Apuntes de Arquitectura de Computadoras (Repaso de temas previos)

Por M. C. Miguelangel Fraga Aguilar
mfraga@itmorelia.edu.mx
http://sagitario.itmorelia.edu.mx/mfraga

Representaciones numéricas

En estos apuntes se usara el posfijo b para denotar un número escrito en binario, d para un número decimal y h para un número hexadecimal.

Números sin Signo.

En las computadoras modernas se representa a los números usando el sistema binario. La representación de números enteros en sistema binario consiste en usar una cadena de dígitos binarios o bits. Cada bit puede tomar el valor de 0 o 1. La posición de cada dígito indica su peso, que es la potencia correspondiente de 2. El valor de cada bit se multiplica por su peso y se suman los productos. De esta forma, el equivalente en decimal V de un número binario se puede conocer usando la ecuación 1.

$$V=\sum_{i=0}^{n-1} d_i 2^i ; \qquad d_i \in \{0,1\} \tag{1}$$

Ejemplo: el equivalente decimal del número binario 10110b se puede obtener por medio de la tabla 1.

i	4	3	2	1	0
Peso	16	8	4	2	1
Dígito (d_i)	1	0	1	1	0

Tabla 1: Ejemplo de un número binario, mostrando el número de bit (i) y el peso correspondiente

V=1x16+0x8+1x4+1x2+0x1=22

Para convertir un número decimal a binario, se divide entre dos, calculando el cociente y el residuo. El residuo sera el bit **menos significativo** del número binario equivalente. Los demás bits se calculan dividiendo el cociente nuevamente entre dos, siendo el residuo el bit siguiente. Esta operación se repite hasta que el cociente sea 0. Ejemplo: se desea convertir el 345 decimal a binario. La siguiente tabla 2 muestra los resultados de las sucesivas divisiones entre dos.

Ejercicios: Convierta los siguientes números a binario: 1234d, 9876d y 2583d

Convierta los siguientes números decimales a binario: 1010 0011b, 1111 0011 1110b

Dividendo	Cociente	Residuo
345	172	1
172	86	0
86	43	0
43	21	1
21	10	1
10	5	0
5	2	1
2	1	0
1	0	1

Tabla 2: Resultados parciales en un ejemplo de conversión de decimal a binario

De este modo, el equivalente de 345d en binario es 1 0101 1001.

Los números binarios también pueden verse como si cada bit se multiplicara por un vector de magnitud igual al peso del bit y el equivalente es el vector resultante, como los muestra la figura 1.

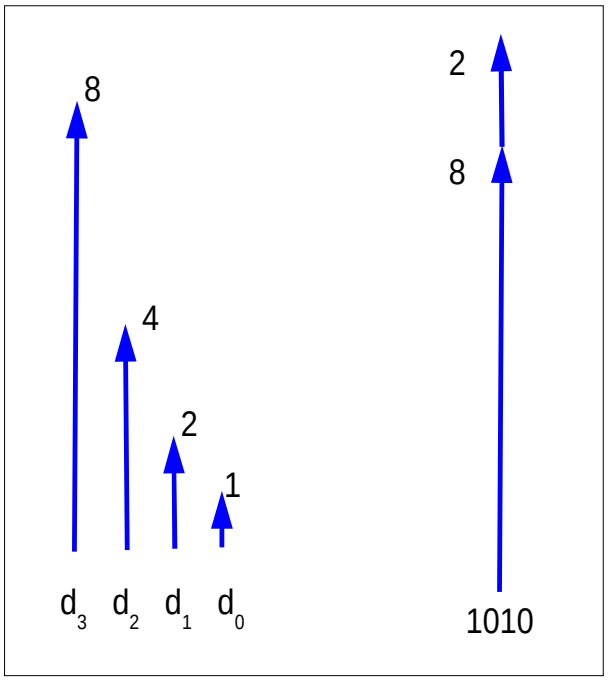

Figura 1: Ejemplo de la interpretación de un número binario como vector

Observe que en el binario natural solo se pueden representar números enteros positivos. Por esta razón, se le conoce a esta representación como números sin signo. En todas las computadoras se cuenta con un número finito de circuitos para almacenar y hacer operaciones con dichos dígitos binarios, lo que hace que solo se pueda trabajar con números binarios de un tamaño finito. Esto implica que el rango de números sin signo con los que se puede trabajar es limitado. Si una computadora utiliza N bits para representar a los números sin signo, estos N bits pueden acomodarse en 2^N combinaciones diferentes. Una de dichas combinaciones se dedica para representar al cero, por lo que solo quedan 2^N-1 combinaciones para representar a otros números. De esta forma, el rango de los números sin signo de N bits es de 0 a 2^N-1.

