1 Codificación da información

Sabemos que o obxectivo dun ordenador é manexar información e que estes equipos son sistemas dixitais. Destas dúas afirmacións se saca en conclusión que será necesario codificar a información nun formato binario axeitado para que os ordenadores a poidan manexar.

1.1 Sumario

1.2 Introdución á representación de números nos sistemas dixitais

Os PCs son sistemas dixitais polo que nos centraremos en como se representan os números neste sistema.
O primeiro é volver a recordar que, no sistema binario, con **n díxitos** podemos representar **2^n símbolos distintos** (se son números sería dende o 0 ata o número 2^n-1).

1.3 Representación binaria sen Signo

O único que hai que ter en conta nesta representación é que o sistema binario e un sistema de numeración ponderado ou posicional, polo que o valor dun díxito depende do símbolo empregado e da posición que ese símbolo ocupa no número.
Así, por exemplo, nun número binario de 8 díxitos, as posicións destes teñen o valor seguinte:

128	64	32	16	8	4	2	1

Como vemos, as cantidades represéntanse de esquerda a dereita como no sistema de representación decimal.
Por exemplo, o número 33 tería una representación decimal 100001. Que, normalmente se separa en grupos de 4 bits para mellorar a súa representación: 0010 0001.
Con este sistema todos os bits están dispoñibles para representar unha cantidade, por conseguinte, con 8 bits podemos representar o rango 0 <= n <= 255.

1.4 Representación binaria con Signo

Moitas veces é preciso traballar con números negativos nos PCs, veremos como se pode facer.

1.4.1 Signo-magnitude

Utilízase un bit para codificar o signo

- ◊ **O:** se o número é positivo.
- ◊ **1:** se o número é negativo.
- ◊ O resto dos bits codifícanse en valor absoluto.
- ◊ **Exemplos:**

+20 = 00010100
-20 = 10010100

1.4.2 Complemento a un

Para representar con **n bits** un número en complemento a un, o único que hai que facer é cambiar cada un dos díxitos do número binario polo seu complementario, isto é, cambiar os uns polos ceros e os ceros polos uns.

- ◊ **Exemplos:**

+86 = 01010110
-86 = 10101001

Como se ve, neste sistema de numeración, temos dúas formas de representar o número 0, polo que fará máis axeitada a utilización da representación en ?complemento a dous?.

+0 = 00000000
-0 = 11111111

1.4.3 Complemento a dous

O complemento a dous resolve o problema do dobre **0** que aparece no *complemento a un*.
Para representar con n bits un número A en complemento a dous faise do seguinte xeito:
- Se A >= 0 o complemento a dous será o número A en binario natural
- Se A < 0 o complemento a dous será igual a 2^n-|A|

O procedemento para codificar un número negativo en n bits será o seguinte:

1. Obter o valor absoluto de A
2. Obter a representación binaria de A en n bits
3. Inverter os bits da representación
4. Sumar 1 ao resultado

- ◊ **Exemplos:** Representar -5 en complemento a 2, utilizando 8 bits:

1. Valor absoluto de -5 = 5
2. Representación en binario de 5 = 00000101
3. Inversión dos bits: 11111010
4. Sumar 1 ao resultado: 11111011

1.5 Representación binaria de decimais

Con un sistema numérico posicional binario é posible representar números reais, vexamos de que xeitos:

1.5.1 Representación en coma fixa

Nesta representación a posición da coma é a que fixa a potencia da base pola que hai que multiplicar o díxito correspondente.
Con esta notación queda moi limitado o número de cantidades a representar e todas elas deben ter a mesma resolución.

