UNIVERSIDAD TECNOLÓGICA NACIONAL

FACULTAD REGIONAL CÓRDOBA

EL LENGUAJE DE LOS DATOS EN LA PC Y SU FORMA DE ALMACENAMIENTO

TRABAJO REALIZADO COMO APOYO PARA LA CATEDRA INFORMATICA I

Autora:

Ing. Norma Mascietti

Ing. Sylvia Arias

FORMAS DE REPRESENTACION

Introducción a los Sistemas de Numeración

En Las Ciencias de la computación y en programación los sistemas de numeración mas utilizados son, el sistema binario y el hexadecimal. Estos sistemas se utilizan para representar números(cantidades) en lugar del conocido sistema decimal. Los sistemas de numeración son lenguajes a través de los cuales se puede comunicar la idea de cantidad para ello se valen de una cierta cantidad de símbolos y de algunas reglas para combinarlos y lograr de esta manera comunicar la idea de cantidad que deseamos. Para poder recibir esta información (idea de cantidad) necesitamos conocer el lenguaje con el cual se expresa la misma. Para lograr este objetivo en primer lugar vamos a definir que es un sistema de numeración posicional, y de como se aplican las mismas reglas para generar numerales en el sistema decimal, el binario y el hexadecimal todos ellos sistemas de numeración posicionales. Además en esta unidad vamos a averiguar como es la organización binaria de los datos (bits, bytes, palabras y palabras dobles), sistemas numéricos con signo y sin signo, operaciones aritméticas, lógicas, en valores binarios, campos de bits, empaquetado de datos y el juego de caracteres ASCII.

El Sistema Numérico Decimal

Hemos utilizado el sistema decimal (de base 10) por tanto tiempo que prácticamente lo tomamos como algo natural. Cuando vemos un numeral, por ejemplo: 313 no pensamos en el valor en sí, en lugar de esto hacemos una representación mental de cuántos elementos representa éste valor. En realidad, el número 313 representa:

$3*10^2 + 1*10^1 + 3*10^0$

ó lo que es lo mismo:

$300 + 10 + 3$

O sea el valor numérico de cada uno de los símbolos (0,1,2,3,4,5,6,7,8,9)de los 10 que componen el sistema decimal depende de la posición en la que se encuentre el símbolo dentro del numeral.
Por ejemplo en el numeral 123.1, no vale lo mismo el uno que figura en el lugar de las centenas que el uno que se encuentra después de la coma, en el lugar de los decimos de unidad.

Lo vemos a continuación mas claramente en la forma expandida del numero, en el primer caso el uno va multiplicado por la base (10 en el sistema decimal) elevada a la potencia 2 (lugar de las centenas) y en el otro caso va multiplicado por la base elevada a la potencia –1.

↓ ↓

$1*10^2 + 2*10^1 + 3*10^0 + 1*10^{-1} + 3*10^{-2}$

1 2 3, 1 3

Sistema decimal ⇒ base =10 o de base 10

10 símbolos {0,1,2,3,4,5,6,7,8,9}

Este valor absoluto propio del símbolo y el valor relativo o posicional la existencia de un neutro y la unidad, el orden monótono creciente de los símbolos la existencia de una operaciones de adición y multiplicación, siendo la base la cantidad de símbolos que posee el sistema determina a un **sistema de numeración posicional** en un sistema de notación posicional cualquier numero debe responder ala notación expandida

Forma expandida

An A0,A-1 A-m=An $*b^n$ +A0$*b^0$ +A-1$*b^{-1}$ + A-m $*b^{-m}$

El sistema numérico binario

El sistema de numeración binario al igual que el sistema decimal es un sistema de numeración posicional pero a diferencia del sistema decimal solo tiene dos símbolos cero y uno y por lo tanto la base es 2 ya que la base de un sistema posicional es igual a la cantidad de símbolos que posee el sistema. Un numero en el sistema binario también puede expresarse en notación expandida según la formula vista anteriormente.
Las computadoras trabajan utilizando la lógica binaria. En ellas se representan valores utilizando dos niveles de voltaje (generalmente 0 Voltios). Y(5 Voltios), con éstos niveles pueden representar exactamente dos valores diferentes, utilizamos los valores cero y uno. Éstos dos valores corresponden a los dígitos utilizados por el sistema binario.
El sistema binario trabaja de forma similar al sistema decimal con dos diferencias, en el sistema binario sólo está permitido el uso de los dígitos 0 y 1 (en lugar de 0~9) y en el sistema binario la base es 2 porque la cantidad de símbolos es 2, por ello es que se utilizan potencias de 2 en lugar de potencias de 10
Al tener únicamente dos símbolos para representar cantidades numéricas la cantidad de dígitos que conforman el numeral binario es mayor que en el sistema decimal
Por ejemplo si tenemos un conjunto que no contiene ningún elemento podemos representarlo con el numeral 0,tanto en binario como en decimal, Si tenemos un conjunto con un elemento podemos representarlo con el numero 1,tanto en binario como en decimal, Pero si tenemos un conjunto con dos elementos podemos representarlo con el numeral 2 en decimal pero en binario se nos terminaron los símbolos por lo tanto tenemos que usar una combinación de los mismos para representar el conjunto así, el numeral en binario que representa este conjunto de dos