Ejemplo: En la tabla 3 se muestran todas las combinaciones que se pueden formar con 3 bits y su equivalente en decimal. El rango para los números sin signo de 3 bits es de 0 a 7.

Número binario	Equivalente decimal
000	0
001	1
010	2
011	3
100	4
101	5
110	6
111	7

Tabla 3: Números sin signo de 3 bits y su equivalente en decimal

Notación hexadecimal

La notación en binario es muy natural para implementarse en circuitos electrónicos digitales, pero al escribirla resulta demasiado larga. Para tener una notación más compacta, es muy común usar la notación hexadecimal. En hexadecimal la base es el numero 16. Esto implica que se necesitan dígitos con valores equivalentes a los números decimales del 10 al 15, que se representan con las letras a, b, c, d, e y f respectivamente. De manera similar al binario, la posición de cada dígito indica su peso, y este es la potencia correspondiente de 16. El equivalente en decimal V de un numero hexadecimal se puede encontrar mediante la ecuación 2.

$$V=\sum_{i=0}^{n-1} d_i 16^i; \qquad d_i \in \{0 \ldots F\} \tag{2}$$

La gran ventaja de la notación hexadecimal es que cada dígito hexadecimal corresponde con cuatro bits, por lo que la conversión entre las dos notaciones es muy directa. La tabla 4 muestra el equivalente en binario y decimal de cada dígito hexadecimal. Usando la información de dicha tabla, es fácil convertir de hexadecimal a binario simplemente

sustituyendo cada dígito hexadecimal por su equivalente en cuatro bits. A modo de ejemplo, considere el número hexadecimal A4F8h, el cual equivale al 1010 0100 1111 1000b. La conversión de binario a hexadecimal puede lograrse agrupando los bits en grupos de cuatro y sustituyendo cada grupo por el dígito hexadecimal correspondiente. Ejemplo: 101 0001 1101 1110b equivale a 51DEh.

Hexadecimal	Binario	Decimal
0	0000	0
1	0001	1
2	0010	2
3	0011	3
4	0100	4
5	0101	5
6	0110	6
7	0111	7
8	1000	8
9	1001	9
A	1010	10
B	1011	11
C	1100	12
D	1101	13
E	1110	14
F	1111	15

Tabla 4: Equivalente de cada dígito hexadecimal en binario y decimal

Números con signo

La notación binaria puede usarse también para representar números negativos. Para esto, alguno de los bits debe usarse para representar al signo y los demás deben codificar de alguna forma el resto del número. Históricamente se han usado varias notaciones para representar a los números con signo en las computadoras. La más utilizada en la actualidad es la de complemento a dos, pero las otras aún son aplicadas en algunos nichos especiales.

Complemento a dos

La notación de complemento a dos es la más utilizada en las computadoras modernas. La formula para encontrar el equivalente decimal de un numero binario escrito en notación con signo de complemento a dos esta dada por la ecuación 3.

$$V = -d_{n-1}2^{n-1} + \sum_{i=0}^{n-2} d_i 2^i; \qquad d_i \in \{0,1\} \tag{3}$$

Observe que la principal diferencia entre la ecuación 1 y la ecuación 3 es que el signo del peso del bit más significativo es negativo. Como el peso del bit más significativo es mayor que la suma de los pesos de los demás bits, si el bit más significativo es uno, el número es negativo y si es cero el número es positivo. Por esta razón al bit más significativo se le conoce como bit de signo. La figura 2 muestra como puede interpretarse un número con signo en complemento a dos como una suma de vectores con magnitud igual al peso de cada bit.