- ◊ **Exemplo:**

Neste exemplo empregamos 8 bits e reservamos 5 para a parte enteira e 3 para a fraccionaria.
Con esta configuración poderemos representar números reais entre 0 e 32,875 con pasos, como moi pequenos, de 0,125 (2^{-3}).
Así:

$10101{,}110 = 1\cdot2^4 + 0\cdot2^3 + 1\cdot2^2 + 0\cdot2^1 + 1\cdot2^0 + 1\cdot2^{-1} + 1\cdot2^{-2} + 0\cdot2^{-3} = 21{,}75$
$01001{,}011 = 0\cdot2^4 + 1\cdot2^3 + 0\cdot2^2 + 0\cdot2^1 + 1\cdot2^0 + 0\cdot2^{-1} + 1\cdot2^{-2} + 1\cdot2^{-3} = 9{,}375$

1.5.2 Representación en coma flotante

Este método serve para representar números reais e permite adaptar a orde de magnitude do valor a representar, isto o consegue, usualmente, trasladando a coma decimal (mediante un exponente) cara a posición da primeira cifra significativa do valor.

Deste xeito, cun número dado de díxitos representativos, obtense maior precisión que coa representación en coma fixa, debido a que o valor destes díxitos é sempre significativo sexa o que sexa a orde de magnitude do número a representar. Debido a esta adaptación, consegue representar un rango moito maior de números (determinado polos valores límite que pode tomar o exponente).

Unha representación en coma flotante componse de tres números (campos) que seguen o seguinte patrón:

$$r = s \cdot m \cdot b^e$$

r: valor real do número a representar
s: signo, para números positivos s = 0, números negativos s = 1.
m: mantisa, díxitos significativos do número. O tamaño deste campo, usualmente fixo e limitado, determina a precisión da representación. Este campo está usualmente normalizado, é dicir, a súa parte enteira só consta dun díxito (que será a primeira cifra significativa do número a representar).
b: base do sistema de representación (10 no sistema decimal, 8 no sistema octal, 2 no sistema binario, etc)
e: exponente, orde de magnitude do significando. O mínimo e máximo valor posible do exponente determinan o rango de valores representables (cando **e** vale cero o valor real coincide coa mantisa).

◊ **O estándar IEEE 754** para aritmética en coma flotante é o estándar máis amplamente usado polas computadoras.
IEEE 754 especifica catro formatos para a representación de valores en punto flotante:

- Precisión simple (32 bits)
- Precisión dobre (64 bits)
- Precisión simple estendida (>= 43 bits, non usada normalmente)
- Precisión dobre estendida (>= 79 bits, usualmente con 80 bits)

◊ **Exemplo:**

Sexa o formato en coma flotante: (S/Exp/Mant) sendo S: 1 bit de signo; Exp: 5 bits de exponente en exceso 16, base 2; Mant: 6 bits de mantisa fraccionaria normalizada. Pídese representar o número -6,125:

Pasos:

1. signo = negativo o bit 12 toma o valor 1
2. pasar a binario a mantisa decimal: 6,125 = 110,001
3. Normalizar, isto é, correr a coma á dereita ou esquerda ata converter o número binario nun numero da forma 1,...
   Así 1,10001 polo que o exponente é 2, que expresado en exceso 16 da 18, en binario: 10010.
1. A mantisa se normaliza e se lle saca o primeiro 1
2. O número final é: **1 10010 100010** (pois hai que completar con 0 á dereita a mantisa).

1.6 Códigos Alfanuméricos

1.6.1 De 6 bits

- **BCDIC (*Binary Coded Decimal Interchange Code*)**
  É un dos primeiros códigos utilizados para representar datos en notación binaria para poder seren manexados por unha computadora. Esta técnica de codificación permite que un conxunto de caracteres alfanuméricos poida ser representado mediante 6 bits, aínda que é máis habitual a versión de 7 bits.

- **FIELDATA**
  É un código utilizado en transmisións de datos de algúns sistemas militares e está orientado á linguaxe máquina.