elementos es el 10 que posee dos dígitos el primer dígito 1 multiplicado por la base elevado a la potencia 1 y el segundo dígito 0 multiplicado por la base elevada a la potencia 0.

Conversión de binario a decimal

El valor binario 10 (2)
Se representa en notación expandida:

$$1*2^1 + 0*2^0$$

$$2 + 0 = 2\ (10) \rightarrow \textbf{Numero en base 10 o Decimal}$$

El valor binario 11001010(2)
Se representa en notación expandida:

$$1*2^7 + 1*2^6 + 0*2^5 + 0*2^4 + 1*2^3 + 0*2^2 + 1*2^1 + 0*2^0$$

$$128 + 64 + 8 + 2$$

$=202_{10} \rightarrow$ Numero en base 10 o Decimal

Formatos binarios

En un sentido estricto, cada número binario contiene una cantidad infinita de dígitos, también llamados bits que es una abreviatura de *binary digits,* por ejemplo, podemos representar el número siete de las siguientes formas:

- 111
- 00000111
- 000000000000111

Por conveniencia ignoraremos cualquier cantidad de ceros a la izquierda, sin embargo, como las instrucciones compatibles con los procesadores Intel 80x86 trabajan con grupos de ocho bits a veces es más fácil extender la cantidad de ceros a la izquierda en un múltiplo de cuatro
ú ocho bits, por ejemplo, el número siete podemos representarlo así: 0111_2 ó 00000111_2. También es conveniente separar en grupos de cuatro dígitos los numerales binarios grandes, por ejemplo, el valor binario 1010111110110010 puede ser escrito así 1010 1111 1011 0010. Además, en una cadena binaria asignaremos al dígito de la extrema derecha como el bit de posición cero y cada bit subsecuente se le asignará el siguiente número sucesivo, de ésta manera un valor binario de ocho bits utiliza los bits cero al siete:

X_7 X_6 X_5 X_4 X_3 X_2 X_1 X_0

Al bit cero se le conoce como el *bit de bajo orden* en tanto que al bit de la extrema izquierda diferente de cero se le llama *bit de alto orden*.

Organización de datos

En términos matemáticos un valor puede tomar un número arbitrario de bits, pero las computadoras por el contrario, generalmente trabajan con un número específico de bits, desde bits sencillos pasando por grupos de cuatro bits (llamados *nibbles*), grupos de ocho bits (*bytes*), grupos de 16 bits (*words*, ó palabras) y aún más. Como veremos mas adelante, existe una buena razón para utilizar éste orden.

Bits

La más pequeña cantidad de información en una computadora binaria es el bit, éste solamente es capaz de representar dos valores diferentes, sin embargo esto no significa que exista una cantidad muy reducida de elementos representables por un bit, todo lo contrario, la cantidad de elementos que se pueden representar con un sólo bit es infinito, considere esto, podemos representar por ejemplo, cero ó uno, verdadero ó falso, encendido ó apagado, masculino ó femenino. Más aún, no estamos limitados a representar elementos antagónicos, un bit sencillo puede representar cualesquiera dos valores, por ejemplo, blanco ó 432, perro ó caliente. Y para ir aún más lejos, dos bits adyacentes pueden representar cosas completamente independientes entre sí, lo que se debe tener en cuenta es que un bit sencillo sólo puede representar dos cosas a la vez. Esta característica otorga a las computadoras binarias un campo infinito de aplicaciones.