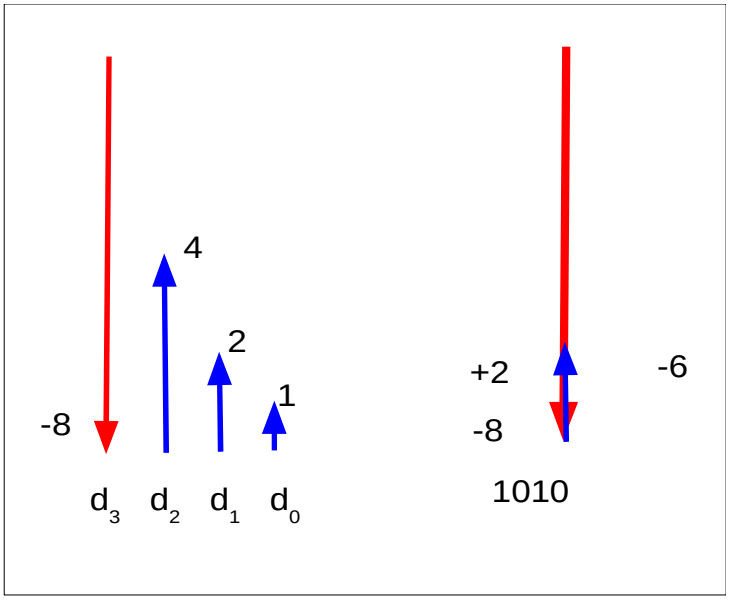

Figura 2: Ejemplo de la interpretación como vectores de un número con signo en complemento a dos.

Ejemplo: Se desea encontrar el equivalente en decimal al número binario con signo en complemento a dos 1001 1010b. Se suma los productos de cada bit multiplicado por el peso respectivo: V=1x(-128)+1x16+1x8+1x2=-102d.

Observe que un numero binario cualquiera puede interpretarse como un número sin signo en binario natural o como un número con signo en complemento a dos. Por ejemplo: el número 1010 0101b puede interpretarse como V=128+32+4+1=165d como numero sin signo o como V=--128+32+4+1= -91d.

Ejercicios: Encuentre los equivalentes en decimal de los siguientes números binarios, interpretándolos tanto como números sin signo como números con signo en complemento a dos. a) 0111 0001b b) 1000 0000b c) 1111 1111b d) F44Dh e) 50DEh f) 9DEAh

La definición matemática del complemento a dos es $C=2^n-V$. Si se dispone de una calculadora con capacidad de realizar operaciones con números hexadecimales, resulta muy practico usar esta definición para obtener el complemento a dos de un número binario expresado en hexadecimal. Se aprovecha el hecho de que 2^n es un uno seguido de n ceros en binario.
Ejemplo: Represente el numero -1234d en notación de magnitud y signo, complemento a uno y complemento a dos en 16 bits.
Solución: 1234d equivale a 04D2h. Para el complemento a dos, usamos la definición: $c=2^{16}$-04d2h=10000h-04D2h=FB2Eh.

Una forma rápida de calcular manualmente el complemento a dos de un número binario es buscar de derecha a izquierda el primer uno y copiar la cadena hasta este uno, y a partir de el colocar los bits restantes negados. A manera de ejemplo, el complemento a dos de 01010100b es 10101100b

La razón principal por la que la notación de complemento a dos es la más usada en las computadoras modernas es que con esta notación las operaciones de suma y resta de números con signo corresponden a las operaciones de suma y resta de números sin signo descartando cualquier acarreo que pudiera generarse, como muestran los siguientes ejemplos:

(+3) 0011	(-3) 1101	(-5) 1011	(+7) 0111
+ (+2) 0010	+ (-2) 1110	+ (+3) 0011	+ (-5) 1011
(+5) 0101	(-5) 11011	(-2) 1110	(+2) 10010

La figura 3 muestra una recta numérica con los números binarios de 3 bits interpretados como números con y sin signo. En ella se puede observar que si se suma 1 al 111b y se descarta el acarreo, se obtiene el 000b. Por esta razón es que el 111b corresponde al -1 en tres bits. El hecho de que al hacer una operación de suma o resta con un numero fijo de bits se obtiene otro número con el mismo número de bits, se puede interpretar también como si la tradicional recta numérica se convirtiera en un circulo numérico, donde se pasa de un extremo a otro igual que en una hoja de papel enrollada. El circulo numérico correspondiente a los números binarios de cuatro bits se muestra en la figura 4.