1.6.2 De 7 bits

- **ASCII (*American Standar Code for Information Interchange*)**
  O uso primordial é facilitar o intercambio de información entre sistemas de procesamento de datos e equipos asociados e dentro de sistemas de comunicación de datos.
  Nun principio cada carácter se codificaba mediante 7 díxitos binarios e foi creado para o xogo de caracteres ingleses máis correntes, polo que non tiña en conta nin caracteres especiais nin caracteres específicos de outras linguas. Isto fixo que que posteriormente se ampliara a 8 díxitos binarios (ASCII Extendido).

1.6.3 De 8 bits

- **ASCII Extendido**
  Como xa se dixo antes é unha ampliación do ASCII, pois neste cada carácter se codificaba mediante 7 díxitos binarios e foi creado para o xogo de caracteres ingleses máis correntes, polo que non tiña en conta nin caracteres especiais nin caracteres específicos doutras linguas. Isto fixo que posteriormente se ampliara a 8 díxitos binarios.

- **EBCDIC (*Extended Binary Coded Decimal Interchange Code*)**
  Este código aparece como unha ampliación do código BCDIC. Nas transmisións de datos é necesario utilizar un grande número de caracteres de control para a manipulación das mensaxes e realización doutras funcións. De aí que o código BCDIC se estendera a unha representación utilizando 8 bits dando orixe ao código EBCDIC.

- **ISO 8859-1**
  ISO 8859-1 é unha norma da ISO que define a codificación do alfabeto latino, incluíndo os diacríticos (como letras acentuadas, ñ, ç), e letras especiais (como Ø), necesarios para a escritura das seguintes linguas orixinarias de Europa occidental: afrikaans, alemán, aragonés, catalán, danés, escocés, español, feroés, finés, francés, gaélico, gallego, inglés, islandés, italiano, neerlandés, noruego, portugués, sueco e vasco.
  Tamén é coñecida como **Alfabeto Latino n.º1** ou **ISO Latín 1**.
  Esta norma pertence ao grupo de xogos de caracteres da ISO coñecidos como **ISO-8859** que se caracterizan por ter a codificación ASCII no seu rango inicial (128 caracteres) e outros 128 caracteres para cada codificación, có que en total utilizan 8 bits.
  Os caracteres de ISO-8859-1 son ademais os primeiros 256 caracteres do estándar **ISO-10646 (Unicode)**.

1.6.4 UNICODE (16 bits)

Sistema de codificación que ten como propósito romper coas limitacións dos códigos de caracteres tradicionais, como os definidos polo estándar ISO-8859. Este problema é que boa parte dos codificadores de caracteres tradicionais comparten o problema de que permiten procesamentos informáticos bilingües (xeralmente usando caracteres latinos e do idioma local), pero non multilingües (procesamento informático de idiomas arbitrarios misturados entre eles). UNICODE trata de resolver este problema.

1.7 Códigos para operacións Aritméticas

1.7.1 Ponderados

- **BCD Natural**
  Cada díxito decimal é codificado cunha secuencia de 4 bits
- **BCD Aiken** (Autocomplementario)
  É similar ao BCD, pero os "pesos" ou "valores" de cada cifra é diferente. No código BCD natural, os pesos son: 8 - 4 - 2 - 1, no código Aiken a distribución é: 2 - 4 - 2 - 1

1.7.2 Non Ponderados

- **BCD Exceso Tres**
  O código **BCD Exceso Tres** obtense sumando "3" a cada combinación do código BCD natural.

Exemplo dos BCD:

Decimal	Natural	Aiken	Exceso 3
0	0000	0000	0011

Decimal	Natural	Aiken	Exceso 3
1	0001	0001	0100
2	0010	0010	0101
3	0011	0011	0110
4	0100	0100	0111
5	0101	1011	1000
6	0110	1100	1001
7	0111	1101	1010
8	1000	1110	1011
9	1001	1111	1100

1.8 Códigos para Contadores

1.8.1 Binarios Continuos e Cíclicos

Un código binario é continuo se as combinacións correspondentes a dous valores sucesivos son adxacentes, é dicir, só difiren nun dos seus díxitos.

