit de signo.
  - El resto de bits se interpreta en binario natural.
  - Ejemplo: -10 en una celda de 5 bits: $\rightarrow (11010)_2$
  - Ejemplo: $+10$ en una celda de 5 bits: $\rightarrow (01010)_2$
- **Rango:** Sea una celda de n bits, el rango es:
  - $\{-(2^{n-1}-1), \ldots, (2^{n-1}-1)\}$. Ejemplo: para $n = 4$, el rango es $\{-7, \ldots, +7\}$
  - Observen que el número de cantidades diferentes que se representan con n bits es $2^n - 1$, en lugar de 2^n. La razón es que hay dos códigos asociados al número 0, son: $000\ldots000$ y $100\ldots000$.

Representación de enteros con signo en complemento a 1 (C1) (I)

- En una celda de n bits, los números positivos se representan en C1 escribiendo el binario natural en el rango entre 0 y $2^{n-1} - 1$.
- Dado el número $k > 0$, el opuesto $(-k)$, se obtiene cambiando en la representación C1 de k, los $1s$ por $0s$ y los $0s$ por $1s$. A esta operación binaria sobre un código binario, se denomina **NOT**
- Ejemplo: $+10$ en una celda de 5 bits: $\rightarrow 01010$.
- Ejemplo: -10 en una celda de 5 bits: $\rightarrow 10101$.
- Observen que $NOT(01010) = 10101$.

Representación de enteros con signo en complemento a 1 (C1) (II)

- Observa, que al igual que en la representacion módulo signo, los enteros positivos tienen su bit más significativo a 0 y los enteros negativos a 1.
- La operación *NOT*, equivale a restar de $2^n - 1$, el entero positivo que se desea pasar a negativo, por ejemplo:
  - Queremos obtener -3 en una celda de 4 bits. En este caso debemos restar 3 de $2^n - 1 = 15$. Es decir $15 - 3 = 12$.
  - Escribimos 12 en binario natural: 1100, que es -3 en C1 para una celda de 4 bits.
- El rango de C1 para una celda de n bits es: $\{-(2^{n-1} - 1), \ldots, (2^{n-1} - 1)\}$. Como vemos coincide con el de módulo signo, ya que el número cero también está representado por dos códigos, en C1 son $000\ldots0$ y $111\ldots1$.

Representación de enteros con signo en complemento a 2 (C2) (I)

- En una celda de n bits, los números positivos se representan en C2 escribiendo el binario natural en el rango entre 0 y $2^{n-1} - 1$.
- Dado el número $k > 0$, el código del opuesto $(-k)$ se obtiene escribiendo en binario natural en una celda de n bits, el resultado de la resta $2^n - k$.
- Ejemplo:
  - $+3$ en un celda de 4 bits sería: 0011
  - Si queremos obtener -3, escribimos en binario natural $2^n - 3 = 13$, que es 1101. Esta es la representación C2 de -3.

Representación de enteros con signo en complemento a 2 (C2) (II)

- El rango es $\{-2^{n-1}, \ldots, 2^{n-1} - 1\}$. Como vemos, tenemos un código más porque el 0 sólo tiene una representación.
- Dado $k > 0$, la operación $2^n - k$, equivale a hacer NOT sobre la representación C2 de k (como en C1) y a continuación hacer la **suma binaria** del resultado con 1.
- Esta última operación binaria se denomina NEG y equivale a copiar los bits de derecha a izquierda hasta encontrar el primer 1, entonces, a partir del siguiente bit vamos haciendo NOT de cada bit hasta el final. Por ejemplo: NEG(011000100) = ...100 (copiamos hasta el primer 1) = 100111100 (Completamos el resto de bits haciendo NOT sobre cada uno)