Lo anterior implica que cuando se utiliza aritmética con un número finito de bits, las operaciones que tengan un resultado que requiera de más bits para representarse tendrán un resultado erróneo. Se llama acarreo cuando se interpreta a los operandos como números sin signo y el resultado de una suma no pertenece al rango de números que se puede representar con el número de bits que se esta utilizando. En el caso de una resta se le conoce como préstamo. Cuando el resultado de una suma o resta de números interpretados como con signo en complemento a dos se sale del rango que se puede representar con la cantidad de bits que se esta usando, se dice que ocurrió un sobreflujo (Overflow en Inglés). Los procesadores modernos señalan la ocurrencia de estas condiciones mediante registros de un bit llamados banderas. Usualmente una bandera indica la existencia de acarreo o prestamo (**CF**) y otra la de sobreflujo (**OF**). Además de estas banderas, es común encontrar otra bandera que almacena una copia del bit de signo de resultado (**SF** Sign Flag) y otra que indica si el resultado fue cero o no (**ZF** Zero Flag).

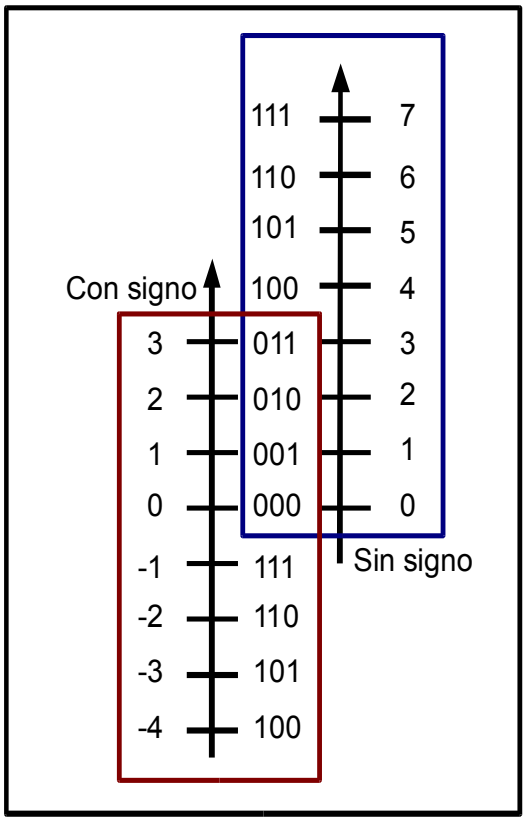

Figura 3: Recta numérica desplazada para el complemento a dos de 3 bits

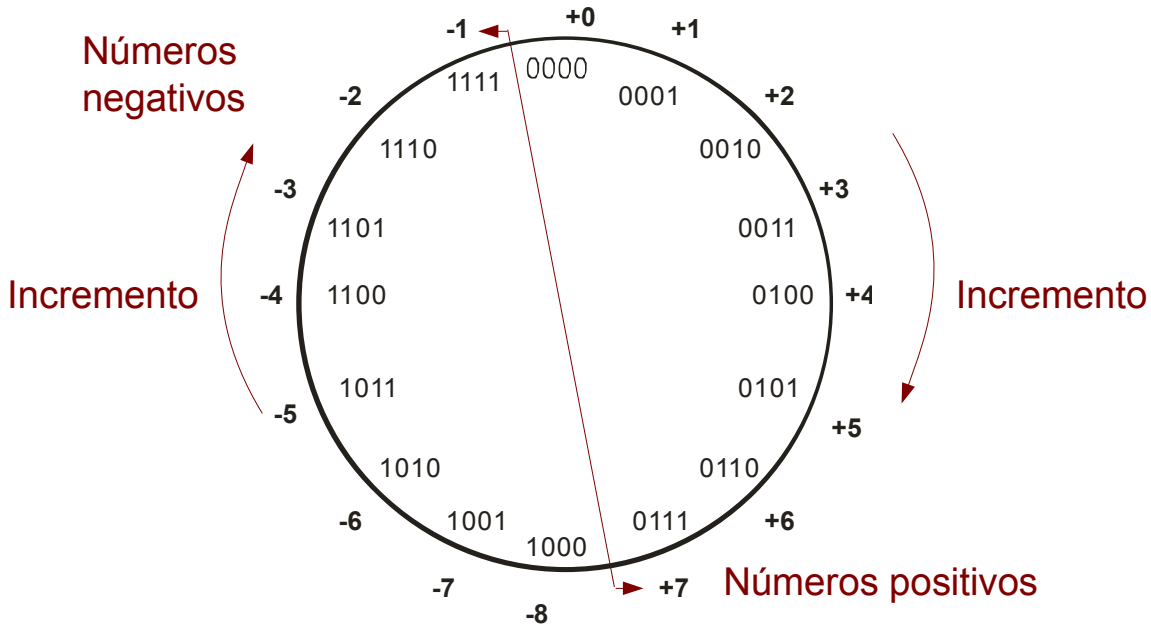

Figura 4: Círculo numérico para los números en complemento a dos de 4 bits

Ejempo: Calcule el resultado de las suma de DEA3h y BABEh, e indique el valor que tomarían las banderas de acarreo, signo, cero y sobreflujo después la suma. Usando una calculadora capaz de sumar en hexadecimal, el resultado es 19961h. Como se esta trabajando con 16 bits, el bit extra se descarta del resultado, pero se guarda en la bandera de acarro, de modo que el resultado en 16 bits es 9961h y cf=1. El bit 15 del resultado es 1, por lo que SF=1 y el resultado es negativo. Zf es 0 ya que el resultado es distinto de cero. Finalmente, para determinar el valor de la bandera de sobreflujo, se revisa si se cumple con la ley de los signos de la adición. El primer sumando es negativo, al igual que el segundo. La suma de dos sumandos negativos debe ser negativa. Como dijimos anteriormente, el resultado es negativo y al cumplirse la ley de los signos de la suma, sabemos que no hay sobreflujo y que OF=0.

Ejercicios: Calcule el resultado de las siguientes operaciones e indique el valor que tendrían las banderas de acarreo (cf), signo (sf), cero (zf) y sobreflujo (of) después de ejecutarse dichas operaciones.