Un código binario é cíclico cando a última combinación é adxacente á primeira.

- **Gray**
  O Código Gray é un caso particular de sistema binario. Consiste nunha ordenación de 2^n números binarios de tal forma que cada número só teña un díxito binario distinto ó seu predecesor.

Secuencia	Binario	Gray	Secuencia	Binario	Gray
0	0000	0000	8	1000	1100
1	0001	0001	9	1001	1101
2	0010	0011	10	1010	1111
3	0011	0010	11	1011	1110
4	0100	0110	12	1100	1010
5	0101	0111	13	1101	1011
6	0110	0101	14	1110	1001
7	0111	0100	15	1111	1000

- **Johnson**
  Denomínase **código Johnson** (Johnson-Mobius) ao código binario continuo e cíclico que ten unha capacidade de codificación que ven dada por 2^n, sendo n o número de bits.

Dígito decimal	Código Johnson
0	00000
1	00001
2	00011
3	00111
4	01111

Dígito decimal	Código Johnson
5	11111
6	11110
7	11100
8	11000
9	10000

1.9 Códigos Detectores de Erros

1.9.1 Códigos de Paridade Constante

Poden detectar erros nun só bit, pero non poden corrixilo.

- **Paridade Par**
- **Paridade Impar**

1.9.2 Códigos de Peso Constante

Son códigos nos que todas as combinacións teñen o mesmo número de uns (1), polo tanto é moi simple detectar erros.

- **Biquinario**
  O código biquinario é un sistema de numeración usado en ábacos e nalgúns dos primeiros ordenadores, como o **Colossus**. O término biquinario refírese a que o código ten unha parte de dous estados (*bi*) e outra de cinco estados (*quin*).

- **2 entre 5**
  Trátase dun código ponderado onde os pesos para os 4 primeiros bits son 1,2,3,6. O bit máis significativo emprégase para para completar a paridade par. Ten dúas características a ter en conta:
  - Non existe codificación para o 0 (hai que elixir unha)
  - Só pode haber dous bits a 1 entre os cinco bits (de aí o seu nome)

1.10 Códigos Correctores de Erros

1.10.1 Código Hamming

O código Hamming é un código detector e corrector de erros que leva o nome do seu inventor, Richard Hamming.
Normalmente o Código Hamming é empregado en aplicacións onde ocorre só un erro por bloque de datos enviado, Hamming (7,4), onde se agregan tres bits adicionais de comprobación por cada catro bits de datos da mensaxe. O código así detecta e corrixe erros de 1 bit.
Para poder detectar (aínda que sen corrixir) erros de dous bits, débese engadir un bit mais, e o código chámase **Hamming extendido.**
O número de bits que se engaden para detectar e corrixir **t erros** denomínase **distancia mínima** e calcúlase coa fórmula: **dmin > 2t+1**.

1.10.2 CRC (Funcións polinómicas)

Os códigos cíclicos tamén chamados CRC (Códigos de Redundancia Cíclica) ou códigos polinómicos son moi potentes.
Estes códigos baséanse no uso dun polinomio xerador G(X) de grado r, e no principio de que n bits de datos binarios pódense considerar coma os coeficientes dun polinomio de orden n-1.
A estes bits de datos se lle engaden r bits de redundancia de forma que o polinomio resultante sexa divisible polo polinomio xerador. O receptor verificará se o polinomio recibido é divisible por G(X). Se non o é, haberá un erro na transmisión.
Así temos:

1. Detecta todos os erros simples.
2. Detecta calquera número de erros impares.
3. Detecta todos os erros dobres.
4. Detecta todos os erros de lonxitude <=r+1
5. Detecta todos os erros de lonxitude r+2 con probabilidade 1-2-r
6. Detecta todos os erros de lonxitude r+3 ou superior con probabilidade 1-2-r-1.

-- Volver