Bytes

Un byte está compuesto de ocho bits y es el elemento de dato más pequeño direccionable por un procesador 80x86, esto significa que la cantidad de datos más pequeña a la que se puede tener acceso en un programa es un valor de ocho bits. Los bits en un byte se enumeran del cero al siete de izquierda a derecha, el bit 0 es el *bit de bajo orden ó el bit menos significativo* mientras que el bit 7 es el *bit de alto orden ó el bit más significativo*. Nos referimos al resto de los bits por su número. Observe que un byte está compuesto Como un byte contiene ocho bits, es posible representar 2^8, ó 256 valores diferentes. Generalmente utilizamos un byte para representar valores numéricos en el rango de 0 ~ 255, números con signo en el rango de –128 ~ +127, códigos de caracter ASCII y otros tipos de datos especiales que no requieran valores diferentes mayores que 256.

Word(Palabra)

Una palabra (word) es un grupo de 16 bits enumerados de cero hasta quince, y al igual que el byte, el bit 0 es el bit de bajo orden en tanto que el número quince es el bit de alto orden. Una palabra contiene dos bytes, el de bajo orden que está compuesto por los bits 0 al 7, y el de alto orden en los bits 8 al 15. Naturalmente, una palabra puede descomponerse en cuatro nibbles. Con 16 bits es posible representar 2^{16} (65,536) valores diferentes, éstos podrían ser el rengo comprendido entre 0 y 65,535, ó como suele ser el caso, de -32,768 hasta +32,767. También puede ser cualquier tipo de datos no superior a 65,536 valores diferentes.

El sistema numérico hexadecimal

Un gran problema con el sistema binario es la verbosidad. Para representar el valor 202_{10} se requieren ocho dígitos binarios, la versión decimal sólo requiere de tres dígitos y por lo tanto los números se representan en forma mucho más compacta con respecto al sistema numérico binario. Desafortunadamente las computadoras trabajan en sistema binario y aunque es posible hacer la conversión entre decimal y binario, ya vimos que no es precisamente una tarea cómoda. El sistema de numeración hexadecimal, o sea de base 16, resuelve éste problema (es común abreviar hexadecimal como hexa). El sistema hexadecimal es compacto y nos proporciona un

mecanismo sencillo de conversión hacia el formato binario, debido a esto, la mayoría del equipo de cómputo actual utiliza el sistema numérico hexadecimal. Como la base del sistema hexadecimal es 16, cada dígito a la izquierda del punto hexadecimal representa tantas veces un valor sucesivo potencia de 16, por ejemplo, el número 1234_{16} es igual a:

$1*16^3 + 2*16^2 + 3*16^1 + 4*16^0$
lo que da como resultado:

$4096 + 512 + 48 + 4 = 4660_{10}$
Cada dígito hexadecimal puede representar uno de dieciséis valores entre 0 y 15_{10}. Como sólo tenemos diez dígitos decimales, necesitamos "inventar" seis dígitos adicionales para representar los valores entre 10_{10} y 15_{10}. En lugar de crear nuevos símbolos para éstos dígitos, utilizamos las letras A a la F. La conversión entre hexadecimal y binario es sencilla, considere la siguiente tabla:

Binario	**Hexadecimal**
0000	0
0001	1
0010	2
0011	3
0100	4
0101	5
0110	6
0111	7
1000	8
1001	9
1010	A
1011	B
1100	C
1101	D
1110	E
1111	F

Ésta tabla contiene toda la información necesaria para convertir de binario a hexadecimal y viceversa. Para convertir un número hexadecimal en binario, simplemente sustituya los correspondientes cuatro bits para cada dígito hexadecimal, por ejemplo, para convertir 0ABCDh en un valor binario:

0 A B C D (Hexadecimal)
0000 1010 1011 1100 1101 (Binario)
Por comodidad, todos los valores numéricos los empezaremos con un dígito decimal; los valores hexadecimales terminan con la letra *h* y los valores binarios terminan con la letra *b*. La conversión de formato binario a hexadecimal es casi igual de fácil, en primer lugar necesitamos asegurar que la cantidad de dígitos en el valor binario es múltiple de 4, en caso contrario agregaremos ceros a la izquierda del valor, por ejemplo el número binario 1011001010, la primera etapa es agregarle dos ceros a la izquierda para que

contenga doce ceros: 001011001010. La siguiente etapa es separar el valor binario en grupos de cuatro bits, así: 0010 1100 1010. Finalmente buscamos en la tabla de arriba los correspondientes valores hexadecimales dando como resultado, 2CA, y siguiendo la convención establecida: 02CAh.