```
a)   ACDEh     b) 56CAh   c) E567h      d) 6234h
    +9876h       +4321h     -6234h        -E567h
```

Temas selectos de lenguaje C que se aplican a los microcontroladores

Tipos de datos enteros en lenguaje C

El lenguaje C esta definido por un estándar ANSI/ISO, el cual se ha ido actualizando con forme pasa el tiempo. La versión de 1999 del estándar, conocida como c99, define los siguientes tipos de datos y garantiza que en cualquier compilador que cumpla con el estándar se podrá representar como mínimo los rangos de números enteros mostrados en la tabla 5. Observe que para los tipos con signo, los rangos son simétricos, y son un subconjunto de los rangos de los números con signo en complemento a dos para 8, 16, 32 y 64 bits, que es lo que se implementa en la mayor parte de los compiladores.

Es muy importante al escribir un programa en cualquier lenguaje, que se realice un estudio de los posibles rangos que puede tomar una variable, y que se escoja un tipo de dato que permita almacenar todos esos valores posibles. No hacerlo es una de las fuentes más comunes de errores intermitentes y de vulnerabilidades de seguridad en los sistemas informaticos.

Tipo de dato de C	Mínimo	Máximo
char	-127	127
unsigned char	0	255
short [int]	-32,767	32,767
unsigned short [int]	0	64,535
int	-32,767	32,767
unsigned [int]	0	64,535

long [int]	-2,147,483,647	2,147,483,647
unsigned long [int]	0	4,294,967,295
long long [int]	-9,223,372,036,854,775,807	9,223,372,036,854,775,807
unsigned long long [int]	0	18,446,744,073,709,551,615

Tabla 5: Rangos para datos enteros garantizados por el estándar C99

Conversión (cast) entre tipos enteros

Cuando se realiza la conversión de un tipo con signo a uno sin signo, los bits no cambian, pero si la interpretación. Por ejemplo, si se tiene una variable con signo de 16 bits con un valor de -12345 decimal, este se representa en binario como CFC7h. Al asignar el contenido de esta variable a otra del mismo tamaño pero sin signo, el valor en binario sigue siendo CFC7h, pero ahora corresponde al +53191 decimal. El fragmento de código en c que se muestra a continuación implementa el ejemplo anterior y muestra los resultados en decimal del equivalente en decimal como número con signo y sin signo.

```
short int v=-12345;//CFC7h
unsigned short int uv=(unsigned short)v;
printf("v=%d,uv=%u\n",v,uv);
```