Suma y resta de enteros con signo

- La suma y la resta son las operaciones básicas, que junto a otras operaciones de manipulación de bits, dan lugar a algoritmos para operaciones más complejas como la multiplicación y la división.
- Observen que la resta equivale a la suma con el sustraendo cambiado de signo, por lo que la operación básica es la suma.
- Dados dos códigos binarios representando dos enteros con signo, la elección de un procedimiento de suma depende de la interpretación que se desee usar (módulo signo, complemento a 1, complemento a 2).
- En este curso nos centraremos en un operador binario útil para la suma en ciertas condiciones denominado “suma binaria”

La suma binaria (I)

- Se trata de una mera operación de manipulación de bits. Por sí misma no se puede asimilar a la suma convencional, ya que esto va poder hacerse sólo para ciertas interpretaciones de los códigos binarios y bajo condiciones particulares.
- La suma binaria de 1 bit puede entenderse como una operación con tres operandos de tipo bit y dos resultados de tipo bit.
- Los tres operandos serían: sumando 1, sumando 2 y acarreo. Las salidas serían resultado y nuevo acarreo.

La suma binaria (II)

- Las reglas de adición binaria se resumen en la siguiente tabla:

Op1	Op2	Acarreo	Resultado	Nuevo Acarreo
0	0	0	0	0
0	1	0	1	0
1	0	0	1	0
1	1	0	0	1
0	0	1	1	0
0	1	1	0	1
1	0	1	0	1
1	1	1	1	1

La suma binaria (II)

- Las tabla anterior se puede resumir en esta otra:

Número de 1s en los tres operandos	Resultado	Nuevo Acarreo
0	0	0
1	1	0
2	0	1
3	1	1

- En otros sitios verán la suma binaria de 1 bit definida con dos operandos. Esta es equivalente a la que acabamos de explicar con el operando acarreo igual a 0.

La suma binaria (III)

- La suma binaria de varios bits se realiza aplicando de forma sucesiva sumas binarias de 1 bit y tomando como acarreo el nuevo acarreo generado por la suma binaria anterior. Veamos un ejemplo:

Ac:	1	0	0	0	0	0
	0	1	1	0	1	0
ADD	0	1	0	0	0	1
C=0	1	0	1	0	1	1

- La fila inferior es el resultado, las dos filas siguientes son los operandos y la fila superior es el acarreo de la suma binaria de 1 bit previa. Observen que empezamos por la columna de la derecha con un acarreo inicial de 0. En el resultado se especifica que el último acarreo es 0.

La suma binaria y la interpretación binario natural (I)

- En binario natural, todos los números son positivos, por lo tanto, la suma sólo contempla que ambos operandos sean enteros positivos.
- La suma de dos números de n bits, puede dar lugar un número de $n+1$ bits. Esto ocurre cuando el resultado supera el rango de la representación de binario natural.
- En binario natural la operación suma binaria puede asimilarse a la operación suma convencional si el resultado se mantiene en el rango. En el caso de que se supere, el resultado correcto implica añadir el bit del último acarreo al resultado.

Ejemplos de sumas binarias bajo la interpretación de binario natural (I)

- Queremos sumar 22+34 usando la suma binaria en una celda de 6 bits con la interpretación de binario natural. El rango para 6 bits es $\{0, \ldots, 63\}$. El resultado está en el rango:

Ac:	0	0	1	1	0	0
	0	1	0	1	1	0
ADD	1	0	0	0	1	0
C=0	1	1	1	0	0	0

- Resulta que el resultado $(111000)_2 = 56 = 22 + 34$.

Ejemplos de sumas binarias bajo la interpretación de binario natural (II)

- Sin embargo, si sumamos 42+26 el resultado no está en el rango. Es necesario incluir el acarreo final como bit más significativo del resultado para que éste sea correcto.

Ac:	1	1	0	1	0	0
	1	0	1	0	1	0
ADD	0	1	1	0	1	0
C=1	0	0	0	1	0	0

- En este caso, resulta que $(000100)_2 = 4 \neq 42 + 26$, pero si añadimos el último acarreo como bit más significativo: $(1000100)_2 = 68 = 42 + 26$.