Operaciones aritméticas y lógicas

Existen varias operaciones aritméticas que se pueden ejecutar en números binarios y hexadecimales, por ejemplo, podemos sumar, restar, multiplicar, dividir y otras operaciones aritméticas más, aunque es aconsejable que Usted sepa ejecutar éstas operaciones a mano, es más recomendable que haga uso de una calculadora apropiada, básicamente para evitar errores ya que nuestro pensamiento está condicionado por años al sistema numérico de base 10. Por otra parte, al intentar ejecutar una operación aritmética en formato binario es fácil caer en errores debido a la verbosidad mencionada, en éste caso es recomendable que primero haga la conversión a formato hexadecimal, ejecute las operaciones necesarias y finalmente vuelva a convertir el resultado a formato binario.

Operaciones lógicas en bits

Existen tres operaciones principales que Usted puede ejecutar en números binarios y hexadecimales: **AND**, **OR**, (OR exclusivo), y **NOT y que tienen sus correspondientes operadores relacionales o logicos en C &&(and) ¦¦ (or) y ! (not)**.. La operación lógica **AND** es como sigue:

0 *and* 0 = 0
0 *and* 1 = 0
1 *and* 0 = 0
1 *and* 1 = 1

Un hecho importante acerca de la operación lógica **AND** es que se puede utilizar para forzar un resultado a cero, si uno de los operandos es cero, el resultado es siempre cero independientemente del otro operando, esto se puede verificar en la tabla de verdad de arriba en donde tanto el renglón como la columna que contienen ceros el resultado es cero, por el contrario, si uno de los operandos contiene 1, el resultado es exactamente el valor del otro operando. Ésta característica de la operación lógica **AND** es muy importante, particularmente con cadenas de bits en donde deseamos forzar algún bit individual de la cadena a cero.
El operador lógico **OR** se define así:

0 or 0 = 0
0 or 1 = 0
1 or 0 = 0
1 or 1 = 1

En palabras decimos: si el primero de los operandos *ó (OR)* el segundo de los operandos (ó ambos) es 1, el resultado es 1, de lo contrario el resultado es 0. A ésta operación lógica también se le conoce como *OR inclusivo*. Si uno de los operandos es uno, el resultado es siempre uno independientemente del valor del segundo operando, si uno de los operandos es cero, el resultado es siempre el valor del segundo operando. Esto es importante como veremos más adelante.

El operador lógico **NOT** acepta solamente un operando y está definido como:

NOT 0 = 1
NOT 1 = 0

Números con signo y sin signo

Hasta éste momento, hemos tratado a los números binarios como valores sin signo, el número binario ...00000 representa al cero, ...00001 representa al uno, ...00010 representa al dos, etc., pero ¿Qué hay con los números negativos? En ésta sección discutiremos cómo representar números negativos utilizando el sistema de numeración binario.
Para representar números con signo utilizando el sistema de numeración binario tenemos que colocar una restricción a nuestros números, éstos deben tener un número finito y fijo de bits.
En lo que concierne a los procesadores 80x86 ésta restricción no es muy importante, después de todo, los 80x86 sólo pueden direccionar un número finito de bits. Para nuestros propósitos limitaremos el número de bits a ocho, 16, 32 ú otro número pequeño de bits.
Con un número fijo de bits sólo podemos representar un cierto número de objetos, por ejemplo, con ocho bits sólo podemos representar $\mathbf{2^8}$ o sea 256 objetos diferentes. Los valores negativos son objetos por su propio derecho al igual que los números positivos, por tanto necesitamos utilizar algunos de los 256 valores diferentes para representar a los números negativos,por lo tanto la cantidad de datos positivos que vamos a poder representar ahora con 8 bits va a ser menor,de esta manera, asignaremos la mitad de las posibles combinaciones para los números negativos y la otra mitad para los números positivos.
Así podemos representar los valores negativos que van del -128...-1 y los valores positivos del 0...127 con un solo byte de ocho bits. Con una palabra de 16 bits podemos representar $\mathbf{2^{16}}$ o sea podemos representar valores en el rango de -32,768 hasta +32,767. Con una palabra doble de 32 bits se pueden representar valores que van de -2,147,483,648 hasta +2,147,483,647. En general, con *n* bits podemos representar los valores con signo en el rango comprendido entre -2^{n-1} hasta $2^{n-1}-1$. Bien, ¿Cómo podemos representar valores negativos? Existen muchas formas pero los procesadores 80x86 utilizan la notación de complemento a dos, en éste sistema, el bit de alto orden de un número es el *bit de signo.* Si el bit de alto orden es cero el número es positivo, si el bit de alto orden es uno, el número es negativo. En el caso de un número positivo, éste es almacenado como un valor binario estándar, pero si el número es negativo éste es almacenado en la forma de complemento de dos, para esto se utiliza el siguiente algoritmo:

Se invierten todos los bits en el número, es decir, se aplica la función lógica NOT.