En el caso de que se pase de un tipo con un mayor número de bits a uno con menos bits, se descartan los bits más significativos y se conservan los menos significativos. Ejemplo: 1234h → 34h. Para el caso contrario, es decir, pasar a un mayor de bits, se trata diferente a los números con signo o sin signo. Un número sin signo se hace crecer siempre añadiendo ceros para completar el numero de bits deseado. Esta operación se conoce como extensión por ceros. En el caso de los números con signo, es deseable que un número negativo siga siendo negativo al aumentar el número de bits empleado para representarlo. Si se usara extensión por ceros, el bit de signo siempre terminaría en cero y el número sería positivo. Es por esto que lo que se hace es copiar el bit de signo a todos los bits que se añaden. Esto es conocido como extensión por signo. La tabla muestra un par de ejemplos de como se convierten dos números de 8 bits a 16 bits con extensión por ceros y por signo.

Número binario de 8 bits	Extensión por ceros	Extensión por signo
0100 1111	0000 0000 0100 1111	0000 0000 0100 1111
1001 1110	0000 0000 1001 1110	1111 1111 1001 1110

Datos multibyte

La mayoría de las computadoras tienen localidades de memoria de un byte (8 bits) de ancho. Cuando se requiere almacenar más de un byte, lo que se hace es utilizar más de una localidad de memoria. Existen dos ordenes en que se pueden almacenar estos datos. El primer orden es comenzar por almacenar el byte menos significativo del número que se desea almacenar en la localidad de dirección más baja. Por ejemplo, si se desea almacenar el número de dieciséis bits ABCDh a partir de la localidad de dirección 100h, se debe almacenar CDh en la localidad 100h y ABh en la localidad 101h. Este ordenamiento se conoce como “little endian”. El orden alternativo, es decir, almacenando el byte más significativo en la localidad de dirección más baja, es conocido como “big endian”. La

figura 5 muestra como se almacena el número de treinta y dos bits 12345678h siguiendo tanto el orden “little endian” como el “big endian”. Little endian es el orden más usado en la actualidad, tanto en las computadoras personales modernas con procesadores compatibles con Intel, como en muchos microcontroladores.

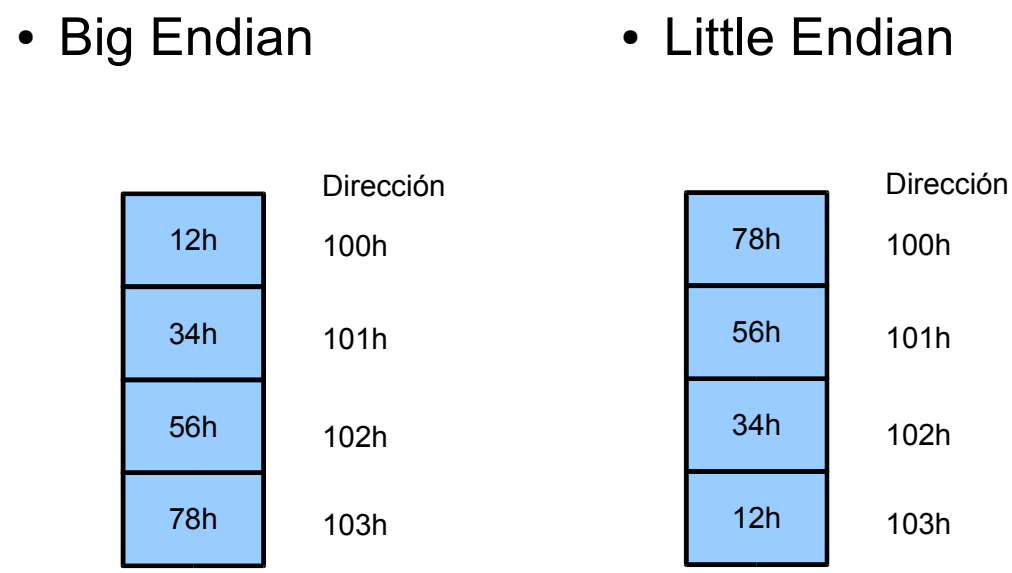

Figura 5: Ejemplo de los dos tipos de orden para datos multibyte.