La suma de enteros con signo

- Vamos a observar aquí una propiedad fundamental de las representaciones de enteros con signo descritas anteriormente.
- La aplición de la suma binaria da resultados correctos bajo una representación en complemento a 2 siempre que los operandos y los resultados respeten el rango.
- En las representaciones de módulo signo y complemento a 1 esto no es así en general.

Ejemplo de suma binaria asumiendo una representación signo - magnitud

Tomemos por ejemplo la suma de $+5 + (-5)$, una celda de 6 bits y la representación módulo-signo:

Ac:	0	0	1	0	1	0
	0	0	0	1	0	1
ADD	1	0	0	1	0	1
C=0	1	0	1	0	1	0

EL RESULTADO NO ES CORRECTO EN LA INTERPRETACIÓN SIGNO - MAGNITUD, DEBERÍA HABER SIDO CERO.

Ejemplo de suma binaria asumiendo una representación C1 (I)

Tomemos el mismo ejemplo de antes:

Ac:	0	0	0	0	0	0
	0	0	0	1	0	1
ADD	1	1	1	0	1	0
C=0	1	1	1	1	1	1

EL RESULTADO ES CORRECTO EN LA INTERPRETACIÓN C1, $5 + (-5) = 0$

Ejemplo de suma binaria asumiendo una representación C1 (II)

Sin embargo, hagamos ahora -5 + -5

Ac:	1	1	0	1	0	0
	1	1	1	0	1	0
ADD	1	1	1	0	1	0
C=1	1	1	0	1	0	0

EL RESULTADO ES EN LA INTERPRETACIÓN C1, -11 NO ES CORRECTO. ES NECESARIO UN PROCEDIMIENTO DE CORRECCIÓN DEL RESULTADO PARA COMPLEMENTO A 1.

Ejemplo de suma binaria asumiendo una representación C2 (I)

Tomemos el primer ejemplo: +5 + (-5)

Ac:	1	1	1	1	1	0
	0	0	0	1	0	1
ADD	1	1	1	0	1	1
C=1	0	0	0	0	0	0

EL RESULTADO ES CORRECTO EN LA INTERPRETACIÓN C2, PERO OJO CON LA INTERPRETACIÓN DEL CARRY (DEBE DESCARTARSE EN EL RESULTADO)

Ejemplo de suma binaria asumiendo una representación C2 (II)

Tomemos el segundo ejemplo: (-5) + (-5)

Ac:	1	1	0	1	1	0
	1	1	1	0	1	1
ADD	1	1	1	0	1	1
C=1	1	1	0	1	1	0

EL RESULTADO ES CORRECTO EN LA INTERPRETACIÓN C2, EL -10.

Aspectos importantes de la suma de enteros con signo

- A nivel de instrucción máquina, no se diferencia entre enteros con signo o sin signo.
- La suma se realiza siempre del mismo modo: se aplica la suma binaria.
- Si los número se interpretan como enteros sin signo (binario natural), **el carry C** representa la condición de "resultado fuera de rango" como hemos visto.
- Sin embargo, en el caso de que se interpreten los operandos como enteros con signo, esto no es así. La condición "fuera de rango" en enteros con signo, queda marcada por el bit de **"overflow" V**. Hoy en día, se determina el bit de "overflow" considerando una interpretación de C2 en la mayoría de procesadores.

Ejemplo:

- Celda de 4 bits y consideramos la suma de $(0110)_2$ y $(0100)_2$ ($6 + 4$, si se interpreta en binario natural).
- El bit de carry se pone a 0 porque no hay fuera de rango bajo la interpretación de enteros sin signo. El bit de overflow se pone a 1 porque sí se da la condición de fuera de rango bajo una interpretación C2.

Ac:	1	0	0	0
	0	1	1	0
ADD	0	1	0	0
C=0 V=1	1	0	1	0