1. Se agrega uno al resultado invertido.

Por ejemplo, para calcular el equivalente en ocho bits de -5:

0000 0101	cinco (en binario)
1111 1010	Se invierten todos los bits.
1111 1011	Se suma uno para obtener el resultado.

Si tomamos el valor de menos cinco y le ejecutamos la operación de complemento de dos obtenemos el valor original, como es de esperarse:

1111 1011	complemento de dos para -5.
0000 0100	se invierten todos los bits.
0000 0101	se suma uno para obtener el resultado (+5).

El código ASCII

El juego de caracteres ASCII (excluyendo los caracteres extendidos definidos por IBM) está dividido en cuatro grupos de 32 caracteres. Los primeros 32 caracteres, del código ASCII 0 hasta el ASCII $1Fh_{16}$ (31_{10}) forman un juego especial de caracteres no imprimibles llamados caracteres de control ya que ejecutan varias operaciones de despliegue/impresión en lugar de mostrar símbolos, ejemplo de éstos son el retorno de carro que posiciona el llamado cursor al lado izquierdo de la actual línea de caracteres, avance de línea que mueve hacia abajo el llamado cursor una línea en el dispositivo de salida. Desafortunadamente, los diferentes caracteres de control ejecutan diferentes operaciones dependiendo del dispositivo de salida ya que existe poca estandarización al respecto.

El segundo grupo de caracteres comprende varios símbolos de puntuación, caracteres especiales y dígitos numéricos, los caracteres mas notables de éste grupo son el caracter de espacio (código ASCII 20h) y los dígitos numéricos (códigos ASCII 30h al 39h). Observe que los digitos numéricos difieren de sus respectivos valores sólo en el nibble de alto orden, restando 30h de un código numérico ASCII dado se obtiene el equivalente numérico.

El tercer grupo de caracteres ASCII está reservado a las letras mayúsculas. Los códigos ASCII para los caracteres "A" a la "Z" están en el rango comprendido entre 41h y 5Ah (65 al 90 decimal). Como éstos caracteres están definidos de acuerdoal alfabeto utilizado en el idioma inglés solo hay 26 diferentes caracteres alfabeticos utilizando los seis códigos restantes para varios símbolos especiales.

El cuarto y último grupo de caracteres ASCII está reservado a las letras minúsculas, cinco símbolos especiales adicionales y otro caracter de control (borrar). Los caracteres ASCII para las letras minúsculas utilizan los códigos 61h al 7Ah. Si Usted convierte a binario los códigos correspondientes a las letras mayúsculas y minúsculas observará que los símbolos para las mayúsculas difieren de sus respectivas minúsculas en una posición de bit. Las letras mayúsculas siempre contienen un cero en la posición cinco en tanto que las letras minúsculas contienen un uno en la misma posición, es posible utilizar éste hecho para convertir d e mayúsculas a minúsculas y viceversa.

De acuerdo con lo ya expuesto podemos afirmar que los bits de posición seis y cinco determinan qué caracteres ASCII estamos utilizando de acuerdo a la siguiente tabla:

Bit 5	**Grupo**

0	0	Caracteres de control
0	1	Dígitos y puntuación
1	0	Letras mayúsculas y caracteres especiales
1	1	Letras minúsculas y caracteres especiales

En el código estándar ASCII el bit de posición siete siempre es cero, esto siginifica que el juego de caracteres ASCII consume la mitad de la capacidad de representación de un byte. IBM utiliza los restantes 128 códigos de caracter para representar diferentes símbolos especiales incluyendo caracteres internacionales (con respecto a EEUU) como letras acentuadas, símbolos matemáticos y caracteres para dibujar líneas. Observe que éstos caracteres adicionales no están estandarizados como una extensión del código ASCII, sin embargo la firma IBM tiene suficiente peso de tal manera que prácticamente todas las computadoras personales basadas en procesadores 80x86 soportan el juego de caracteres extendidos IBM/ASCII. Esto también es válido para las impresoras